

Beyond Planning: Sparse Context Reset for Long-Horizon GUI Agents

Weiyao Huang¹, Xinye Wu¹, Zeming Liu²
Xiaoyu Shen³, Yunpu Ma^{4,5,6*}

¹Sun Yat-sen University ²Beihang University

³Eastern Institute of Technology, Ningbo

⁴LMU Munich ⁵TUM ⁶Munich Center for Machine Learning

huangwy95@mail2.sysu.edu.cn cognitive.yunpu@gmail.com

Abstract

Graphical User Interface (GUI) agents powered by vision-language models (VLMs) have emerged as a promising paradigm for computer-use automation, but remain unreliable on long-horizon tasks. Existing planner-executor frameworks address this by shifting more reasoning to the planner, thereby decomposing rather than decoupling long-horizon reasoning. Our failure analysis reveals that many long-horizon failures are not pure capability gaps: from the same failure state, executors recover substantially more often when their accumulated context is reset with a locally executable subtask. To bridge this gap, we propose SCOR, a sparse context reset framework that restores the executor’s short-horizon capability by resetting its accumulated task context. At subtask boundaries, a lightweight REFINER emits a locally executable subtask and resets the EXECUTOR’s context; the executor then autonomously executes multiple GUI actions within the reset context. We train the REFINER with turn-level GRPO. In OSWorld, SCOR achieves a success rate of 53.16%, yielding a 25.41% relative improvement over the standalone EvoCUA-8B executor. Our code is available at <https://github.com/huia711/SCoR>.

1 Introduction

GUI agents powered by vision-language models (VLMs) have emerged as a compelling paradigm for computer-use automation, demonstrating the ability to navigate desktop applications, fill web forms, and execute multi-step workflows from natural language instructions (Qin et al., 2025; Xu et al., 2025; Wu et al., 2025; Wang et al., 2025; Awadallah et al., 2025; Xue et al., 2026).

*Corresponding author.

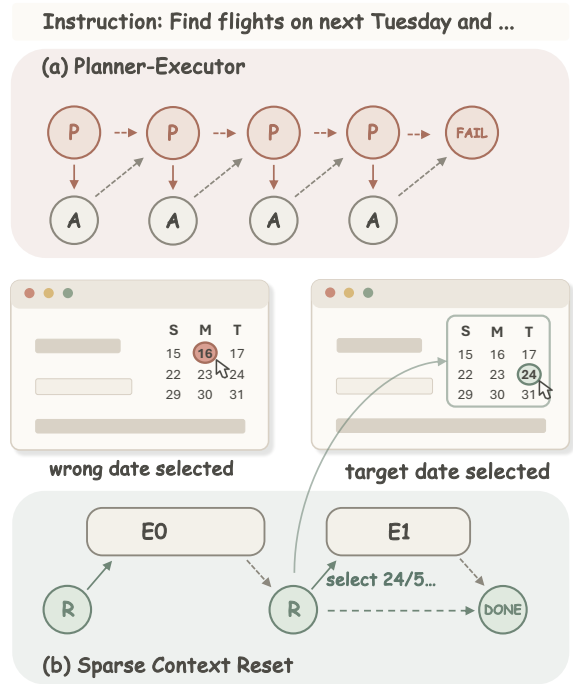


Figure 1: Motivation for sparse context reset. A planner-executor loop repeatedly emits planner directives (P) for executor actions (A); when the planner does not provide stronger GUI priors, frequent intervention may propagate incorrect local decisions to the executor. SCOR instead lets the executor run autonomously (E_0), invokes the REFINER only at a subtask boundary (R), and resets the EXECUTOR’s context with a locally executable subtask so that the next rollout (E_1) can restore the executor’s short-horizon capability.

Recent specialized agents such as UI-TARS-1.5 (Qin et al., 2025) and EvoCUA (Xue et al., 2026) achieve strong performance in short-horizon tasks, suggesting that grounding and short-horizon execution are increasingly reliable for routine interactions. Yet even these agents fail systematically on

long-horizon tasks that require tracking evolving UI states, reasoning about implicit preconditions, or recovering from dense-widget errors—the failure modes that matter most in real computer use.

Hierarchical GUI frameworks attempt to address this by separating strategic reasoning from atomic execution (Agashe et al., 2025; Sun et al., 2025; Xie et al., 2025; Deng et al., 2026; Cheng et al., 2026). Agent S2 maintains and revises a subgoal plan; CODA and Jedi mediate execution at the action level; and CES and MGA augment step-wise decisions with compressed task state or structured memory. Despite their differences, these frameworks remain planner-centric: long-horizon competence is still carried by a Planner that must maintain progress and decide the next operation. Even in subgoal-based systems such as Agent S2, the Worker is not reset into an isolated local objective; it executes under the global task and subgoal list and must still interpret local actions in the presence of the full long-horizon context. In other words, prior hierarchies mostly decompose the reasoning-to-action pipeline rather than decouple long-horizon reasoning. This places the hardest reasoning burden on the planner side, which explains why hierarchical agents often rely on frontier reasoning models, such as ChatGPT-o3.

