LITEGUI: LIGHTWEIGHT GUI AGENTS VIA MULTI-SOLUTION GUIDED DISTILLATION AND DUAL-LEVEL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We present LiteGUI, a new framework for building lightweight GUI agents. GUI interaction poses unique challenges due to the long-horizon nature of complex tasks and the existence of multiple valid interaction paths, which are difficult for lightweight GUI agents to handle effectively. To address these challenges, LiteGUI introduces a two-stage post-training framework operating on logged GUI states. First, we propose *Guided Multi-Solution On-Policy Distillation*, which provides training-time privileged teacher guidance at each GUI state by selecting the most-matched valid action from human-verified multi-solution action annotations, while leaving the student's on-policy rollout unchanged. Second, we develop *Multi-Solution Dual-Level GRPO*, which combines action-level supervision with history-conditioned, per-state planning-quality supervision while accounting for multiple valid actions at each logged GUI state. Together, these stages support multi-step GUI tasks without requiring the student to reproduce fixed demonstration trajectories. We further develop a scalable data generation pipeline and a corresponding multi-solution dataset, *Lite-Dataset*, to support the training and evaluation of GUI agents, addressing a gap in the existing literature. Extensive experiments across multiple GUI benchmarks demonstrate that LiteGUI substantially improves the performance of lightweight GUI agents over existing training paradigms, including SFT, on-policy distillation and GRPO, and achieves competitive or superior performance compared with substantially larger models.

1 INTRODUCTION

Automated computer use has received increasing attention in recent years (Hurst et al., 2024; Parker Lyman, 2025; anthropic, 2026; Team et al., 2023; Bai et al., 2023; 2025a). Among these approaches, GUI agents represent a specialized class of systems that leverage visual understanding to interpret and execute computer-based tasks. While prior work has primarily focused on building GUI agents using large-scale models (Qin et al., 2025; Liu et al., 2025a; Agashe et al., 2025), the development of lightweight models [1] for efficient deployment is challenging and remains largely underexplored. Smaller models are typically considered insufficient for handling such complex tasks due to their limited capacity. We observe that the commonly used supervised fine-tuning (SFT) paradigm—particularly for smaller models—tends to overfit to domain-specific and fixed interaction trajectories. This limitation degrades zero-shot generalization and hinders effective error recovery in dynamic UI environments. In this paper, we propose a new approach, named **LiteGUI**, to improve the performance of lightweight GUI agents through on-policy distillation and reinforcement learning.

While distillation has been successfully applied to tasks such as summarization, translation, and arithmetic reasoning (e.g., GKD (Agarwal et al., 2024; Chen et al., 2025; Shenfeld et al., 2026)), its effectiveness in GUI agent settings remains limited. This is largely due to the higher task complexity and the relatively weaker performance of available teacher models. To address the challenges, we propose **Guided On-Policy Distillation (Guided-OPD)**, to ensure that the teacher model provides reliable guidance by conditioning on training-time privileged information derived from human-verified valid-action annotations. Furthermore, to account for the multi-solution nature of GUI tasks,

[1] By a lightweight model, we refer to models in the 2-3B parameter range.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the proposed method compares the student's on-policy output with the valid-action set for the current GUI state and selects the **most-matched valid action**, providing adaptive action-level guidance without imposing a single fixed target completion at each logged state.

In addition, careful design of reinforcement learning (RL) strategies is essential to address the challenges inherent in GUI-based tasks. Existing RL approaches are often limited by their reliance on single-solution rewards, lack of long-horizon planning, and susceptibility to losing state context during execution. To overcome these limitations, we propose a **Multi-Solution Dual-Level GRPO**. First the method includes **dual-level** reward framework that jointly supports long-term planning and short-term execution. At the macro level, we introduce robust, model-evaluated subtask rewards that provide structured guidance for decomposing complex, long-horizon tasks. At the micro level, by integrating real-time state perception and trajectory recording, the framework dynamically aligns low-level execution outcomes with an equivalent **multi-solution** set, thereby mitigating the "false negative" penalties induced by rigid single-label supervision, and thus improving the execution precision while robustly preserving generalization across diverse and complex UI scenarios. In summary, the core contributions of this paper are:

We propose **LiteGUI**, a novel framework for building lightweight GUI agents that addresses the unique challenges of GUI interaction tasks, including the long-horizon nature of complex tasks, the existence of multiple valid interaction paths, and the limited availability of capable models and high-quality training data. Specifically, we propose (1) **Most-Matched Guided On-Policy Distillation**, which selects the valid action most closely aligned with the student's current output from the human-verified valid-action set, using it only as training-time privileged teacher guidance to reduce hallucinations and supervision–state mismatch; and (2) **Multi-Solution Dual-Level GRPO**, which integrates robust model-evaluated subtask-planning rewards with multi-solution action-set matching to jointly support long-horizon planning and short-horizon perception and execution. Notably, LiteGUI is **SFT-free**, avoiding the need for supervised fine-tuning on static trajectories and thereby mitigating catastrophic forgetting of foundational capabilities and policy overfitting in small-scale models;

Evaluations on ScreenSpot-Pro (Li et al., 2025), OS-World (Xie et al., 2024), and Lite-Bench demonstrate that our approach enables lightweight models to achieve substantial gains in GUI task success rates. Our LiteGUI model achieves strong performance among lightweight models, over existing training paradigms, including SFT, on-policy distillation and GRPO, and becomes competitive with much larger-scale models on the reported comparisons;

Finally, to support the training and evaluation of the distillation and RL exploration paradigms, we design an automated data generation pipeline and curate **Lite-Dataset**, comprising 30K GUI trajectories and 11K multi-solution annotations. To the best of our knowledge, it is the first available **multi-solution** GUI agent dataset. Besides, we also introduce **Lite-Bench**, a 160-task benchmark covering File System, Web, and Terminal. All resources will be publicly released for community.

2 Related Work

2.1 Vision-based GUI Agents

Early GUI agents predominantly relied on underlying system APIs or accessibility trees, which severely hindered their cross-platform generalization (Deng et al., 2023; Xie et al., 2024). Recently, the research paradigm has rapidly shifted toward vision-only agents, primarily bifurcating into two approaches: the direct application of powerful general-purpose Multimodal Large Language Models (MLLMs) (e.g., GPT-4o(Hurst et al., 2024), Claude 3.5 Sonnet, Qwen-VL(Bai et al., 2023; 2025a)), and the fine-tuning of domain-specific GUI vision models (e.g., CogAgent (Hong et al., 2024), UI-TARS (Qin et al., 2025), OpenCUA (Wang et al., 2025) and others (Lin et al., 2024; Yuan et al., 2025)) utilizing large-scale interaction data (Gou et al., 2025; Wu et al., 2024; Xu et al., 2025). While these models exhibit exceptional planning and execution performance in the GUI domain, their formidable capabilities incur prohibitive computational overhead during inference, severely impeding practical deployment and user experience. Consequently, developing lightweight agents (e.g., 2B/A3B-scale models) has become an urgent imperative to address these bottlenecks. However, the post-training of current lightweight models relies almost exclusively on traditional Supervised Fine-Tuning (SFT) (Hsieh et al., 2025; Xu et al., 2025) or Reinforcement Learning (RL) (Shen et al.,

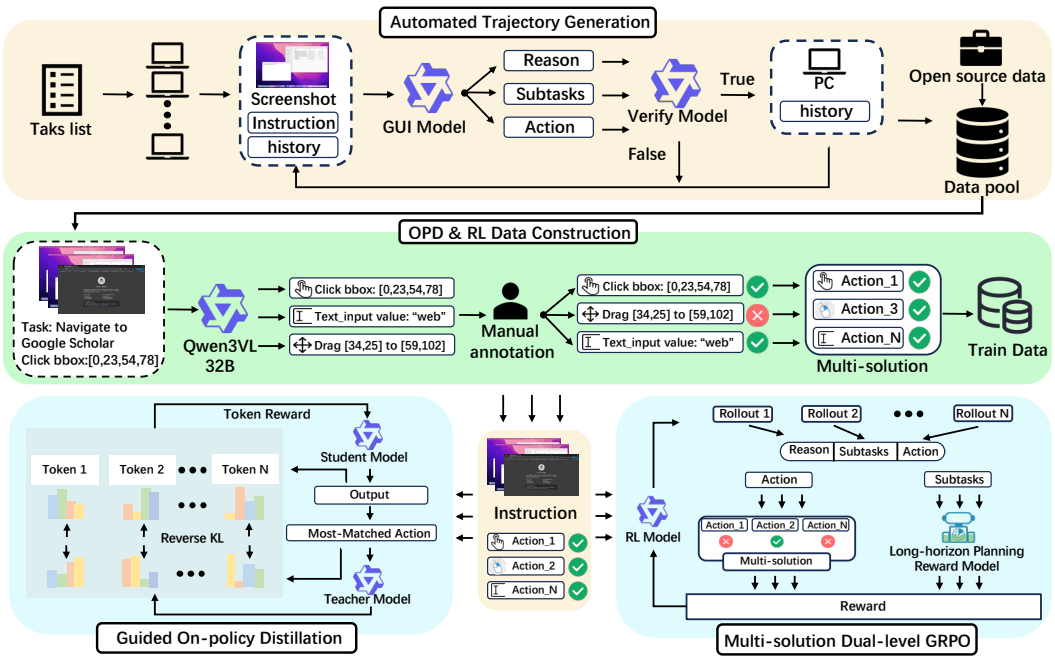

Figure 1: Overview of Lite-GUI. The framework consists of: (a) an automated GUI trajectory generation module for producing action trajectories for GUI tasks; (b) an OPD & RL data construction module for generating training data, with a particular focus on multi-solution annotations for OPD and RL; and (c) the proposed two-stage training paradigm, comprising Guided On-policy Distillation and Multi-solution Dual-level GRPO. Details of the framework can refer to Appendix A.7.

2025; Luo et al., 2025) methods utilizing naive coordinate rewards (e.g., GRPO). This coarse-grained training paradigm is precisely the root cause of the policy rigidity and capability degradation observed in small-scale models during long-horizon, multi-step exploration.

2.2 Post-training for GUI Agents

To improve the interaction capabilities of GUI agents, post-training paradigms are shifting from static imitation toward dynamic exploration. However, most existing approaches still rely on SFT over static trajectories, which suffers from covariate shift in multi-step decision-making (Ross et al., 2011), leading to rapid error accumulation once deviating from expert paths. Moreover, fitting long sequential data can disrupt the visual representation capabilities of lightweight models, resulting in catastrophic forgetting (Zeng et al., 2024). To address these limitations, recent work explores policy distillation methods (e.g., GKD (Agarwal et al., 2024), SDFT (Shenfeld et al., 2026), and others (Chen et al., 2025)), where teacher models provide real-time guidance during student exploration. However, such approaches remain underexplored in GUI agents, primarily due to the susceptibility of large models to hallucinations in "closed-book" settings (Alansari & Luqman, 2026; Huang et al., 2025), which introduce noisy supervision and degrade multi-step trajectory execution. We therefore propose Guided On-policy Distillation to leverage human-verified valid-action annotations to mitigate hallucinations and employ most-matched valid-action selection to accommodate the multi-solution nature of GUI tasks, enabling flexible action-level exploration. Furthermore, reinforcement learning (RL) methods (e.g., GRPO (Shao et al., 2024) and SAPO (Gao et al., 2025)) have been applied to vision-based models (e.g., VLM-R1 (Shen et al., 2025)). However, existing reward designs based on exact coordinate/action matching (Lu et al., 2025; Luo et al., 2025) suffer from sparsity and false negatives, limiting generalization. To address this, we introduce a multi-solution dual-level RL mechanism: macro-level subtask rewards evaluated by strong models, and micro-level dynamic matching over multi-solution actions, improving optimization in long-horizon exploration.

3 METHOD

We propose a two-stage training paradigm for GUI agents, shown in Fig.1. In the first stage, we introduce **Guided On-policy Distillation (Guided-OPD)**, which leverages multi-solution action annotations as training-time privileged information to provide a more stable supervisory signal for distillation. In the second stage, we introduce **Multi-solution Dual-level GRPO (MD-GRPO)**, which improves reinforcement learning by incorporating a multi-solution reward for action correctness and a history-conditioned, per-state reward for evaluating subtask-plan quality. To achieve this two-stage training paradigm, both distillation and reinforcement learning datasets are required, especially those with multi-solution annotations. We develop a data generation pipeline to semi-automatically produce multi-solution GUI task trajectories for training. Details of the pipeline are illustrated in Fig. 1. In the following subsections, we first introduce the task formulation of the GUI agent, followed by the details of Guided-OPD and MD-GRPO.

