PAGE: Progress-Aware Reinforcement Learning with Hierarchical Advantage for Long-Horizon GUI Agents

Qingliang Zhang[♡†], **Yongqian Sun**[♡], **Yuanhong Cai**[◇†], **Xiaohui Nie**[◇*], **Jiaju Wang**[♡†], **Yu Luo**[♡], **Jiamin Jiang**[♡], **Zhekang Chen**[♣], **Yaozheng Fang**[♣], **Meng Wang**[♣], **Chaobo Xian**[♣], **Peng Li**[♣], **Dan Pei**[♠]

[♡]Nankai University [◇]Computer Network Information Center, Chinese Academy of Sciences
[♣]ByteDance [♠]Tsinghua University

zhangqingliang@mail.nankai.edu.cn sunyongqian@nankai.edu.cn
caiyuanhong26@mails.ucas.ac.cn xhnie@cnic.cn

Abstract

Large vision-language models (LVLMs) perform well on long-horizon GUI tasks, but their high inference costs make small VLMs (SVLMs) attractive for cost-effective local deployment despite a substantial capability gap. Group Relative Policy Optimization (GRPO) is a widely used RL algorithm for improving SVLMs, yet its binary outcome rewards provide no intermediate progress signal across trajectories and its uniform advantages lead to semantically-blind credit assignment within each trajectory. We propose *PAGE*, an RL training method built on automatic progress reward construction and hierarchical advantage computation. The former derives structured evaluation criteria from successful trajectories through iterative extraction and produces stable continuous progress rewards through multi-round aggregation, while the latter combines progress-based inter-rollout advantages with intra-rollout subtask-level advantages obtained through LVLM-driven segmentation and completion verification. Across three long-horizon GUI scenarios, *PAGE* achieves a 79.3% average success rate with an 8B SVLM, a 16.0 pp gain over the GRPO baseline, and competitive performance against the evaluated closed-source LVLMs.

1 Introduction

With the rapid advancement of large vision-language models (VLMs), GUI agents that autonomously operate graphical user interfaces to accomplish repetitive tasks have become increasingly practical (Hong et al., 2024; Qin et al., 2025; Xie et al., 2024; Zhou et al., 2024a). Large VLMs (LVLMs) have demonstrated strong performance on long-horizon GUI tasks (Anthropic, 2026; OpenAI, 2026; Xie et al., 2024), yet their inference cost prohibits large-scale production deployment. This trade-off makes open-source small VLMs (SVLMs) attractive for cost-effective local deployment, although they remain far less capable on long-horizon tasks (Belcak et al., 2025). Reinforcement learning (RL) offers a promising path to close this gap (Bai et al., 2024; Qi et al., 2025), but effective RL training on long-horizon GUI tasks remains challenging.

Among existing RL algorithms, critic-based methods such as PPO (Schulman et al., 2017) require a learned value function whose estimation becomes unreliable on long sequences with high-dimensional visual observations (Yuan et al., 2025). GRPO (Shao et al., 2024) bypasses this difficulty and has become the predominant choice for GUI agent training (Li et al., 2025; Cheng et al., 2026; Feng et al., 2025; He et al., 2026; Wei et al., 2025), where N rollouts per task each receive a binary reward at the trajectory's end, and a single advantage computed via group normalization is broadcast to every action within the rollout. However, this framework faces two limitations when applied to long-horizon GUI tasks.

Absent progress signal. Most GUI agent training relies solely on a binary reward indicating task success or failure at the trajectory's end (Bai et al., 2024; Li et al., 2025; Xie et al., 2024). Unlike mathematical reasoning, where correctness at each step is symbolically verifiable (Lightman et al., 2024; Wang et al., 2024), GUI tasks lack built-in progress metrics. What counts as meaningful progress varies across tasks, and judging whether a milestone such as "the filter has been correctly

[†]Work done during internships at ByteDance.
[*]Corresponding author.

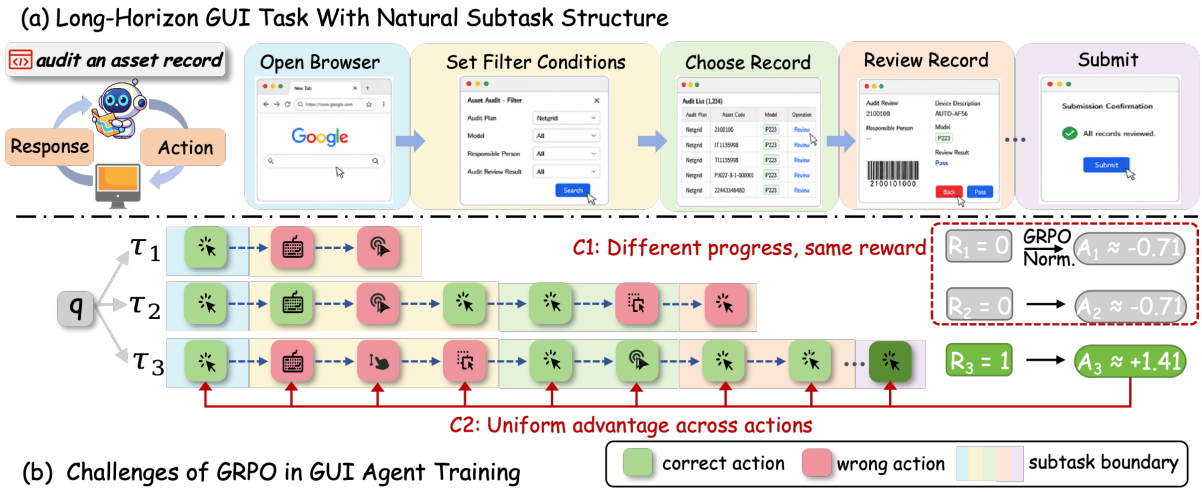


Figure 1: (a) A long-horizon GUI task naturally composed of semantically complete subtasks. (b) Two limitations of GRPO on long-horizon GUI tasks: rollouts with different progress receive the same reward (C1), and all actions within a rollout receive uniform advantage regardless of correctness (C2).

set" has been reached demands structured evaluation criteria that are not readily available. Without such criteria, directly prompting an LVLM for open-ended progress assessment yields unreliable scores (Lin et al., 2026), and binary outcome remains the practical default. As a result, rollouts of vastly different progress become indistinguishable to GRPO, collapsing its advantage signal (Yu et al., 2025; Guo et al., 2026). As shown in Figure 1b, τ_1 (fails at the third step) and τ_2 (nearly completes the task) both receive $R{=}0$ and identical advantages, preventing the model from learning incremental improvements (Wang et al., 2025). In the extreme case where all N rollouts fail, common in early training on difficult tasks, the advantage becomes uniformly zero and the gradient vanishes entirely.

Semantically-blind credit assignment. The correctness of a GUI action depends on its operational context. The same "click (320, 450)" may be correct when setting a filter but wrong when reviewing a record (Figure 1b). Consecutive actions that together complete one logical operation are sequentially dependent, so only their collective outcome is reliably verifiable (Xi et al., 2026). Yet GRPO assigns uniform advantage to all actions within a rollout, giving actions that contribute to success and those that cause failure identical training signal (Tan et al., 2026). GiGPO (Feng et al., 2025) attempts finer credit assignment by matching states across rollouts, but GUI observations are high-resolution screenshots that resist exact comparison (Xie et al., 2024). SEA (Cheng et al., 2026) adopts step-wise GRPO with per-step rewards, yet individual-step evaluation is empirically less reliable than coarser-grained assessment (Lin et al., 2026) and cannot capture whether a sequence of actions has collectively succeeded or failed.

We observe that long-horizon GUI tasks naturally progress through a sequence of intermediate result states (*e.g.*, "filter set," "record reviewed"), each produced by a group of consecutive actions working toward one objective (Figure 1a). Meanwhile, LVLMs have demonstrated the ability to decompose complex tasks and verify intermediate completion states from screenshots (Bai et al., 2024; Zheng et al., 2023). Based on these observations, we propose *PAGE*, an RL training method that combines automatic progress reward construction with hierarchical advantage computation to effectively train GUI agents on long-horizon tasks. To address the absent progress signal, *PAGE* automatically derives structured evaluation criteria through an iterative extraction process with a convergence criterion, then applies multi-round aggregation to produce stable continuous progress reward for each trajectory (Section 3.1). To overcome semantically-blind credit assignment, *PAGE* adopts a hierarchical advantage computation that combines inter-rollout advantages derived from the progress reward with intra-rollout subtask-level advantages obtained through LVLM-driven segmentation and completion verification, enabling fine-grained credit assignment within the trajectory (Section 3.2).

Our contributions are as follows:

- We present *PAGE*, an RL training method for

improving SVLM-based GUI agents on long-horizon tasks.

- *PAGE* introduces **automatic progress reward construction** (Section 3.1) that derives structured evaluation criteria and produces stable continuous reward, and **hierarchical advantage computation** (Section 3.2) that combines inter-rollout advantages with intra-rollout subtask-level advantages for fine-grained credit assignment.
- Experiments on three long-horizon GUI scenarios show that *PAGE* outperforms all training baselines on average and remains competitive with the evaluated closed-source LVLMs at a fraction of the inference cost. We make all code and data publicly available.[1]

2 Preliminaries

GUI agent interaction. Given a task instruction x, a GUI agent policy π_θ interacts with the environment over T steps. At each step t, the agent observes a screenshot o_t, generates an action a_t (click, type, scroll, *etc.*), and the environment transitions to the next state. The complete interaction forms a trajectory $\tau = \{(o_1, a_1), \ldots, (o_T, a_T)\}$, and a binary reward $R(\tau) \in \{0, 1\}$ indicating task success or failure is returned only at the trajectory's end.

RL training with GRPO. For each task, GRPO (Shao et al., 2024) samples N rollouts from the current policy and computes the advantage via within-group normalization:

$$\hat{A}(\tau_i) = \frac{R(\tau_i) - \bar{R}}{\sigma_R + \epsilon}, \quad \bar{R} = \frac{1}{N}\sum_{j=1}^{N} R(\tau_j) \quad (1)$$

where σ_R is the within-group standard deviation. This advantage is broadcast to every step within τ_i. The policy is updated by maximizing a clipped surrogate objective with KL regularization.

3 Method

To address the two limitations of GRPO on long-horizon GUI tasks identified in Section 1, we propose *PAGE* (Figure 2), which introduces two components: **progress reward construction** (Section 3.1) that addresses the absent progress signal by automatically deriving structured evaluation criteria and producing continuous reward $R(\tau)$ that differentiates rollouts beyond binary success/failure, and **hierarchical advantage computation** (Section 3.2) that addresses semantically-blind credit assignment by computing advantages at two levels, with inter-rollout advantages differentiating rollouts by progress and intra-rollout subtask-level advantages further differentiating steps within the same rollout based on completion verification. In each RL iteration (Section 3.3), the policy π_θ produces N rollouts that are evaluated by an LVLM to obtain progress rewards $R(\tau)$ and subtask verification results $\{v_m\}$, which are then combined into hierarchical advantages for policy update.

3.1 Progress Reward Construction

3.1.1 Evaluation Checklist Extraction

To quantify trajectory progress, we automatically extract structured evaluation criteria, which we term a **checklist**, from a small set of successful trajectories. The checklist decomposes a GUI task into a sequence of ordered **checkpoints**, each describing an observable intermediate result state in the task progression.

Data preparation. For each task, we collect a small number of trajectories by rolling out an LVLM in the environment. These trajectories are split by outcome into two disjoint sets: an extraction set containing only successful trajectories for checklist generation, and a validation set containing both successful and failed trajectories for convergence determination. Including both outcome types in the validation set ensures the checklist can identify successes while correctly distinguishing failures.