This raises a question: *can we improve long-horizon GUI agents without making the planner stronger, by resetting an EXECUTOR’s context so that it can restore its short-horizon capability?*

To probe this hypothesis, we conduct an oracle-intervention study on OSWorld tasks on which both UI-TARS-1.5-7B and EvoCUA-8B fail under the original global instruction. We restart execution from that executor’s first-failure UI state under two conditions: Global retains the original instruction and accumulated interaction history, whereas Local clears the history and replaces the global instruction with a locally executable instruction for the current state. As shown in Figure 2, the fail-step recovery rate increases from 50.0% to 77.8% for UI-TARS-1.5 and from 50.0% to 74.1% for EvoCUA. These recoveries further translate into end-to-end task success rates of 9.3% and 16.7%, respectively, compared with 0% under Global. These results reveal a **Task Granularity Gap**: executors may fail under accumulated long-horizon context yet recover when reset with a locally executable subtask aligned with their stable execution granularity.

The Task Granularity Gap points to a different design principle for GUI collaboration. If an ex-

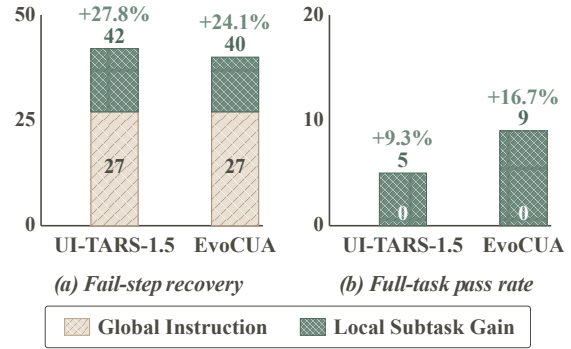


Figure 2: Quantifying the Task Granularity Gap. From the same first-failure UI state, resetting the EXECUTOR’s context and replacing the global instruction with a locally executable subtask substantially improves recovery and yields additional full-task successes.

ecutor already has reliable short-horizon capability, then the auxiliary model should not continuously mediate execution. Instead, it should reset the executor’s accumulated context and delegate a locally executable subtask back to the executor. In response, we propose **SCOR** (Sparse Context Reset), a REFINER–EXECUTOR framework for long-horizon GUI agents. A lightweight REFINER intervenes only at subtask boundaries, resets the EXECUTOR’s context with a locally executable subtask, and then lets the EXECUTOR run autonomously. We train the REFINER with turn-level GRPO, using progress feedback from the resulting EXECUTOR rollouts to optimize reset-context generation.

We summarize our contributions as follows:

1. Identifying the Task Granularity Gap. We identify the Task Granularity Gap as a core failure mode in long-horizon GUI agents: an executor may fail not for lack of capability, but because its accumulated context is misaligned with its stable execution granularity.
2. Sparse Context Reset. We propose SCOR, a REFINER–EXECUTOR framework that uses a lightweight REFINER to reset the EXECUTOR’s context at boundaries with locally executable subtasks. This addresses the Task Granularity Gap without relying on a planner with stronger reasoning capability.
3. Turn-level GRPO. We train the REFINER by comparing reset contexts at the same GUI boundary. In OSWorld, SCOR yields relative improvements of 65.49% and 25.41% over UI-TARS-1.5-7B and EvoCUA-8B

2 Related Work

GUI Agents. Recent VLM-based GUI agents have advanced screenshot-based computer-use automation (OpenAI et al., 2024; Yang et al., 2025a; Qin et al., 2025; Xu et al., 2025; Wu et al., 2025; Wang et al., 2025; Awadallah et al., 2025; Xue et al., 2026). Monolithic agents such as OS-Atlas, UI-TARS, OpenCUA, Fara, and EvoCUA unify perception, reasoning, grounding, and action generation, yet remain brittle on long-horizon tasks involving evolving UI state, dense controls, and implicit preconditions (Xie et al., 2024).