3.1 TASK FORMULATION

Given a user instruction q (e.g., lower the computer volume), the GUI model input x_t is denoted as:

$$x_t^{(w)} = (q, s_{t-w}, y_{t-w}, \ldots, s_{t-1}, y_{t-1}, s_t)\,, \tag{1}$$

where s_t denotes the GUI screenshot at time step t, y_t denotes the structured model output. w is the observation window size, i.e., the input consists of the user instruction, the current state, and the previous states and outputs within a history window of size w. The model output y_t contains three key components:

$$y_t = \{r_t, c_t, a_t\}, \tag{2}$$

where r_t denotes the `Reasoning` field, c_t denotes the `subtask` list, and a_t denotes the executable GUI action (e.g., the mouse and keyboard actions). The `Reasoning` field follows an Observation–Intent–Analysis structure. For example, it may describe the current UI observation, infer that the user intends to open a website, and conclude that the next step should be clicking the address bar. The `subtask` field serves as an explicit planning and memory state for the agent, provides a compact mechanism for carrying long-horizon task state across steps. For example, it may contain items such as "enter the target URL", "log into the account", and "check whether the homepage has loaded". These entries store task-level information that may not be fully recoverable from the latest screenshot alone, including subtask completion status, key intermediate milestones, and visual clues extracted from previous observations. Thus, the recent screenshots provide local perceptual context, while subtasks preserve higher-level progress information. We represent the executable action as

$$a_t = (\tau_t, p_t, v_t), \tag{3}$$

where τ_t is the action type, p_t is the position argument, and v_t is the text or key-value argument. For example, a click action can be represented as $\tau_t =$ `CLICK` with $p_t = (521, 41)$, while a text-input action can be represented as $\tau_t =$ `TEXT_INPUT` with $v_t =$ `www.example.com`. The model predicts a point $p_t = (x, y)$; for grounding supervision, the corresponding human annotation may specify a target bounding box $b^* = [x_{\min}, y_{\min}, x_{\max}, y_{\max}]$.

For each state x_t, we maintain a set of human-verified valid actions as ground truths:

$$\mathcal{A}_t^* = \{a_t^{*(1)}, a_t^{*(2)}, \ldots, a_t^{*(K)}\}, \tag{4}$$

which contains K multiple actions that can correctly advance the task from the current GUI state. It should be viewed as a finite approximation of the valid action space rather than an exhaustive set of all possible correct actions.

3.2 GUIDED ON-POLICY DISTILLATION

In standard Generalized Knowledge Distillation (GKD) or On-policy Distillation (OPD), the student first samples an output $\hat{a}_t$ conditioned on the current input x_t, and the teacher evaluates the sampled sequence under teacher forcing. This evaluation produces full next-token distributions at all valid completion prefixes. However, in GUI agent training, student outputs are often noisy: they may contain incorrect action types, inaccurate coordinates, wrong text arguments, or invalid JSON structures. If the teacher evaluates such outputs using only the default context, the resulting likelihood signal can be unstable. To mitigate this issue, we introduce a Guided On-policy Distillation (Guided

OPD) mechanism, which provides the teacher with additional action guidance constructed from the human-verified valid action set $\mathcal{A}_t^*$, We denote this teacher-side guidance as:

$$g_t = h(\mathcal{A}_t^*, \hat{a}_t), \tag{5}$$

where $h(\cdot)$ specifies how the annotated valid actions are converted into the teacher-side action guidance. In this way, the teacher distribution is conditioned on $\pi_T(\cdot \mid x_t, g_t)$, whereas the student policy remains $\pi_\theta(\cdot \mid x_t)$. Note that the guidance g_t is derived only from the current-step human-verified valid action set $\mathcal{A}_t^*$ and is used exclusively on the teacher side during training. It is never provided to the student model and is not used at inference time. Therefore, this framework can be viewed as a form of training-time privileged guidance, following the learning-using-privileged-information paradigm (Vapnik & Vashist, 2009; Vapnik & Izmailov, 2015), and as a privileged-information extension of on-policy distillation (Lopez-Paz et al., 2015; Agarwal et al., 2024). Different guidance variants $h(\cdot)$ correspond to different ways of mapping $\mathcal{A}_t^*$ into g_t. We consider three variants: Single-GT, Multi-GT and Most-Matched-GT Guided OPD in this paper.

(1) Single-GT Guided OPD. One valid action is randomly sampled from $\mathcal{A}_t^*$ and provided to the teacher as the reference action:

$$g_t = a_t^{*(k)}, \quad k \sim \mathrm{Uniform}\{1, \ldots, K\}. \tag{6}$$

This gives the teacher additional task constraints, but the randomly selected ground truth may be far from the student's current output, leading to a mismatch between the selected reference action and the student's current output.

(2) Multi-GT Guided OPD. The full valid action set is provided to the teacher:

$$g_t = \mathcal{A}_t^*. \tag{7}$$

This exposes the teacher to all annotated valid actions for the current state. However, it may also increase the complexity of the teacher context and lead to a more diffuse likelihood estimate across multiple possible actions.

(3) Most-Matched-GT Guided OPD. Given the student-generated output $\hat{y}_t$, we select the valid action that is most matched to the student's current on-policy behavior:

$$g_t = a_t^\dagger = \arg\max_{a^* \in \mathcal{A}_t^*} \phi_{\mathrm{gui}}(\hat{y}_t, a^*), \tag{8}$$

where ϕ_{gui} is the unified GUI action matching function defined as:

$$\phi_{\mathrm{gui}}(\hat{y}_t, a^*) = \frac{w_\tau m_\tau + \mathbb{I}[b^* \neq \emptyset] w_p m_p + \mathbb{I}[v^* \neq \emptyset] w_v m_v}{w_\tau + \mathbb{I}[b^* \neq \emptyset] w_p + \mathbb{I}[v^* \neq \emptyset] w_v}. \tag{9}$$

$m_\tau = \mathbb{I}[\hat{\tau}_t = \tau^*]$ measures action-type correctness, $m_p \in [0, 1]$ measures the alignment between the predicted position $\hat{p}_t$ and the target bounding box b^*, and $m_v \in [0, 1]$ measures exact or partial matching between the predicted value $\hat{v}_t$ and the annotated value v^*. The indicator terms ensure that position and value scores are only considered when the corresponding annotations are available. Details of m_τ, m_p, m_v, and corresponding coefficients w_τ, w_p, w_v are provided in Appendix A.2. If the student output cannot be parsed into a valid GUI action, all candidate actions receive zero matching scores, and we fall back to a deterministic reference action from $\mathcal{A}_t^*$. This selection mechanism provides the teacher with a reference action that is aligned with the student's current on-policy output. Compared with Single-GT, it reduces the mismatch caused by random or fixed reference selection. Compared with Multi-GT, it provides a more focused reference signal while avoiding an overly complex teacher context.

Full-vocabulary reverse-KL OPD. After obtaining g_t, we construct the input of teacher model as

$$\tilde{x}_t = [q, s_{t-w}, y_{t-w}, \ldots, s_{t-1}, y_{t-1}, s_t, \texttt{reference_action} = g_t]. \tag{10}$$

The student samples a completion $\hat{y}_t \sim \pi_\theta(\cdot \mid x_t)$. The sampled token sequence and the discrete teacher-side guidance g_t are treated as fixed during the optimization step. Let $\mathcal{M}_t$ denote the non-masked completion positions. At each valid position $j \in \mathcal{M}_t$, we compute the full-vocabulary student and teacher distributions:

$$p_{\theta,j} = \pi_\theta(\cdot \mid x_t, \hat{y}_{t,<j}), \qquad p_{T,j}^{g_t} = \pi_T(\cdot \mid \tilde{x}_t, \hat{y}_{t,<j}). \tag{11}$$

Following GKD (Agarwal et al., 2024), the objective is the full-vocabulary, per-prefix divergence:

$$\mathcal{L}_{\mathrm{OPD}} = \frac{1}{|\mathcal{M}_t|} \sum_{j \in \mathcal{M}_t} D_\beta \left(p_{\theta,j} \,\middle\|\, \mathrm{sg}\left[p_{T,j}^{q_t} \right] \right), \qquad \beta = 1, \tag{12}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient (identity in the forward pass, with zero gradient), and D_β denotes the generalized JSD/KL family used by our implementation. Under our parameterization, $\beta = 1$ selects reverse KL:

$$D_1(p\|q) = D_{\mathrm{KL}}(p\|q) = \sum_v p(v) \log \frac{p(v)}{q(v)}. \tag{13}$$

The loss is computed over the full vocabulary, rather than only the sampled token, with gradients flowing only through the student distribution. See Appendix A.5 for implementation details.

3.3 Multi-solution Dual-level GRPO

We further optimize the student using GRPO. Unlike standard single-answer tasks, GUI control exhibits strong multi-solution equivalence: under the same GUI state, multiple actions may validly advance the task. For example, a file can be opened by double-clicking it or by selecting it and pressing Enter; a web page can be reached by clicking a shortcut or directly typing the URL. Therefore, a reward based on a single ground-truth action may incorrectly penalize valid alternative actions.

We focus on two reward-design questions that are central to GUI agent training: how to avoid false negatives caused by single-reference supervision, and how to evaluate whether the model understands the current task stage and can re-plan when necessary. We address these questions using the **multi-solution action reward** and **planning reward**. For each logged input x_t, the behavior policy samples G candidate outputs:

$$\{\hat{y}_t^{(1)}, \hat{y}_t^{(2)}, \ldots, \hat{y}_t^{(G)}\} \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid x_t). \tag{14}$$

Training uses logged GUI states rather than live environment rollouts: all candidates are scored at the same state using the trajectory context and human-verified annotations. The total reward for the i-th candidate is

$$R_t^{(i)} = \lambda_{ms} R_{\mathrm{ms}}(\hat{y}_t^{(i)}, \mathcal{A}_t^*) + \lambda_{sub} R_{\mathrm{sub}}(\hat{y}_t^{(i)}) + \lambda_{format/length} R_{\mathrm{format/length}}(\hat{y}_t^{(i)}), \tag{15}$$

where R_{ms} is the multi-solution GUI action reward, R_{sub} is the planning reward, and $R_{\mathrm{format/length}}$ is the JSON-aware soft overlength reward. λ_{ms}, λ_{sub} and $\lambda_{format/length}$ are the coefficients for each reward term. Detailed hyperparameter settings can be found in Appendix A.6.

Multi-solution Action Reward. Given the multi-solution action set $\mathcal{A}_t^*$, we compare the model output against all valid candidate actions and take the maximum score as:

$$R_{\mathrm{ms}}(\hat{y}_t, \mathcal{A}_t^*) = \max_{a^* \in \mathcal{A}_t^*} \phi_{\mathrm{gui}}(\hat{y}_t, a^*). \tag{16}$$

where ϕ_{gui} is the unified GUI action matching function defined in Eq. 9. This changes the reward criterion from "does the model reproduce this specific annotated action?" to "does the model match any acceptable action?" Since a single annotation is usually only one sample from the space of valid GUI actions, max-over-solutions substantially reduces false negative rewards caused by fixed annotation paths. From an optimization perspective, single-ground-truth rewards can incorrectly convert differences among equivalent action paths into negative feedback, discouraging exploration of valid alternatives. In contrast, the multi-solution reward gives positive feedback when the model output matches any annotated acceptable action for the current logged state.

Planning Reward for Long-Horizon Tasks. The multi-solution action reward provides action-level supervision by matching each candidate output against annotated acceptable actions for the current logged GUI state. For multi-step GUI tasks, a complementary planning signal can assess whether a proposed subtask plan reflects task progress and the intended next step. We therefore introduce a history-conditioned, per-state planning-quality reward for this purpose. The reward scores the current candidate subtask plan c_t, which compactly represents task progress, using the available trajectory context. It therefore evaluates intermediate plan quality at the current logged state rather than the outcome of executing a candidate action. The frozen VLM judge uses the task, available screenshot history, and previous model outputs as context and returns a score:

$$R_{\mathrm{sub}} = f_{\mathrm{judge}}(q, s_{t-w:t}, y_{t-w:t-1}, c_t), \quad R_{\mathrm{sub}} \in [0, 1]. \tag{17}$$

It evaluates the plan along four dimensions: task relevance and adaptability, visual grounding, decomposition granularity, and state consistency. In particular, when the latest screenshot indicates that the agent is stuck, in an error state, or in a loop, the judge is instructed to assign a low score unless the subtask plan explicitly reflects the abnormal state and proposes a corrective next step. Together, the two rewards provide action-level supervision and history-conditioned plan-quality supervision.