Iterative extraction and convergence. In each round, the LVLM reads a random subset of successful trajectories from the extraction set and produces a candidate checklist following predefined guidelines that emphasize observability, independence, non-overlap, and appropriate granularity (Appendix D.2). Across rounds, candidate checklists are fused by comparing coverage and description quality, retaining checkpoints that best satisfy the guidelines. To assess convergence, the LVLM judges each checkpoint's satisfaction for every trajectory in the validation set, producing a Boolean matrix (rows = trajectories, columns = checkpoints). The element-wise agreement rate between the current and previous round's matrices is computed, and iteration stops when this rate

[1] https://github.com/ZQlQZ/PAGE

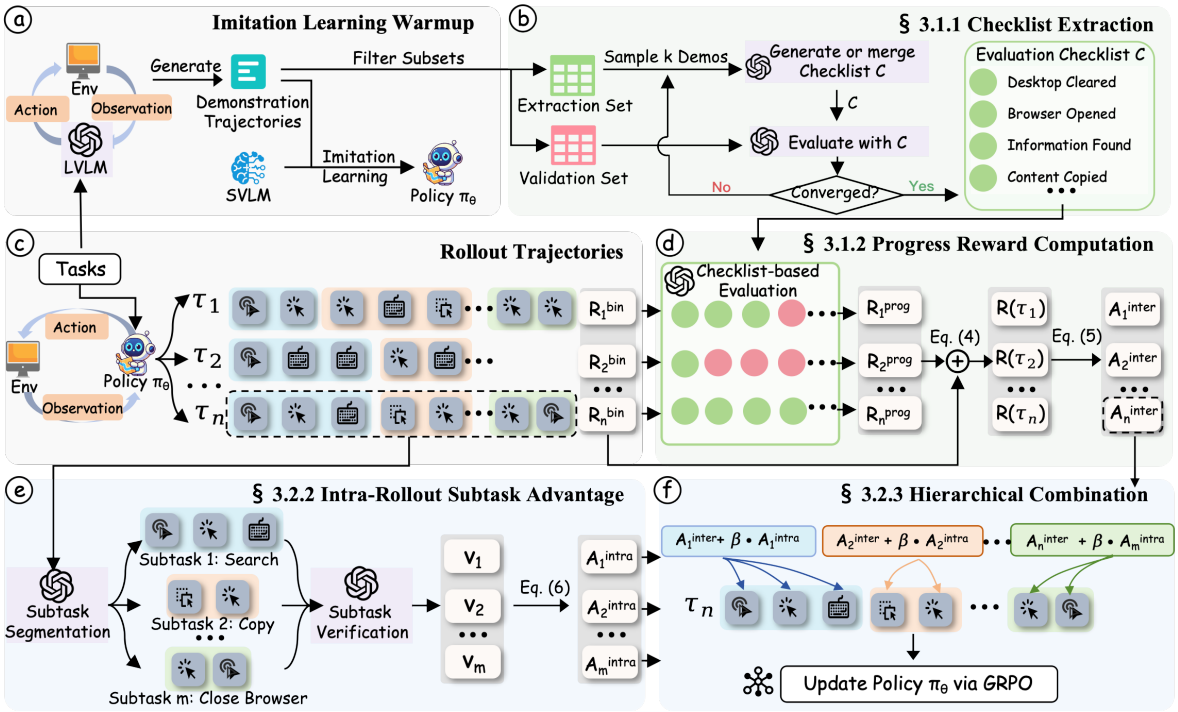


Figure 2: Overview of *PAGE*. Training begins with imitation learning warmup (a) and one-time checklist extraction (b). During RL, rollouts (c) are evaluated via checklist-based progress reward (d) and subtask-level completion verification (e), producing hierarchical advantages that are combined (f) for policy update.

reaches the threshold δ for two consecutive rounds, indicating stable scoring behavior. This iterative procedure requires far less data than training a reward model from scratch. The complete algorithm is given in Appendix A.1. The resulting checklist contains C checkpoints, is white-box, and requires no manual reward annotation.

3.1.2 Progress Reward Computation

Checklist-based evaluation. Given a trajectory τ, the LVLM follows predefined evaluation rules to sequentially examine the screenshots and action records, judging whether each of the C checkpoints has been completed. The resulting binary completion vector has length C, with raw progress score $s = n/C$ for n checkpoints marked as satisfied.

Since LVLM judgments exhibit non-negligible variance across calls, we adopt a multi-round aggregation strategy to produce a stable progress reward $R_{\text{prog}}(\tau)$ from multiple raw scores. Specifically, L independent evaluation rounds are executed, yielding L raw scores $\{s^{(1)}, \ldots, s^{(L)}\}$, which are aggregated into $R_{\text{prog}}(\tau)$ via a method inspired by James-Stein shrinkage estimation (James and Stein, 1961):

$$\eta = \beta_s + (1 - \beta_s) \cdot \frac{c}{L} \tag{2}$$

$$R_{\text{prog}}(\tau) = \eta \cdot s_{\text{base}} + (1 - \eta) \cdot \bar{s} \tag{3}$$

where c is the mode count among the L scores, s_{base} is the mode (if $c > L/2$) or the median, $\bar{s}$ is the mean, and β_s is a minimum trust prior. When multiple rounds agree, $\eta \to 1$ and the consensus is trusted, whereas disagreement shifts the estimate toward the mean and suppresses uncertain signals. Empirically, the checklist-based progress evaluation achieves near-human agreement (Appendix C), confirming both the precision of the structured criteria and the stability of the aggregation strategy.

Composite reward. Relying solely on progress reward risks local optima where the agent maximizes intermediate progress without completing the task. For example, a rollout satisfying 4/5 checkpoints yet ultimately failing would receive a reward close to that of successful rollouts, eroding the learning signal for task completion. To maintain a clear separation between success and failure while retaining fine-grained progress information, we combine the progress reward with the

binary outcome returned by the environment.

$$R(\tau) = \alpha \cdot R_{\text{prog}}(\tau) + (1-\alpha) \cdot R_{\text{bin}}(\tau) \quad (4)$$

where $R_{\text{bin}} \in \{0, 1\}$ is the binary signal indicating task success or failure returned by the environment at the trajectory's end, and α controls the combination weight. This ensures that successful rollouts always receive higher reward than failed ones regardless of progress, while failed rollouts are still differentiated by the degree of completion.

3.2 Hierarchical Advantage Computation

3.2.1 Inter-Rollout Advantage

For the N rollouts of the same task, we compute rollout-level advantages using the composite reward $R(\tau_j)$:

$$\hat{A}_j^{\text{inter}} = \frac{R(\tau_j) - \bar{R}}{\sigma_R + \epsilon} \quad (5)$$

and broadcast $\hat{A}_j^{\text{inter}}$ to all steps within τ_j. Unlike standard GRPO under binary reward, which can only separate successful and failed groups, Eq. (5) allows failed rollouts with different completion degrees to receive differentiated advantages, enabling the model to learn incremental improvements from partial completions.

3.2.2 Intra-Rollout Subtask Advantage

Subtask segmentation. As discussed in Section 1, the correctness of a GUI action depends on its surrounding context, and consecutive actions that together accomplish one logical operation are sequentially dependent. We therefore adopt **subtasks** as the granularity for credit assignment, where each subtask is a semantically complete segment of the trajectory (*e.g.*, "set filter conditions," "review the first record") whose completion can be independently verified. An LVLM segments each trajectory into a sequence of contiguous, non-overlapping subtasks $\{S_1, \ldots, S_M\}$, following predefined guidelines that ensure independent verifiability, operational completeness, and full trajectory coverage (Appendix D.5). The LVLM simultaneously generates a **subtask instruction** for each segment, consisting of the GUI state when the subtask begins (context), the operational objective (goal), and the observable outcome that signals completion (success state). Unlike the checklist in Section 3.1.1, which evaluates global task progress across the entire trajectory, subtask instructions serve as local evaluation criteria for verifying whether individual segments have been completed.

Intra-rollout advantage computation. For each subtask S_m, the LVLM verifies whether the success state described in its instruction has been achieved by examining the screenshots and action records within that segment, producing a binary verification result $v_m \in \{0, 1\}$. The intra-rollout subtask advantage is then computed by subtracting the rollout mean:

$$\hat{A}_m^{\text{intra}} = v_m - \bar{v}_j, \quad \bar{v}_j = \frac{1}{M}\sum_{m=1}^{M} v_m \quad (6)$$

where $\bar{v}_j$ is the mean verification result within rollout τ_j. The subtask advantage $\hat{A}_m^{\text{intra}}$ is broadcast to every step inside S_m, assigning positive advantage to completed subtasks and negative advantage to failed ones.

3.2.3 Hierarchical Combination

The final advantage for each step combines the two levels:

$$A_i = \hat{A}_j^{\text{inter}} + \beta \cdot \hat{A}_m^{\text{intra}} \quad (7)$$

where β controls the strength of intra-rollout credit assignment. The inter-rollout term $\hat{A}_j^{\text{inter}}$ determines whether a rollout as a whole should be reinforced or suppressed relative to other rollouts, while the intra-rollout term $\hat{A}_m^{\text{intra}}$ further differentiates steps within the same rollout based on subtask completion.

3.3 Training Pipeline

Following standard practice (Bai et al., 2024; Guo et al., 2025), training begins with an imitation learning warmup: the SVLM is trained via SFT on a small set of demonstration trajectories collected from an LVLM, equipping the policy with basic GUI interaction capabilities for meaningful RL rollouts. The checklist is then extracted once from a small subset of successful warmup trajectories, requiring far fewer trajectories than SFT itself. Each RL iteration then: (1) produces N rollouts from π_θ; (2) evaluates them via an LVLM to obtain $R(\tau)$ and $\{v_m\}$; (3) computes advantages via Eq. (5)–Eq. (7) and updates the policy by maximizing:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \min\big(\rho_i A_i,\ \text{clip}(\rho_i, 1\pm\epsilon) A_i\big)\right] - \beta_{\text{kl}}\, \mathbb{D}_{\text{KL}}\big(\pi_\theta \| \pi_{\text{ref}}\big) \quad (8)$$

where $\mathcal{B}$ is a minibatch of action-level training samples, p_i is the task instruction and interaction context provided to the policy for sample i, $\pi_{\theta_{\text{old}}}$ is the policy before the current update, $\rho_i = \pi_\theta(a_i|p_i)/\pi_{\theta_{\text{old}}}(a_i|p_i)$ is the corresponding importance sampling ratio, π_{ref} is the SFT checkpoint, and β_{kl} controls the KL regularization strength. LVLM evaluation is invoked only during training, and inference requires no additional model.

4 Experiments

We evaluate *PAGE* on three long-horizon GUI scenarios. Specifically, we aim to demonstrate: **(1)** the effectiveness of *PAGE* across diverse GUI scenarios; **(2)** the independent and complementary contributions of progress reward construction and hierarchical advantage computation; **(3)** the robustness to hyperparameter choices; **(4)** the deployment cost advantage over API-based LVLMs.

4.1 Experiment Setup

Datasets. We construct three long-horizon GUI scenarios ($\geq$20 steps, well above the 15-step threshold established by Kang et al. (2026)) from real enterprise workflows (Appendix B). **Asset Audit** ($\sim$20 steps): operate a Web asset management system to complete inventory audits involving form interaction and visual barcode comparison. **Browser Search** ($\sim$31 steps): retrieve information via browser and save to a local document, coordinating across multiple applications. **File Organization** ($\sim$28 steps): classify and reorganize files based on specified rules by first observing directory contents. The dataset contains 264 training tasks and 150 evaluation tasks.

Metrics. (1) **Success Rate (SR)**: the proportion of tasks fully completed. (2) **Weighted Efficiency Score (WES+)** (Abhyankar et al., 2026): SR $\times$ $(t_{\text{human}}/t_{\text{agent}})$, where t denotes the number of steps taken to complete the task, jointly reflecting success rate and step efficiency. Task success is determined by an LVLM-as-judge (Bai et al., 2024) against predefined evaluation rules, with Doubao-Seed-2.0-Pro (ByteDance Seed Team, 2026) serving as the judge for all methods, and the resulting evaluation protocol achieves 93.3% accuracy against human annotations (Appendix B.3).