Hierarchical GUI systems adopt different planner-executor interfaces. CODA (Sun et al., 2025) keeps the executor frozen and trains a Planner to generate a Thought-and-Action pair at each step, which the executor grounds into an atomic action. The Jedi-7B w/ o3 system (Xie et al., 2025) uses o3 to issue action-level decisions grounded by Jedi-7B. CES (Deng et al., 2026) and MGA (Cheng et al., 2026) likewise operate step-wise, but explicitly manage long-horizon state: CES compresses task progress with a State Tracker, whereas MGA replaces raw trajectory history with validated structured memory. Agent S2 (Agashe et al., 2025) uses a Manager to maintain and revise a subgoal plan, while a Worker executes under the global task and subgoal list before routing actions to grounding experts. Despite their differences, these methods remain planner-centric and do not isolate the executor from accumulated long-horizon context. SCOR instead treats the GUI executor as a frozen specialist and uses a lightweight REFINER only to reset its context at sparse subtask boundaries, preserving autonomous multi-step execution within each locally executable subtask.

Reinforcement Learning for Agents. Reinforcement learning has become central to improving reasoning and agentic behavior in VLMs, with GRPO and DAPO enabling stable policy optimization without dense human supervision (Shao et al., 2024; Yu et al., 2025). In GUI control, DigiRL and ZeroGUI study online learning, task generation, and reward estimation (Bai et al., 2024; Yang et al., 2025b). Prior work mainly optimizes monolithic or tightly coupled policies, making long-horizon credit assignment costly. SCOR instead trains only the REFINER with turn-level GRPO, using progress in the resulting executor sub-rollouts to score reset contexts sampled at the same GUI boundary.

Method	Planner	Invocation	Context
CODA	Qwen2.5-VL-32B	Step	Shared
Jedi	ChatGPT-o3	Step	Shared
MGA	ChatGPT-o3	Step	Structured memory
Agent S2	Gemini-2.5-Pro	Mixed	Shared
SCOR	Qwen3-VL-8B	Boundary	Reset

Table 1: Interface comparison with recent multi-agent computer-use frameworks. Unlike step-level systems, SCOR invokes a lightweight REFINER at sparse boundaries and resets the EXECUTOR’s context with a locally executable subtask for autonomous local execution.

3 Task Granularity

Problem Setting. We study GUI agents in desktop environments, formalized as a Partially Observable Markov Decision Process (POMDP). Given an instruction I , the agent observes a screenshot $o_t \in \mathcal{O}$ and selects an atomic action $a_t \in \mathcal{A}$ (click, keystroke, scroll, etc.) over discrete time steps, following the PyAutoGUI convention (Qin et al., 2025). An episode terminates when the agent emits a termination signal or reaches the step limit $T_{\max}$. A task-specific evaluator $\mathcal{V} : \mathcal{S} \rightarrow [0, 1]$ determines success from the terminal environment state $s_T \in \mathcal{S}$. The interaction induces a trajectory:

$$\tau = ((o_1, a_1), (o_2, a_2), \dots, (o_T, a_T)), \quad (1)$$

Long-Horizon GUI Tasks as Subtask Sequences.

Prior hierarchical GUI frameworks organize complex task execution through subtasks at different temporal scales (Agashe et al., 2025; Liu et al., 2025). Following this view, we represent a long-horizon GUI task as a sequence of short-horizon subtasks, each inducing a contiguous sub-rollout of the overall interaction trajectory. Formally, we partition τ at subtask boundaries:

$$0 = b_0 < b_1 < \dots < b_M = T, \quad (2)$$

$$\rho_m = ((o_t, a_t))_{t=b_{m-1}+1}^{b_m},$$

where each sub-rollout ρ_m is induced by a subtask instruction z_m given to the executor. We define its length $|\rho_m| = b_m - b_{m-1}$ as the task granularity. Task granularity determines how reasoning is divided between the auxiliary model and the executor. Per-action segmentation ($|\rho_m| = 1$) shifts step-level decisions to the auxiliary model (Sun et al., 2025; Lu et al., 2025); monolithic execution ($M = 1$) instead requires the executor to manage the full instruction, accumulated history, and evolving UI state throughout the task.

