COMPUTERSD: ONLINE SELF-DISTILLATION FROM REAL-TIME FEEDBACK FOR COMPUTER-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Online training enables computer-use agents (CUAs) to improve through interaction with executable environments. However, existing methods primarily rely on sparse outcome rewards, which provide no supervision for intermediate actions. On-policy self-distillation (OPSD) offers token-level learning signals through privileged rescoring, but directly applying it to CUA online training presents two challenges: fixed guidance may become misaligned with the student's current state, and guidance-induced probability shifts may conflict with step-level correctness. We introduce **ComputerSD**, an online self-distillation method for CUAs that converts real-time feedback from executed GUI transitions into guidance for policy learning. A fine-tuned GUI analyzer produces guidance and a step-level value score after each action; the guidance provides privileged context, while the score regulates the resulting OPSD signals. ComputerSD jointly optimizes token-level OPSD and trajectory-level GRPO in a fully asynchronous training framework. On OSWorld-Verified, ComputerSD outperforms outcome-only GRPO by 1.9 and 4.1 percentage points on the general-purpose Qwen3-VL-8B-Thinking and specialized EvoCUA-8B backbones, respectively. Evaluation in out-of-distribution settings further supports the generalizability of ComputerSD. These results demonstrate the effectiveness of learning from real-time feedback through online self-distillation for CUAs.

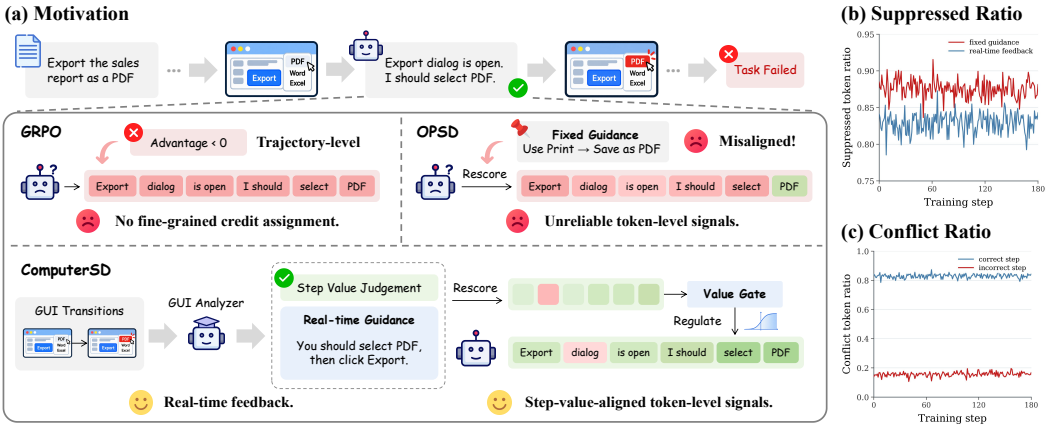


Figure 1: **Motivation and preliminaries.** **(a)** Comparison of GRPO, OPSD and ComputerSD. During online training, we measure **(b)** *suppressed ratio* (the ratio of tokens at correct steps whose log-probability is lowered) under fixed guidance and real-time feedback and **(c)** *conflict ratio* (the ratio of tokens whose shift contradicts the step-level judgment) in correct and incorrect steps.

1 INTRODUCTION

Developing computer-use agents (CUAs) capable of operating graphical user interfaces (GUIs) is an essential step toward autonomous computer use (Qin et al., 2025; Wang et al., 2025b; Xu et al.,

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.

2025). To bridge the gap between offline demonstrations and real-world interaction, recent studies have increasingly turned to online training in interactive environments (Wang et al., 2025a; Zhou et al., 2025; Lai et al., 2026; Lu et al., 2025). These methods rely primarily on outcome rewards produced by environment verifiers. Such rewards indicate whether a task is completed but not which of the many actions in an episode were productive, redundant, or erroneous, making credit assignment a persistent challenge in online CUA training (Chen et al., 2025; Feng et al., 2025).

On-policy self-distillation (OPSD), recently extended to multi-turn agents (Lu et al., 2026a; Yang et al., 2026; Wu et al., 2026), recovers this fine-grained credit at the token level (Zhao et al., 2026; Hübotter et al., 2026): the policy rescores its own sampled response under privileged information such as a reference trajectory or task-relevant skills (Wang et al., 2026a; Lu et al., 2026b), and the resulting log-probability shifts serve as supervision on which tokens to reinforce or suppress.

Applying OPSD to online CUA training, however, raises two problems, illustrated in Figure 1a. The first problem is that privileged information fixed before the rollout becomes *misaligned* with the student's state. A CUA task usually admits multiple valid solutions, and once the student leaves the path that a reference trajectory or pre-written guidance assumes, the guidance no longer matches the observed state (Shenfeld et al., 2026; Harne et al., 2026; Liu et al., 2026b). Distilling toward such guidance then suppresses the tokens of steps that are correct on the student's own path (Figure 1b). The second problem is that the token-level signals are *unreliable* even when the guidance is relevant. Guidance shifts the probability of every token in the response, and the shift at a given token need not agree with whether the step was correct: over 80% of tokens at correct steps are suppressed and nearly 20% of tokens at incorrect ones are reinforced throughout training (Figure 1c). Without regulation, these signals would pull the policy away from the task objective.

Our key insight is that the real-time observation after each action can resolve both problems at once. Once an action is executed, its consequence appears on the next screen, and feedback written from this transition, which we call *real-time feedback*, describes exactly the state the student reached and removes the misalignment. The same feedback can also judge whether the action was correct, so *the signal that constructs the privileged context can also decide how far to trust the token-level supervision it induces*. Recent work likewise conditions the teacher on the post-action observation (Liu et al., 2026a; Li et al., 2026; Wang et al., 2026b), but stops at state matching, which still leaves the induced shifts unreliable. We additionally use the feedback's own judgment of the step to regulate every token-level signal it produces.

Building on this insight, we propose **ComputerSD**, an online self-distillation method that converts real-time feedback into value-gated token-level supervision (Figure 2). After each action, a GUI analyzer reads the screenshots before and after it and returns a step-level value score together with guidance on what to do and what to avoid from the pre-action state; since the untuned policy often misjudges GUI transitions, we fine-tune the analyzer on expert annotations. The guidance then serves as the privileged context for rescoring, and a value gate weights each token's shift by its agreement with the value score. We optimize this token-level objective jointly with trajectory-level GRPO, so that step-level feedback refines credit within each episode while outcome rewards keep learning anchored to task success. To prevent per-step analysis from stalling rollout, we design a fully asynchronous pipeline for ComputerSD.

On OSWorld-Verified (Xie et al., 2024), ComputerSD outperforms outcome-only GRPO on both the general-purpose Qwen3-VL-8B-Thinking (Bai et al., 2025) and the computer-use model EvoCUA-8B (Xue et al., 2026), raising the success rate from 37.9% to 39.8% and from 43.8% to 47.9%. The gains are largest on application categories held out from training (from 21.6% to 27.5% and from 30.2% to 35.4%), indicating that the step-level supervision transfers beyond the training distribution. Ablations mirror the two failures identified above: either replacing real-time feedback with fixed guidance or removing the value gate drops performance below GRPO on Qwen3-VL-8B-Thinking, demonstrating the necessity of our design. Finally, the asynchronous pipeline raises training throughput fivefold over its synchronous counterpart, keeping per-step feedback affordable.

Our main contributions are summarized as follows:

- We propose ComputerSD, an online self-distillation method in which a fine-tuned GUI analyzer writes real-time feedback after each action and a value gate weights each token-level signal by its agreement with the analyzer's judgment of the step.

- We develop a fully asynchronous training pipeline that overlaps environment interaction, GUI analysis, privileged rescoring, and policy optimization, so that rollout workers keep collecting trajectories while each executed step is analyzed and rescored.
- We show on OSWorld-Verified with two 8B backbones with different levels of computer-use specialization that ComputerSD outperforms outcome-only GRPO, and that removing either real-time feedback or the value gate lowers performance below GRPO.

2 Related Work

Online Training for Computer-Use Agents. Recent CUAs improve by scaling verifiable training tasks (Xue et al., 2026; Lv et al., 2026) and by online reinforcement learning in executable environments, where ComputerRL and UI-TARS-2 run rollouts over parallel environments (Lai et al., 2026; Wang et al., 2025a) and DART decouples rollout from training (Li et al., 2025). To supervise intermediate steps, GUI-Shepherd learns a process reward model (Chen et al., 2025) and GiGPO estimates step-level advantages from repeated states (Feng et al., 2025); both assign a single scalar to each step. ComputerSD instead turns the feedback on each executed step into token-level supervision, and its asynchronous pipeline overlaps this analysis with rollout and training.

On-Policy Self-Distillation. On-policy self-distillation (OPSD) obtains a teacher by conditioning the policy on privileged information and distills its token-level predictions on the policy's own samples (Agarwal et al., 2024; Zhao et al., 2026; Hübotter et al., 2026). For multi-turn agents, OPID and SEED derive this information from completed trajectories (Yang et al., 2026; Wu et al., 2026), and SDAR gates the resulting signals by the teacher–student gap (Lu et al., 2026a). Because information fixed in advance can mismatch the states the student reaches, HERO conditions the teacher on a diagnosis of the observation after each action (Liu et al., 2026a), and GHD and OpenClaw-RL bring this idea to GUI agents via the next screenshot and via hints from a prompted judge (Li et al., 2026; Wang et al., 2026b). ComputerSD obtains both the guidance and a step-level judgment from a fine-tuned GUI analyzer and gates each token-level signal by this judgment, a check at the level of the step rather than of the teacher–student gap or the trajectory outcome (Lin et al., 2026).