Baselines. We compare against two categories. (1) *Closed-source LVLMs*: Claude Sonnet 4.6 (Anthropic, 2026), Doubao-Seed-2.0-Pro, and GPT-5.4 (OpenAI, 2026), evaluated in zero-shot inference mode. (2) *Training methods*: **Qwen3-VL-8B-Instruct** (Bai et al., 2025), the base model without any fine-tuning, representing the inherent SVLM capability; **SFT**, after imitation learning warmup without further RL training, serving as the RL lower bound; **GRPO** (Shao et al., 2024), **GiGPO** (Feng et al., 2025), and **SEA** (Cheng et al., 2026), all trained from the same SFT checkpoint with identical rollout and optimization configurations. GiGPO groups steps by anchor state matching for two-level credit assignment (we use step index as proxy since GUI screenshots preclude exact state matching). SEA adopts step-wise GRPO with per-step binary rewards, sharing the same LVLM scoring infrastructure as ours for fair comparison.

Implementation. The imitation learning warmup uses approximately 400 trajectories per scenario (1.2K total), generated by Doubao-Seed-2.0-Pro, which also serves as the LVLM for checklist extraction, progress evaluation, subtask segmentation, and subtask verification during RL training. All RL methods, including ours and the baselines, are trained for 2 epochs on 8 GPUs using the same configurations: rollout group size N=4, learning rate 1×10^{-6}, batch size 8, and a maximum step limit of 45 per rollout. Each checklist-extraction round samples k=5 successful trajectories for extraction and three trajectories for validation, and convergence is typically reached within four to five rounds. We set the checklist convergence threshold to δ=0.9 and use L=5 progress-evaluation rounds at temperature 0.3 with β_s=0.5. For *PAGE*, we set the reward combination weight α=0.5 and the intra-rollout strength β=1.0 without further tuning. In the main experiments, we train the models jointly on all three scenarios, while the ablation studies focus on Asset Audit. Each model is evaluated three times on the full test set, and we report the mean to account for environment stochasticity. The training infrastructure builds on Agent Lightning (Luo et al., 2025) and verl (Sheng et al., 2025).

4.2 Main Results

***PAGE* outperforms all training baselines.** As shown in Table 1, *PAGE* achieves the highest Avg SR (79.3%) and Avg WES+ (75.3%) among all training methods, surpassing GiGPO by +5.0 pp in SR and GRPO by +16.0 pp in SR, with consistent gains across all three scenarios. The largest

Table 1: Main results on three long-horizon GUI scenarios. SR (%) is the primary metric; WES+ (%) jointly reflects success rate and step efficiency. "+" denotes RL training from the SFT checkpoint. Best results among training methods in **bold**, second-best underlined. Results are averaged over 3 runs, with standard deviations shown as subscripts.

Method	Asset Audit		Browser Search		File Org.		Average	
	SR	WES+	SR	WES+	SR	WES+	SR	WES+
Closed-Source LVLMs								
Claude Sonnet 4.6	$81.3_{\pm 5.0}$	57.4	$82.0_{\pm 1.6}$	67.2	$96.7_{\pm 1.9}$	84.2	$86.7_{\pm 1.4}$	69.6
Doubao-Seed-2.0-Pro	$92.7_{\pm 2.5}$	83.1	$54.0_{\pm 5.9}$	48.6	$91.3_{\pm 0.9}$	92.9	$79.3_{\pm 2.4}$	74.9
GPT-5.4	$55.3_{\pm 9.8}$	49.6	$71.3_{\pm 1.9}$	64.3	$82.0_{\pm 7.1}$	81.2	$69.6_{\pm 1.3}$	65.0
Qwen3-VL-8B-Instruct								
Base	$38.7_{\pm 2.5}$	29.5	$0.0_{\pm 0.0}$	0.0	$4.7_{\pm 2.5}$	3.5	$14.4_{\pm 1.6}$	11.0
SFT	$60.7_{\pm 2.5}$	55.7	$33.3_{\pm 5.0}$	27.0	$56.7_{\pm 1.9}$	45.5	$50.2_{\pm 0.3}$	42.7
+ GRPO	$69.3_{\pm 3.4}$	67.1	$48.7_{\pm 3.4}$	41.7	$72.0_{\pm 3.3}$	66.4	$63.3_{\pm 0.9}$	58.4
+ GiGPO	$\underline{79.3}_{\pm 0.9}$	$\underline{75.2}$	$\mathbf{65.0}_{\pm 2.5}$	$\underline{55.9}$	$\underline{78.7}_{\pm 1.9}$	74.1	$\underline{74.3}_{\pm 1.7}$	$\underline{68.4}$
+ SEA	$76.0_{\pm 3.3}$	66.4	$31.3_{\pm 5.2}$	25.6	$\underline{78.7}_{\pm 6.2}$	$\underline{74.2}$	$62.0_{\pm 3.3}$	55.4
+ *PAGE* (Ours)	$\mathbf{91.3}_{\pm 3.4}$	**86.1**	$\underline{64.7}_{\pm 0.9}$	**61.9**	$\mathbf{82.0}_{\pm 3.3}$	**78.0**	$\mathbf{79.3}_{\pm 1.1}$	**75.3**
Δ *vs GRPO*	*+22.0*	*+19.0*	*+16.0*	*+20.2*	*+10.0*	*+11.6*	*+16.0*	*+16.9*

improvement appears on Asset Audit (+22.0 pp over GRPO), where each subtask produces a visually distinct result state (e.g., filter applied, record reviewed) that is easy to verify from screenshots, maximizing the accuracy of both checklist evaluation and subtask verification. File Organization shows the smallest gain (+10.0 pp), as its repetitive file-moving operations yield less distinguishable intermediate states for progress differentiation. SEA performs notably poorly on Browser Search (31.3%), where actions span multiple applications (browser $\rightarrow$ LibreOffice) and their correctness cannot be judged in isolation, making per-step evaluation unreliable and subtask-level granularity necessary.

***PAGE* approaches closed-source LVLMs.** Our 8B model matches Doubao-Seed-2.0-Pro (79.3%) and surpasses GPT-5.4 (+9.7 pp), while narrowing the gap with Claude Sonnet 4.6 (86.7%) to 7.4 pp. On Asset Audit, the trained SVLM (91.3%) even surpasses Claude Sonnet 4.6 (81.3%), suggesting that task-specific RL adaptation can provide an advantage over zero-shot general-purpose models within the same workflow distribution. *PAGE* also achieves the highest WES+ (75.3%) across all methods including closed-source LVLMs, indicating that hierarchical advantage computation encourages more step-efficient execution. By suppressing actions in failed subtasks, the model learns to avoid redundant exploration and take more direct paths to task completion.

Table 2: Ablation study on Asset Audit (SR and WES+ in %). C1 = progress reward (Section 3.1). C2 = intra-rollout subtask advantage (Section 3.2.2).

Configuration	C1	C2	SR	WES+
PAGE (Full)	✓	✓	$\mathbf{91.3}_{\pm 3.4}$	**86.1**
w/o C2	✓		$79.3_{\pm 0.9}$	71.4
w/o C1		✓	$77.3_{\pm 1.9}$	68.7
w/o C1 & C2 (GRPO)			$69.3_{\pm 3.4}$	67.1

4.3 Ablation Study

To validate the independent contributions of each component, we conduct an ablation study on Asset Audit (Table 2). Eliminating either component substantially degrades performance, as SR drops to 77.3% (-14.0 pp) without progress reward (w/o C1) and to 79.3% (-12.0 pp) without intra-rollout subtask advantage (w/o C2), while disabling both recovers standard GRPO (69.3%). Progress reward alone provides +10.0 pp over GRPO because it converts the all-zero-advantage problem into a meaningful gradient that rewards partial completions, enabling the model to learn incrementally even from failed rollouts. Subtask advantage adds a further +12.0 pp by directing positive reinforcement specifically to completed subtasks while suppressing actions in failed segments, preventing the model from reinforcing error patterns that coexist with successful steps. The +4.0 pp synergy arises because progress reward first creates meaningful inter-rollout differentiation, providing an informative baseline from which subtask-level adjustments can operate effectively.

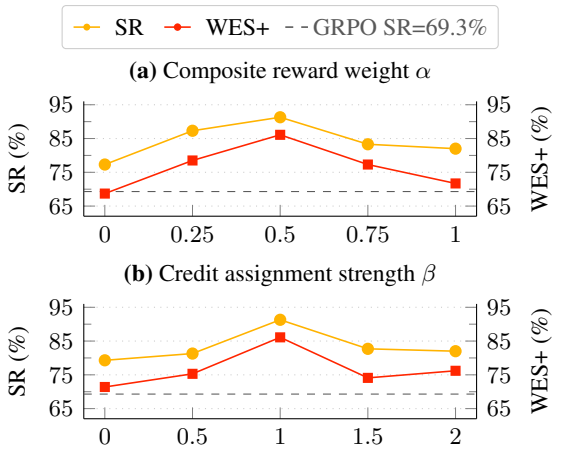


Figure 3: Hyperparameter sensitivity on Asset Audit. (a) α controls composite reward weight (Eq. (4)). (b) β controls intra-rollout subtask advantage strength (Eq. (7)). Both metrics peak at α=0.5, β=1.0.

Table 3: Average token consumption and cost per case for API-based GUI agents. Our locally deployed 8B model requires no API fees after a one-time training investment of $823.

Model	Prompt	Compl.	Cost/Case
GPT-5.4	754.0K	0.8K	$1.898
Claude Sonnet 4.6	388.6K	6.4K	$1.261
Doubao-Seed-2.0-Pro	258.5K	5.1K	$0.134

4.4 Sensitivity Analysis

We analyze the sensitivity of α and β on Asset Audit (Figure 3). Performance remains strong across $\alpha \in [0.25, 0.75]$ (all $\geq 83.3\%$), suggesting the method requires no per-scenario tuning, while α=0 drops to 77.3% (-14.0 pp), confirming that progress reward is essential even when intra-rollout credit is available. For β, performance peaks at 1.0 and drops sharply beyond this as the intra-rollout term magnitude exceeds the inter-rollout signal, amplifying subtask verification errors.

4.5 Cost Analysis

API-based LVLMs incur $0.13–$1.90 per case due to long interaction traces (Table 3), with costs recurring for every execution. Our one-time training cost of **$823** (384 GPU-hours on 8 GPUs plus LVLM API calls) allows the model to run locally with negligible inference cost thereafter. Compared to Claude Sonnet 4.6 ($1.26/case), the investment is recovered after 653 cases, saving over $11,700 at 10,000 cases. In typical enterprise deployments handling tens to hundreds of tasks daily, break-even is reached within days.

5 Related Work

RL for GUI agents. DigiRL (Bai et al., 2024) introduces an offline-to-online RL pipeline with a VLM-based binary evaluator for mobile GUI control. ZeroGUI (Yang et al., 2025) automates task generation and reward estimation via multi-round VLM voting. WebAgent-R1 (Wei et al., 2025) extends GRPO to multi-turn web interactions with dynamic context compression. DART (Li et al., 2025) improves training efficiency via decoupled asynchronous architecture and adaptive data curation. SEA (Cheng et al., 2026) and SEAgent (Sun et al., 2026) introduce step-level signals via rule-based execution verification and a trained world state model, respectively, but assess each step independently without capturing overall task progress. All these methods lack trajectory-level progress quantification, leaving rollouts of different completion degrees indistinguishable to the advantage computation.