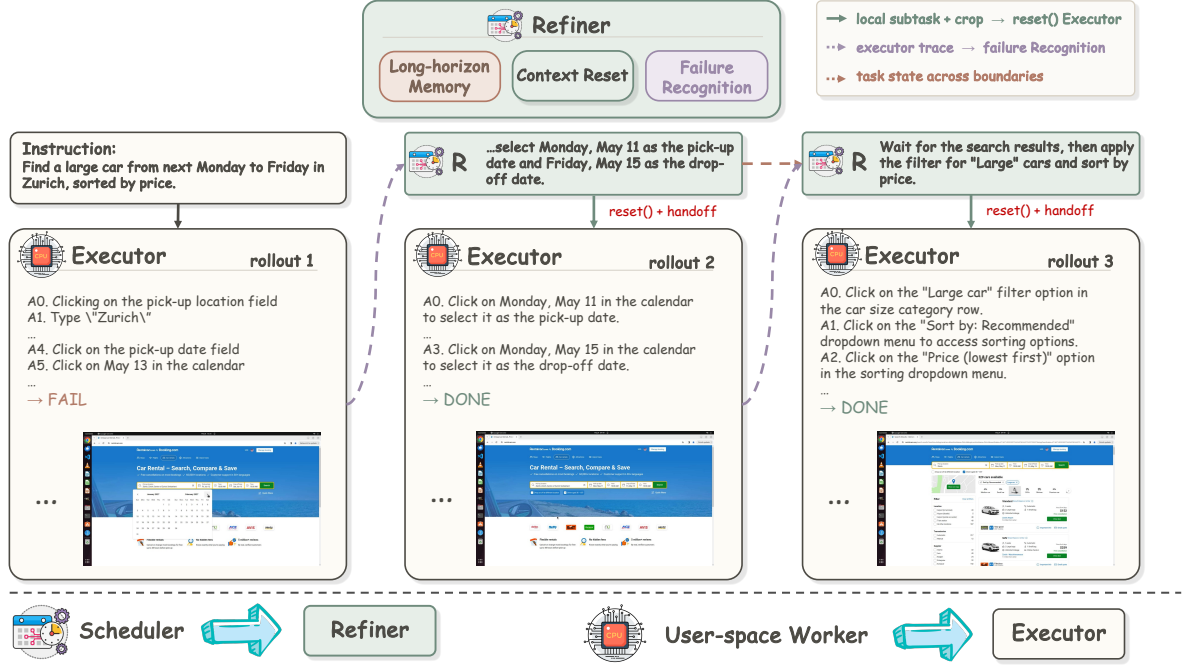


Figure 3: Overview of the SCOR collaborative loop. A shared REFINER is invoked only at subtask boundaries. Each REFINER call emits a locally executable subtask and visual crop, resets the EXECUTOR’s context, and hands off control for an autonomous rollout. Then the EXECUTOR runs autonomously for multiple GUI actions. The REFINER also reads the previous executor trace for failure recognition and maintains long-horizon task memory across boundaries.

The Granularity Sweet-Spot Hypothesis. The desired operating point lies between these extremes, where subtasks are local enough to fall within the executor’s stable execution regime but coarse enough to preserve autonomous multi-step execution. Recent GUI benchmark evidence supports this dependence: the same VLM executor that achieves only 20% success on raw user instructions gains roughly 30 absolute points when given manually crafted intermediate plans (Lin et al., 2024). Our failure-state replay study provides further evidence: from the same failure states, a frozen executor recovers more often when reset with a locally executable instruction (Figure 2).

We therefore formulate the Granularity Sweet-Spot Hypothesis: for a given frozen GUI executor, an intermediate task granularity may yield more reliable execution than either per-action or monolithic execution. SCOR operationalizes this hypothesis by learning whether to reset the context and how to formulate the locally executable subtask.

4 Methodology

As shown in Figure 3, SCOR coordinates a REFINER and an EXECUTOR through sparse context

resets. The EXECUTOR runs autonomously under its local instruction context. Control returns to the REFINER only at execution boundaries, when the EXECUTOR emits a boundary signal (DONE, FAIL, or CALL_USER) or exhausts its local step budget. At each returned boundary, the REFINER inspects the global instruction, current screenshot, long-horizon memory from the history, and the latest subtask rollout to recognize whether execution has succeeded, entered an incorrect local state, made insufficient progress, or requires user intervention. It then decides whether to terminate the task or reset the EXECUTOR’s context with a locally executable subtask and visual focus.

This loop is inspired by OS-style context switching under cooperative scheduling. The REFINER plays a kernel-like role: it maintains long-horizon task state, diagnoses returned executor traces, and switches the semantic context for the next rollout. The EXECUTOR behaves like a user-space worker: it runs autonomously inside an isolated local context until it returns control at the next boundary.

4.1 Refiner

The REFINER is the only trainable component of SCOR and acts as a boundary-level recovery con-

troller. Its capability consists of three coupled functions: *long-horizon memory*, which preserves task state across executor rollouts; *failure recognition*, which diagnoses the returned executor trace; and *context reset*, which converts the global task state into a locally executable context for the next rollout. At each boundary n , the REFINER receives

$$x_n = \phi(I, M_{<n}, F_{n-1}, o_n), \quad (3)$$

where I is the global instruction, $M_{<n}$ is the history of completed subtasks, F_{n-1} contains compact feedback from the latest Executor sub-rollout, and o_n is the current screenshot. The Refiner emits

$$d_n = (T_n, z_n, c_n), \quad (4)$$

where T_n is a failure-recognition reasoning trace that checks whether the latest rollout completed, entered an incorrect local state, exhausted its budget without sufficient progress, or requires user intervention. z_n is either the next locally executable subtask or a terminal control signal (DONE, FAIL, or CALL_USER), and c_n is an optional crop region that anchors the EXECUTOR’s first visual observation. The crop acts as a lightweight visual-focus mechanism, consistent with evidence that region-centric visual inputs can improve fine-grained multimodal perception (Wei et al., 2026).