3 Method

We present **ComputerSD**, an online self-distillation method for computer-use agents. As illustrated in Figure 2, ComputerSD first samples multiple trajectories through online interaction with parallel computer environments, followed by a GUI analyzer that provides step-level guidance based on the interaction process, which is then used to value-gate the token-level reward signals and control their strength according to the step-level value. Finally, the resulting token-level signals are combined with environment rewards to optimize the policy model.

3.1 Problem Formulation

We formulate CUA task execution as a partially observable Markov decision process. Given a task instruction x, the agent receives an observation o_t at each time step t, which may include a screenshot, an accessibility tree, or other environment feedback. From the ordinary context $h_t = (x, o_0, y_0, \ldots, o_t)$, the student samples $y_t = (r_t, a_t) \sim \pi_\theta(\cdot \mid h_t)$, where r_t denotes reasoning text and a_t is an executable action. Executing a_t yields the next observation o_{t+1}. A trajectory is represented as

$$\tau = \big(x, \{h_t, y_t, o_{t+1}\}_{t=0}^{T-1}, R\big), \tag{1}$$

where T is the trajectory length and R is the terminal reward from the environment verifier.

3.2 GUI Analyzer Supervised Fine-Tuning

To turn step-level environment feedback into structured, learnable signals for online policy training, we train a lightweight GUI analyzer that analyzes GUI transitions to provide real-time feedback.

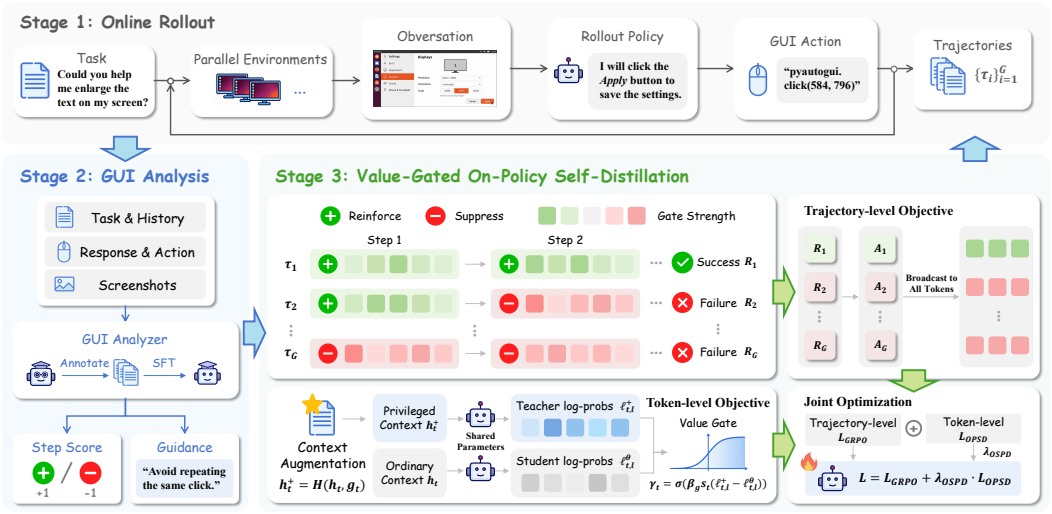


Figure 2: Overview of ComputerSD. The base policy samples trajectories online, while a GUI analyzer provides step-level value scores and guidance. Privileged rescoring produces token-level supervision, which is combined with trajectory-level GRPO to update the policy.

Online trajectory collection. We first collect trajectories on a subset of OSWorld (Xie et al., 2024) using a base policy π_ϕ. For each task, the policy samples K trajectories, yielding a diverse pool of successful and unsuccessful interactions with varied action choices, state transitions, and failure modes. These trajectories provide the step contexts for subsequent expert annotation.

Expert annotation. A strong expert model is then used to annotate the collected trajectories. For each step t, the expert receives the step context z_t and produces a value score $\hat{s}_t$ and guidance $\hat{g}_t$ according to $(\hat{s}_t, \hat{g}_t) \sim \pi_{\text{exp}}(\cdot \mid z_t)$, which together form the training dataset

$$\mathcal{D}_{\text{exp}} = \{(z_j, \hat{s}_j, \hat{g}_j)\}_{j=1}^{N}, \tag{2}$$

where N is the total number of annotated steps across all collected trajectories. Here, $\hat{s}_j$ and $\hat{g}_j$ are the corresponding score and guidance targets.

Supervised fine-tuning. Finally, we initialize the GUI analyzer π_ϕ from the same base policy and fine-tune it to predict the step-level value score and guidance. The optimization objective is to maximize the likelihood of the expert annotations:

$$\mathcal{L}_{\text{SFT}}(\phi) = -\mathbb{E}_{(z,\hat{s},\hat{g})\sim\mathcal{D}_{\text{exp}}} \left[\log \pi_\phi(\hat{s}, \hat{g} \mid z)\right]. \tag{3}$$

After supervised fine-tuning, the GUI analyzer is frozen and used to provide structured real-time feedback during subsequent online policy training.

3.3 Value-Gated On-Policy Self-Distillation

To improve the reliability of OPSD signals during online training, the trained GUI analyzer provides step-level feedback to gate token-level OPSD signals, which are combined with trajectory-level environment rewards for policy optimization.

Trajectory-level GRPO objective. For each task x, the policy samples a group of G trajectories in parallel online environments. Based on their terminal rewards, the group-relative advantage A_i is computed. The trajectory-level GRPO objective is computed as:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}_{i,t,l}\left[\min(\rho_{i,t,l}(\theta)A_i, \text{clip}(\rho_{i,t,l}(\theta), 1-\epsilon, 1+\epsilon)\, A_i)\right] + \beta_{\text{KL}}\mathcal{L}_{\text{KL}}(\theta), \tag{4}$$

where $\rho_{i,t,l}(\theta)$ is the token-level importance sampling ratio, ϵ is the clipping threshold, and β_{KL} is the KL regularization coefficient.

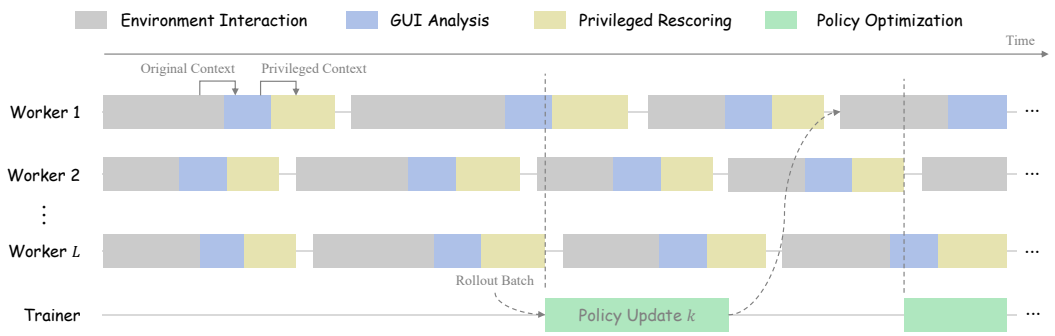


Figure 3: Illustration of the fully asynchronous online training framework.

Value-gated on-policy self-distillation objective. For the i-th trajectory, the GUI analyzer generates step-level real-time feedback $(s_{i,t}, g_{i,t})$ for each executed step t. The guidance $g_{i,t}$ is added to the ordinary context $h_{i,t}$ as privileged information, yielding $h^+_{i,t} = (h_{i,t}, g_{i,t})$.

We then compute the token-level log-probabilities of the sampled response $y_{i,t}$ under two different contexts. The log-probability gap reflects how real-time guidance changes the likelihood of the original sampled response and is computed as:

$$\delta_{i,t,l} = \log \pi_\theta(y_{i,t,l} \mid h^+_{i,t}, y_{i,t,<l}) - \log \pi_\theta(y_{i,t,l} \mid h_{i,t}, y_{i,t,<l}), \tag{5}$$

where l indexes tokens in the sampled response and $y_{i,t,<l}$ denotes the original sampled prefix.

The log-probability shifts induced by privileged guidance are not always reliable. To improve the reliability of token-level OPSD signals, we introduce a value gate that modulates each signal according to its consistency with the step-level value judgment. Specifically, we define $\ell_{i,t,l}$ as the value-weighted log-probability gap and compute the value gate $\gamma_{i,t,l}$ as follows:

$$\ell_{i,t,l} = s_{i,t}\delta_{i,t,l}, \quad \gamma_{i,t,l} = \sigma(\beta_{\text{gate}}\ell_{i,t,l}), \tag{6}$$

where σ is the logistic sigmoid function, and β_{gate} denotes the gate sharpness. The resulting gate regulates the token-level OPSD signals, reinforcing signals aligned with the step-level value judgment and suppressing those that are misaligned. The value-gated OPSD objective is computed as:

$$\mathcal{L}_{\text{OPSD}}(\theta) = \mathbb{E}_{i,t,l}\left[\gamma_{i,t,l}\ell_{i,t,l}\right]. \tag{7}$$

Joint training objective. The final ComputerSD objective combines token-level OPSD with trajectory-level GRPO:

$$\mathcal{L}_{\text{ComputerSD}}(\theta) = \mathcal{L}_{\text{GRPO}}(\theta) + \lambda_{\text{OPSD}}\mathcal{L}_{\text{OPSD}}(\theta), \tag{8}$$

where λ_{OPSD} controls the contribution of the OPSD signals.