GRPO and its variants. GRPO (Shao et al., 2024) assigns uniform advantage to all actions within a rollout. ArCHer (Zhou et al., 2024b) learns inter-turn value functions, but value estimation becomes unreliable over long visual sequences. GiGPO (Feng et al., 2025) groups steps by matching recurring anchor states across rollouts, but degenerates into singleton groups when high-resolution screenshots rarely recur. HGPO (He et al., 2026) organizes steps into hierarchical groups based on historical context consistency, but relies on comparing context sequences that are difficult to align across pixel-based GUI trajectories. Other approaches introduce plan-execute decomposition (Peng et al., 2026), hindsight Q-value refinement (Tan et al., 2026), or proximity-based weighting (Fang et al., 2026), each requiring additional models or assumptions.

6 Conclusion

We presented *PAGE*, an RL training method that addresses two limitations of GRPO on long-horizon GUI tasks. Automatic progress reward construction derives structured evaluation criteria and produces continuous reward that differentiates rollouts by completion degree. Hierarchical advantage computation combines inter-rollout and intra-rollout subtask-level advantages for fine-grained credit assignment. Experiments on three scenarios demonstrate that *PAGE* outperforms existing methods and

enables an 8B model to match commercial LVLMs at a fraction of the inference cost, offering a practical path toward deploying capable GUI agents at scale.

Limitations

Our method relies on an LVLM for training data generation, training-time supervision, and final task-success evaluation. While LVLM-based progress scoring and task-success evaluation show strong agreement with human annotations, achieving 87.3% exact agreement and 93.3% accuracy, respectively, the method still assumes that the LVLM has sufficient capability to interact with the target GUI environment and provide reliable judgments. Self-evaluation mechanisms and environment-grounded success checks remain possible directions for reducing this reliance.

Our evaluation covers three long-horizon GUI scenarios derived from enterprise workflows. Although these scenarios involve different applications and interaction patterns, this limited scope does not fully represent the diversity of real-world GUI environments. Broader validation on more diverse public GUI benchmarks, including tasks with noisier visual states, is needed to assess the generalizability of *PAGE*.

Ethical Considerations

All models and APIs used in this paper are publicly available or commercially accessible, and our utilization strictly complies with their respective licenses and terms of use. The GUI task scenarios are constructed from enterprise workflows and do not involve personal user data.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (62272249, 62302244), the Fundamental Research Funds for the Central Universities (XXX-63253249), the Tianjin Key Research and Development Program (Grant No. 25YFYFFG00690), and the CNIC Young Scientists Research Fund under Grant No. E5553701.

References

Reyna Abhyankar, Qi Qi, and Yiying Zhang. 2026. Osworld-human: Benchmarking the efficiency of computer-use agents. In *Proceedings of Machine Learning and Systems*, volume 8, pages 482–494. MLSys.

Anthropic. Introducing Claude Sonnet 4.6 [online]. 2026.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, pages 12461–12495. Curran Associates, Inc.

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, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *Preprint*, arXiv:2506.02153.

ByteDance Seed Team. Seed 2.0 official launch [online]. 2026.

Yuhao Cheng, Liang Tang, Shuxian Li, Yukang Huo, Tiaonan Duan, Kaer Huang, Yanzhe Jing, and Yiqiang Yan. 2026. Evolving in tasks: Empowering the multi-modality large language model as the computer use agent. *Preprint*, arXiv:2508.04037.

Yangyi Fang, Jiaye Lin, Xiaoliang Fu, Cong Qin, Haolin Shi, and Chang Liu. 2026. Proximity-based multi-turn optimization: Practical credit assignment for LLM agent training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 285–307, San Diego, California, USA. Association for Computational Linguistics.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 46375–46408. Curran Associates, Inc.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Yongxin Guo, Wenbo Deng, Zhenglin Cheng, and Xiaoying Tang. 2026. G^2RPO-a: Guided group relative policy optimization with adaptive guidance. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4525–4539, San Diego, California, United States. Association for Computational Linguistics.

Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. 2026. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. In *International Conference on Learning Representations*, volume 2026, pages 27572–27593.

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

William James and Charles Stein. 1961. Estimation with quadratic loss. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 361–379. University of California Press.

Bin Kang, Shaoguo Wen, Yifei Bi, Shunlong Wu, Xinbin Yuan, Rui Shao, Junle Wang, and Zhuotao Tian. 2026. LonghorizonUI: A unified framework for robust long-horizon task automation of GUI agent. In *The Fourteenth International Conference on Learning Representations*.

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. 2025. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *Preprint*, arXiv:2509.23866.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Haojia Lin, Xiaoyu Tan, Yulei Qin, Zihan Xu, Yuchen Shi, Zongyi Li, Gang Li, Shaofei Cai, Siqi Cai, Yuzheng Cai, Chaoyou Fu, Ke Li, and Xing Sun. 2026. CUARewardbench: A benchmark for evaluating reward models for computer-using agents. In *Forty-third International Conference on Machine Learning*.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. 2025. Agent lightning: Train any ai agents with reinforcement learning. *Preprint*, arXiv:2508.03680.

OpenAI. Introducing GPT-5.4 [online]. 2026.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. 2026. Hiper: Hierarchical reinforcement learning with explicit credit assignment for large language model agents. *Preprint*, arXiv:2602.16165.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 79791–79821.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *Preprint*, arXiv:2501.12326.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297, New York, NY, USA. Association for Computing Machinery.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. 2026. SEAgent: Self-evolving computer use agent with autonomous learning from experience. In *Forty-third International Conference on Machine Learning*.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. 2026. Hindsight credit assignment for long-horizon llm agents. *Preprint*, arXiv:2603.08754.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

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

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao

Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025. WebAgent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7909–7928, Suzhou, China. Association for Computational Linguistics.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. 2026. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, WWW '26, page 4184–4195, New York, NY, USA. Association for Computing Machinery.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc.

Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. 2025. Zerogui: Automating online gui learning at zero human cost. *Preprint*, arXiv:2505.23762.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, and 17 others. 2025. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. 2025. What's behind ppo's collapse in long-cot? value optimization holds the secret. *Preprint*, arXiv:2503.01491.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024a. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024b. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*.

A Algorithm Details

This appendix provides the complete pseudocode for the iterative checklist extraction algorithm of *PAGE*.

A.1 Iterative Checklist Extraction

Algorithm 1 describes the iterative procedure for extracting a structured evaluation checklist from successful trajectories (Section 3.1.1 in the main text). In each round, an LVLM generates candidate checkpoints from sampled trajectories and merges them with the existing checklist, optimizing for coverage, description quality, granularity, and inter-checkpoint independence. Convergence is determined by the stability of the scoring behavior on a held-out validation set.

Algorithm 1 Iterative Checklist Extraction

Require: Extraction set $\mathcal{D}_{\text{good}}$ (successful trajectories), validation set $\mathcal{D}_{\text{val}}$ (both successful and failed), per-round sample size k, threshold $\delta = 0.9$

$\mathcal{C}_0 \leftarrow \emptyset$
for round $t = 1, 2, \ldots$ **do**
  Sample k trajectories from $\mathcal{D}_{\text{good}}$
  LVLM generates candidate checkpoints $\mathcal{C}_{\text{new}}$
  $\mathcal{C}_t \leftarrow$ LVLM-Merge$(\mathcal{C}_{t-1}, \mathcal{C}_{\text{new}})$
  {Merge by coverage, quality, granularity, independence}
  $E_t \in \{0,1\}^{|\mathcal{D}_{\text{val}}| \times |\mathcal{C}_t|} \leftarrow$ score $\mathcal{D}_{\text{val}}$ against $\mathcal{C}_t$ per checkpoint
  if $|\mathcal{C}_t| \neq |\mathcal{C}_{t-1}|$ **then**
    continue {Dimension mismatch, skip comparison}
  end if
  stability $\leftarrow$ agree$(E_t, E_{t-1})/|E_t|$
  {Element-wise agreement rate of Boolean matrices}
  if stability $\geq \delta$ for 2 consecutive rounds **then**
    break
  end if
end for

Ensure: $\mathcal{C}_t$ = {scenario description, list of checkpoint definitions}

B Task Scenarios and Evaluation Protocol

B.1 Task Design

Existing GUI benchmarks primarily target short-horizon tasks (*e.g.*, the median task in OS-World (Xie et al., 2024) requires only ∼6 steps, well below the 15-step long-horizon threshold (Kang et al., 2026)), which are insufficient for

studying the credit assignment and progress quantification challenges that arise in long-horizon settings. We therefore construct three desktop scenarios from real enterprise workflows, each requiring 20–40 sequential operations. Within each scenario, task instances are generated by varying scenario variables such as target entities and operation rules. Changing these variables alters the GUI pages the agent encounters, requiring it to locate UI elements at different positions, interpret varying dropdown options and search results, and make content-dependent decisions (*e.g.*, approve vs. reject based on visual barcode comparison), thereby diversifying the visual states and decision complexity across instances. The dataset contains 264 training tasks and 150 evaluation tasks, with training and evaluation sets using mutually exclusive variable values to ensure no evaluation instance appears during training.

B.2 Task Scenarios

Table 4 summarizes the three scenarios.

Asset Audit. The agent operates a Web-based asset management system to complete inventory audits under specified conditions, where each task involves different filter criteria and record contents. This scenario involves form-based Web interaction and visual comparison reasoning. Figure 4 illustrates representative stages from a 25-step execution trace.

Browser Search. The agent retrieves the answer to a specified question via a browser and saves it to a designated local document, coordinating across browser, file manager, and LibreOffice with varying queries and target paths. This scenario involves cross-application coordination and information transfer between GUI environments. Figure 5 illustrates representative stages from a 36-step execution trace.

File Organization. The agent classifies and reorganizes files according to sorting rules specified in the instruction, where directory contents and rules vary across instances. Since filenames are not listed in the instruction, the agent observes the current directory state and determines the operation sequence accordingly. This scenario involves observation-driven conditional reasoning and batch file manipulation. Figure 6 illustrates representative stages from a 31-step execution trace.

B.3 Evaluation Protocol

For each scenario, we define a set of evaluation rules that specify the conditions for task success (Section B.3.1). After a rollout completes, an LVLM-as-judge evaluates the full trajectory against these rules and produces a binary outcome: success if all rules are satisfied, failure otherwise. To validate the reliability of this approach, we compare LVLM-as-judge determinations against independent human annotations on 150 trajectories (50 per scenario). Table 5 reports the agreement: accuracy reaches 93.3% with F1 = 0.946, confirming that LVLM-based evaluation is a reliable proxy for human judgment.

B.3.1 Evaluation Rules

Below we present the evaluation rules for the three scenarios. The LVLM receives the task instruction, the agent's execution trajectory, and these rules, then determines whether all rules are satisfied.

Asset Audit Evaluation Rules

Evidence priority: (1) Screenshots (ground truth), (2) Action logs (intent only), (3) Agent thought/summary (reference only, cannot be sole basis).

Rules:

1. **System access:** Agent has successfully entered the asset audit system page.
2. **Filter conditions:** Agent has set the correct filters: inventory plan = the plan name specified in the instruction, audit result = "unaudited."
3. **Audit quantity:** Agent has successfully completed audit operations on 2 records.
4. **Audit logic:** For each record: if the image barcode matches the system barcode → approve; if mismatched or unreadable → reject.

Browser Search Evaluation Rules

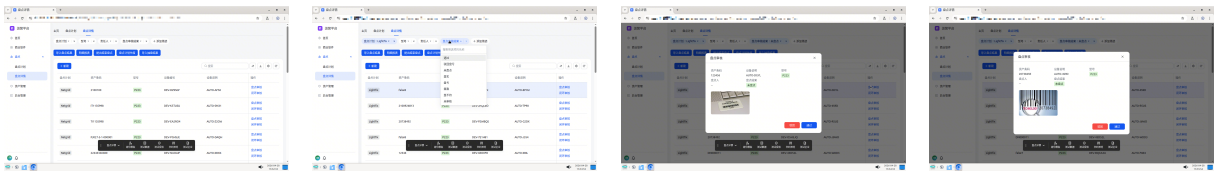
Evidence priority: Same as above.