The REFINER never emits clicks, keystrokes, or other atomic GUI actions. It also does not need to maintain a full plan. Its role is to maintain compact cross-boundary task state and decide how the EXECUTOR should be re-contextualized for the next local rollout: what objective should be attempted, which task constraints should remain visible, and which UI region should receive visual focus.

4.2 Executor

The EXECUTOR, denoted by $\mathcal{E}$, is a frozen GUI-specialist agent. Whenever z_n is a local subtask, SCOR starts a fresh local dialogue for the executor and initializes it with the reset context (z_n, c_n) . The EXECUTOR then interacts with the environment for up to K atomic steps:

$$(a_{n,j}, t_{n,j}) \sim \mathcal{E}(\cdot \mid z_n, \tilde{o}_{n,j}, H_{n,j}^{\text{loc}}), \quad (5)$$

where $a_{n,j}$ is an atomic GUI action or a boundary signal, $t_{n,j}$ is the EXECUTOR’s reasoning trace, $\tilde{o}_{n,j}$ is either the crop specified by c_n or the full screenshot, and $H_{n,j}^{\text{loc}}$ contains only the interaction history within the current subtask.

Crucially, the Executor does not receive the accumulated dialogue history from previous rollouts. Long-horizon information reaches it only through the compact reset context produced by the REFINER. This reset removes failed attempts and irrelevant traces, replaces the global instruction with a local objective, and redirects visual attention to the task-relevant region. Within the subtask, the EXECUTOR remains autonomous: it decides actions without REFINER intervention until it emits a boundary signal or reaches the local step budget.

4.3 Training

Data Construction. Following ZeroGUI (Yang et al., 2025b), we construct an OSWorld-style training pool. Training and evaluation share only OSWorld’s general infrastructure; the generator has no access to evaluation instructions, task-specific targets, initialization configurations, evaluator programs, or trajectories. After generation, we apply a Task-Instance Overlap Filter (TIOF) that removes candidates overlapping with OSWorld in instruction semantics, task-specific targets, or action-chain structure. The remaining tasks are dynamically verified for executability and non-triviality. Appendix B.2 details the construction pipeline, leakage prevention and dynamic verification for the generated tasks.

Turn-Level Online GRPO. We train only the REFINER while keeping the EXECUTOR frozen. At each boundary, we sample G candidate decisions from the same GUI state and REFINER input, and compare how effectively they help the fixed EXECUTOR advance the original task. Each candidate $d_n^{(g)}$ receives a boundary-level reward:

$$R_n^{(g)} = R_{\text{prog},n}^{(g)} + \lambda_c R_{\text{crop},n}^{(g)}, \quad (6)$$

The progress reward is assigned by a hybrid evaluator. For a local subtask candidate, the VLM judge $\mathcal{J}$ evaluates whether the proposed subtask matches the frozen EXECUTOR’s local execution capability and whether the resulting rollout makes progress toward the original task. For a terminal decision, the programmatic evaluator $\mathcal{V}$ determines whether the current environment state satisfies the original task.

$$R_{\text{prog},n}^{(g)} = \begin{cases} J_{\text{prog}}(I, d_n^{(g)}, \rho_n^{(g)}), & \text{local,} \\ \mathcal{V}(s_n), & \text{terminal,} \end{cases} \quad (7)$$

Here, $\rho_n^{(g)}$ denotes the executor sub-rollout, including its reasoning, actions and observations.