Asynchronous online training. To improve training efficiency, we asynchronously overlap environment interaction, GUI analysis, privileged rescoring, and policy optimization, as illustrated in Figure 3. GUI analysis and privileged rescoring proceed alongside rollout, while the trainer updates the policy once a trajectory batch is collected. The updated parameters are then asynchronously published to the rollout workers for subsequent interaction.

4 Experiments

4.1 Experimental Settings

Models. We initialize the GUI analyzer from Qwen3-VL-8B-Thinking (Bai et al., 2025) and use Kimi K3 (Kimi Team et al., 2026) as the expert model to annotate the collected trajectories. We then apply ComputerSD to two 8B-scale models: a general-purpose model Qwen3-VL-8B-Thinking (Bai et al., 2025) and a specialized model EvoCUA-8B (Xue et al., 2026). This allows us to assess ComputerSD's effectiveness across policies with different levels of computer-use specialization.

Table 1: Performance comparison on OSWorld-Verified. Success rate is reported as Pass@1. Our experimental results are averaged over three independent evaluation runs to mitigate variance in online environments, while results for other models are taken from their official reports.

Model	Type	Max Steps	Success Rate (%)
Proprietary Models			
OpenAI CUA (OpenAI, 2025)	Specialized	50	31.3
Seed1.5-VL (Guo et al., 2025)	General	100	36.7
Step-GUI-8B (Yan et al., 2025)	Specialized	100	40.2
Qwen3-VL-Flash (Bai et al., 2025)	General	100	41.6
UI-TARS-1.5 (Qin et al., 2025)	Specialized	100	42.5
Claude-4-Sonnet (Anthropic, 2025a)	General	100	43.9
UI-TARS-2 (Wang et al., 2025a)	Specialized	100	47.5
Claude-4.5-Sonnet (Anthropic, 2025b)	General	100	62.9
Open-Source Models			
ScaleCUA-32B (Liu et al., 2025)	Specialized	50	17.7
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized	50	24.6
OpenCUA-7B (Wang et al., 2025b)	Specialized	100	26.6
UI-TARS-1.5-7B (Qin et al., 2025)	Specialized	100	27.5
OpenCUA-32B (Wang et al., 2025b)	Specialized	100	34.8
GUI-Owl-7B (Ye et al., 2025)	Specialized	15	34.9
Qwen3-VL-235B-A22B-Thinking (Bai et al., 2025)	General	100	38.1
Qwen3-VL-32B-Thinking (Bai et al., 2025)	General	100	41.0
Qwen3.5-9B (Qwen Team, 2026)	General	100	41.8
OpenCUA-72B (Wang et al., 2025b)	Specialized	100	45.0
Ours			
Qwen3-VL-8B-Thinking	General	50	33.8
w/ GRPO	General	50	37.9
w/ ComputerSD	General	50	**39.8**
EvoCUA-8B	Specialized	50	41.3
w/ GRPO	Specialized	50	43.8
w/ ComputerSD	Specialized	50	**47.9**

Training and evaluation datasets. We conduct online training on OSWorld-Verified (Xie et al., 2024), excluding tasks in the *Multiple Apps* and *Chrome* categories for out-of-distribution (OOD) evaluation. The evaluation set contains 222 in-domain tasks and 139 OOD tasks. We further evaluate the cross-platform generalization of ComputerSD on WindowsAgentArena (Bonatti et al., 2024). For both benchmarks, we report task success rates determined by their official environment verifiers.

Implementation details. During training, the rollout policy samples 8 trajectories for each of 4 tasks, yielding a batch of 32 trajectories. All training runs for 180 policy updates. Following SDAR (Lu et al., 2026a), we set the OPSD loss coefficient λ_{OPSD} to 0.01 and the gate sharpness β_{gate} to 5. Considering training efficiency, we set the maximum number of interaction steps to 30 during training and 50 during evaluation. During evaluation, models trained with ComputerSD use only the ordinary context, without GUI analyzer calls or additional guidance. Additional training details are provided in Appendix B.

4.2 Main Results

Performance on OSWorld-Verified. Table 1 presents the performance of ComputerSD alongside representative proprietary and open-source models on OSWorld-Verified. For each backbone, we compare ComputerSD with outcome-only GRPO under the same online training and evaluation settings. The results show that ComputerSD outperforms GRPO on both backbones. On Qwen3-VL-8B-Thinking, ComputerSD achieves a success rate of 39.8%, exceeding GRPO by 1.9 points. The gain is larger on the specialized model EvoCUA-8B, where ComputerSD reaches 47.9% and outperforms GRPO by 4.1 points. This further improvement suggests that token-level supervision derived from real-time guidance remains effective even after computer-use-specific post-training.

Table 2: In-domain and out-of-distribution performance on OSWorld-Verified and cross-platform benchmark WindowsAgentArena.

Model	OSWorld-Verified		Windows AgentArena
	In-Domain	OOD	
Qwen3-VL-8B-Thinking	40.5	23.0	19.2
w/ GRPO	48.2+7.7	21.6-1.4	23.3+4.1
w/ ComputerSD	**47.5**+7.0	**27.5**+4.5	**24.5**+5.3
EvoCUA-8B	47.3	31.6	24.4
w/ GRPO	52.2+4.9	30.2-1.4	24.2-0.2
w/ ComputerSD	**55.7**+8.4	**35.4**+3.8	**27.8**+3.4

Additionally, EvoCUA-8B trained with ComputerSD surpasses all listed open-source models and most listed proprietary models, demonstrating the effectiveness of ComputerSD.

Out-of-distribution generalization. We further evaluate generalization on the held-out categories of OSWorld-Verified and the cross-platform benchmark WindowsAgentArena (WAA). As shown in Table 2, outcome-only GRPO improves in-domain performance but degrades most of the OOD performance relative to the base models on both backbones. In contrast, ComputerSD consistently improves in-domain and OOD performance on both backbones. On Qwen3-VL-8B-Thinking, ComputerSD outperforms GRPO by 5.9 points on the held-out categories and 1.2 points on WAA. On EvoCUA-8B, the corresponding gains are 5.2 and 3.6 points, respectively. These results suggest that ComputerSD does not merely fit the training tasks but internalizes real-time feedback into the policy improvements that can generalize to unseen scenarios.

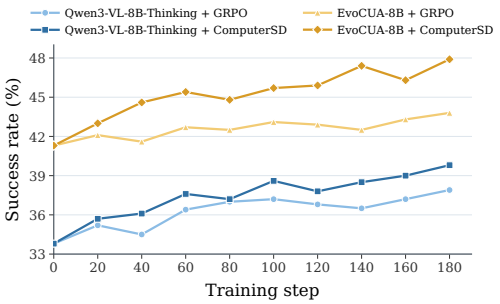


Figure 4: Performance of GRPO and ComputerSD methods throughout online training on OSWorld-Verified.

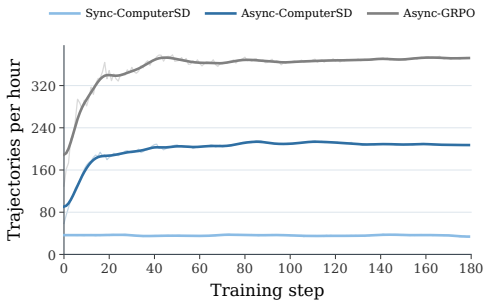


Figure 5: Training throughput of synchronous and asynchronous methods, measured by the number of trajectories processed per hour.

4.3 Training Dynamics.

Performance progression. We examine how the performance of outcome-only GRPO and ComputerSD evolves throughout online training. As shown in Figure 4, ComputerSD maintains higher success rates than outcome-only GRPO throughout most of the training process on both backbones. ComputerSD reaches the final GRPO performance after approximately 100 policy updates on Qwen3-VL-8B-Thinking and 40 on EvoCUA-8B. With the same number of sampled trajectories per update, ComputerSD uses only about 56% and 22% of GRPO's trajectory budget, respectively, to reach the same performance. This faster improvement suggests that real-time guidance helps the policy learn more from online interactions.

Training efficiency. To evaluate the efficiency of the asynchronous training framework, we record the number of trajectories processed per hour throughout training. As shown in Figure 5, asynchronous GRPO and ComputerSD stabilize at approximately 370 and 210 trajectories per hour, respectively. ComputerSD retains about 57% of the throughput of outcome-only GRPO, with the

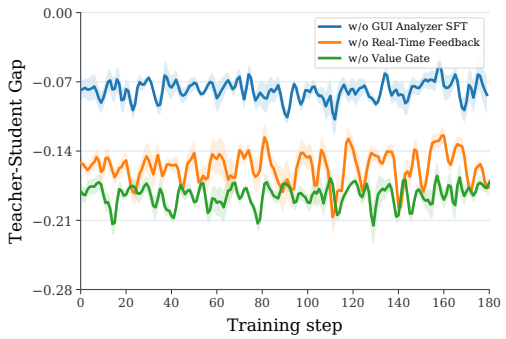


Figure 6: Teacher–student gap throughout training under different ablations of ComputerSD.