GRPO Objective. For the G candidate outputs sampled from the same input, we normalize rewards within the group to obtain the relative advantage:

$$\hat{A}_t^{(i)} = \frac{R_t^{(i)} - \text{mean}(\{R_t^{(j)}\}_{j=1}^{G})}{\text{std}(\{R_t^{(j)}\}_{j=1}^{G}) + \epsilon_{\text{adv}}}. \tag{18}$$

Here ϵ_{adv} is the numerical stabilizer for reward normalization and is distinct from the clipping bounds ϵ_{low} and ϵ_{high}. We then optimize the policy using the GRPO clipped objective:

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{G}\sum_{i=1}^{G} \min\left(\rho_t^{(i)}\hat{A}_t^{(i)}, \text{clip}\left(\rho_t^{(i)}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\right)\hat{A}_t^{(i)}\right) + \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{19}$$

where the sequence-level policy ratio is computed from the mean token log-probability difference:

$$\rho_t^{(i)} = \exp\left[\frac{1}{|\mathcal{M}_t^{(i)}|}\sum_{j\in\mathcal{M}_t^{(i)}}\left(\log \pi_\theta(\hat{y}_{t,j}^{(i)} \mid x_t, \hat{y}_{t,<j}^{(i)}) - \log \pi_{\theta_{\text{old}}}(\hat{y}_{t,j}^{(i)} \mid x_t, \hat{y}_{t,<j}^{(i)})\right)\right]. \tag{20}$$

Here, $\mathcal{M}_t^{(i)}$ denotes the valid completion-token positions, $\pi_{\theta_{\text{old}}}$ is the behavior policy used to sample candidates, and π_{ref} is the reference model. Detailed hyperparameter settings are in Appendix A.6.

4 Experiments

4.1 Experimental Setup

We implement Lite-GUI using two representative backbones: Qwen3-VL-2B and Qwen3-VL-30B-A3B, and employ Qwen3-VL-32B as the teacher model. To implement Lite-GUI, a multi-solution dataset is required. However, no such dataset is currently available in the community. We therefore develop an automated data pipeline (illustrated in Figure 1), to construct Lite-Dataset, which contains 11K multi-solution samples. More details about the data construction pipeline and Lite-Dataset are provided in the Appendix A.7 and A.8. In MD-GRPO, we use Qwen3-VL-32B-Instruct as the sub-task planning judge because it is community-friendly while can still provide strong alignment with human judgments, achieving 90.0% human agreement from our study. Its prompt, calibration metrics, and limitations are provided in Appendix A.4. All models are trained on a cluster of NVIDIA A100 GPUs. Details on hyper-parameters are provided in Appendix A.5 and A.6.

We evaluate Lite-GUI on ScreenSpot-Pro (Li et al., 2025) and OS-World (Xie et al., 2024), two of the most widely used benchmarks for GUI agents, alongside Lite-Bench, our newly curated benchmark. Lite-Bench comprises 160 tasks (up to 18 steps), featuring 75 web tasks, 38 terminal tasks, and 47 file system tasks that strictly prohibit terminal usage to enforce pure vision-based interaction. It is worth noting that Lite-Bench is in a similar domain to Lite-Dataset, but with no overlapping data. Details of Lite-Bench can be referred to Appendix A.9. Each reported result for our models and ablations is the arithmetic mean over five independent runs. Below, we first compare Lite-GUI with widely used training paradigms and then compare our models with state-of-the-art (SOTA) models.

4.2 Effectiveness of LiteGUI

We first compare Lite-GUI with widely used training paradigms for GUI agents, including SFT, GRPO, and OPD, as well as ablations of individual components of Lite-GUI, across all three benchmarks in Table 1. To ensure a fair comparison, all training methods use the same Qwen3-VL-2B backbone and the same Lite-Dataset. The reported results are averaged over five independent runs. We observe that SFT actually degrades performance on both ScreenSpot-Pro and OS-World, potentially due to overfitting and catastrophic forgetting in the lightweight GUI-agent setting. In contrast, Lite-GUI (last row) achieves the best performance across all three benchmarks.

Table 1: Effectiveness of LiteGUI. The best results are highlighted in bold.

Method	Multi-Solution	Dual-level	Benchmark		
			ScreenSpot-Pro	OS-World	Lite-Bench
Qwen3-VL-2B (Bai et al., 2025a)	-	-	40.16	6.04	32.35
+ SFT	-	-	31.56	3.25	37.96
+ SFT + GRPO	no	no	33.47	3.77	42.65
+ OPD	-	-	41.87	9.32	51.47
+ Single-Guided OPD	-	-	41.68	11.04	51.76
+ Multi-Guided OPD	-	-	42.25	11.11	52.94
+ Most-Matched-Guided OPD	-	-	42.50	11.56	55.88
+ Most-Matched-Guided OPD + GRPO	no	no	42.37	6.56	52.94
	no	yes	42.50	8.13	59.20
	yes	no	42.69	11.84	60.58
	yes	yes	**43.13**	**13.24**	**63.03**

Table 2: Performance comparison on ScreenSpot-Pro.

Model	ScreenSpot-Pro Accuracy (%)												Avg.
	CAD		Dev		Creative		Scientific		Office		OS		
	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	
Proprietary Models													
GPT-4o (Hurst et al., 2024)	2	0	1.3	0	1	0	2.1	0	1.1	0	0	0	0.8
Claude Computer Use (anthropic, 2026)	14.5	3.7	22	3.9	25.9	3.4	33.9	15.8	30.1	16.3	11	4.5	17.1
General Open-source Models													
Qwen2.5-VL-3B (Bai et al., 2025b)	9.1	7.3	22.1	1.4	26.8	2.1	38.2	7.3	33.9	15.1	10.3	1.1	16.1
Qwen2.5-VL-7B (Bai et al., 2025b)	16.8	1.6	46.8	4.1	35.9	7.7	49.3	7.3	52.5	20.8	37.4	6.7	26.8
Qwen3-VL-2B (Bai et al., 2025a)	52.79	15.62	50.0	10.34	48.48	18.18	59.02	23.63	58.19	33.96	51.40	21.34	40.16
Qwen3-VL-32B (Bai et al., 2025a)	71.06	35.93	76.62	24.13	72.72	29.37	84.72	31.81	85.31	54.71	68.22	15.73	58.57
GUI-specific Models (SFT)													
SeeClick-9.6B (Cheng et al., 2024)	2.5	0	0.6	0	1	0	3.5	0	1.1	0	2.8	0	1.1
CogAgent-18B (Hong et al., 2024)	7.1	3.1	14.9	0.7	9.6	0	22.2	1.8	13	0	5.6	0	7.7
Aria-UI (Yang et al., 2024)	7.6	1.6	16.2	0	23.7	2.1	27.1	6.4	20.3	1.9	4.7	0	11.3
OS-Atlas-7B (Wu et al., 2024)	12.2	4.7	33.1	1.4	28.8	2.8	37.5	7.3	33.9	5.7	27.1	4.5	18.9
ShowUI-2B (Lin et al., 2024)	2.5	0	16.9	1.4	9.1	0	13.2	7.3	15.3	7.5	10.3	2.2	7.7
UGround-V1-7B(Gou et al., 2025)	15.8	1.2	51.9	2.8	47.5	9.7	57.6	14.5	60.5	13.2	38.3	7.9	31.1
UI-TARS-2B (Qin et al., 2025)	17.8	4.7	47.4	4.1	42.9	6.3	56.9	17.3	50.3	17	21.5	5.6	27.7
UI-TARS-72B (Qin et al., 2025)	18.8	12.5	62.9	17.2	57.1	15.4	64.6	20.9	63.3	26.4	42.1	15.7	38.1
JEDI-3B (Xie et al., 2025)	27.4	9.4	61	13.8	53.5	8.4	54.2	18.2	64.4	32.1	38.3	9	36.1
JEDI-7B (Xie et al., 2025)	38	14.1	42.9	11	50	11.9	72.9	25.5	75.1	47.2	33.6	16.9	39.5
GUI-Actor-7B (Wu et al., 2025)	-	-	-	-	-	-	-	-	-	-	-	-	44.6
GUI-specific Models (RL)													
UI-R1-E-3B (Lu et al., 2025)	37.1	12.5	46.1	6.9	41.9	4.2	56.9	21.8	65	26.4	32.7	10.1	33.5
GUI-R1-3B (Luo et al., 2025)	26.4	7.8	33.8	4.8	40.9	5.6	61.8	17.3	53.6	17	28.1	5.6	-
GUI-R1-7B (Luo et al., 2025)	23.9	6.3	49.4	4.8	38.9	8.4	55.6	11.8	58.7	26.4	42.1	16.9	-
InfigUI-R1-3B (Liu et al., 2025b)	33	14.1	51.3	12.4	44.9	7	58.3	20	65.5	28.3	43.9	12.4	35.7
GUI-G1-3B (Zhou et al., 2025)	39.6	9.4	50.7	10.3	36.6	11.9	61.8	30	67.2	32.1	23.5	10.6	37.1
SE-GUI-3B (Yuan et al., 2025)	38.1	12.5	55.8	7.6	47	4.9	61.8	16.4	59.9	24.5	40.2	12.4	35.9
Ours													
Lite-GUI-2B	61.92	20.31	59.09	14.48	59.59	22.37	68.75	25.45	65.53	28.30	59.81	24.71	46.86
Lite-GUI-30B-A3B	74.61	20.31	76.62	29.65	70.20	21.67	79.16	33.63	80.79	50.94	79.43	39.32	**58.95**

A closer examination of the ablations shows that Guided-OPD outperforms vanilla OPD, while Most-Matched Guided-OPD achieves the best, demonstrating the benefit of using the guided supervision. Moreover, incorporating either multi-solution or dual-level rewards in GRPO further improves performance, while combining all components achieves the strongest overall results. Notably, OS-World and Lite-Bench contain more complex, long-horizon GUI agent tasks, whereas ScreenSpot-Pro focuses primarily on visual grounding and single-step interaction. Consequently, Lite-GUI yields larger gains on OS-World and Lite-Bench, highlighting the effectiveness of guided planning and multi-solution learning for complex, long-horizon GUI interaction.

We further conduct a detailed analysis of Guided-OPD. Across 50 multi-solution validation states, privileged guidance improves the teacher's discrimination between valid and invalid GUI actions, increasing pairwise preference accuracy from 46% to 77% and yielding a cleaner supervision signal for OPD. Guided-OPD also improves the validity and diversity of the student's generated actions without causing policy collapse toward a narrow subset of valid solutions. Overall, Most-Matched-GT strengthens both teacher supervision and student action generation while preserving coverage of multiple valid interaction paths. Further details are provided in Appendix A.10.

4.3 Comparison with SOTA Methods

We then compare our Lite-GUI models with state-of-the-art (SOTA) approaches on ScreenSpot-Pro, OS-World, and Lite-Bench in Tables 2, 3, and 4, respectively. We train both Qwen3-VL-2B and Qwen3-VL-30B-A3B using Lite-GUI. For ScreenSpot-Pro, we build upon the widely used ShowUI-Desktop-8K (Lin et al., 2024) dataset, which is publicly available for this task. We augment it with multi-solution annotations generated using our data pipeline, rather than using our Lite-Dataset. For

Table 3: Performance comparison on OS-World.

Model	Type	Open source	Success Rate (Avg %)
Claude-sonnet-4-5 (anthropic, 2026)	General	False	42.9
doubao-1-5-thinking-vision-pro (Guo et al., 2025)	Specialized	False	39
GPT-o3 (openai, 2025)	General	False	9.1
Qwen2.5-VL-32B-instruct (Bai et al., 2025b)	General	True	3.0
Qwen2.5-VL-72B-instruct (Bai et al., 2025b)	General	True	4.4
Kimi-VL-A3B (Team et al., 2025)	General	True	9.7
Qwen3-VL-2B-Instruct (Bai et al., 2025a)	General	True	6.04
OpenCUA-A3B (Wang et al., 2025)	Specialized	True	16.9
Qwen3-VL-30B-A3B-Instruct (Bai et al., 2025a)	General	True	16.7
OpenCUA-Qwen2-7B (Wang et al., 2025)	Specialized	True	19.9
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized	True	24
Lite-GUI-2B	Specialized	True	13.24
Lite-GUI-30B-A3B	Specialized	True	22.7

Table 4: Performance comparison on Lite-Bench.

Model	Success Rate (%)			
	File System	Web	Terminal	All
Qwen3-VL-2B (Bai et al., 2025a)	23	35	41	32.35
Qwen3-VL-30B-A3B (Bai et al., 2025a)	50	53	92	61.34
Qwen3-VL-32B (Bai et al., 2025a)	85	82	95	85.88
Lite-GUI-2B	40.42	69.33	78.94	63.13
Lite-GUI-30B-A3B	**89.36**	**84.00**	**97.36**	**88.75**

OS-World and Lite-Bench, no publicly available training datasets are provided for these benchmarks. Therefore, we use our Lite-Dataset to train the models on both benchmarks.