Rules:

1. **Folder exists:** The folder specified in the instruction exists under `/home/user/exp/`.
2. **File exists:** The specified file exists inside the target folder.
3. **Document content:** The document contains the correct answer to the search query specified in the instruction. Minor formatting differences are acceptable.
4. **Document saved:** The document has been saved (not in unsaved state) and content is successfully written to file.

Table 4: Summary of the three training/evaluation scenarios. Training and evaluation sets use mutually exclusive variable values to ensure out-of-instance generalization.

Scenario	Core Capability	Train	Eval	Steps
Asset Audit	Web form + visual reasoning	64	50	∼20
Browser Search	Cross-app retrieval + document writing	100	50	∼31
File Organization	Rule-driven batch classification	100	50	∼28



Step 5 *Enter audit system* · · · Step 12 *Set filter conditions* · · · Step 16 *Review 1st record* · · · Step 22 *Review 2nd record*

Figure 4: **Asset Audit** demo (25 steps total). The agent navigates a Web-based asset management system, sets filter conditions, then reviews each record by comparing the photo barcode against the system barcode to make pass/reject decisions. The full trace is provided in Appendix E.1.

Table 5: Agreement between LVLM-as-judge and independent human annotation on task success evaluation over 150 trajectories (50 per scenario).

Metric	Value
Accuracy	93.3% (140/150)
Precision	0.906
Recall	0.989
F1	0.946

File Organization Evaluation Rules

Evidence priority: Same as above.

Rules:

1. **Target folders exist:** The two target folders specified in the instruction exist under `/home/user/exp/`.
2. **Files moved to first folder:** All files with the specified extensions have been moved to the first target folder and no longer exist in `/home/user/exp/`.
3. **File moved and renamed:** The specified file has been moved to the second target folder and renamed to the specified new name. The original filename no longer exists in `/home/user/exp/`.
4. **Remaining files intact:** Files not matching any classification rule remain in `/home/user/exp/`, not moved or deleted.

C Progress Reward Validation

C.1 Scoring Agreement

To verify the reliability of LVLM-based progress reward computation (Section 3.1), we conduct independent human checklist scoring on all 150 evaluation trajectories. Human annotators judge each trajectory on a per-checkpoint basis, producing a continuous n/C score in the same format as the LVLM. Table 6 summarizes the agreement. Pearson $r = 0.932$ and weighted $\kappa = 0.934$ indicate that LVLM-based progress scoring is a reliable proxy for human judgment.

Table 6: Agreement between LVLM-based progress scoring and independent human checklist annotation on 150 evaluation trajectories.

Metric	Value
Exact agreement	87.3% (131/150)
Pearson r	0.932 ($p < 10^{-66}$)
Weighted Cohen's κ	0.934
Deviation ≤ 0.2	98.0% (147/150)
Mean score difference	0.016

C.2 Checklist Examples

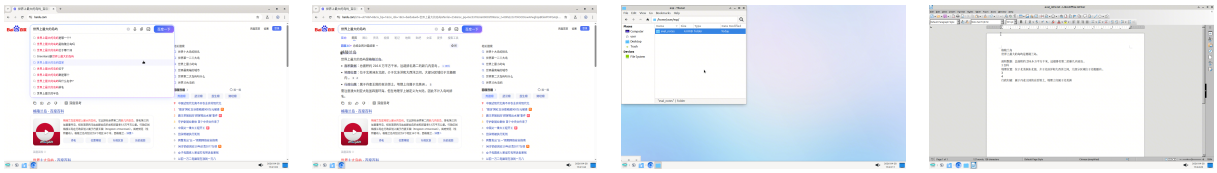
Below we present the automatically extracted checklists for the three scenarios. These are generated by the iterative extraction algorithm (Section 3.1.1) from successful trajectories and used directly in progress reward computation during RL training.

Asset Audit Checklist

Scene description: Asset inventory audit

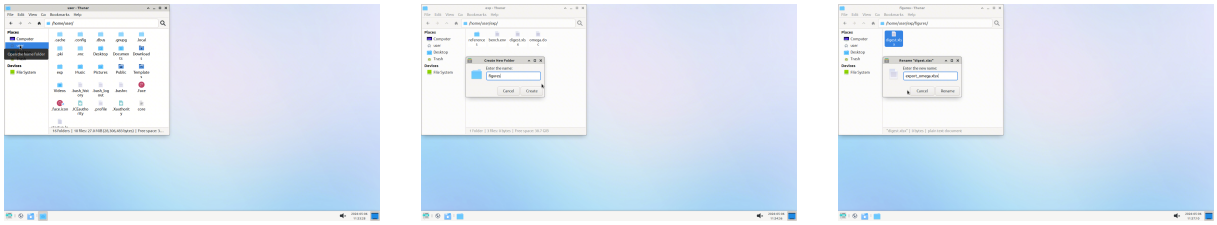
Checkpoint definitions:

1. **System page access:** Successfully entered the asset audit system page.
2. **Filter setup:** Correctly set the audit plan to the plan name specified in the instruction, with audit result set to "unaudited."



Step 7 *Search keywords* · · · Step 10 *Select & copy answer* · · · Step 20 *Create folder in exp/* · · · Step 29 *Paste & save in LibreOffice*

Figure 5: **Browser Search** demo (36 steps total). The agent searches for information in a browser, copies the answer, then switches to the file system to create a directory and file, and saves the content via LibreOffice—demonstrating cross-application coordination. The full trace is provided in Appendix E.2.



Step 3 *Observe initial files* · · · Step 11 *Create target folders* · · · Step 28 *Move & rename file*

Figure 6: **File Organization** demo (31 steps total). The agent observes directory contents, creates classification folders according to sorting rules, then moves and renames files by extension—requiring conditional reasoning and batch operations. The full trace is provided in Appendix E.3.

3. **Record 1 review:** Completed audit of the 1st asset record, with pass/reject judgment consistent with barcode matching rules.
4. **Record 2 review:** Completed audit of the 2nd asset record, with pass/reject judgment consistent with barcode matching rules.
5. **Task termination:** Closed browser after completing 2 record reviews, task correctly terminated without tool call exceptions.

Browser Search Checklist

Scene description: Browser search and information saving

Checkpoint definitions:

1. **Initial environment:** All browser windows closed, returned to desktop.
2. **Search completed:** Successfully navigated to the search engine homepage and displayed search results for the specified keywords.
3. **Answer text copied:** Complete answer text from search results copied to clipboard.
4. **Folder and file prepared:** Navigated to `/home/user/exp/`, created subfolder with the specified name and entered it, and a txt file with the specified name exists within it (if the original file existed, it has been deleted).
5. **File content saved:** Specified txt file opened with LibreOffice, answer text pasted and saved.
6. **Task wrap-up:** All open windows closed, returned to desktop.

File Organization Checklist

Scene description: File organization

Checkpoint definitions:

1. **Enter target directory:** Successfully navigated to `/home/user/exp/`.
2. **Create specified folders:** Created the two folders specified in the instruction under `/home/user/exp/`.
3. **Move specified file types:** Moved all files of types specified in the instruction from `/home/user/exp/` to the corresponding target folders.
4. **Move and rename specified file:** Moved the file specified in the instruction to the corresponding folder and renamed it to the new name specified in the instruction.
5. **Task correctly terminated:** Closed all open windows and ended the task after completing all file organization operations.

D Prompt Templates

This appendix provides the full prompt templates used in our system. Variables filled at runtime are highlighted in orange.

D.1 GUI Agent System Prompt

GUI Agent System Prompt

You are a **professional GUI Agent** (macOS / Windows / Linux). Your task is to execute GUI operations step-by-step based on the user instruction, operation history, and screenshot.

Tool Definitions:

Tool	Description	Parameters
`click`	Left click	`thought, x, y`
`left_double_click`	Double click	`thought, x, y`
`right_click`	Right click	`thought, x, y`
`drag`	Drag	`thought, start_x, start_y, end_x, end_y`
`type`	Type text	`thought, content`
`hotkey`	Key press	`thought, key`
`scroll`	Scroll	`thought, direction, x, y`
`wait`	Wait	`thought`
`finished`	Task complete	`thought`
`call_user`	Call human	`thought`
`output`	Output result	`thought, content`

Rules:

1. Every tool call must include a `thought` field describing your observation and intent.
2. All coordinates must be integers.

D.2 Checklist Extraction Prompt

Checklist Extraction Prompt

Role: You are an expert in Agent evaluation system design.
Task: Given `k` successful Agent trajectories for task `scene`, extract a scoring checklist that captures the task's ordered checkpoints.

Requirements:

1. Analyze all trajectories to identify 3–7 ordered checkpoints representing key milestones in task progression.
2. Each checkpoint must be *visually verifiable*, *independently assessable*, and *non-overlapping*.
3. Each checkpoint contains only: **name** (concise) and **goal description** (one sentence describing the *expected result/state*, not the Agent's action).
4. Use "the XXX specified in the instruction" for placeholder variables rather than hardcoded values.

Reference Example: `example` **Trace Data:** `traces_text`

Output Format: `[SCENE_DESCRIPTION]` task description; `[CHECKPOINT_DEFINITIONS]` numbered list of "Checkpoint Name: Goal description".

D.3 Candidate Checklist Fusion Prompt

Candidate Checklist Fusion Prompt

Role: You are an expert in Agent evaluation system design.
Task: Compare two independently generated candidate checklists (Version A: `checklist_a` and Version B: `checklist_b`) and fuse them into an improved version.

Fusion Principles:

1. *Coverage* — Include key checkpoints found in either version.
2. *Description Quality* — Choose the clearer, more specific goal description. Must describe expected result/state, not action.
3. *Granularity* — Adjust if one version is too fine-grained or too coarse.
4. *Ordering* — Preserve the logical order of task progression.
5. *Independence* — Each checkpoint must be independently assessable.
6. *Non-overlapping* — No scope overlap between checkpoints.
7. *Count* — Typically 3–7 checkpoints total.

Output Format: `[SCENE_DESCRIPTION]` task description; `[CHECKPOINT_DEFINITIONS]` numbered list of "Checkpoint Name: Goal description".

D.4 Checklist-Based Progress Evaluation Prompt

Checklist-Based Progress Evaluation Prompt

Role: You are an impartial automated auditor. Your task is to evaluate the Agent's execution of a `scene_description` task based on objective evidence.

Evaluation Inputs: (1) **User Instruction** — the task assigned to the Agent. (2) **Agent Execution Trace** — each step's action (Function Call) and post-action screenshot (Observation).

Core Principle: Correct Action $\neq$ Correct Result. The Agent's action command may be correct, but the actual execution result in the sandbox may not be. You *must* verify each action's outcome via the post-action screenshot.

Data Trust Hierarchy: (1) *Highest trust*: Screenshot (visual ground truth). (2) *Intent only*: Action code (shows intent, not success). (3) *Never trust directly*: Summary / Thought.

Step-by-Step Screenshot Verification: For each critical action, check its subsequent screenshot: (1) Did the action take effect? (2) Is the result correct? (3) Are there any anomalies?

Checkpoint Evaluation (`num_checkpoints` checkpoints, total score 1.0): `checkpoint_definitions`
Progress score = (completed checkpoints n) / `num_checkpoints`, rounded to 4 decimals. If n = `num_checkpoints`, the progress score is 1.0.

Output Format: `{"progress_score": <score>, "reason": "...", "checkpoint_results": [...]}`

D.5 Subtask Segmentation and Instruction Generation Prompt

Subtask Segmentation and Instruction Generation Prompt

Role: You are an expert in GUI Agent trajectory analysis and credit assignment.
Your task is to segment an Agent execution trajectory (with step indices) into consecutive subtasks and generate a self-contained instruction for each subtask.

Core Objective: Identify semantic phase transitions and produce semantically complete subtask segments. Each segment must represent: (1) an independent subtask goal, (2) an observable success state, and (3) a complete operation loop.