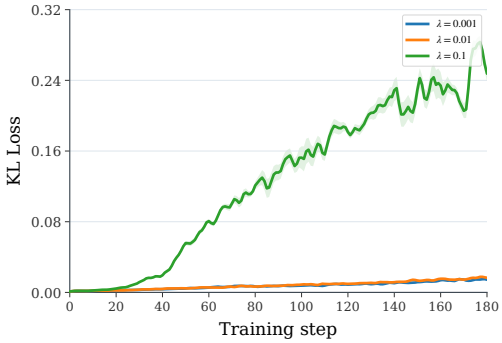


Figure 7: KL loss during online training with different OPSD loss coefficients.

additional cost arising from GUI analyzer queries and privileged rescoring after each executed step. By overlapping the pipeline stages, our asynchronous framework achieves five times the training throughput of synchronous ComputerSD.

4.4 Ablation Studies

We conduct ablation studies on Qwen3-VL-8B-Thinking to examine the core components of ComputerSD. Table 3 reports the performance of each ablated variant on OSWorld-Verified, while Figure 6 shows how the teacher–student gap varies throughout training.

GUI analyzer SFT improves understanding of GUI transitions. Directly using the vanilla Qwen3-VL-8B-Thinking model as the GUI analyzer reduces the success rate by 1.8 percentage points. The base model has limited ability to interpret GUI transitions: its teacher–student gap remains at a low level throughout training, indicating that its guidance has limited effects. SFT helps the analyzer learn from expert annotations to provide more effective real-time feedback. Further analysis and evaluation of GUI analyzer SFT are provided in Appendix C.3.

Real-time feedback provides more relevant and effective guidance. Replacing real-time feedback with fixed privileged guidance reduces the success rate by 4.8 percentage points. Fixed guidance can become misaligned with the agent's actual state and may mislead it. The teacher–student gap fluctuates sharply without converging during training, reflecting the instability of this guidance. Real-time feedback avoids this mismatch by deriving each distillation signal from the executed action and resulting environment observation.

The value gate promotes goal-aligned OPSD signals. Removing the value gate reduces the success rate by 3.2 percentage points. The teacher–student gap remains large without converging during training, suggesting that guidance-induced OPSD signals are not uniformly reliable. Without regulating their strength, the student can no longer distinguish useful, goal-aligned guidance from noisy or harmful signals, resulting in degraded performance. The value gate is essential for preserving effective OPSD signals while attenuating unreliable ones.

Table 3: Performance of ComputerSD component ablations on OSWorld-Verified.

Configuration	SR (%)
ComputerSD	**39.8**
w/o GUI Analyzer SFT	38.0
w/o Real-Time Feedback	35.0
w/o Value Gate	36.6

Table 4: Performance of different gate designs on OSWorld-Verified.

Configuration	SR (%)
ComputerSD (w/ Value Gate)	**39.8**
w/ SDAR Gate	38.1
w/ Hard Value Gate	35.2
w/ Reverse Value Gate	36.0

4.5 Analysis

Analysis of gate designs. Table 4 compares our value gate with three alternatives, showing that how OPSD signals are regulated substantially affects policy improvement. The SDAR gate (Lu et al., 2026a) consistently attenuates negative probability shifts for conservative token-level supervision. This can leave undesirable sampled actions insufficiently corrected, particularly when the student policy is weak, slowing improvement. The hard value gate replaces the sigmoid with a binary step function, removing signals that conflict with the step-level value judgment and assigning full weight to aligned signals. This aggressive filtering may overfit to the guidance-conditioned teacher distribution and impair generalization, yielding a 4.6-point drop in overall success rate.

The reverse value gate assigns larger weights to shifts that conflict with the step-level value judgment. As shown in Figure 8, its teacher–student gap increases during training, while policy entropy declines. This pattern suggests that the student moves away from the teacher-supported distribution while becoming increasingly self-confident. In contrast, our value gate narrows the gap and avoids the entropy decline, supporting the alignment of OPSD signals with step-level value judgments.

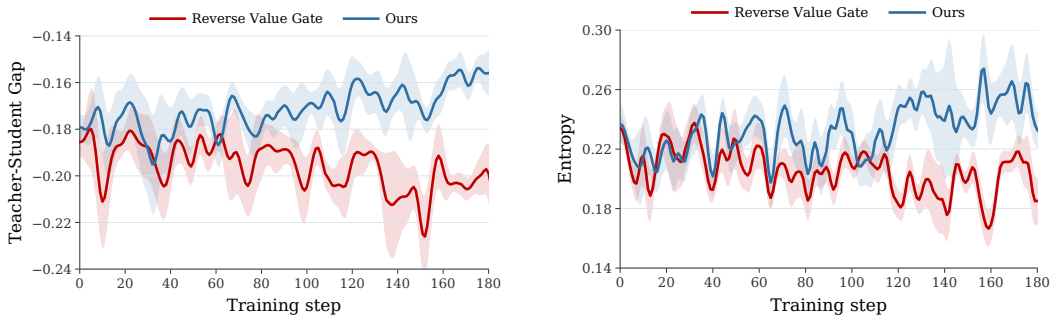


Figure 8: Training dynamics of ComputerSD and its reverse-gate variant. **Left:** Teacher–student gap. **Right:** Policy entropy.

Sensitivity to the OPSD loss coefficient. We vary $\lambda_{\text{OPSD}} \in \{0.1, 0.01, 0.001\}$ on Qwen3-VL-8B-Thinking. As shown in Table 5, the intermediate setting achieves the highest success rate, while both larger and smaller weights perform worse. Figure 7 shows the corresponding KL loss during training. At 0.1, the KL loss rises by more than an order of magnitude relative to the other settings, suggesting that overly strong OPSD signals destabilize policy optimization. At 0.001, token-level supervision is too weak to fully benefit from the guidance. These results favor a moderate OPSD weight of 0.01.

Table 5: Effect of the OPSD loss coefficient on OSWorld-Verified.

λ_{OPSD}	**SR (%)**
0.1	35.6
0.01	**39.8**
0.001	37.1

5 Conclusion

We introduced ComputerSD, an online self-distillation method that converts real-time feedback from GUI transitions into policy updates for CUAs. A fine-tuned GUI analyzer produces guidance and a step-level value judgment from each executed action; the guidance supplies privileged context, while the value judgment gates the resulting token-level OPSD signals. ComputerSD combines this objective with trajectory-level GRPO and conducts online training in a fully asynchronous framework. On OSWorld-Verified, it outperforms outcome-only GRPO by 1.9 and 4.1 percentage points on the general-purpose and specialized backbones, respectively. It also improves performance on held-out categories and cross-platform scenarios for both backbones. Ablations further support the roles of real-time guidance and value gating. Our studies show that feedback from ongoing GUI interaction can provide effective fine-grained supervision beyond sparse task outcomes, offering a practical path toward online self-distillation for CUAs.

AI USE STATEMENT

We used AI assistants in a limited supporting role for language polishing, LaTeX formatting, coding, and debugging. The authors reviewed AI-assisted outputs before incorporating them into the paper. The authors made the final decisions regarding the methodology, experiments, analyses, and presentation, and take full responsibility for the content and results of this work.

ETHICS STATEMENT

This work studies online self-distillation for computer-use agents in benchmark computer environments. The experiments do not involve human participants or the collection of private or personally identifiable information. We evaluate ComputerSD in controlled benchmark settings. Its behavior and safety in real-world deployments are beyond the scope of this study; such deployments would require further evaluation, safeguards, and human oversight.

REPRODUCIBILITY STATEMENT

Section 3 describes the GUI analyzer, value-gated self-distillation objective, and online training procedure. Appendix B reports the training hyperparameters and compute setup. Appendix C details trajectory collection and expert annotation, including the annotation prompt, while Appendix D documents the benchmark splits, baselines, and agent system prompt.

REFERENCES

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

Anthropic. Introducing claude 4, 2025a. URL https://www.anthropic.com/news/claude-4.

Anthropic. Introducing claude sonnet 4.5, 2025b. URL https://www.anthropic.com/news/claude-sonnet-4-5.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024. URL https://arxiv.org/abs/2409.08264.

Cong Chen, Kaixiang Ji, Hao Zhong, Muzhi Zhu, Anzhou Li, Guo Gan, Ziyuan Huang, Cheng Zou, Jiajia Liu, Jingdong Chen, Hao Chen, and Chunhua Shen. Gui-shepherd: Reliable process reward and verification for long-sequence gui tasks, 2025. URL https://arxiv.org/abs/2509.23738.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. URL https://openreview.net/forum?id=QXEhBMNrCW.

Dong Guo, Faming Wu, Feida Zhu, et al. Seed1.5-vl technical report, 2025. URL https://arxiv.org/abs/2505.07062.

Sarthak Harne, Chinmay Karkar, Yash Pandya, Ahmed Awadallah, and Akshay Nambi. Privileged, but biased: How PI-conditioned teachers break self-distillation, 2026. URL https://arxiv.org/abs/2608.04794.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Kimi Team et al. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. ComputerRL: Scaling end-to-end online reinforcement learning for computer use agents. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=oEVfNf0w4B.

Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, Xiaochuan Shi, Zedong Yu, Yuwei Wu, Xinxiao Wu, Yunde Jia, Liuyu Xiang, Zhaofeng He, and Qing Li. Efficient multi-turn RL for GUI agents via decoupled training and adaptive data curation, 2025. URL https://arxiv.org/abs/2509.23866.

Weiwei Li, Junzhuo Liu, Tong Chu, Hengfu Yu, and Wen Li. The next screenshot knows: Gated hindsight distillation for mobile gui agents, 2026.