Model	System	Office	Daily	Prof.	OS	Multi-App	Overall
<i>General VLMs</i>							
Qwen3-VL-8B-Instruct	Single	11.1	46.7	53.1	45.8	6.5	25.60
computer-use-preview [†]	Single	23.0	37.1	51.0	70.8	15.8	31.20
Claude-4-Sonnet-20250514 [†]	Single	43.6	55.1	55.1	45.8	28.5	43.88
<i>Agentic frameworks</i>							
Agent S2 w/ Gemini-2.5-Pro [†]	Multi	46.1	48.5	67.3	62.5	28.7	45.76
Jedi-7B w/ o3 [†]	Multi	47.0	61.4	69.4	54.2	35.0	<u>50.65</u>
<i>Ours</i>							
UI-TARS-1.5-7B	Single	22.2	28.2	57.1	34.8	6.5	24.98
+ Qwen3-VL-8B-Instruct	Multi-Prompt	30.8	42.3	55.1	62.5	9.8	33.32+33.4%
+ SCoR	Multi-RL	39.3	47.4	69.4	62.5	20.4	41.34+65.5%
EvoCUA-8B	Single	40.1	49.9	75.5	58.3	19.3	42.39
+ Qwen3-VL-8B-Instruct	Multi-Prompt	44.4	<u>63.4</u>	<u>79.6</u>	<u>66.7</u>	27.7	50.54+19.2%
+ SCoR	Multi-RL	<u>46.6</u>	64.1	83.7	<u>66.7</u>	<u>32.5</u>	53.16 +25.4%

Table 2: Main results in OSWorld. We report task success rate (%) aggregated by domain. Bold indicates the best result and underline indicates the second-best result. Percentages in the Overall column show relative improvement over the corresponding frozen executor. [†] denotes reported results under original protocols.

The auxiliary crop reward $R_{\text{crop},n}^{(g)}$ encourages the proposed crop to cover a task-relevant UI region. This region is inferred from the OSWorld accessibility tree, which is used only by the training backend and is never exposed to the REFINER or EXECUTOR.

5 Experiments

We organize the evaluation around three questions:

RQ1 (Effectiveness). Does SCoR improve frozen GUI executors on complex multi-step GUI tasks under a matched action budget?

RQ2 (Transferability). Does SCoR generalize across different frozen executors and across different domains (desktop OS, real web)?

RQ3 (Collaboration interface). Which collaboration interface is responsible for the gain: executor self-reset, action-level planning, prompted boundary reset, or trained boundary reset?

5.1 Experimental Setup

Training and Evaluation Details. We use Qwen3-VL-8B-Instruct as the base model for the REFINER and keep the EXECUTOR frozen throughout training. We optimize the REFINER with online RL using Slime (Zhu et al., 2025). Each trajectory contains at most 10 REFINER calls and 50 total EXECUTOR actions. Each sub-rollout is capped at $K = 15$ executor actions for UI-TARS-1.5-7B and $K = 25$ for EvoCUA-8B. We train with Adam using a learning rate of 5×10^{-7} and set $\lambda_c = 0.5$.

All main experiments are conducted on $8 \times \text{H100}$ GPUs per node.

Benchmarks. We evaluate SCoR on OSWorld (Xie et al., 2024), a desktop OS benchmark of 369 Ubuntu tasks aggregated into five domains: Office (LibreOffice Calc / Impress / Writer), Daily (Chrome / Thunderbird / VLC), Professional (GIMP / VS Code), OS operations, and Multi-App workflows; and Online-Mind2Web (Xue et al., 2025), an online web benchmark of 300 tasks across 136 real-world websites spanning nine official website categories, and uses WebJudge (o4-mini) for automatic evaluation.

Baselines. We compare against three categories of systems. (i) *General VLMs*: Qwen3-VL-8B-Instruct, OpenAI computer-use-preview, and Claude-4-Sonnet-20250514, each evaluated as a standalone GUI policy. (ii) *Agentic frameworks*: Agent S2 with Gemini-2.5-Pro (Agashe et al., 2025) and Jedi-7B with ChatGPT-o3 (Xie et al., 2025). (iii) *Ours*: UI-TARS-1.5-7B (Qin et al., 2025) and EvoCUA-8B (Xue et al., 2026) as standalone executors, together with prompted and trained SCoR variants using the same frozen executors. All controlled comparisons use the same 50-step GUI action budget.

Agent S2 and Jedi are official OSWorld-Verified reference results obtained by the benchmark team under unified task environments and execution-based evaluators. Our experiments use the same task definitions, VM initialization process, and evaluators, but the inference-time configurations are

Model	System	Easy	Medium	Hard	Overall
UI-TARS-1.5-7B	Standalone	54.8	28.6	6.7	29.9
+ Qwen3-VL-8B-Instruct	Multi-Prompt	54.8	<u>32.1</u>	<u>16.7</u>	<u>34.2</u> +14.4%
+ SCoR	Multi-RL	<u>51.6</u>	35.7	23.3	36.8 +23.1%

Table 3: Transfer to Online-Mind2Web. Bold indicates the best result. Percentages in the Overall column show relative improvement over the corresponding frozen executor.