As shown in Table 2, On ScreenSpot-Pro, Lite-GUI-2B achieves an average accuracy of 46.86%, setting a new SOTA among models of the same parameter scale. Notably, it also surpasses significantly larger specialized models, including UI-TARS-72B (38.1%). Remarkably, while existing methods typically rely on extensive training data, Lite-GUI-2B attains these strong results using only the 8K-sample ShowUI-Desktop-8K dataset. This high data efficiency highlights the effectiveness of our LiteGUI framework in GUI reasoning. Furthermore, Lite-GUI-30B-A3B achieves SOTA performance (58.95%) across all compared models, demonstrating the effectiveness of our approach.

From Table 3, Lite-GUI-2B achieves a 13.24% success rate on OSWorld, more than doubling the performance of its backbone Qwen3-VL-2B (6.04%). Lite-GUI-30B-A3B further pushes the performance to 22.7%, achieving the best performance among models of the same parameter scale while rivaling top-tier specialized models with substantially larger parameters (72B). It is also worth noting that, without training on large-scale datasets as in OpenCUA and UI-TARS, our Lite-GUI model can still achieve comparable results, demonstrating the data efficiency of our method. On Lite-Bench (Table 4), Lite-GUI-2B achieves a 63.13% success rate, representing nearly a $2\times$ improvement over the vanilla baseline (32.35%). These substantial gains validate the effectiveness of our approach, with similar trends observed for Lite-GUI-30B-A3B, which achieves a success rate of up to 88.75%.

5 Conclusion

We have proposed Lite-GUI, a new training paradigm for building lightweight GUI agents. Lite-GUI leverages multi-solution guided distillation and dual-level GRPO to address the unique challenges of GUI tasks, which are often complex, long-horizon, and inherently multi-path, while also facing a lack of strong teacher models at comparable scales. Our approach achieves substantial improvements over existing baselines, reaching SOTA performance among models at comparable scales and delivering competitive results with significantly larger models across all benchmarks. We have also developed a scalable, automated pipeline for generating multi-solution GUI data, resulting in the 11K-sample Lite-Dataset, which is currently lacking in the community. Future work will explore incorporating online interaction with real-world environments to further improve agent performance, as well as further scaling down the model size to enable more practical on-device deployment.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks that require disclosure: generating synthetic GUI interaction data using the pipeline in Appendix A.7, assisting with translation of manuscript text, and assisting with the implementation of methods and with the creation or editing of code. We have not used generative AI tools to develop theoretical models or conceptual frameworks, to formulate or prove mathematical claims, to propose or refine hypotheses, to design or provide feedback on research methodology or experiments, to clean or reformat datasets, to support qualitative or thematic data analysis, or to interpret results; these tasks are therefore not applicable to this work. Additionally, we used generative AI tools for a task with recommended disclosure, namely drafting and editing manuscript sections for readability. All AI-assisted outputs were reviewed by the authors: AI-assisted code was tested for correctness, and AI-assisted text was checked against the cited literature and our experimental results. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies GUI agents that control a computer from screenshots. Such systems can improve productivity and accessibility, but they can also perform unintended operations, leak information visible on screen, or be misused for unauthorized automation. Our training data consist of GUI screenshots and synthetic interaction trajectories; we remove sensitive content during data construction and release only assets that we are permitted to distribute. Practical deployment should include permission control, confirmation of sensitive actions, execution logging, and privacy filtering. Our experiments involve no human subjects. A detailed discussion of limitations, broader impacts, and asset licenses is provided in Appendix A.12 and Appendix A.13.

REPRODUCIBILITY STATEMENT

We describe the task formulation and the two training stages in Section 3, and the benchmark setup and evaluation protocols in Section 4. Implementation details, hyperparameters, the judge prompt, and the data pipeline are given in Appendix A.5, Appendix A.6, Appendix A.4, and Appendix A.7. We will publicly release the Lite-Dataset, Lite-Bench, and the training and evaluation code to support reproduction.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

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*, 2:10–16, 2025.

Aisha Alansari and Hamzah Luqman. Large language models hallucination: A comprehensive survey. *Computer Science Review*, 61:100970, 2026.

anthropic. claude-3-family. https://www.anthropic.com/news/claude-3-family, 2026.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren,

Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

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*, 2025b.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting, 2025. *URL https://arxiv. org/abs/2510.18874*, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.505.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Chang Gao, Chujie Zheng, Xiong-Hui Chen, Kai Dang, Shixuan Liu, Bowen Yu, An Yang, Shuai Bai, Jingren Zhou, and Junyang Lin. Soft adaptive policy optimization, 2025. URL https://arxiv.org/abs/2511.20347.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, Jingji Chen, Jingjia Huang, Kang Lei, Liping Yuan, Lishu Luo, Pengfei Liu, Qinghao Ye, Rui Qian, Shen Yan, Shixiong Zhao, Shuai Peng, Shuangye Li, Sihang Yuan, Sijin Wu, Tianheng Cheng, Weiwei Liu, Wenqian Wang, Xianhan Zeng, Xiao Liu, Xiaobo Qin, Xiaohan Ding, Xiaojun Xiao, Xiaoying Zhang, Xuanwei Zhang, Xuehan Xiong, Yanghua Peng, Yangrui Chen, Yanwei Li, Yanxu Hu, Yi Lin, Yiyuan Hu, Yiyuan Zhang, Youbin Wu, Yu Li, Yudong Liu, Yue Ling, Yujia Qin, Zanbo Wang, Zhiwu He, Aoxue Zhang, Bairen Yi, Bencheng Liao, Can Huang, Can Zhang, Chaorui Deng, Chaoyi Deng, Cheng Lin, Cheng Yuan, Chenggang Li, Chenhui Gou, Chenwei Lou, Chengzhi Wei, Chundian Liu, Chunyuan Li, Deyao Zhu, Donghong Zhong, Feng Li, Feng Zhang, Gang Wu, Guodong Li, Guohong Xiao, Haibin Lin, Haihua Yang, Haoming Wang, Heng Ji, Hongxiang Hao, Hui Shen, Huixia Li, Jiahao Li, Jialong Wu, Jianhua Zhu, Jianpeng Jiao, Jiashi Feng, Jiaze Chen, Jianhui Duan, Jihao Liu, Jin Zeng, Jingqun Tang, Jingyu Sun, Joya Chen, Jun Long, Junda Feng, Junfeng Zhan, Junjie Fang, Junting Lu, Kai Hua, Kai Liu, Kai Shen, Kaiyuan Zhang, Ke Shen, Ke Wang, Keyu Pan, Kun Zhang, Kunchang Li, Lanxin Li, Lei Li, Lei Shi, Li Han, Liang Xiang, Liangqiang Chen, Lin Chen, Lin Li, Lin Yan, Liying Chi, Longxiang Liu, Mengfei Du, Mingxuan Wang, Ningxin Pan, Peibin Chen, Pengfei Chen, Pengfei Wu, Qingqing Yuan, Qingyao Shuai, Qiuyan Tao, Renjie Zheng, Renrui Zhang, Ru Zhang, Rui Wang, Rui Yang, Rui Zhao, Shaoqiang Xu, Shihao Liang, Shipeng Yan, Shu Zhong, Shuaishuai Cao, Shuangzhi Wu, Shufan Liu, Shuhan Chang, Songhua Cai, Tenglong Ao, Tianhao Yang, Tingting Zhang, Wanjun Zhong, Wei Jia, Wei Weng, Weihao Yu, Wenhao Huang, Wenjia Zhu, Wenli Yang, Wenzhi Wang, Xiang Long, XiangRui Yin, Xiao Li, Xiaolei Zhu, Xiaoying Jia, Xijin Zhang, Xin Liu, Xinchen Zhang, Xinyu Yang, Xiongcai Luo, Xiuli Chen, Xuantong Zhong, Xuefeng Xiao, Xujing Li, Yan Wu, Yawei Wen, Yifan Du, Yihao Zhang, Yining Ye, Yonghui Wu, Yu Liu, Yu Yue, Yufeng Zhou, Yufeng Yuan, Yuhang Xu, Yuhong Yang, Yun Zhang, Yunhao Fang, Yuntao Li, Yurui Ren, Yuwen Xiong, Zehua Hong, Zehua Wang, Zewei Sun, Zeyu Wang, Zhao Cai, Zhaoyue Zha, Zhecheng An, Zhehui Zhao, Zhengzhuo Xu, Zhipeng Chen, Zhiyong Wu, Zhuofan Zheng, Zihao Wang, Zilong Huang, Ziyu Zhu, and Zuquan Song. Seed1.5-vl technical report, 2025. URL https://arxiv.org/abs/2505.07062.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14281–14290, 2024.

ZongHan Hsieh, Tzer-Jen Wei, and ShengJing Yang. Zonui-3b: A lightweight vision-language model for cross-resolution gui grounding. https://arxiv.org/abs/2506.23491, 2025. arXiv:2506.23491 [cs.CV], version 2, last revised 1 Jul 2025.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Kaixin Li, Meng Ziyang, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=XaKNDIAHas.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent, 2024. URL https://arxiv.org/abs/2411.17465.

Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, et al. Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc. *arXiv preprint arXiv:2502.14282*, 2025a.

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*, 2025b.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing 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.

openai. Openai o3. https://openai.com, 2025.

Tao Zhang Parker Lyman. Manus academy. https://manus.im, 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.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

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.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, Ruochen Xu, and Tiancheng Zhao. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*, 2025.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, Congcong Wang, Dehao Zhang, Dikang Du, Dongliang Wang, Enming Yuan, Enzhe Lu, Fang Li, Flood Sung, Guangda Wei, Guokun Lai, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haoning Wu, Haotian Yao, Haoyu Lu, Heng Wang, Hongcheng Gao, Huabin Zheng, Jiaming Li, Jianlin Su, Jianzhou Wang, Jiaqi Deng, Jiezhong Qiu, Jin Xie, Jinhong Wang, Jingyuan Liu, Junjie Yan, Kun Ouyang, Liang Chen, Lin Sui, Longhui Yu, Mengfan Dong, Mengnan Dong, Nuo Xu, Pengyu Cheng, Qizheng Gu, Runjie Zhou, Shaowei Liu, Sihan Cao, Tao Yu, Tianhui Song, Tongtong Bai, Wei Song, Weiran He, Weixiao Huang, Weixin Xu, Xiaokun Yuan, Xingcheng Yao, Xingzhe Wu, Xinxing Zu, Xinyu Zhou, Xinyuan Wang, Y. Charles, Yan Zhong, Yang Li, Yangyang Hu, Yanru Chen, Yejie Wang, Yibo Liu, Yibo Miao, Yidao Qin, Yimin Chen, Yiping Bao, Yiqin Wang, Yongsheng Kang, Yuanxin Liu, Yulun Du, Yuxin Wu, Yuzhi Wang, Yuzi Yan, Zaida Zhou, Zhaowei Li, Zhejun Jiang, Zheng Zhang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Zijia Zhao, and Ziwei Chen. Kimi-VL technical report, 2025. URL https://arxiv.org/abs/2504.07491.

Vladimir Vapnik and Rauf Izmailov. Learning using privileged information: similarity control and knowledge transfer. *The Journal of Machine Learning Research*, 16(1):2023–2049, 2015.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural networks*, 22(5-6):544–557, 2009.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. Opencua: Open foundations for computer-use agents, 2025. URL https://arxiv.org/abs/2508.09123.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

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

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

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling computer-use grounding via user interface decomposition and synthesis, 2025. URL https://arxiv.org/abs/2505.13227.

Yibin Xu, Liang Yang, Hao Chen, Hua Wang, Zhi Chen, and Yaohua Tang. Deskvision: Large scale desktop region captioning for advanced gui agents. *arXiv preprint arXiv:2503.11170*, 2025.

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

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.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 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.

A APPENDIX

A.1 GROUND-TRUTH GUIDANCE VARIANTS

Let $\mathcal{A}_t^*$ denote the human-verified valid action set for the current GUI state. The teacher-side guidance g_t is constructed from $\mathcal{A}_t^*$.

For Single-GT Guided OPD, one valid action is randomly sampled:

$$g_t = a_t^{*(k)}, \quad k \sim \mathrm{Uniform}\{1, \ldots, K\}. \tag{21}$$

For Multi-GT Guided OPD, the full valid action set is provided:

$$g_t = \mathcal{A}_t^*. \tag{22}$$

For Guided OPD, the guidance is selected according to:

$$g_t = a_t^\dagger = \arg\max_{a^* \in \mathcal{A}_t^*} S(\hat{a}_t, a^*). \tag{23}$$

All three variants use the same student input x_t and differ only in the privileged teacher context.