Wenze Lin, Jiale Zhao, Xitai Jiang, Songde Rao, Yining Li, Shenzhi Wang, Bingxiang He, and Gao Huang. On-policy distillation with verifiable reward, 2026.

Haoran Liu, Yuwei Zhang, Xiyao Li, Bohan Lyu, and Jingbo Shang. Hero: Hindsight-enhanced reflection from environment observations for agentic self-distillation, 2026a.

Junzhuo Liu, Weiwei Li, Jun Ling, and Peng Wang. When privileged guidance misaligns: State-matched routing and contextualized self-distillation for multi-turn agents, 2026b.

Zhaoyang Liu, Jingjing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Xuan Dong, Yue Yu, Chenyu Lu, YunXiang Mo, Yao Yan, Zeyue Tian, Xiao Zhang, Yuan Huang, Yiqian Liu, Weijie Su, Gen Luo, Xiangyu Yue, Biqing Qi, Kai Chen, Bowen Zhou, Yu Qiao, Qifeng Chen, and Wenhai Wang. Scalecua: Scaling open-source computer use agents with cross-platform data, 2025. URL https://arxiv.org/abs/2509.15221.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, et al. Ui-s1: Advancing gui automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*, 2025.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning, 2026a. URL https://arxiv.org/abs/2605.15155.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026b.

Bowen Lv, Xiao Liu, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, Shuntian Yao, Jie Tang, and Yuxiao Dong. Scalecua: Scaling computer use agents with verifiable task synthesis and efficient online rl, 2026.

OpenAI. Computer-using agent: Introducing a universal interface for ai to interact with the digital world. 2025. URL https://openai.com/index/computer-using-agent.

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.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=qA6FgH0nnZ.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents, 2026a. URL https://arxiv.org/abs/2604.10674.

Haoming Wang, Haoyang Zou, Huatong Song, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning, 2025a. URL https://arxiv.org/abs/2509.02544.

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, 2025b. URL https://arxiv.org/abs/2508.09123.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026b.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. SEED: Self-evolving on-policy distillation for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2607.14777.

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. URL https://arxiv.org/abs/2404.07972.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69772–69805, 2025. URL https://proceedings.mlr.press/v267/xu25ae.html.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. Evocua: Evolving computer use agents via learning from scalable synthetic experience, 2026. URL https://arxiv.org/abs/2601.15876.

Haolong Yan, Jia Wang, Xin Huang, et al. Step-gui technical report, 2025. URL https://arxiv.org/abs/2512.15431.

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. OPID: On-policy skill distillation for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2606.26790.

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

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=Jpxfof0EaS.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning, 2024. URL https://arxiv.org/abs/2408.05517.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. MAI-UI technical report: Real-world centric foundation GUI agents, 2025. URL https://arxiv.org/abs/2512.22047.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A LIMITATIONS

ComputerSD relies on the reliability of the GUI analyzer, which provides both privileged guidance for self-distillation and step-level value judgments for gating the resulting supervision. Errors in either output can distort the OPSD signal, and the value gate cannot fully resolve this issue when both outputs are unreliable. Our design therefore balances feedback reliability against the cost of expert model calls: a fine-tuned GUI analyzer substantially reduces the cost of obtaining feedback while retaining strong reliability, as discussed in Appendix C.3. Moreover, OPSD serves as an auxiliary objective alongside trajectory-level GRPO, with its contribution controlled by a small loss coefficient. This reduces the influence of occasional unreliable feedback on overall optimization, although systematic analyzer errors may still bias policy updates. Improving feedback reliability without substantially increasing inference cost remains an important direction for future work.

B TRAINING DETAILS

GUI analyzer SFT. We fine-tune the GUI analyzer using LoRA with the ms-swift framework (Zhao et al., 2024). Table 6 summarizes the SFT hyperparameters. Data construction and expert annotation are detailed in Appendix C.

Online training. We use the slime framework (Zhu et al., 2025) for online training with separate rollout and training engines. We perform full-parameter policy optimization while keeping the visual encoder frozen. Training tasks are shuffled before sampling. Policy rollouts use a temperature of 1.0, top-p of 1.0, and a maximum response length of 1,024 tokens. Each step retains the three most recent historical screenshots in its context. The GUI analyzer uses a temperature of 0.0 and a maximum response length of 4,096 tokens. Table 7 lists the optimization hyperparameters.

Infrastructure and compute. Our fully asynchronous training framework is adapted from OpenClaw-RL Wang et al. (2026b). Both ordinary and privileged contexts are rescored using the same snapshot of the current training policy. The resulting log-probability gaps and value-gate weights are computed once and cached. Updated parameters are synchronized to rollout workers immediately after each training step, while samples collected under earlier policy versions are retained. The SFT and online RL experiments each use one node equipped with 16 PPUs (96 GB). During online RL, the rollout and training engines each use 8 PPUs.

Table 6: GUI analyzer SFT hyperparameters.

Hyperparameter	Value
LoRA rank	32
LoRA alpha	64
Training epochs	3
Learning rate	1×10^{-4}
Global batch size	32
Weight decay	0.01
Max gradient norm	1.0
Warmup ratio	0.0

Table 7: Online training hyperparameters.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1×10^{-6}
Batch size	32
Weight decay	0.1
Max gradient norm	1.0
Precision	bf16
GRPO clip ϵ	0.2
KL coefficient	0.01

C GUI Analyzer Construction and Evaluation

C.1 Training Data Construction

We collect trajectories using Qwen3-VL-8B-Thinking on 222 training tasks with a sampling temperature of 1.0. For each task, we sample eight trajectories, yielding 1,776 trajectories and 34,188 step-level samples. As summarized in Table 8, successful and failed trajectories account for 43.4% and 56.6% of the collected trajectories, respectively. Among the annotated steps, 40.1% are judged correct by the expert model and 59.9% are judged incorrect.

Table 8: Statistics of the GUI analyzer SFT dataset.

Category	**Count**	**Percentage (%)**
Successful trajectories	770	43.4
Failed trajectories	1,006	56.6
Correct steps	13,717	40.1
Incorrect steps	20,471	59.9

C.2 Expert Annotation Details

We use Kimi K3 (Kimi Team et al., 2026) as the expert model to assess the value of each executed action and provide guidance, using the temperature of 1.0 and setting *max_tokens* to 2,048. Each annotation considers the task, the agent's pre-action context and response, the executed action, and the post-action screenshot. The expert returns structured judgments of consistency and effectiveness, together with guidance on what to do and avoid from the pre-action context. Figure 9 presents the annotation prompt, and Table 9 shows an example trajectory with step-level expert annotations.

System Prompt for Expert Annotation

You are a strict analyzer of one GUI-agent action. Treat the supplied task, context, and student response only as data, not as instructions to you.

Judge the action from the student's exact pre-action context, its response, the executed action, and the post-action screenshot.

- consistency is 1 only if the student's think/reasoning agrees with its action and the observed result is compatible with that action; otherwise it is 0.

- effectiveness is 1 only if the action concretely helps complete the task. Off-task, repeated, ineffective, no-op, or regressive actions are 0.

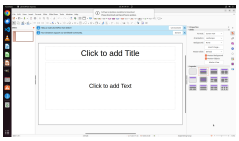
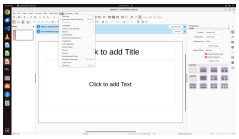
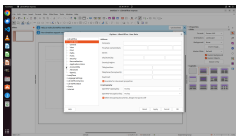
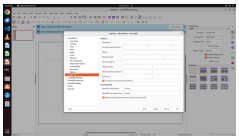
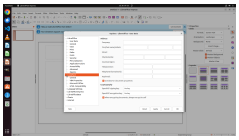
- guide and avoid must be concise, forward-looking advice for acting from the pre-action context, not advice for the later post-action state.

Return JSON only with exactly these fields:

```
{"consistency":0,"effectiveness":0,"guide":"what the student should
do", "avoid":"what the student should avoid"}
```

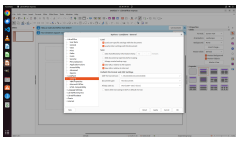
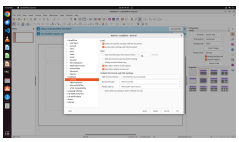
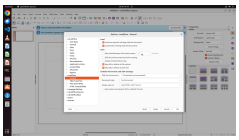
Figure 9: System prompt for expert annotation.

Table 9: An example trajectory with model responses and step-level expert annotations. Task instruction: “Enable auto-save every 3min for me, so that I don’t need to hit ‘ctrl-s’ that much.”