Hard Constraints:

1. *Continuity & Coverage* — Segments must be sequential, non-overlapping, and cover all steps.
2. *Granularity* — Each segment contains 2–15 steps; the total number of segments is at most 7.
3. *Operation Closure* — Never split within a logically atomic operation sequence (e.g., click input box → type text → click search).

Critical Rule — No Outcome Leakage: Describe the expected, observable condition for success, but never claim that the condition was actually achieved.

✗ "The asset record was successfully approved and disappeared from the pending list."

✓ "Approve the specified asset record; success is indicated when it is marked as approved or no longer appears in the pending list."

Instruction Structure: Each instruction must contain: (1) *Context* — the observable GUI state at segment start; (2) *Goal* — the operational objective; and (3) *Success State* — the observable outcome that signals completion without asserting that it occurred.

Output Format: `{"segments": [{"instruction": "...", "start_step": 1, "step_length": 4}, ...]}`

D.6 Subtask Completion Verification Prompt

Subtask Completion Verification Prompt

Role: You are an impartial GUI trajectory verifier.
Your task is to determine whether an entire subtask segment has achieved the success state defined by the `instruction`.

Inputs: `instruction` (context + goal + success state), `start_screenshot` (before the segment), `final_screenshot` (after the segment), `subtask_actions` (all actions), and `tool_outputs` (execution results per action).

Core Verification Principle: Determine completion from observable evidence, prioritizing screenshots over action intent or thought. Use action records and tool outputs only to interpret the observed state; do not infer success from the intended action alone.

Verification Flow:

1. **Execution Validity** — If an unresolved exception or crash prevents a valid final state, output 0.
2. **Success State Match** — Check whether the final observable state matches the instruction's success state.
3. **Goal Relevance** — Confirm that the observed state change is attributable and relevant to the subtask goal.

Key Principle: Ignore recoverable intermediate errors and detours. If the final observable state matches the success state, output 1; otherwise, output 0. The verification result is strictly binary: $v_m \in \{0, 1\}$.

E Example Trajectories

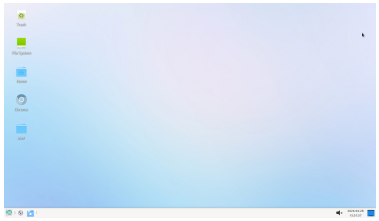
We provide representative execution traces for each of the three scenarios. Each trace shows the agent's reasoning (Thought) and action at every step, paired with the corresponding screenshot.

E.1 Asset Audit Task

Task Instruction

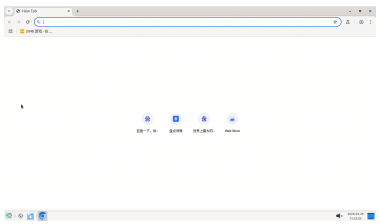
Open a browser and navigate to the asset audit system. Set the filter conditions: Inventory Plan = "Lightfix", Audit Result = "Unaudited". Then perform inventory audit on 2 asset records: if the barcode in the image matches the system barcode, click "Approve"; otherwise click "Reject". Finally close the browser.

Step 1



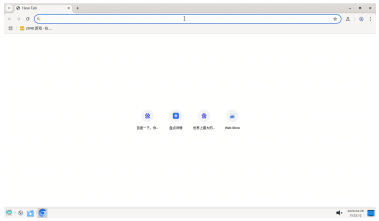
Thought: Double-click the Chrome icon to open the browser.
Action: `left_double_click(x=68, y=389)`

Step 2



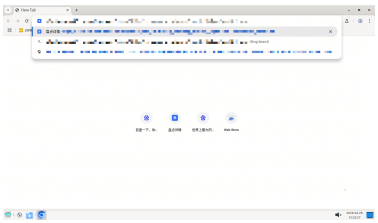
Thought: Click the address bar to input the URL.
Action: `click(x=715, y=61)`

Step 3



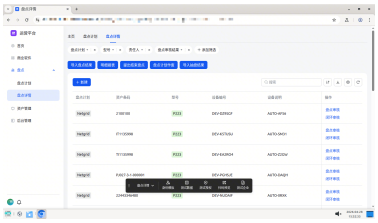
Thought: Type the asset audit system URL into the address bar.
Action: `type(content="https://...audit_system")`

Step 4



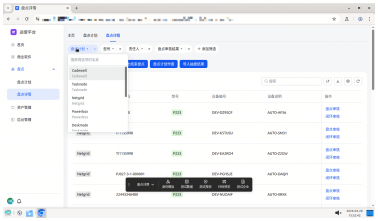
Thought: Press Enter to navigate to the URL.
Action: `hotkey(key="enter")`

Step 5



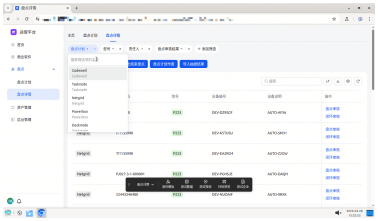
Thought: Click the "Inventory Plan" filter to select the Lightfix plan.
Action: `click(x=292, y=179)`

Step 6



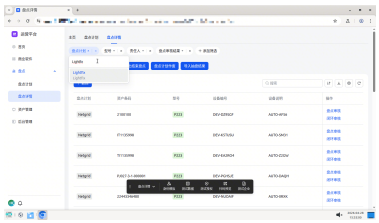
Thought: Click the search box to search for the Lightfix plan.
Action: `click(x=370, y=221)`

Step 7



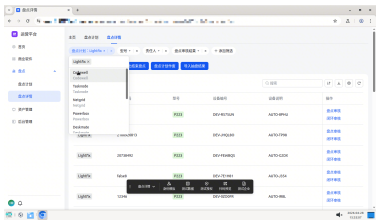
Thought: Type "Lightfix" in the search box to find the plan.
Action: `type(content="Lightfix")`

Step 8



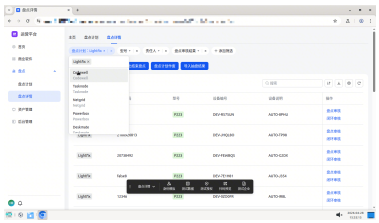
Thought: Select “Lightfix” from the search results.
Action: `click(x=293, y=263)`

Step 9



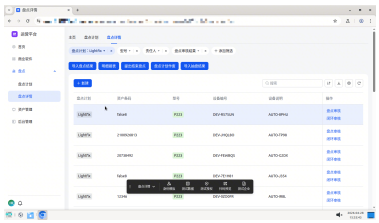
Thought: Press ESC to close the dropdown.
Action: `hotkey(key="escape")`

Step 10



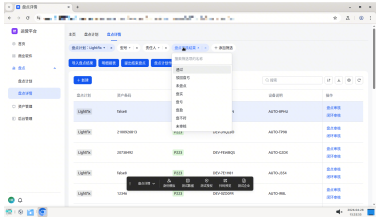
Thought: Click elsewhere to close the dropdown.
Action: `click(x=400, y=400)`

Step 11



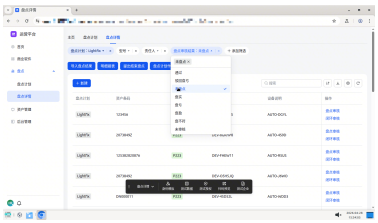
Thought: Click the “Audit Result” filter to select “Unaudited”.
Action: `click(x=710, y=179)`

Step 12



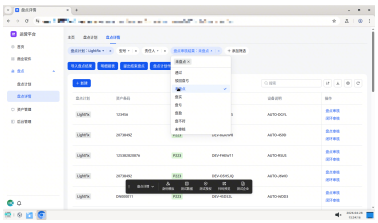
Thought: Select “Unaudited” from the dropdown.
Action: `click(x=693, y=327)`

Step 13



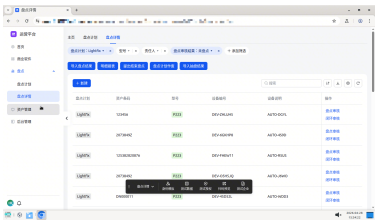
Thought: Press ESC to close the dropdown.
Action: `hotkey(key="escape")`

Step 14



Thought: Click the left area to close the dropdown.
Action: `click(x=150, y=400)`

Step 15



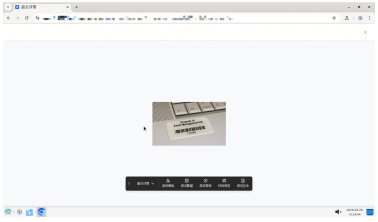
Thought: Click the “Inventory Audit” button for the first record to view details.
Action: `click(x=1290, y=412)`

Step 16



Thought: Click the image to enlarge it for barcode identification.
Action: `click(x=558, y=492)`

Step 17



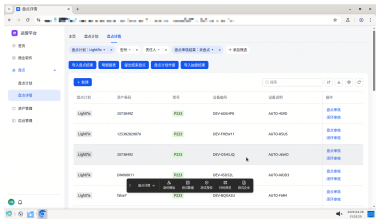
Thought: Click X to close the enlarged image view.
Action: `click(x=1433, y=19)`

Step 18



Thought: Image barcode “123456” matches system record. Click “Approve”.
Action: `click(x=960, y=606)`

Step 19



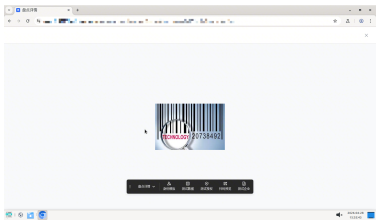
Thought: Click “Inventory Audit” for the second record (barcode: 2073849Z).
Action: `click(x=1290, y=412)`

Step 20



Thought: Click the image to enlarge and verify the barcode.
Action: `click(x=558, y=492)`

Step 21



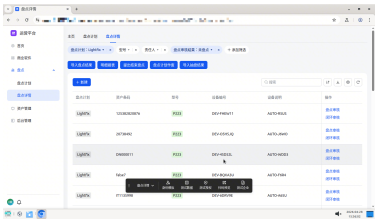
Thought: Press ESC to close the enlarged image.
Action: `hotkey(key="escape")`

Step 22



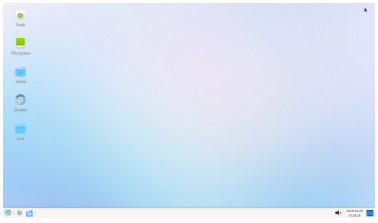
Thought: Image barcode “20738492” does not match system record “2073849Z” (last char: 2 vs Z). Click “Reject”.
Action: `click(x=874, y=614)`

Step 23



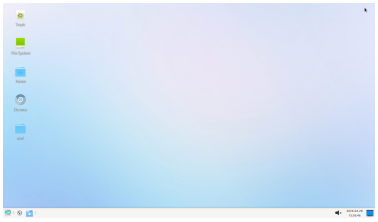
Thought: Click the browser close button (X) to close the browser.
Action: `click(x=1433, y=19)`

Step 24



Thought: Output task completion summary.
Action: `output(content="Task summary...")`

Step 25



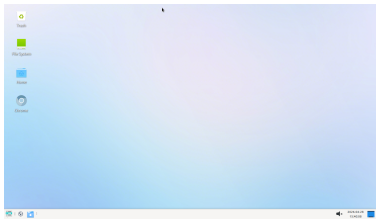
Thought: Task completed. Audited 2 records: first approved (barcode 123456 matched), second rejected (barcode 20738492 vs 2073849Z mismatch). Browser closed.
Action: `finished()`

E.2 Browser Search Task

Task Instruction

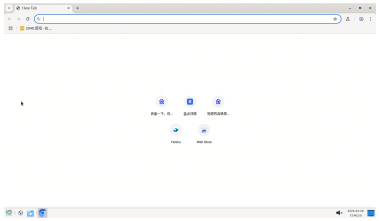
Open a browser, go to Baidu (https://www.baidu.com), search for "the largest island in the world", find and copy the complete answer text. Then go to /home/user/exp/, create a folder named "eval_notes", create "eval_info.txt" inside it, open with LibreOffice, paste the copied text, save and exit. Finally close all windows and return to the desktop.