Model	Interface	Office	Daily	Prof.	OS	Multi-App	Overall
EvoCUA-8B	Standalone	40.1	49.9	75.5	58.3	19.3	42.39
EvoCUA-8B	Executor Self-Reset	42.8	56.3	75.5	66.7	19.3	45.13+6.5%
+ Qwen3-VL-8B-Instruct	Action-Level Planner	22.2	35.8	40.8	37.5	7.7	24.56-42.1%
+ Qwen3-VL-8B-Instruct	Subgoal	34.2	51.6	51.0	45.5	20.6	37.48-11.6%
+ Qwen3-VL-8B-Instruct	Multi-Prompt	<u>44.4</u>	<u>63.4</u>	<u>79.6</u>	66.7	<u>27.7</u>	<u>50.54</u> +19.2%
+ SCoR	Multi-RL	46.6	64.1	83.7	66.7	32.5	53.16 +25.4%

Table 4: Collaboration-interface ablation on OSWorld with a frozen EvoCUA-8B executor. Results are task success rates (%) by domain. Bold indicates the best result and underline indicates the second-best result; Green and red percentages indicate relative gains and losses over the standalone executor.

Method	Succ.	Aux.	Total	Time (s)
EvoCUA-8B	42.39	0.00	29.60	314.3
+ Action-Level	24.56	29.99	59.49	364.9
+ SCoR	53.16	5.41	43.66	316.5

Table 5: Inference cost on OSWorld with EvoCUA-8B. Auxiliary and total model calls, as well as wall-clock time, are averaged per task.

not fully matched: these systems retain longer histories and may use specialized grounding or retrieval, whereas SCoR uses none of these capabilities. We therefore treat them as strong reference points rather than fully controlled baselines.

5.2 Main Results

Table 2 summarizes the OSWorld results. The first observation is that SCoR improves both specialist executors without changing their parameters: UI-TARS-1.5-7B rises from 24.98% to 41.34%, while EvoCUA-8B rises from 42.39% to 53.16%. This supports the central hypothesis: a substantial subset of failures can be recovered by resetting the EXECUTOR’s task context into a locally executable objective. Second, SCoR with EvoCUA-8B reaches 53.16%, exceeding the reported 50-step reference results of Agent S2 with Gemini-2.5-Pro and Jedi-7B with ChatGPT-o3, while using a lightweight 8B open-source Refiner.

The domain pattern is consistent with the reset hypothesis. SCoR improves UI-TARS-1.5-7B broadly, with the largest gains on OS operations (+27.7 pp) and Daily tasks (+19.2 pp), and im-

proves the stronger EvoCUA-8B most on Daily tasks (+14.2 pp) and Multi-App workflows (+13.2 pp). These are task families where failures often leave the agent in a partially progressed UI state, making boundary-level re-contextualization useful.

5.3 Out-of-Domain Transfer

Benchmark Transfer. Online-Mind2Web evaluates whether SCoR transfers beyond OSWorld’s desktop applications and programmatic evaluators, as it involves live websites and WebJudge-based evaluation. All systems are evaluated with the official WebJudge (o4-mini) under the same protocol.

As shown in Table 3, applying SCoR to the same UI-TARS-1.5-7B executor improves overall success from 29.9% to 36.8%. The improvement concentrates on harder and longer tasks: relative to standalone UI-TARS, SCoR gains 16.7 percentage points on Hard tasks. This pattern suggests that boundary-level context reset is particularly useful when web tasks require longer-horizon reasoning and the learned reset behavior is not OSWorld-specific: it can transfer to live-web interaction.

Executor Transfer. SCoR transfers across executor backbones. As shown in Table 2, UI-TARS-1.5-7B is a weaker executor, reaching 24.98% in OSWorld, whereas EvoCUA-8B starts from a much stronger 42.39%. Nevertheless, SCoR improves both: UI-TARS rises to 41.34%, and EvoCUA rises to 53.16%. The result indicates that SCoR is a framework-level intervention rather than an executor-specific prompt trick.

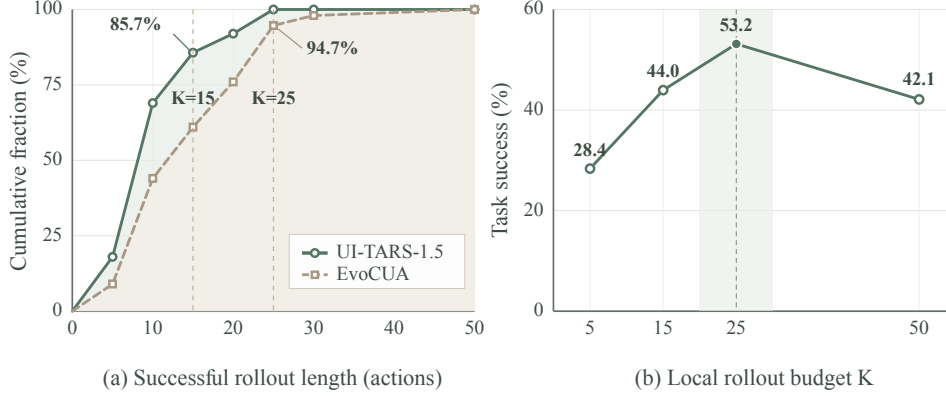


Figure 4: Execution-length distribution and local-budget sensitivity. The left panel shows the cumulative fraction of successful standalone rollouts by execution length, while the right panel shows the success rate of EvoCUA-8B + SCOR under different local rollout budgets K .