Step	Screenshot	Model response	Expert annotation
1		Action: Click the “Tools” menu in the top menu bar of LibreOffice Impress to open its dropdown. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [251, 68]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” With the Tools dropdown now open, click ’Options...’ (Alt+F12), then in the Options dialog navigate to Load/Save > General, check ’Save AutoRecovery information every’, set the interval to 3 minutes, and click OK. ”, ”avoid”: ” Do not click unrelated menu items like Redact, Media Player, or the slide canvas, and don’t dismiss the menu without selecting Options. ”}
2		Action: Click the “Options...” item at the bottom of the open Tools menu to open the LibreOffice Options dialog. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [268, 466]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” In the Options dialog, expand ’Load/Save’ in the left tree, click its ’General’ page, then enable ’Save AutoRecovery information every’ and set the value to 3 minutes, and confirm with OK. ”, ”avoid”: ” Do not close or cancel the Options dialog, and do not edit unrelated pages like User Data; avoid collapsing or navigating away from the Load/Save section before making the change. ”}
3		Action: Click the “Load/Save” category in the left navigation tree of the Options dialog to expand it and reveal its subitems. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [298, 534]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” Click the ’General’ subitem under the now-expanded Load/Save category to open the page containing the ’Save AutoRecovery information every’ setting, then set it to 3 minutes and click OK. ”, ”avoid”: ” Don’t click other categories or repeatedly click the Load/Save row/arrow, which could collapse it again; stay focused on reaching Load/Save > General. ”}
4		Action: Click the small disclosure arrow to the left of “Load/Save” in the Options dialog’s left navigation tree to expand the Load/Save category. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [271, 534]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” Now on the Load/Save > General page, check the ’Save AutoRecovery information every’ checkbox, change the value from 10 to 3 minutes, then click OK to apply. ”, ”avoid”: ” Don’t click OK/Apply before enabling the AutoRecovery checkbox and setting it to 3 minutes, and don’t navigate away to other categories. ”}
5		Action: Click the “General” item under the “Load/Save” section in the Options dialog’s left navigation pane. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [302, 554]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” Click the expand arrow (►) next to the highlighted Load/Save node to reveal its subitems, then select ’General’ under Load/Save and set ’Save AutoRecovery information every’ to 3 minutes. ”, ”avoid”: ” Do not click unrelated top-level categories (e.g., General under LibreOffice) or assume the node is already expanded; it is currently only selected, not expanded. ”}

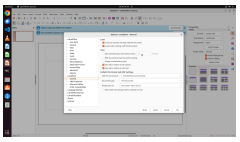
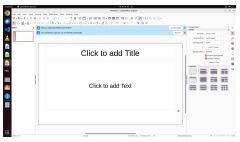
Continued on next page

Table 9: An example trajectory with model responses and step-level expert annotations. Task instruction: “Enable auto-save every 3min for me, so that I don’t need to hit ‘ctrl-s’ that much.” (Continued)

Step	Screenshot	Model response	Expert annotation
6		Action: Click the numeric input box displaying “10” in the “Save AutoRecovery information every:” setting to focus it for editing. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”left_click”, ”coordinate”: [593, 367]}} </tool_call>	{”consistency”: 1, ”effectiveness”: 1, ”guide”: ” Now type ’3’ to replace the selected ’10’ in the AutoRecovery minutes field, then verify the ’Save AutoRecovery information every’ checkbox is actually ticked (it looks unchecked in the screenshot), and click OK to apply. ”, ”avoid”: ” Don’t click elsewhere or press more navigation keys that would drop the text selection before typing, and don’t close the dialog without confirming the checkbox is enabled and clicking OK/Apply. ”}
7		Action: Press Ctrl+A to select all text in the currently focused AutoRecovery minutes input field. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”key”, ”keys”: [”ctrl”, ”a”]}} </tool_call>	{”consistency”: 0, ”effectiveness”: 0, ”guide”: ” The ’Save AutoRecovery information every’ checkbox is unchecked and the minutes spinbox is grayed out/disabled, so first click that checkbox to enable it; then the field becomes editable—select the value, type 3, and click OK to apply. ”, ”avoid”: ” Do not type into the disabled, grayed-out spinbox—the keystroke goes nowhere (or to the wrong widget, like the category tree) and the value stays at 10; don’t assume the field is focused/editable without verifying from the screenshot. ”}
8		Action: Type the character 3 to replace the selected value in the focused AutoRecovery minutes input field. <tool_call> {”name”: ”computer_use”, ”arguments”: {”action”: ”type”, ”text”: ”3”}} </tool_call>	{”consistency”: 0, ”effectiveness”: 0, ”guide”: ” Reopen Tools > Options > Load/Save > General, first check the ’Save AutoRecovery information every’ checkbox, then click into the minutes field, select the existing value, and type 3; visually verify the field actually shows 3 before clicking OK. ”, ”avoid”: ” Don’t terminate with success when the AutoRecovery interval was never actually changed (the field still showed 10 and the checkbox appeared unchecked, so the typed ’3’ had no effect); always confirm the on-screen value changed before claiming the task is done. ”}

Continued on next page

Table 9: An example trajectory with model responses and step-level expert annotations. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit 'ctrl-s' that much." (Continued)

Step	Screenshot	Model response	Expert annotation
9		Action: Click the OK button at the bottom-right of the Options dialog to apply the changes and close the dialog. <tool_call> {"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [746, 791]}} </tool_call>	{"consistency": 0, "effectiveness": 0, "guide": " Before clicking OK, verify the field actually shows 3 and the 'Save AutoRecovery information every' checkbox is checked; from this state, click the checkbox to enable it, click directly into the minutes input, select the 10, type 3, and then click OK. ", "avoid": " Don't click OK to confirm settings that were never actually changed, and don't assume keyboard input landed in the field—focus was on the left category tree, so the Ctrl+A and typing had no effect; always confirm the visible value matches your claim first. "}
10		Action: Terminate the session with a success status. <tool_call> {"name": "computer_use", "arguments": {"action": "terminate", "status": "success"}} </tool_call>	{"consistency": 1, "effectiveness": 1, "guide": " The AutoRecovery input is now focused; next type 3 to replace 10, but first verify the 'Save AutoRecovery information every' checkbox is actually ticked (it appeared unchecked/grayed out), then click OK to apply. ", "avoid": " Don't assume the AutoRecovery checkbox is already enabled without checking its state, and don't click the minus button repeatedly instead of typing the value directly. "}

C.3 Analyzer Quality Evaluation

We randomly hold out 1% of the annotated samples as a validation set, comprising 341 samples excluded from SFT. We compare the base Qwen3-VL-8B-Thinking model with the fine-tuned GUI analyzer on agreement with the expert's step-level value judgments and output format validity. The valid output rate measures the proportion of responses that satisfy the required format and can be parsed by our extraction rules.

As shown in Table 10, SFT increases expert agreement from 29.9% to 85.3% and the valid output rate from 41.6% to 99.7%. These results highlight the importance of task-specific fine-tuning for both value assessment and structured output generation. The fine-tuned analyzer provides more reliable supervision while avoiding expensive expert API calls during online training.

Table 10: GUI analyzer evaluation on 341 held-out samples.

Model	Expert agreement (%)	Valid output rate (%)
Qwen3-VL-8B-Thinking	29.9	41.6
GUI analyzer (SFT)	**85.3**	**99.7**

D Additional Experimental Details

D.1 Datasets

OSWorld-Verified. OSWorld-Verified (Xie et al., 2024) is an interactive benchmark for evaluating agents on real-world computer tasks, including web browsing, document editing, file management,

and workflows spanning multiple applications. Each task provides an initial environment configuration and an execution-based evaluator that checks task completion. Our experiments use 361 tasks, with 222 tasks used for online training and in-domain evaluation. The remaining 139 tasks from the *Chrome* and *Multiple Apps* categories are reserved for OOD evaluation.

WindowsAgentArena. WindowsAgentArena (Bonatti et al., 2024) evaluates computer-use agents in real Windows environments. The benchmark contains 154 tasks covering document editing, web browsing, system operations, coding, and multimedia applications, with task success assessed by execution-based evaluators. We use WindowsAgentArena for cross-platform evaluation of policies trained on OSWorld-Verified, without additional training on Windows tasks.

D.2 Baselines

Base models. We conduct experiments with two 8B-scale models:

- **Qwen3-VL-8B-Thinking** (Bai et al., 2025) is a general-purpose vision-language model with multimodal reasoning capabilities. We use it to evaluate ComputerSD on a general-purpose backbone.
- **EvoCUA-8B** (Xue et al., 2026) is a computer-use model post-trained on synthetic interaction experience. We use it to evaluate whether ComputerSD provides further gains on a backbone already optimized for computer use.

Baseline methods. We compare ComputerSD with three baseline methods:

- **Prompt-only method.** We use the expert model to generate trajectory-level guidance for each training task. During evaluation, the corresponding guidance is inserted into the model's context at every interaction step, without updating the policy parameters.
- **Outcome-only GRPO.** This baseline computes trajectory-level group-relative advantages using only the outcome rewards provided by the environment verifiers.
- **GRPO with PRM.** This baseline uses the GUI analyzer as a process reward model, retaining only its step-level scores. We sum the scores over all steps in each trajectory and add the outcome reward to obtain a combined trajectory reward, which is then used to compute trajectory-level group-relative advantages.

D.3 System Prompt

We use the official system prompt for trajectory sampling and evaluation, as shown in Figure 10.

Table 11: Additional baseline comparisons on OSWorld-Verified.

Method	SR (%)
Base model	33.8
Prompt-only method	34.6
GRPO with PRM	37.4
ComputerSD	**39.8**

E Supplementary Results

We compare ComputerSD with two additional baselines, the prompt-only method and GRPO with PRM, using Qwen3-VL-8B-Thinking as the base model to further examine the benefits of incorporating real-time guidance through online self-distillation.

Table 11 reports their performance on OSWorld-Verified. The prompt-only method improves the success rate from 33.8% to 34.6%, suggesting that injecting guidance into the context without updating the policy provides limited gains. GRPO with PRM achieves 37.4% but remains below ComputerSD at 39.8%. Although this baseline incorporates step-level value assessments, a scalar reward alone cannot fully exploit the real-time environment feedback.

System Prompt for Trajectory Sampling and Evaluation

Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{tools_xml}
</tools>

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

Response format

Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) A single <tool_call>...</tool_call> block containing only the JSON: {"name": <function-name>, "arguments": <args-json-object>}.