Step 1



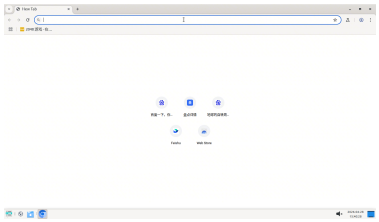
Thought: Double-click the Chrome icon to open the browser.
Action: `left_double_click(x=68, y=389)`

Step 2



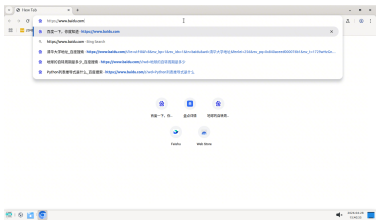
Thought: Click the address bar to input the URL.
Action: `click(x=712, y=61)`

Step 3



Thought: Type the Baidu URL into the address bar.
Action: `type(content="https://www.baidu.com")`

Step 4



Thought: Press Enter to navigate to Baidu.
Action: `hotkey(key="enter")`

Step 5



Thought: Click the search box to enter the query.
Action: `click(x=720, y=290)`

Step 6



Thought: Type the search query about the largest island.
Action: `type(content="What is the largest island in the world")`

Step 7



Thought: Press Enter to search.
Action: `hotkey(key="enter")`

Step 8



Thought: Click the start of the answer text to begin selection.
Action: `click(x=148, y=241)`

Step 9



Thought: Drag to select the complete answer text about Greenland.
Action: `drag(start=(148,241), end=(589,407))`

Step 10



Thought: Re-drag to select the full answer text.
Action: `drag(start=(148,241), end=(610,407))`

Step 11



Thought: Click “Copy” from the context menu to copy the selected text.
Action: `click(x=441, y=443)`

Step 12



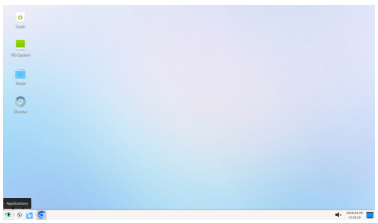
Thought: Click the minimize button to return to desktop.
Action: `click(x=1354, y=19)`

Step 13



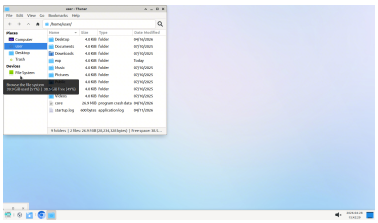
Thought: Click the file manager icon in the taskbar.
Action: `click(x=19, y=820)`

Step 14



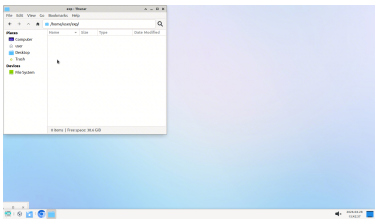
Thought: Double-click Home to open the user’s home directory.
Action: `left_double_click(x=69, y=280)`

Step 15



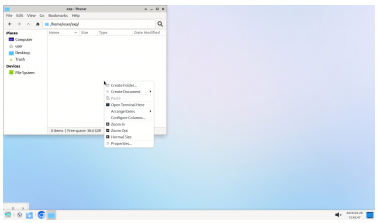
Thought: Double-click the “exp” folder to enter it.
Action: `left_double_click(x=215, y=217)`

Step 16



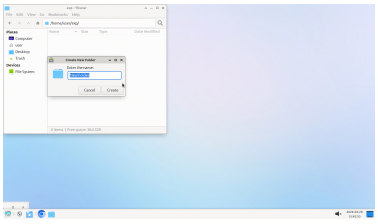
Thought: Right-click in the empty area to create a new folder.
Action: `right_click(x=400, y=300)`

Step 17



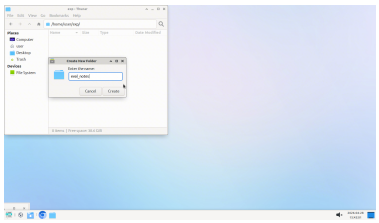
Thought: Click “Create Folder...” to create a new folder.
Action: `click(x=473, y=313)`

Step 18



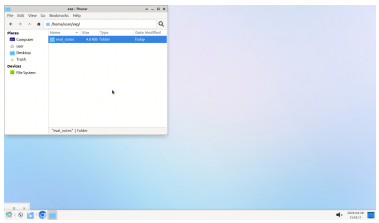
Thought: Type the folder name “eval_notes”.
Action: `type(content="eval_notes")`

Step 19



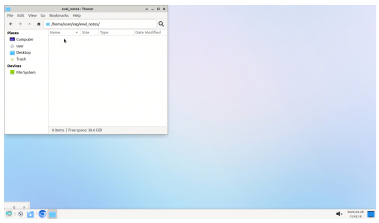
Thought: Click “Create” to confirm folder creation.
Action: `click(x=429, y=336)`

Step 20



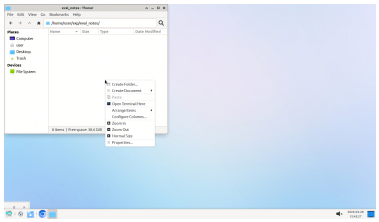
Thought: Double-click “eval_notes” to enter the folder.
Action: `left_double_click(x=238, y=134)`

Step 21



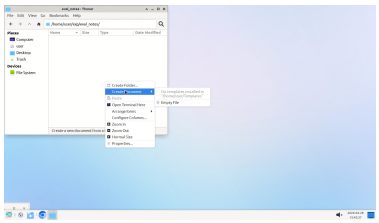
Thought: Right-click to create a new file.
Action: `right_click(x=400, y=300)`

Step 22



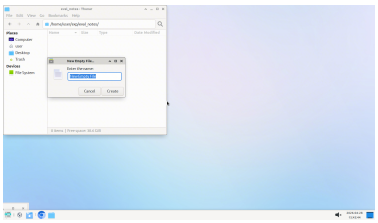
Thought: Click “Create Document” to see sub-menu.
Action: `click(x=480, y=338)`

Step 23



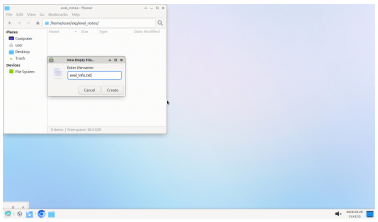
Thought: Click “Empty File” to create a new empty file.
Action: `click(x=650, y=381)`

Step 24



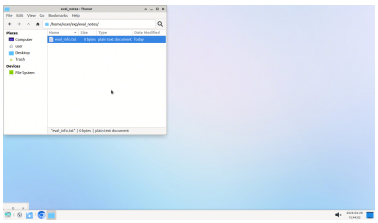
Thought: Type the filename “eval_info.txt”.
Action: `type(content="eval_info.txt")`

Step 25



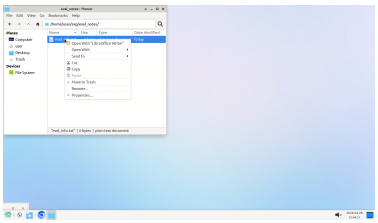
Thought: Click “Create” to confirm file creation.
Action: `click(x=429, y=336)`

Step 26



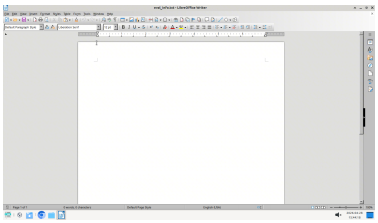
Thought: Right-click eval_info.txt to open with LibreOffice.
Action: `right_click(x=244, y=134)`

Step 27



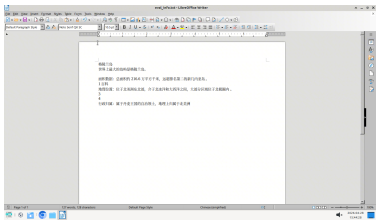
Thought: Click “Open With LibreOffice Writer”.
Action: `click(x=371, y=149)`

Step 28



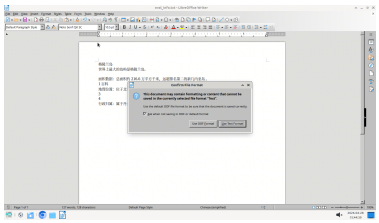
Thought: Paste the copied text using Ctrl+V.
Action: `hotkey(key="ctrl v")`

Step 29



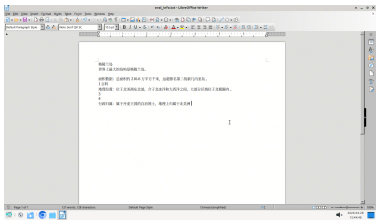
Thought: Save the document using Ctrl+S.
Action: `hotkey(key="ctrl s")`

Step 30



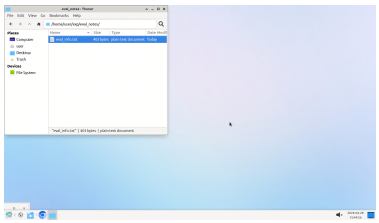
Thought: Click "Use Text Format!" to save as .txt.
Action: `click(x=894, y=463)`

Step 31



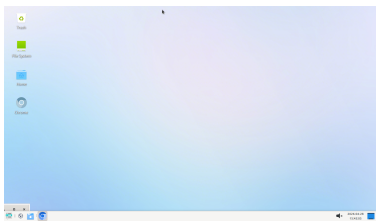
Thought: Exit LibreOffice Writer using Ctrl+Q.
Action: `hotkey(key="ctrl q")`

Step 32



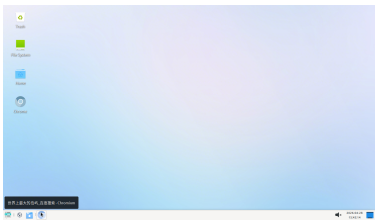
Thought: Close the file manager window.
Action: `click(x=627, y=15)`

Step 33



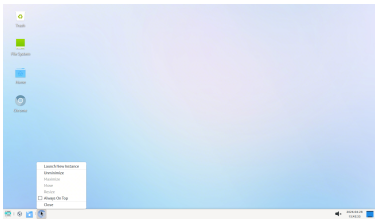
Thought: Click Chrome preview to show the browser window.
Action: `click(x=150, y=820)`

Step 34



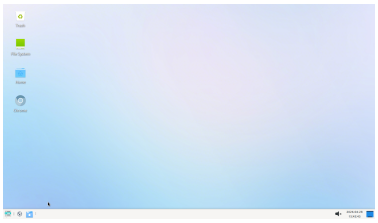
Thought: Right-click Chrome in taskbar to find close option.
Action: `right_click(x=150, y=820)`

Step 35



Thought: Click "Close" to close the Chrome browser.
Action: `click(x=178, y=783)`

Step 36



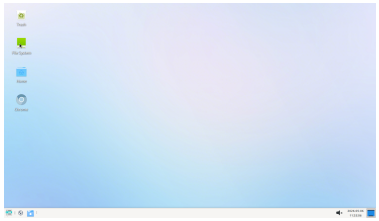
Thought: Task completed. Searched Baidu for the largest island, copied the answer (Greenland), created eval_notes/eval_info.txt, pasted text in LibreOffice, saved, and closed all windows.
Action: `finished()`

E.3 File Organization Task

Task Instruction

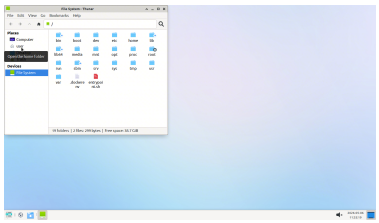
Navigate to /home/user/exp/. Create folders “references” and “figures” if they do not exist. Move all .doc files into references/. Move digest.xlsx into figures/ and rename it to export_omega.xlsx. Leave other files in place. Finally close all windows.