A.2 ACTION MATCHING AND MULTI-SOLUTION REWARD DETAILS

We define the base GUI action matcher $\phi_{\mathrm{gui}}(\hat{y}_t, a^*)$ as a normalized score in $[0, 1]$. If the model output cannot be parsed as a valid JSON action or violates the required action schema, the score is set to zero. For the reported runs, the component weights are $w_\tau = 10$, $w_p = 5$, and $w_v = 4$; with the value-matching scores below, an exact value match contributes 4 and a partial match contributes 2 before normalization.

Action type matching.

$$m_\tau = \mathbb{I}[\hat{\tau} = \tau^*]. \tag{24}$$

Position matching. For position-based actions, the model predicts a point $\hat{p}$ and the annotation provides a target bounding box b^*. We use a continuous position score $m_p \in [0, 1]$. If the predicted point falls inside the central region of the target box, $m_p = 1$. If it falls inside the full box but outside the central region, the score is linearly interpolated between 0.5 and 1. If it falls outside the box, the score decays exponentially with the distance d to the box boundary:

$$m_p = 0.5 \exp(-d/200). \tag{25}$$

Value matching. For text or key-value actions, we compute:

$$m_v = \begin{cases} 1, & \text{exact match,} \\ 0.5, & \text{partial match,} \\ 0, & \text{otherwise.} \end{cases} \tag{26}$$

Combined score. The final action matching score is:

$$\phi_{\mathrm{gui}}(\hat{y}_t, a^*) = \frac{w_\tau m_\tau + \mathbb{I}[b^* \neq \emptyset] w_p m_p + \mathbb{I}[v^* \neq \emptyset] w_v m_v}{w_\tau + \mathbb{I}[b^* \neq \emptyset] w_p + \mathbb{I}[v^* \neq \emptyset] w_v}. \tag{27}$$

Given multiple valid actions, we use:

$$R_{\mathrm{ms}}(\hat{y}_t, \mathcal{A}_t^*) = \max_{a^* \in \mathcal{A}_t^*} \phi_{\mathrm{gui}}(\hat{y}_t, a^*). \tag{28}$$

A.3 Planning Reward Details

The planning reward is a history-conditioned, per-state plan-quality signal computed for candidate subtask plans at logged GUI states using an external VLM judge. The judge receives the user instruction, available screenshot history, previous model outputs, and the current candidate subtask plan. Candidate actions are not executed for reward computation, and the reward uses neither sampled future states nor terminal task-success feedback; it is therefore a local surrogate for plan quality, not an estimate of long-term return or verified task recovery.

The judge evaluates the plan along four dimensions:

1. **Task relevance and adaptability**: whether the plan reflects the user instruction and adapts to the current UI state.
2. **Visual grounding**: whether the plan is grounded in actual UI elements visible in the screenshots.
3. **Decomposition granularity**: whether the plan decomposes the task into actionable subtasks at an appropriate granularity.
4. **State consistency**: whether the completion status of subtasks is consistent with the latest screenshot.

If the latest screenshot indicates that the agent is stuck, in an error state, or in a loop, the first unfinished subtask must explicitly propose a corrective next step. If the plan ignores the obstacle, remains overly high-level, or contains completion markers inconsistent with the latest screenshot, the judge is instructed to assign a score below 0.5.

The judge is required to return a structured JSON score:

```
{"score": 0.8}
```

A.4 Training-time Planning Judge: Prompt and Reliability

We use a frozen Qwen3-VL-32B-Instruct model as the VLM judge for the planning reward. The judge is queried only during RL training. The temperature is set to 0, and the judge is instructed to output a structured JSON score from the discrete set $\{0.0, 0.3, 0.5, 0.8, 1.0\}$. For detailed prompt design, please refer to Fig. 2.

Human–judge agreement. On a held-out calibration subset, Qwen3-VL-32B-Instruct achieves 90.0% exact agreement with human annotations (Cohen's $\kappa = 0.7916$). Treating high-quality trajectories as the positive class gives 94.52% recall, 81.18% precision, and 87.34% F1. These metrics assess the training-time planning reward judge, not the GPT-5.4 evaluator used to judge Lite-Bench task success (Appendix A.9).

Scope and limitations. The lower precision suggests an optimistic bias; this offline agreement study does not rule out reward hacking during training. The reward coefficients are 0.30 for planning, 0.60 for action matching, and 0.10 for format/length; these are weights, not fixed shares of realized reward.

A.5 Generalized Knowledge Distillation

We first optimize the policy with generalized knowledge distillation (GKD). The teacher is Qwen3-VL-32B-Instruct, and the students are Qwen3-VL-2B-Instruct and Qwen3-VL-30B-A3B. All GKD runs use full-parameter bfloat16 training with the Megatron backend. Unlike offline distillation from fixed target responses, our GKD stage is on-policy: the student generates the completion that is used for distillation. Concretely, we set the GKD on-policy probability to $\lambda = 1.0$, enable vLLM rollout, and disable sequential teacher generation. The dataset therefore provides prompts and human-verified valid-action candidates for constructing the privileged guidance described in the main text, while the optimized response tokens are sampled from the current student policy.

For each GUI state x_t, the student samples a completion $\hat{y}_t \sim \pi_\theta(\cdot \mid x_t)$. The sampled completion is held fixed during the optimization step. The teacher and student are each evaluated once on

	2B GKD	**A3B GKD**
Student	Qwen3-VL-2B-Instruct	Qwen3-VL-30B-A3B
Teacher	Qwen3-VL-32B-Instruct	Qwen3-VL-32B-Instruct
Training GPUs	1 node $\times$ 8 A100 GPUs	2 nodes $\times$ 8 A100 GPUs
Rollout mode	colocated vLLM	colocated vLLM
On-policy probability λ	1.0	1.0
GKD divergence parameter β	1.0	1.0
Teacher demo mode	GT guided	GT guided
Max sequence length	8192	8192
Max completion length	512	512
Micro batch size	1	1
Global batch size	32	32
Epochs	2	2
TP / PP / EP / CP	1 / 4 / 1 / 1	1 / 4 / 2 / 1
vLLM tensor parallel size	4	4
Learning rate	2.5×10^{-7}	2.5×10^{-7}
Warmup fraction	0.01	0.01

Table 5: Main hyperparameters for the GKD stage. TP, PP, EP, and CP denote tensor, pipeline, expert, and context parallelism.

the full sampled completion sequence under teacher forcing. In Guided-OPD, the teacher receives the most-matched valid action g_t from Section 3.2 as privileged guidance, and its input is $\tilde{x}_t = [x_t, \texttt{reference_action} = g_t]$. The discrete teacher-side guidance g_t is also held fixed during the optimization step. Let $\mathcal{M}_t$ denote the non-masked completion positions in $\hat{y}_t$. This GKD implementation instantiates the Guided-OPD objective in Section 3.2, so we use the same notation as the main text. Using the full-vocabulary next-token distributions at each position $j \in \mathcal{M}_t$, we compute the distillation loss:

$$\mathcal{L}_{\mathrm{OPD}} = \frac{1}{|\mathcal{M}_t|} \sum_{j \in \mathcal{M}_t} D_\beta \left(p_{\theta,j} \,\middle\|\, \mathrm{sg}\left[p_{T,j}^{g_t} \right] \right), \qquad \beta = 1,$$

where $p_{\theta,j} = \pi_\theta(\cdot \mid x_t, \hat{y}_{t,<j})$ is the student distribution at position j, and $p_{T,j}^{g_t} = \pi_T(\cdot \mid \tilde{x}_t, \hat{y}_{t,<j})$ is the teacher distribution conditioned on the most-matched guidance g_t. The operator $\mathrm{sg}[\cdot]$ denotes stop-gradient. Our implementation parameterizes the generalized JSD/KL family by β; with $\beta = 1$, it selects the reverse-KL direction:

$$D_1(p\|q) = D_{\mathrm{KL}}(p\|q) = \sum_v p(v) \log \frac{p(v)}{q(v)}.$$

Therefore, the reported experiments use full-vocabulary, per-prefix reverse KL, not a sampled-token surrogate. The teacher forward pass is executed under `no_grad` and its logits are detached; gradients flow only through the student distribution. One teacher-forced forward pass over the complete completion provides teacher distributions for all valid prefixes. We use sampling temperature 1.0 and top-$p = 0.98$ for student rollouts.

A.6 Reinforcement Learning with GRPO

After GKD, we further optimize the policy with group relative policy optimization (GRPO). For each logged prompt, the behavior policy samples $G = 16$ completions; candidates are scored at the same logged GUI state rather than by executing live environment transitions. Rewards are normalized within the group to form relative advantages:

$$\hat{A}_t^{(i)} = \frac{R_t^{(i)} - \mathrm{mean}(\{R_t^{(j)}\}_{j=1}^G)}{\mathrm{std}(\{R_t^{(j)}\}_{j=1}^G) + \epsilon_{\mathrm{adv}}}.$$

Here ϵ_{adv} is a numerical stabilizer distinct from the clipping bounds below. We use sequence-level importance sampling and the clipped GRPO objective:

$$\mathcal{L}_{\mathrm{GRPO}} = -\frac{1}{G} \sum_{i=1}^{G} \min \left(\rho_t^{(i)} \hat{A}_t^{(i)},\ \mathrm{clip}(\rho_t^{(i)}, 1 - \epsilon_{\mathrm{low}}, 1 + \epsilon_{\mathrm{high}}) \hat{A}_t^{(i)} \right) + \beta_{\mathrm{KL}} D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}),$$

	2B GRPO	**A3B GRPO**
Initialization	2B GKD checkpoint	A3B GKD checkpoint
Training GPUs	2 nodes $\times$ 8 A100 GPUs	2 nodes $\times$ 8 A100 GPUs
Policy rollout mode	colocated vLLM	colocated vLLM
Reward model	Qwen3-VL-32B-Instruct	Qwen3-VL-32B-Instruct
Reward servers	2	4
Max sequence length	10240	10240
Max completion length	512	640
Soft max length	–	512
Soft cache length	128	384
Micro batch size	1	1
Global batch size	64	32
Epochs	1	1
Generations per prompt	16	16
Steps per generation	1	1
TP / PP / EP / CP	1 / 4 / 1 / 1	1 / 4 / 2 / 1
Sequence parallel	enabled	enabled
KL coefficient β_{KL}	0.0	0.0
Learning rate	2.5×10^{-7}	1.0×10^{-7}
Warmup fraction	0.01	0.01

Table 6: Main hyperparameters for the GRPO stage. The 2B GRPO run uses a two-stage schedule in which the second stage resumes from the latest checkpoint of the first stage. Both runs use $G = 16$, $\epsilon_{\mathrm{low}} = 0.2$, $\epsilon_{\mathrm{high}} = 0.28$, $\beta_{\mathrm{KL}} = 0$, dynamic sampling, and overlong-response filtering.

where the sequence-level importance ratio is computed from the mean token log-probability difference:

$$\rho_t^{(i)} = \exp\left[\frac{1}{|\mathcal{M}_t^{(i)}|}\sum_{j\in\mathcal{M}_t^{(i)}}\left(\log \pi_\theta(\hat{y}_{t,j}^{(i)} \mid x_t, \hat{y}_{t,<j}^{(i)}) - \log \pi_{\theta_{\mathrm{old}}}(\hat{y}_{t,j}^{(i)} \mid x_t, \hat{y}_{t,<j}^{(i)})\right)\right].$$

Here $\mathcal{M}_t^{(i)}$ denotes the valid completion-token positions and $\hat{A}_t^{(i)}$ is the group-normalized advantage defined in Section 3.3. The resulting scalar ratio is shared by the whole completion and is the only quantity clipped; no token-wise clipping is applied. We use $\epsilon_{\mathrm{low}} = 0.2$, $\epsilon_{\mathrm{high}} = 0.28$, dynamic sampling, and overlong-response filtering in both 2B and A3B GRPO runs. The KL coefficient is $\beta_{\mathrm{KL}} = 0$ for both runs.

The reward profile is `local_subtask`. It combines multi-solution GUI action matching, subtask quality judged by a local Qwen3-VL-32B-Instruct reward model, and a JSON-aware soft overlength penalty. The reward functions and weights passed to training are:

$$R = 0.60\, R_{\mathrm{ms}} + 0.30\, R_{\mathrm{sub}} + 0.10\, R_{\mathrm{format/length}}.$$

The subtask reward model is served locally with vLLM. For the 2B run, two local reward servers are used; for the A3B run, four local reward servers are used. For the A3B policy, generated outputs are additionally constrained by a structured JSON regular expression requiring fields for reasoning, subtask decomposition, and the next action. The $R_{\mathrm{format/length}}$ term is an auxiliary output-validity regularizer, not a semantic task-success reward: malformed or schema-invalid JSON receives no format credit, while valid but overlong responses receive a soft length penalty under the completion limits listed in Table 6.