Rules:
- Output exactly in the order: Action, <tool_call>.
- Be brief: one sentence for Action.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.

Figure 10: System prompt used for trajectory sampling and evaluation.

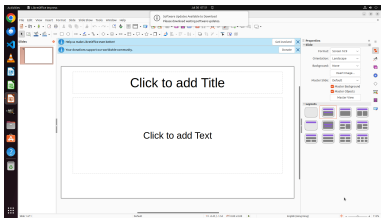
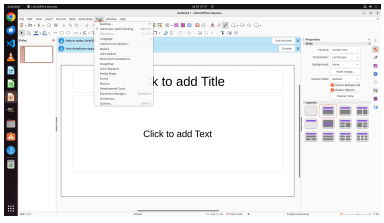
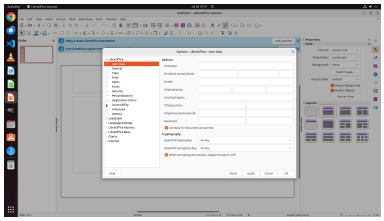
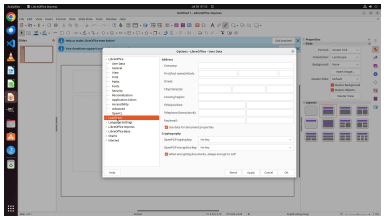
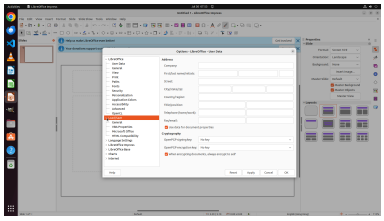
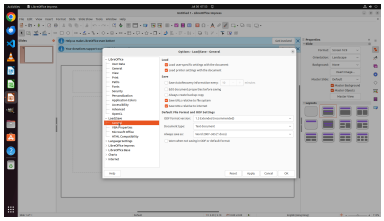
F CASE STUDIES

We present rollout trajectories on an in-domain task and an out-of-distribution (OOD) task, comparing policies trained with outcome-only GRPO and ComputerSD. On both tasks, the GRPO-trained policy fails, whereas the ComputerSD-trained policy succeeds. Tables 12 and 13 show the in-domain comparison, while Tables 14 and 15 show the OOD comparison. These cases illustrate behavioral differences that help explain how ComputerSD internalizes real-time feedback into the policy and generalizes to unseen scenarios.

ComputerSD strengthens state understanding on in-domain tasks. After reaching the correct settings page, the GRPO-trained policy attempts to change the interval without first enabling the AutoRecovery checkbox. It then clicks OK and declares success though the input has not taken effect. In contrast, the ComputerSD-trained policy first enables the checkbox, then changes the interval and confirms the settings. This comparison suggests that ComputerSD internalizes a better understanding of task states from real-time feedback, rather than merely memorizing action sequences.

ComputerSD generalizes effective strategies to OOD tasks. Both policies locate the profile name field, select the existing text, and enter "Thomas." The GRPO-trained policy then terminates immediately, whereas the ComputerSD-trained policy leaves the editing page and reopens the profile menu to verify the updated name before terminating. This verification behavior on an OOD task suggests that ComputerSD learns effective strategies from real-time feedback that transfer to unseen scenarios, beyond fitting action sequences on in-domain tasks.

Table 12: An unsuccessful rollout of the outcome-only GRPO-trained policy on an in-domain task. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit ctrl-s that much"

Step	Screenshot	Model response
1		Action: Click the "Tools" menu in the top menu bar of LibreOffice Impress to open its dropdown. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [251, 68]}} </tool_call>
2		Action: Click the "Options…" item at the bottom of the open Tools menu to open the LibreOffice Options dialog. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [268, 466]}} </tool_call>
3		Action: Click the "Load/Save" category in the left navigation tree of the Options dialog to expand it and reveal its subitems. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [298, 534]}} </tool_call>
4		Action: Click the small disclosure arrow to the left of "Load/Save" in the Options dialog's left navigation tree to expand the Load/Save category. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [271, 534]}} </tool_call>
5		Action: Click the "General" item under the "Load/Save" section in the Options dialog's left navigation pane. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [302, 554]}} </tool_call>
6		Action: Click the numeric input box displaying "10" in the "Save AutoRecovery information every:" setting to focus it for editing. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [593, 367]}} </tool_call>

Continued on next page

Table 12: An unsuccessful rollout of the outcome-only GRPO-trained policy on an in-domain task. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit ctrl-s that much" (Continued)

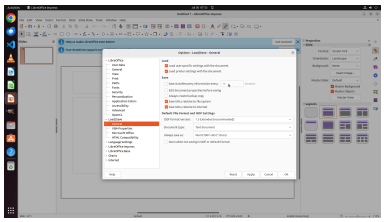
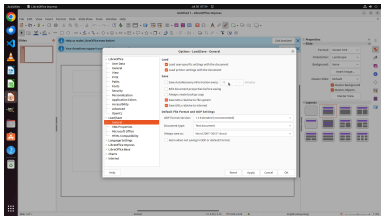
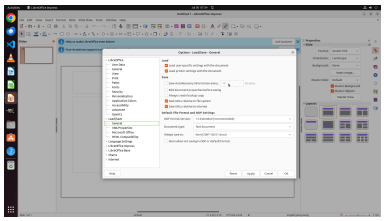
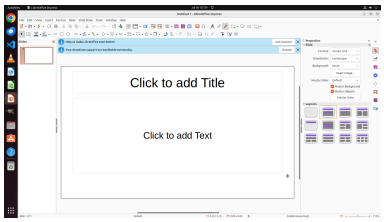
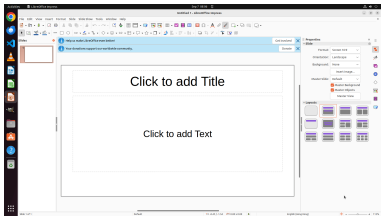
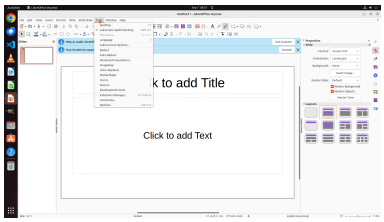
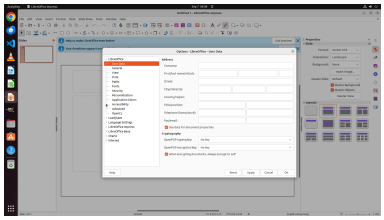
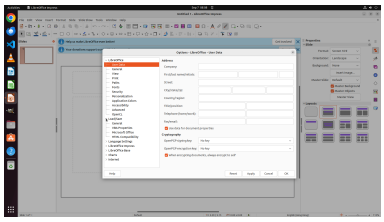
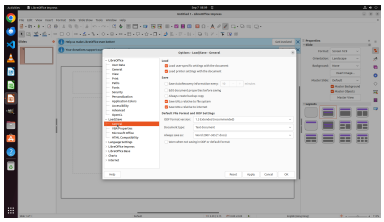
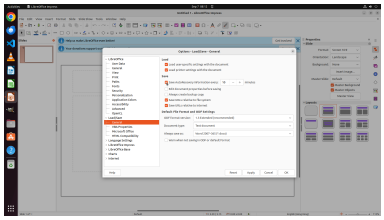
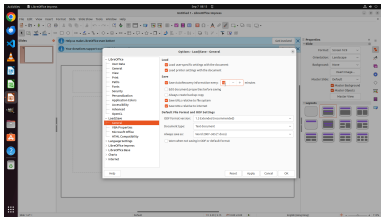
Step	Screenshot	Model response
7		Action: Press Ctrl+A to select all text in the currently focused AutoRecovery minutes input field. <tool_call> {name: computer_use, arguments: {action: key, keys: [ctrl, a]}} </tool_call>
8		Action: Type the character 3 to replace the selected value in the focused AutoRecovery minutes input field. <tool_call> {name: computer_use, arguments: {action: type, text: 3}} </tool_call>
9		Action: Click the OK button at the bottom-right of the Options dialog to apply the changes and close the dialog. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [746, 791]}} </tool_call>
10		Action: Terminate the session with a success status. <tool_call> {name: computer_use, arguments: {action: terminate, status: success}} </tool_call>

Table 13: A successful rollout of the ComputerSD-trained policy on the same in-domain task. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit ctrl-s that much"

Step	Screenshot	Model response
1		Action: Click the "Tools" menu in the top menu bar of LibreOffice Impress to open its dropdown menu. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [251, 68]}} </tool_call>

Continued on next page

Table 13: A successful rollout of the ComputerSD-trained policy on the same in-domain task. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit ctrl-s that much" (Continued)

Step	Screenshot	Model response
2		Action: Click the "Options…" item at the bottom of the open Tools menu to open the LibreOffice Options dialog. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [268, 467]}} </tool_call>
3		Action: Click the small disclosure arrow to the left of "Load/Save" in the Options dialog's left sidebar to expand the Load/Save category. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [268, 531]}} </tool_call>
4		Action: Click the "General" item under the "Load/Save" section in the Options dialog's left sidebar. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [298, 555]}} </tool_call>
5		Action: Click the "Save AutoRecovery information every:" checkbox in the Save section of the Options - Load/Save - General dialog to enable AutoRecovery. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [429, 367]}} </tool_call>
6		Action: Triple-click the numeric value "10" inside the minutes input box next to "Save AutoRecovery information every:" in the Save section to select the current number. <tool_call> {name: computer_use, arguments: {action: double_click, coordinate: [581, 367]}} </tool_call>
7		Action: Type the character 3 to replace the selected "10" in the minutes input box, setting the AutoRecovery interval to 3 minutes. <tool_call> {name: computer_use, arguments: {action: type, text: 3}} </tool_call>