Step 1



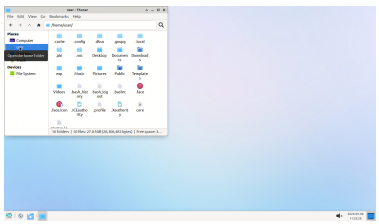
Thought: Double-click the File System icon to open the file manager.
Action: `left_double_click(x=69, y=168)`

Step 2



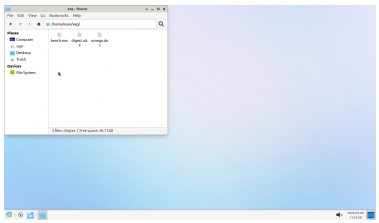
Thought: Click the user folder in the sidebar to navigate to home directory.
Action: `click(x=62, y=161)`

Step 3



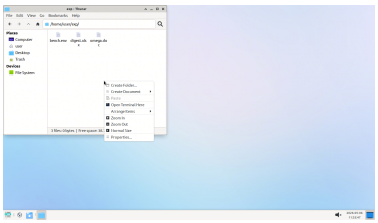
Thought: Double-click the “exp” folder to enter it.
Action: `left_double_click(x=216, y=264)`

Step 4



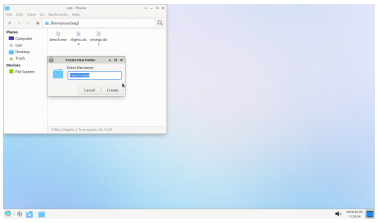
Thought: Right-click in empty area to create a new folder.
Action: `right_click(x=400, y=300)`

Step 5



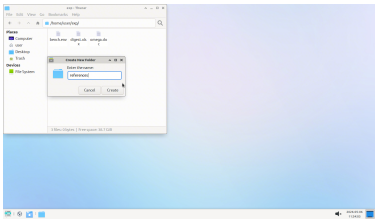
Thought: Click “Create Folder...” to create the references folder.
Action: `click(x=473, y=313)`

Step 6



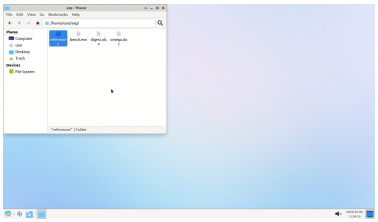
Thought: Type the folder name “references”.
Action: `type(content="references")`

Step 7



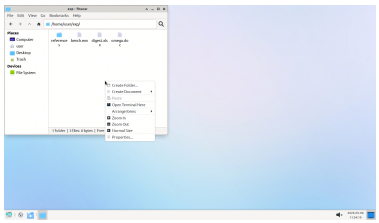
Thought: Click “Create” to confirm folder creation.
Action: `click(x=429, y=336)`

Step 8



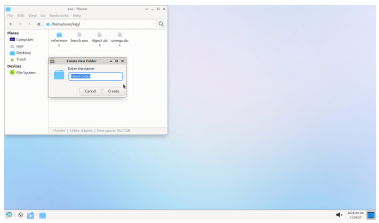
Thought: Right-click in empty area to create the second folder.
Action: `right_click(x=400, y=300)`

Step 9



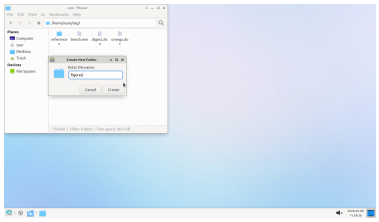
Thought: Click “Create Folder...” to create the figures folder.
Action: `click(x=473, y=313)`

Step 10



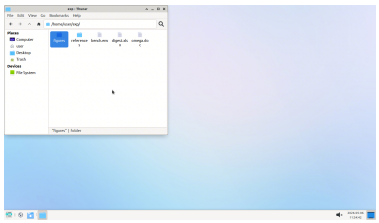
Thought: Type the folder name “figures”.
Action: `type(content="figures")`

Step 11



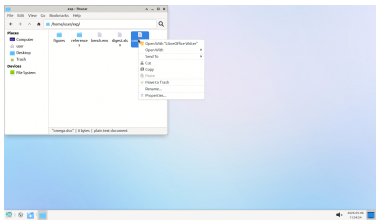
Thought: Click “Create” to confirm folder creation.
Action: `click(x=429, y=336)`

Step 12



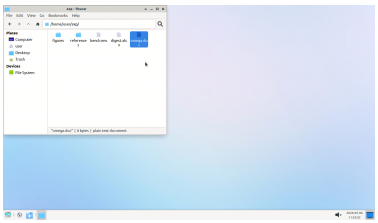
Thought: Right-click omega.doc to cut it for moving to references/.
Action: `right_click(x=532, y=135)`

Step 13



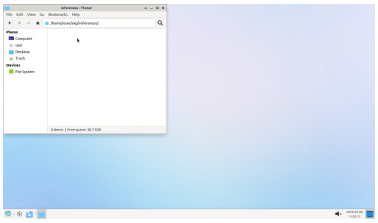
Thought: Click “Cut” to cut omega.doc.
Action: `click(x=564, y=226)`

Step 14



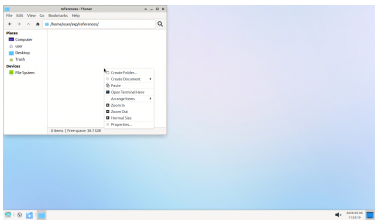
Thought: Double-click references/ folder to enter it.
Action: `left_double_click(x=294, y=135)`

Step 15



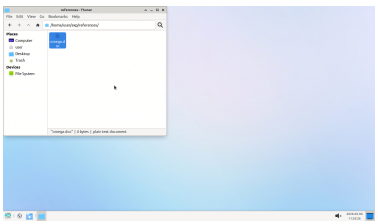
Thought: Right-click in empty area to paste omega.doc here.
Action: `right_click(x=400, y=250)`

Step 16



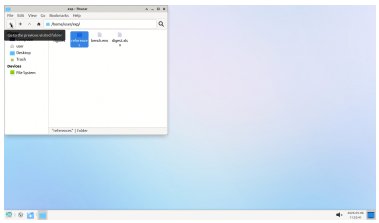
Thought: Click “Paste” to move omega.doc into references/.
Action: `click(x=441, y=314)`

Step 17



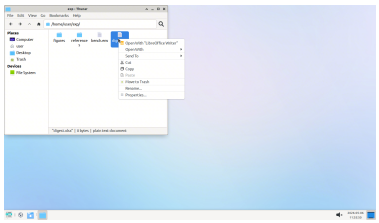
Thought: Click the back button to return to /home/user/exp/.
Action: `click(x=27, y=74)`

Step 18



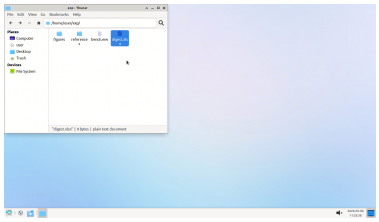
Thought: Right-click digest.xlsx to cut it for moving to figures/.
Action: `right_click(x=453, y=133)`

Step 19



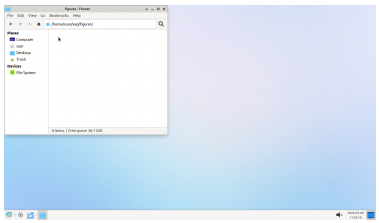
Thought: Click “Cut” to cut digest.xlsx.
Action: `click(x=486, y=224)`

Step 20



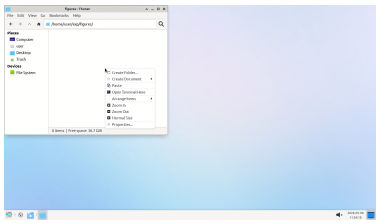
Thought: Double-click figures/ folder to enter it.
Action: `left_double_click(x=216, y=127)`

Step 21



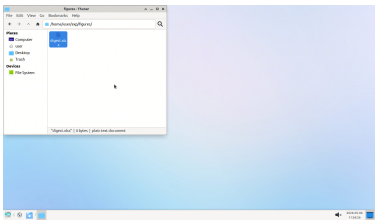
Thought: Right-click in empty area to paste digest.xlsx here.
Action: `right_click(x=400, y=250)`

Step 22



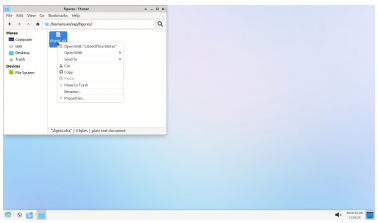
Thought: Click “Paste” to move digest.xlsx into figures/.
Action: `click(x=441, y=314)`

Step 23



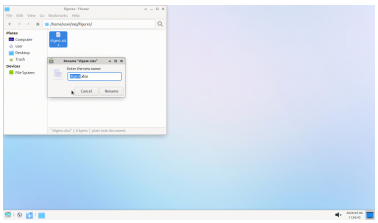
Thought: Right-click digest.xlsx to rename it to export_omega.xlsx.
Action: `right_click(x=215, y=146)`

Step 24



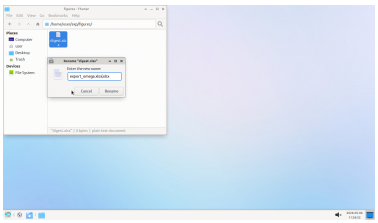
Thought: Click “Rename...” to start renaming.
Action: `click(x=272, y=338)`

Step 25



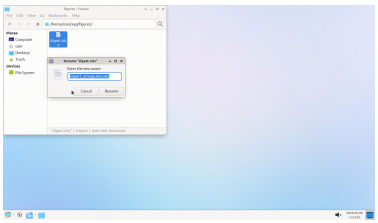
Thought: Type the new filename “export_omega.xlsx”.
Action: `type(content="export_omega.xlsx")`

Step 26



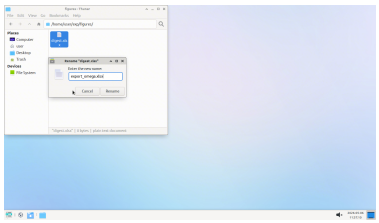
Thought: Select all text to re-enter the correct filename.
Action: `hotkey(key="ctrl a")`

Step 27



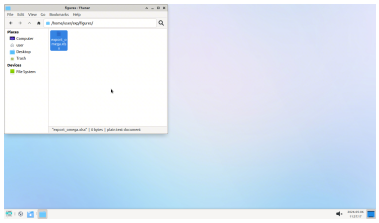
Thought: Re-type the filename “export_omega.xlsx”.
Action: `type(content="export_omega.xlsx")`

Step 28



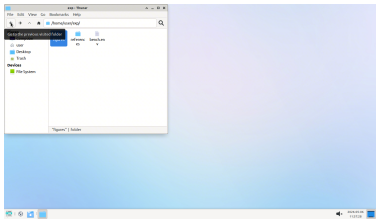
Thought: Click “Rename” to confirm the rename operation.
Action: `click(x=424, y=336)`

Step 29



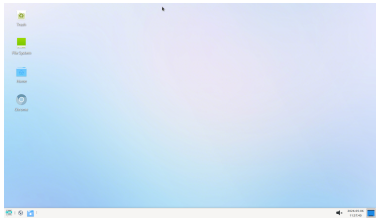
Thought: Click the back button to return to /home/user/exp/ and verify.
Action: `click(x=27, y=74)`

Step 30



Thought: Click the close button to close the file manager window.
Action: `click(x=627, y=15)`

Step 31



Thought: Task completed. Created references/ and figures/ folders, moved omega.doc to references/, moved digest.xlsx to figures/ and renamed it to export_omega.xlsx. All windows closed.
Action: `finished()`

F Use of LLMs

We used LLMs exclusively as writing assistants to refine language. In particular, their use was restricted to grammar correction, style improvement, and phrasing adjustments for clarity and conciseness.