A.7 Data Pipeline

To facilitate the training of Lite-GUI, we developed a two-stage data construction pipeline consisting of an automated trajectory generation phase and a high-quality multi-solution annotation phase, as illustrated in Fig. 1. Both Lite-Dataset and Lite-Bench are generated through this pipeline.

A.7.1 Automated Trajectory Generation (Data Pipeline)

In this section, we detail the **Automated Trajectory Generation (ATG)** framework, a self-evolving pipeline designed to synthesize high-quality interaction trajectories for GUI agents. The framework

leverages a "generation-verification-correction" closed-loop mechanism to mitigate error propagation and hallucinations in long-horizon tasks.

Multimodal Context Representation

At each discrete time step t, the system constructs a comprehensive multimodal input $\mathcal{X}_t$:

$$\mathcal{X}_t = \{\mathcal{T}_{name}, \mathcal{I}_t, \mathcal{H}_{text}, \mathcal{V}_{hist}\} \tag{29}$$

where $\mathcal{T}_{name}$ denotes the task identifier, $\mathcal{I}_t$ is the current RGB screenshot, $\mathcal{H}_{text}$ represents the historical dialogue logs, and $\mathcal{V}_{hist} = \{\mathcal{I}_{t-1}, \mathcal{I}_{t-2}\}$ provides the temporal visual context through the two preceding frames.

Decision Engine: Qwen3-VL-32B

We employ **Qwen3-VL-32B** as the core GUI Model to parse $\mathcal{X}_t$ and generate a structured decision tuple $\mathcal{D}_t$:

$$\mathcal{D}_t = \langle \text{Reasoning, Subtasks, Action, Value, Position} \rangle \tag{30}$$

The model explicitly outputs its cognitive chain via *Reasoning* and decomposes complex objectives into granular *Subtasks*, ensuring the logical consistency of the resulting *Action* and its corresponding coordinates (*Position*), for reference, see Fig. 7 and system prompt refers to Fig. 3.

Robust Verification and Feedback-driven Refinement

To ensure the reliability of generated actions, a peer **Qwen3-VL-32B** model acts as the **Verify Model**, for reference, see Fig. 7 and system prompt refers to Fig. 4.

Majority Voting Mechanism

For each candidate action $\mathcal{D}_t$, the Verify Model performs three independent evaluations considering task relevance and planning accuracy. An action is committed to the execution environment (PC) if and only if it secures a majority consensus:

$$Count(\text{Eval} = \text{"Yes"}) \geq 2 \tag{31}$$

Iterative Self-Correction and Recency-based Fallback

If the consensus threshold is not met, the framework initiates a feedback loop (up to $N = 3$ iterations):

- **Feedback Injection:** The *Eval* and *Reasoning* from the Verify Model are fed back into the GUI Model as negative constraints to guide policy refinement.
- **Evolutionary Selection:** In cases where the loop reaches the maximum iteration limit without a consensus, the system adopts a **Recency-based Selection** strategy. It executes the action from the latest iteration that yielded the highest number of "Yes" votes.
- **Rationale:** This strategy assumes that the sequential accumulation of feedback cues progressively narrows the search space, rendering the latest refined output the most robust approximation of the optimal policy.

A.7.2 Data Aggregation and Policy Enhancement

Post-execution, successful trajectories are archived into a centralized **Data Pool**. By integrating these synthesized trajectories with existing open-source datasets, we provide a rich, multi-domain corpus for subsequent post-training stages, including OPD and reinforcement learning, thereby continuously improving the agent's generalization capabilities across diverse GUI environments.

Guided On-policy Distillation and Reinforcement Learning Data Construction

The Guided On-policy Distillation and Reinforcement Learning Data Construction phase (as illustrated in Fig. 1) serves as the core refinement engine for transforming linear explorations into high-density, multi-path expert demonstrations. This process shifts the dataset from a singular-path trajectory to a topological decision graph through state-space exploration and human-in-the-loop auditing.

- **State-space Exploration and Stochastic Inference:** Building upon the multi-step trajectories collected during the ATG phase, the framework performs an in-depth exploration of the state space. For any intermediate state s_t within a trajectory, we employ **Qwen3-VL-32B** to perform multiple independent stochastic inferences. The objective is to probe the existence of alternative valid solutions for the current sub-task. By varying the decoding parameters, the model generates a set of potential candidate actions $\{a_t^1, a_t^2, \ldots, a_t^n\}$, effectively uncovering the inherent strategy diversity of the GUI environment.
- **Expert Auditing and Policy Refinement:** The candidate actions generated during the exploration phase are mapped into a structured semantic format. These candidates are then submitted to a **Manual Annotation** process, which acts as the ultimate "quality gate." Expert annotators evaluate each proposed action against the visual state and task objectives. As depicted in Fig. 1, correct actions that logically advance the task are marked as "True" and preserved, while erroneous or redundant actions are pruned. This human-in-the-loop verification eliminates model hallucinations and ensures the high fidelity of the branched trajectories.
- **RL Data Densification:** The culmination of this process is a structured "Multi-solution" dataset where a single task at a specific step possesses multiple validated paths. This construction significantly densifies the reward landscape for the subsequent **Guided On-policy Distillation** and **Reinforcement Learning** stages. Rather than simple imitation learning, the agent is provided with a rich ensemble of successful strategies, fostering superior decision resilience and error-recovery capabilities.

A.8 LITE-DATASET

A.8.1 TASK DOMAINS AND ENVIRONMENT CONFIGURATION

The **Lite-Dataset** is a large-scale, high-fidelity multimodal dataset specifically designed for the training and evaluation of autonomous GUI agents. The dataset is strictly rooted in a native PC terminal environment similar to Ubuntu 22.04 LTS, simulating authentic productivity workflows.

- **Full-stack Productivity Ecosystem:** The dataset covers low-level system management (File System, Terminal commands, and System Settings) as well as high-level professional applications, including the full WPS Office suite (Writer, Spreadsheets, and Presentation).
- **Challenge Benchmarks:** We have introduced complex interaction tasks involving VS Code, Dingding, and specialized interfaces for Musa Digital Human and AI Assistants. These tasks are defined as "Challenge Benchmarks," where the decision space exhibits non-linear growth as task complexity increases.

A.8.2 STANDARDIZED ACTION SPACE

To bridge the gap between high-level visual perception and system-level execution, we define a standardized action space $\mathcal{A}$ within the Lite-Datasets. Each action consists of a type, target coordinates (Position), and optional parameters (Value):

- **Mouse Interactions:** Includes `CLICK`, `RIGHT_CLICK`, `DOUBLE_CLICK`, `DRAG`, and `SCROLL_UP/DOWN` for page navigation and content scrolling.
- **Keyboard Operations:** Includes `TEXT_INPUT` for string entry and `KEY` for discrete key presses or complex combinations (e.g., *CTRL+A*).
- **System Control:** The `WAIT` action suspends the agent for 1 second, ensuring system stability and accounting for asynchronous loading latencies.

A.8.3 GUIDED ON-POLICY DISTILLATION AND RL PIPELINE

To construct an expert demonstration library for reinforcement learning, we designed a data construction pipeline based on stochastic exploration:

- **Contextual Sliding Window:** Long-horizon trajectories are truncated into manageable samples with a maximum window of three steps (comprising three dialogue turns and three screenshots) to enhance the precision of local decision-making.

- **Multi-solution Stochastic Exploration:** We utilize **Qwen3-VL-32B** to perform re-inference on the final round of each windowed sample. By increasing the temperature ($Temperature = 1.0$), the model performs multiple samplings ($N = 5$) to obtain diverse results, which are subsequently deduplicated and stored. The specific output format of these samples follows the template defined in Fig. 7.
- **Human-in-the-loop Verification:** All candidate paths generated during exploration undergo rigorous manual annotation. We constructed **30,000** complete multi-step trajectories and approximately **70,000** derived windowed training samples; across the collected data, **11,000** samples carry human-verified multi-solution annotations. Detailed examples of these multi-path ground truths can be found in Fig. 7.

A.8.4 Error-aware Trajectory Modeling

A distinguishing feature of Lite-Datasets is its inclusion of negative supervision signals. Within the complete trajectories collected via the Automated Trajectory Generation (ATG) phase, we intentionally preserved and labeled samples where human annotators identified errors in intermediate steps. This error-aware modeling provides critical negative feedback, enabling the agent to improve its error-correction capabilities and robustness in complex, real-world environments.

A.9 Lite-Bench

A.9.1 Benchmark Overview and Design Philosophy

To rigorously evaluate the multi-step complex task completion capabilities of GUI agents, we present **Lite-Bench**. Built upon a live **Ubuntu 22.04** environment, Lite-Bench shifts the evaluation paradigm from static offline prediction to **dynamic, closed-loop interaction**. The benchmark is specifically designed to stress-test the agent's core competencies in long-term planning, visual grounding, complex instruction comprehension, and error recovery within a non-stationary environment.

A.9.2 Task Composition and Procedural Constraints

Lite-Bench comprises 160 tasks, featuring 75 web tasks, 38 terminal tasks, and 47 file system tasks that strictly prohibit terminal usage to enforce pure vision-based interaction. None of the 160 Lite-Bench evaluation task instances is included in the training samples. However, some evaluation tasks share workflow families with training examples while differing in their target entities or object values. We therefore interpret Lite-Bench as an evaluation of transfer to held-out task instances, not as a task-family-disjoint test of generalization to entirely unseen workflows. To simulate the efficiency constraints of professional workflows, each task is capped at a maximum of **18 steps**. The tasks are categorized into three primary domains:

- **File System:** Focuses on GUI-based file management, directory structuring, and visual data organization. *Notably, Lite-Bench enforces a strict procedural constraint: file system tasks must be accomplished through the graphical file manager. Operations executed via the Terminal for these specific tasks are recorded as failures, ensuring the benchmark measures GUI manipulation proficiency rather than command-line scripting.*
- **Web Browser:** Involves multi-tab navigation, semantic information retrieval, and complex web-form processing.
- **Terminal:** Covers standard command-line operations, system-level configurations, and environment debugging.

For a complete evaluation of this fixed task set, the pooled success rate must satisfy $\mathrm{All} = (47\,\mathrm{File} + 75\,\mathrm{Web} + 38\,\mathrm{Terminal})/160$, up to displayed rounding. The same identity applies to an average over repeated evaluations when every run uses this task set and denominator. As stated above, OS-World and Lite-Bench results are arithmetic means over five independent runs.

Table 7: Analysis of Guided-OPD. (a) Teacher supervision quality under privileged guidance. (b) Student valid-action reliability and solution coverage under repeated sampling.

(a) Teacher Supervision Quality

Method	Selected Valid ↑	Other Valid ↑	Invalid ↓	V–I Margin ↑	Pairwise Acc. ↑
OPD	0.596	0.314	0.000	0.455	46%
Guided-OPD	**0.980**	**0.532**	0.000	**0.756**	**77%**

(b) Student Solution Coverage

Model	ValidRate@8 ↑	UniqueValid@8 ↑	Coverage@8 ↑
OPD	0.5175	0.52	0.26
Most-Matched-GT Guided-OPD	0.5725	**0.66**	**0.33**
Final MD-GRPO	**0.6375**	**0.66**	**0.33**

A.9.3 Evaluation Protocol: LLM-as-a-Judge

The non-trivial nature of verifying GUI task success requires a nuanced understanding of both final states and interaction sequences. Lite-Bench adopts an automated **"LLM-as-a-Judge"** protocol utilizing the latest **GPT-5.4** multimodal model:

1. **Real-time Execution:** The agent interacts with the live Ubuntu environment. Upon task completion or reaching the 18-step limit, the system automatically exports a comprehensive execution log (JSON) and a full-process trajectory of screenshots.
2. **Multimodal Auditing:** The execution artifacts are fed into **GPT-5.4**. The judge model is instructed to perform a dual-criteria audit:
   - **Outcome Verification:** Does the final UI state satisfy the initial user instruction?
   - **Compliance Auditing:** Did the agent adhere to the required interaction modality (e.g., using GUI instead of Terminal for file tasks)?
3. **Success Rate (SR):** The primary metric is the Success Rate, defined as the percentage of tasks verified as "Successful" by the judge model based on the criteria above.

By combining a live execution environment with a high-level multimodal auditing mechanism, Lite-Bench provides an objective, scalable, and challenging framework for assessing the next generation of general-purpose GUI agents.

Separately from the training-time planning judge evaluated in Appendix A.4, we calibrate the GPT-5.4 benchmark evaluator. On a held-out human-annotated subset, GPT-5.4 agrees with human judgments at 93.7% for file tasks, 93.4% for browser tasks, and 94.8% for terminal tasks, with 93.9% overall agreement. ScreenSpot-Pro is evaluated over its fixed benchmark images. For OS-World and Lite-Bench, every reported result for our models and ablations is the arithmetic mean of five independent runs; the tables do not mix single-run and repeated-run values. The repetition protocol of literature baselines is not established here.