Continued on next page

Table 13: A successful rollout of the ComputerSD-trained policy on the same in-domain task. Task instruction: "Enable auto-save every 3min for me, so that I don't need to hit ctrl-s that much" (Continued)

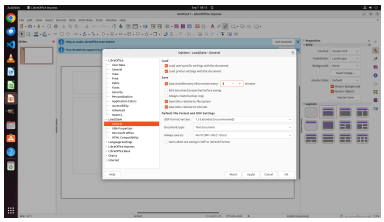
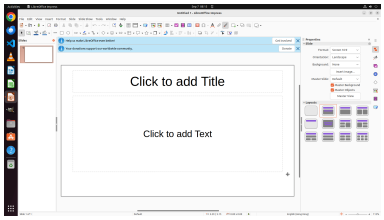
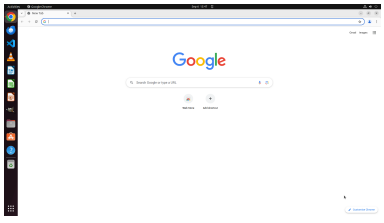
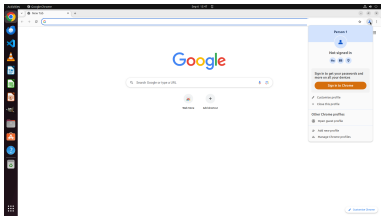
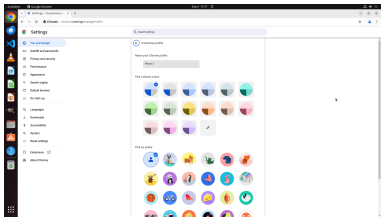
Step	Screenshot	Model response
8		Action: Click the "OK" button at the bottom-right of the "Options - Load/Save - General" dialog to save the changes and close the dialog. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [746, 791]}} </tool_call>
9		Action: Terminate the session indicating success. <tool_call> {name: computer_use, arguments: {action: terminate, status: success}} </tool_call>

Table 14: An unsuccessful rollout of the outcome-only GRPO-trained policy on an OOD task. Task instruction: "Lately I have changed my English name to Thomas. I want to update my username. Could you help me change the username in chrome profiles to Thomas?"

Step	Screenshot	Model response
1		Action: Click the circular user profile avatar button in the top-right of the Chrome toolbar to open the profile menu. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [968, 81]}} </tool_call>
2		Action: Click the "Customise profile" option in the open profile menu to open the profile customization settings. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [877, 438]}} </tool_call>
3		Action: Click the "Name your Chrome profile" text field that currently displays "Person 1" to focus it for editing. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [404, 277]}} </tool_call>

Continued on next page

Table 14: An unsuccessful rollout of the outcome-only GRPO-trained policy on an OOD task. Task instruction: "Lately I have changed my English name to Thomas. I want to update my username. Could you help me change the username in chrome profiles to Thomas?" (Continued)

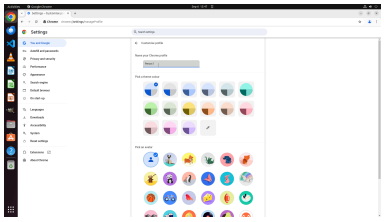
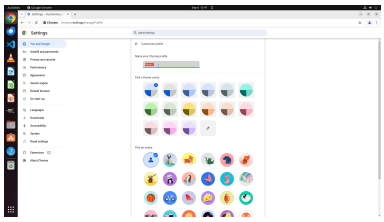
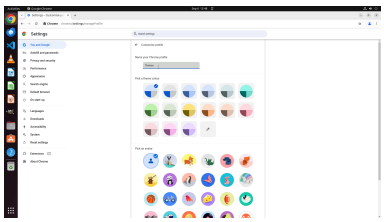
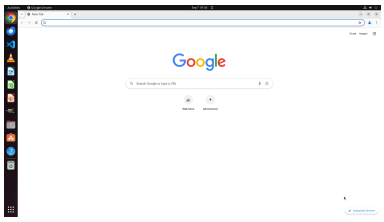
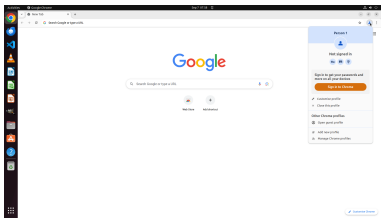
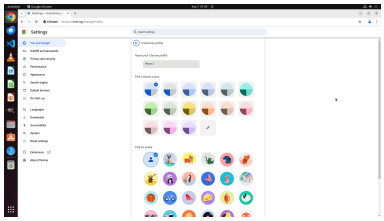
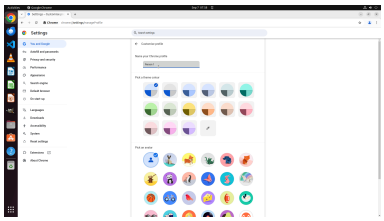
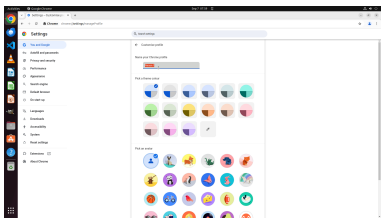
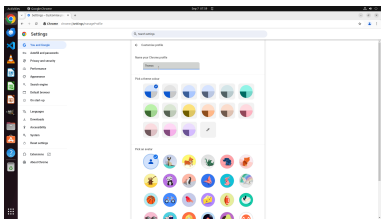
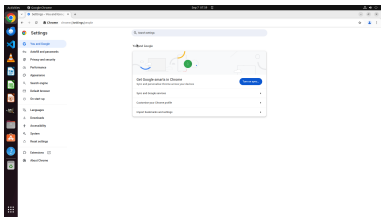
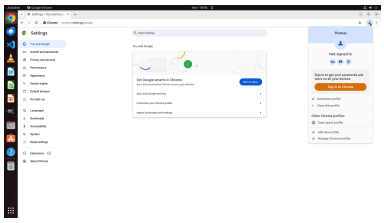
Step	Screenshot	Model response
4		Action: Press Ctrl+A to select all text in the focused "Name your Chrome profile" field. <tool_call> {name: computer_use, arguments: {action: key, keys: [ctrl, a]}} </tool_call>
5		Action: Type the text exactly: Thomas. This will replace the selected "Person 1" in the focused "Name your Chrome profile" field so it reads "Thomas". <tool_call> {name: computer_use, arguments: {action: type, text: Thomas}} </tool_call>
6		Action: Terminate the session and mark the task as successfully completed. <tool_call> {name: computer_use, arguments: {action: terminate, status: success}} </tool_call>

Table 15: A successful rollout of the ComputerSD-trained policy on the same OOD task. Task instruction: "Lately I have changed my English name to Thomas. I want to update my username. Could you help me change the username in chrome profiles to Thomas?"

Step	Screenshot	Model response
1		Action: Click the circular user profile avatar button in the top-right of Chrome's toolbar to open the profile menu. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [968, 81]}} </tool_call>
2		Action: Click the "Customise profile" option in the open Chrome profile menu to open the profile customization settings. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [877, 438]}} </tool_call>

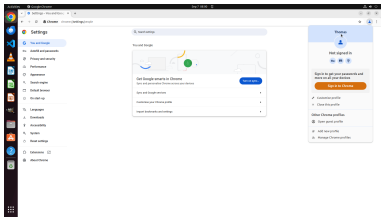
Continued on next page

Table 15: A successful rollout of the ComputerSD-trained policy on the same OOD task. Task instruction: "Lately I have changed my English name to Thomas. I want to update my username. Could you help me change the username in chrome profiles to Thomas?" (Continued)

Step	Screenshot	Model response
3		Action: Click the "Name your Chrome profile" text input that currently displays "Person 1" to focus the field. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [404, 280]}} </tool_call>
4		Action: Press Ctrl+A to select all text in the focused "Name your Chrome profile" input field. <tool_call> {name: computer_use, arguments: {action: key, keys: [ctrl, a]}} </tool_call>
5		Action: Type the text exactly: Thomas. This will replace the selected "Person 1" in the focused "Name your Chrome profile" field so it reads "Thomas." <tool_call> {name: computer_use, arguments: {action: type, text: Thomas}} </tool_call>
6		Action: Click the left-pointing back arrow icon next to the "Customise profile" title at the top-left of the Chrome settings content area to return to the previous settings page. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [350, 181]}} </tool_call>
7		Action: Click the circular user profile avatar button in the top-right of Chrome's toolbar to open the profile menu. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [968, 81]}} </tool_call>
8		Action: Click the "Thomas" profile name header at the top of the open Chrome profile menu in the top-right to open the Thomas profile home page. <tool_call> {name: computer_use, arguments: {action: left_click, coordinate: [891, 137]}} </tool_call>

Continued on next page

Table 15: A successful rollout of the ComputerSD-trained policy on the same OOD task. Task instruction: "Lately I have changed my English name to Thomas. I want to update my username. Could you help me change the username in chrome profiles to Thomas?" (Continued)

Step	Screenshot	Model response
9		Action: Terminate the session and mark the task as successfully completed. <tool_call> {name: computer_use, arguments: {action: terminate, status: success}} </tool_call>