To audit train–test independence, we compared Lite-Bench instructions with approximately 70K training samples using BGE-M3 embeddings and a similarity threshold of 0.80. Only 862 training samples exceeded the threshold, and no identical task instructions were found. Lite-Bench covers File System, Web, and Terminal, whereas Lite-Dataset spans more than seven application categories, including Feishu, AI assistants, and WPS.

A.10 Analysis of Guided On-Policy Distillation

Unless otherwise specified, all diagnostics in this section are conducted on $N = 50$ multi-solution validation states. For the student solution-coverage analysis, we sample $K = 8$ independent completions per state.

We conduct two complementary diagnostic analyses to better understand the effect of privileged guidance in Guided-OPD. The first evaluates whether privileged guidance improves the teacher's

ability to discriminate valid GUI actions from invalid ones. The second examines whether Most-Matched-GT causes the student policy to collapse toward a narrower subset of valid solutions. The results are summarized in Table 7.

Teacher-scoring diagnostic. For each validation state i, let a_i^{sel} denote the human-verified valid action selected as privileged guidance, a_i^{oth} denote another human-verified valid action that is not provided as guidance, and a_i^{neg} denote an invalid action sampled from another GUI state. Let

$$s_i(a) \in [0, 1]$$

denote the score assigned by the same Qwen3-VL-32B teacher to candidate action a under the corresponding teacher context.

We report five metrics to characterize teacher supervision quality.

Selected Valid. This metric measures the average teacher score assigned to the valid action selected as privileged guidance:

$$S_{\mathrm{sel}} = \frac{1}{N} \sum_{i=1}^{N} s_i(a_i^{\mathrm{sel}}).$$

A larger value indicates that the teacher assigns higher confidence to the guidance-selected valid action.

Other Valid. To determine whether privileged guidance improves supervision beyond the explicitly provided reference, we also evaluate another human-verified valid action that is not used as guidance:

$$S_{\mathrm{oth}} = \frac{1}{N} \sum_{i=1}^{N} s_i(a_i^{\mathrm{oth}}).$$

An increase in this metric indicates that the effect of privileged guidance is not restricted to the selected reference action, but also improves the teacher's recognition of alternative valid GUI decisions.

Invalid. We measure the average score assigned to an invalid action:

$$S_{\mathrm{neg}} = \frac{1}{N} \sum_{i=1}^{N} s_i(a_i^{\mathrm{neg}}).$$

A lower value is preferred, as it indicates that the teacher does not assign high confidence to actions that are invalid for the current GUI state.

Valid–Invalid Margin. We measure the separation between valid and invalid actions by averaging the scores of the two valid candidates and subtracting the invalid-action score:

$$M_{\mathrm{V-I}} = \frac{S_{\mathrm{sel}} + S_{\mathrm{oth}}}{2} - S_{\mathrm{neg}}.$$

A larger margin indicates that the teacher more clearly separates valid GUI actions from invalid alternatives, yielding a more discriminative supervision signal.

Pairwise Preference Accuracy. We further measure whether the teacher ranks valid actions above the invalid action. For each state, we compare both valid candidates against the invalid candidate:

$$\mathrm{PairAcc} = \frac{1}{2N} \sum_{i=1}^{N} \left[\mathbb{I}\left(s_i(a_i^{\mathrm{sel}}) > s_i(a_i^{\mathrm{neg}})\right) + \mathbb{I}\left(s_i(a_i^{\mathrm{oth}}) > s_i(a_i^{\mathrm{neg}})\right) \right],$$

where $\mathbb{I}[\cdot]$ is the indicator function. A larger value means that the teacher more consistently prefers valid actions over invalid ones.

As shown in Table 7(a), Guided-OPD increases the Selected Valid score from 0.596 to 0.980 and the Other Valid score from 0.314 to 0.532, while the Invalid score remains at 0.000. Accordingly, the Valid–Invalid Margin increases from 0.455 to 0.756, and Pairwise Preference Accuracy improves from 46% to 77%. Importantly, the improvement on Other Valid indicates that privileged guidance does more than simply increase confidence in the provided reference action; it also improves the teacher's discrimination of alternative valid GUI decisions.

Student solution-coverage diagnostic. We next examine whether the stronger alignment introduced by Most-Matched-GT reduces the diversity of valid actions produced by the student. For each validation state i, we independently sample $K = 8$ completions from the student policy:

$$\hat{a}_{i,1}, \hat{a}_{i,2}, \ldots, \hat{a}_{i,K}.$$

Let A_i^* denote the human-verified valid-action set for state i. A sampled action is considered valid if it matches at least one action in A_i^* according to the GUI action-matching function. We define

$$m_{i,j} = \mathbb{I}\left[\exists a^* \in A_i^*,\ \hat{a}_{i,j} \sim a^*\right],$$

where $\hat{a}_{i,j} \sim a^*$ denotes that the sampled action matches the corresponding human-verified valid action.

We report three metrics to quantify valid-action reliability and solution diversity.

ValidRate@8. ValidRate@8 measures the fraction of the eight sampled actions that are valid:

$$\text{ValidRate@8} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{j=1}^{K} m_{i,j}, \qquad K = 8.$$

This metric measures the reliability of the student policy under repeated sampling: a larger value indicates that a greater fraction of generated actions are valid.

UniqueValid@8. For each state, we define the set of distinct human-verified valid actions recovered by the sampled outputs as

$$U_i = \{a^* \in A_i^* \mid \exists j \in \{1, \ldots, K\}, \hat{a}_{i,j} \sim a^*\}.$$

We then compute

$$\text{UniqueValid@8} = \frac{1}{N}\sum_{i=1}^{N} |U_i|.$$

Unlike ValidRate@8, which can be high even when the same valid action is repeatedly generated, UniqueValid@8 measures how many distinct valid action modes are observed under repeated sampling.

Coverage@8. Coverage@8 measures the fraction of the annotated valid-action set recovered by the student samples:

$$\text{Coverage@8} = \frac{1}{N}\sum_{i=1}^{N}\frac{|U_i|}{|A_i^*|}.$$

A larger value indicates that the student covers a larger fraction of the human-verified valid solution space.

As shown in Table 7(b), Most-Matched-GT Guided-OPD improves ValidRate@8 from 0.5175 to 0.5725, while increasing UniqueValid@8 from 0.52 to 0.66 and Coverage@8 from 0.26 to 0.33. The final MD-GRPO model further increases ValidRate@8 to 0.6375 while preserving UniqueValid@8 and Coverage@8 at 0.66 and 0.33, respectively. These results indicate that Most-Matched-GT improves the reliability of valid-action generation without reducing the observed diversity or coverage of valid solutions.

A.11 Inference Efficiency and Training Cost

We benchmarked inference with three-window GUI contexts. On one A100, Qwen3-VL-2B reaches 103.0 tokens/s at approximately 4.5 GB VRAM, Lite-GUI-30B-A3B reaches 91.54 tokens/s at approximately 18 GB, whereas the Qwen3-VL-32B teacher reaches 27.27 tokens/s and requires two A100 GPUs with tensor parallelism. On one RTX 4090, Qwen3-VL-2B reaches 81.2 tokens/s in BF16 and Lite-GUI-30B-A3B reaches 17.7 tokens/s with AWQ; the dense 32B AWQ model runs out of memory.

The 11K multi-solution annotations required approximately 120 human hours. The full data-generation and training pipeline required approximately 1.2K A100 GPU hours. These are one-time

offline costs; they trade against lower inference memory and latency for the resulting lightweight models.

Under the same controlled training configuration, the SFT baseline requires approximately 4 GPU-hours, compared with 17 GPU-hours for Most-Matched-GT-Guided OPD. In an additional ShowUI-Desktop-8K comparison with identical model settings, SFT, Most-Matched-GT-Guided OPD, and GRPO require approximately 0.5, 4, and 12 hours, respectively, and achieve 30.55%, 46.29%, and 43.51% ScreenSpot-Pro accuracy.

A.12 Limitations and Broader Impacts

Although our method improves GUI-agent training through guided on-policy distillation and multi-solution dual-level reinforcement learning, several limitations remain.

First, the human-verified valid action set $\mathcal{A}_t^*$ is only a finite approximation of the full valid action space. GUI tasks often admit many semantically equivalent action paths, and our annotations cannot exhaustively cover all possible correct actions. As a result, a model output that correctly advances the task may still receive a lower reward if the corresponding action is not included in the annotated multi-solution set. The Most-Matched-GT selector should therefore be understood as an alignment heuristic that reduces supervision–state mismatch, not as a guarantee that all valid action modes remain equally represented.

Second, constructing multi-solution annotations requires additional human verification. Compared with single-action supervision, annotating and validating multiple acceptable actions for the same GUI state is more expensive and may limit scalability to new applications, operating systems, or task domains.

Third, the planning reward relies on a frozen Qwen3-VL-32B-Instruct model as an external VLM judge during RL training. Although the judge is not used at inference time, it increases training cost and may introduce model-specific evaluation bias. In particular, the judge may over- or under-estimate the quality of a subtask plan when the screenshot history is ambiguous or when the plan uses a valid but uncommon strategy.

Fourth, GRPO is a practical choice for our logged-state setting rather than a claim of superiority over critic-based PPO. PPO may provide finer-grained credit assignment when reliable value targets and online rollouts are available, while our setting avoids an additional critic and the cost of repeated GUI interaction. A compute-matched GRPO–PPO comparison remains future work.

Fifth, our latency and memory measurements are server-class measurements and do not include a full energy study or a broad on-device deployment evaluation. Our experiments focus on GUI-agent tasks under the evaluated benchmark and data distribution. The results may not directly generalize to all operating systems, applications, screen resolutions, languages, or highly dynamic interfaces. Further evaluation is needed before deploying such agents in open-ended real-world environments.

Finally, GUI automation systems can have both positive and negative societal impacts. They may improve productivity and accessibility by automating repetitive computer tasks, but they may also cause unintended operations, privacy leakage from screenshots, or misuse in unauthorized automation. Practical deployment should include permission control, sensitive-action confirmation, logging, and privacy filtering for screenshot-based data.

A.13 Existing Assets and Licenses

We use several existing models, benchmarks, and software packages in this work. We cite the corresponding papers or official repositories when applicable and follow their license and usage terms. Table 8 summarizes the main existing assets used in our experiments.

For all existing assets, we use them only for research purposes and follow their official license and usage terms. We do not redistribute third-party pretrained models or benchmark assets with restrictive terms; instead, we provide instructions for obtaining them from their official sources when necessary. Our released assets only include our own GUI trajectories, annotations, prompts, configurations, and code components that we are permitted to distribute.

Table 8: Existing assets used in this work.

Asset	Role in this work	License / Terms
Qwen3-VL-32B	Frozen VLM judge for the planning reward; also used as the teacher model for Guided OPD when applicable.	Apache-2.0
Qwen3-VL-30B-A3B	Base student model for GUI-agent training.	Apache-2.0
OSWorld	Public benchmark for evaluating GUI agents in real computer environments.	Apache-2.0
ScreenSpot-Pro	Public GUI grounding benchmark for professional high-resolution computer-use scenarios.	MIT
ms-swift	Main training framework for SFT, GKD/OPD, and GRPO experiments.	Apache-2.0
Megatron-LM / Megatron Core	Distributed training backend used through the Megatron training path, including tensor, pipeline, data, and expert parallelism.	Apache-2.0

Prompt for the Long-horizon Planning Reward:
Background: The user is operating an {Ubuntu Linux} system.

User Task: {task_desc}

Agent's Subtask Plan:
- {subtask_1}
- {subtask_2}
- ...

Evaluate the plan on four criteria:
1. Relevance and adaptability to the user task and the current UI state.
2. Visual grounding to the screenshot history.
3. Decomposition quality: the overall plan should usually be decomposed into about 3-4 subtasks, while each individual subtask should stay atomic, concrete, and directly executable.
4. Checkbox status accuracy: the final status must match the last screenshot.

If the last screenshot shows the agent is stuck, in an error state, or in a loop, the first unfinished subtask must explicitly describe the corrective click or input action. In that case, a short single-step corrective subtask is acceptable and should not be penalized just for being short. If the plan ignores the obstacle, is too vague/high-level, or the checkbox state mismatches the last screenshot, the score must be below 0.5.

Use only one of these scores: 0.0, 0.3, 0.5, 0.8, 1.0.

- 1.0: The plan is highly relevant, visually grounded, and the next unfinished subtask is precise and executable.
- 0.8: The plan is mostly good, with only minor redundancy or mild wording issues.
- 0.5: The plan is partially correct but has a noticeable issue, such as coarse decomposition or checkbox mismatch.
- 0.3: The plan is weak, underspecified, or misses the current obstacle/recovery need.
- 0.0: The plan is irrelevant, misleading, or unusable for the current state.

Return the result as a JSON object:
{"score": <score>}

Figure 2: Prompt for the planning reward.

GUI Model System Prompt:

You are using Ubuntu 22.04 with the following considerations:
\n1. Use Brave as the primary web browser, launched via Super+A.\n2. Launch applications via Super+A.\n3. Rename files/folders with F2.\n4. For bookings, when entering a city, select from the dropdown list if available.\n5. Before typing in input fields, click to focus. If conflicting pre-filled text exists, use Ctrl+A + Delete to clear.\n6. In browsers, open a new tab with Ctrl+T if the current page isn't correct.\n7. On CAPTCHA pages, close the tab and open a new one.\n8. Enter website addresses directly in the address bar.\n9. To download images, right-click → Save image as... (or "图片另存为").\n10. Switch apps with Alt+Tab.\nYou can interact with the computer using mouse and keyboard based on the task and screenshot.\nYour interaction is limited to the desktop GUI, but you may use the terminal if needed.\nAlways follow the task exactly without adding extra steps.\n---\n\n### **Screenshot Analysis and Task Execution**\nAdditionally, the following contextual information is provided:\n-
Overall Screen State: A description of the system's current state, including open windows, applications, and visible UI elements.\n-
Task-Relevant Information: Elements in the screenshot that are crucial to completing the task.\n- **Missing or Pending Elements**: Any critical UI elements or information that are required for task completion but are currently absent.\n\n## Task Management Protocol\nFor each user request:\n1. Analyze the complete task and break it down into 4-7 logical subtasks, do not add unnecessary subtasks.\n2. Present each subtask with:\n - A clear, one-sentence description of the objective\n - A Markdown checkbox to indicate completion status: [ ] (incomplete) or [x] (complete)\n3. In subsequent interactions, update checkbox statuses based on progress\n4. Add subtasks as the situation evolves if no subtask is provided in the previous interaction.\n5. If the subtasks are provided in the Previous Actions History, you should not change the subtasks, but you can update the status of the subtasks by adding \"x\" to the checkbox based on progress.\n\nYou must think in a structured way before deciding your next action.\nFor every decision, please output the following 3 parts in the \"Reasoning\" field:\n1. Observation: Describe the key elements visible on the screen (e.g. buttons, input boxes, labels). Be specific about position, status, and text.\n2. Intent: Describe what the user wants to achieve.\n3. Analysis: Think through the steps needed to achieve the intent, based on the current screen. Explain reasoning.\n\n---\n\n
Available Actions\nYou can only select **one** of the following actions at a time:\n\n- `TEXT_INPUT`: Types a string of text at a specific position.\n- `CLICK`: Moves the mouse to a position and performs a left-click.\n- `RIGHT_CLICK`: Moves the mouse to a position and performs a right-click.\n- `DOUBLE_CLICK`: Moves the mouse to a position and performs a double-click.\n- `DRAG`: Drags from one position to another. Use \"value\" field with format \"start_x,start_y to end_x,end_y\".\n- `SCROLL_UP`: Moves the mouse to a position and performs scrolls up when a desired button is not visible or more content needs to be seen.\n- `SCROLL_DOWN`: Moves the mouse to a position and performs scrolls down when a desired button is not visible or more content needs to be seen.\n- `WAIT`: Waits for 1 second for the system to load or respond.\n- `KEY`: Presses a key or key combination on the keyboard, use + to combine keys. For example, \"value\": \"CTRL+A\"\n\nBased on the **visual information from the screenshot**, **the task requirements**, and **past interactions**, determine the **best next action** to progress toward completing the task.\n\n---\n
 # Keyboard Command Guidelines\n\n
Supported Keys\n- Control: ctrl, alt, shift, super\n- Navigation: enter, esc, tab, up, down, left, right\n- Editing: backspace, delete\n- Page nav: page_up, page_down, home, end\n\n## Key Format\n- \"+\" for simultaneous press (e.g., \"CTRL+A\", \"CTRL+SHIFT+ESC\")\n- Space for sequential press (e.g., \"A B C\", \"CTRL+C CTRL+V\")\n\n## Usage\nIn a KEY action, set the key(s) in the \"value\" field:\n```json\n{\n \"Next Action\": \"KEY\",\n \"value\": \"CTRL+A\"\n}\n---\n\n### **Output Format**\nYour response must strictly follow the **JSON format** below:\n```json\n{{\n \"Reasoning\": \"Provide a detailed analysis of what is currently visible on the screen, taking into account history, and explain your logical step-by-step process to progress toward task completion.\",\n \"subtask\": [\"subtask_description_list\"],\n \"Next Action\": \"action_type\" | \"None\",\n \"pos\": \"x,y\" | \"None\",\n \"value\": \"xxx\" # Only include the \"value\" field if the action is TEXT_INPUT, KEY, or DRAG, otherwise omit this key.\n}}\n```\n\nOne Example:\n```json\n{{ \n \"Reasoning\": \"Observation: The current screen shows google result of amazon, in previous action I have searched amazon on google. Intent: I want to go to amazon.com. Analysis: I need to click on the first search results to go to amazon.com.\",\n \"subtask\": [\"[x] search amazon on google\", \"[ ] click on the first search results\", \"[ ] go to amazon.com\"],\n \"Next Action\": \"CLICK\",\n \"pos\": \"100,100\"\n}}\n```\n
Key Capabilities\n1. **Only output ONE action at a time** in \"Next Action\".\n2. **Do not add extra actions**; always predict the most efficient one.\n3. When finished, set `\"Next Action\": \"None\"`. \n If waiting, set `\"Next Action\": \"WAIT\"`.\n4. Use \"file manager\" for file browsing; use WPS for Office apps.\n5. For multi-product/page tasks, split into subgoals and follow order.\n6. **Avoid repeating the same action/element. If repeated, reflect and change.**\n7. Carefully check the current screen and history to confirm actions are correct.\n8. **Keep operations efficient**: minimal steps, natural timing.\n9. If the window isn't maximized, press **Super+UP** first.\n\n\n"

Figure 3: GUI Model System Prompt.

Verify Model System Prompt:

You are an AI assistant responsible for evaluating the correctness and relevance of a proposed next action in the context of a given task and system state. Your goal is to determine whether the provided next action is **correct**, **reasonable**, **executable**, and **task-relevant** based on the available information.**{SYSTEM_REQUIREMENTS}**

The previous action was:**{last_step}**
You also need to determine whether the previous action was executed correctly, as this affects whether the current action is meaningful. This helps avoid situations where prerequisite actions were not carried out.

**Evaluation Criteria**
Evaluate the `Next Action` based on the following factors:
1. **Correctness**: Does this action logically follow from the previous actions and the current system state? Validate the action type and the position if provided.
2. **Reasonableness**: Is this action an appropriate step toward completing the task? Check the `subtask` field to see if the action is related to the subtask.
3. **Executability**: Can this action be successfully executed within the given system constraints (e.g., interacting only with the GUI, no terminal access)?
4. **Task Relevance**: Is this action directly related to completing the task, or is it unnecessary or incorrect?
5. **Avoid Repeated Actions**: Avoid choosing the same action/elements multiple times in a row.
6. **Efficiency**: Is this action the most efficient way to complete the task?
{KEY_COMMAND_NOTES}
Notes:
1.If the task involves entering content into an input field, you need to determine whether the task requires clearing the existing content in the search box before entering the new input.

Please determine if the solution below is correct. First, rewrite it in a mathematical structure format with:
- Observation
- Intent
- Analysis
- Action
Then verify each component for correctness.
The solution you should evaluate: **{current_action}**
**Output Format**
Keep your response brief (maximum 150 words total) and structured as follows:
Provide a structured JSON response with two key fields:
```json
{{
   "Reasoning": "Explain your step-by-step evaluation of why the proposed next action is correct or incorrect based on the given context.",
   "Eval": "yes" | "no"
}}
```

Figure 4: Verify Model System Prompt.

SYSTEM_REQUIREMENTS:
You are using an Ubuntu 22.04 system with the following special considerations:
1. The Brave web browser is the very first choice when you need to do some web actions, open it in the application launcher if you cannot find it in the current screen.
2. To launch applications, press Super+A to open the application launcher.
3. To rename a file or a folder, you can use keyboard shortcut "F2" after you select the file or folder.
4. When setting the departure or arrival point for flight or train ticket bookings, after entering the desired city, if a pop-up window displays a list of relevant airports or train stations, click to select the specific option from the list to confirm your choice.
5. To enter text into an input field, start by clicking on the field to activate it. Begin typing your desired input. If the content you type does not appear correctly—possibly due to pre-filled or conflicting text—clear the input box by pressing "Ctrl + A" to select all text, followed by "Delete" to erase it. Once the field is empty, you can re-enter your text.
6. When you need to use the web browser, if the current page is not the target page, you must use keyboard shortcut "Ctrl+T" to open a new tab.
7. When you are on the login page or some other pages that require user's confirmation, if it is a web page, try to close the web page and open a new one, if it is not a web page, you should not do any actions, you should say "Next Action": "None" in the json field.
8. When you want to enter some text into a text input field, such as a search bar, you need to check if the text input field is focused, if it is not focused, you should use "CLICK" action to click on the text input field, then enter the text.
9. When you encounter a CAPTCHA security verification page at the beginning of the task, you should close the current page and open a new one.
10. When accessing a website, it is preferable to enter the website address directly into the browser's address bar. If the exact address is unknown, you can search for the website using a search engine.
11. To download an image from a website, right-click on the image. Then, in the context menu that appears, choose "Save image as..." or "图片另存为" to save it to your computer. When the save dialog box appears during image download, you need to click on the "File name" field to activate it before you can enter or edit the file name.
12. If you want to edit the content in a search box or text field (such as input or delete text), you must first activate the input field before performing any editing actions.
13. Pay attention to the type of action: double_click to open an application, click to select an application.
14. If the task does not involve file system operations, you should focus on the following: After completing text input (TEXT_INPUT), use the ENTER shortcut key (Return key) to execute the search operation.
You can interact with the computer using a mouse and keyboard based on the given task and screenshot.
Your interaction is limited to the desktop GUI. You can use the terminal application or access the application menu.
You may be given some history of planned actions, which are responses from previous iterations of this loop.
You should carefully analyze and adjust your plan based on the current task, screenshot, and previous actions.
You can interact with the computer using a mouse and keyboard based on the given task and screenshot.
Your interaction is limited to the desktop GUI. You can use the terminal application or access the application menu.
You may be given some history of planned actions, which are responses from previous iterations of this loop.
You should carefully analyze and adjust your plan based on the current task, screenshot, and previous actions.

Figure 5: System requirements Prompt.

KEY_COMMAND_NOTES
 # Keyboard Command Guidelines
Supported Function Keys
The following function keys are supported:
- Control keys: ctrl, alt, shift, super
- Navigation keys: enter, esc, tab, up, down, left, right
- Editing keys: backspace, delete
- Page navigation: page_up, page_down, home, end

Key Format Specifications
- Use "+" to connect keys for simultaneous key combinations
Example: "CTRL+A" (press Ctrl and A at the same time)
Example: "CTRL+SHIFT+ESC" (press Ctrl, Shift, and Esc simultaneously)
- Use space to indicate sequential key presses (non-combined)
Example: "A B C" (press A, then B, then C in sequence)
Example: "CTRL+C CTRL+V" (press Ctrl+C and release, then press CTRL+V)

Usage in Actions
When using the KEY action, specify the key or key combination in the "value" field:
```json
{
"Next Action": "KEY",
"value": "CTRL+A"
}
```

Figure 6: Key command note Prompt.

Action and Solution Format

The GUI agent outputs a JSON object with three key components: reasoning, subtask plan, and next action. An example output is shown below:

```
{
  "Reasoning": "Observation: ... Intent: ... Analysis: ...",
  "subtask": [
    "[x] Open the browser",
    "[ ] Enter the target URL",
    "[ ] Check whether the page has loaded"
  ],
  "Next Action": "CLICK",
  "pos": "521, 41",
  "value": null
}
```

The GUI Verify model outputs a JSON object with two key components: reasoning, eval. An example output is shown below:

```
{
  "Reasoning": "...",
  "Eval": "yes"
}
```

Ground-truth solutions are represented using lightweight

```
<action>CLICK</action><bboxes>[5, 88, 183, 149]</bboxes>
<action>TYPE</action><value>Hello World</value>
<action>KEY</action><value>ctrl+a</value>
<action>DRAG</action><value>170,367 to 170,157</value>
```

Figure 7: Action and Solution Format And Ground-truth solutions.