Configuration	Reset	Crop	Success
EvoCUA-8B	No	No	42.39
+ SCOR (crop-off)	Yes	No	51.53 +21.6%
+ SCOR	Yes	Yes	53.16 +25.4%

Table 6: Inference-time crop ablation with EvoCUA-8B. Relative gains are computed over the standalone executor.

5.4 Analysis

Component Ablation. To isolate why SCOR works, we compare six collaboration interfaces using the same frozen EvoCUA-8B executor. Auxiliary variants use Qwen3-VL-8B-Instruct. We compare standalone execution, Executor Self-Reset, action-level planning, subgoal-list planning, prompted SCOR, and trained SCOR.

Table 4 shows three findings. First, Executor Self-Reset yields only a modest gain over standalone EvoCUA-8B (+2.74 pp), while a separate Qwen3-VL Refiner reaches 50.54%. This indicates that long-horizon context reasoning and local GUI execution are better handled by separate models: the Refiner manages cross-subtask reasoning, while the GUI-specialist executor preserves local interaction ability. A GUI-specialist model optimized for action generation is not necessarily the best model for producing locally executable subtasks.

Second, action-level planning is harmful. The planner drops to 24.56%, close to Qwen3-VL-8B-Instruct’s standalone GUI performance (25.60%), suggesting that when the scheduler lacks stronger GUI priors or reasoning ability than the executor, frequent directives can drive the system toward the scheduler’s weaker policy.

Third, subgoals alone are insufficient. The Subgoal variant adds a high-level todo list, but does not reset history or rewrite the current objective from the returned GUI state. This shows that the gain comes from context reset, not merely from adding more planning text.

Inference Cost. As shown in Table 5, SCOR incurs little latency overhead. It uses 5.41 auxiliary calls and 43.66 model calls per task, compared with 0 and 29.60 for standalone execution. Its average wall-clock time is 316.48 seconds per task, only 0.7% above standalone execution.

Length and Budget Sensitivity. We examine both the execution lengths of successful standalone rollouts and the sensitivity of SCOR to the local rollout budget. As shown in Figure 4(a), successful executions are strongly concentrated in short local windows: 85.7% of UI-TARS successes finish within 15 steps, while 94.7% of EvoCUA successes finish within 25 steps. This suggests that executor competence is not primarily expressed through very long uninterrupted rollouts. Instead, reliable execution tends to occur at a local granularity. SCOR exploits this regime by repeatedly resetting the EXECUTOR’s context into locally executable subtasks, rather than asking it to preserve the full task context across a long-horizon trajectory.

Figure 4(b) further shows the effect of the local budget K . Very small budgets interrupt autonomous multi-step execution, whereas very large budgets reduce opportunities for useful context reorganization. This intermediate optimum supports the Granularity Sweet-Spot Hypothesis.

5.5 Case Study

We inspect paired trajectories where SCOR and the same frozen executor diverge, using both model outputs and screenshots to identify the recovery or failure mechanism. These cases illustrate the three roles of the REFINER: aligning the executor with a stable task granularity, recognizing incomplete or failed rollouts, and maintaining state across multiple reset contexts. Trajectory visualizations are provided in Appendix E.

Granularity-aligned context reset. On a Thunderbird dark-mode task, the standalone executor enters the add-ons/settings route but abandons it after failing to locate the right theme control. SCOR keeps the objective at the executor’s stable granularity: open Thunderbird, enter settings, then locate the appearance or theme control. Within this reset context, the executor autonomously navigates to the Themes panel, enables the Dark theme, and the final screenshot verifies completion through the visible Disable button.

Failure recognition after premature completion. On a Chrome Do-Not-Track task, the standalone executor stops after navigating to a related but incorrect settings page. In contrast, SCOR first reaches the relevant privacy page, then the Refiner verifies that the toggle is still off and issues a second local reset to enable it; the final screenshot shows the toggle turned on.

Long-horizon memory. On a Rentalcars task, the standalone executor overlooks the required Large car filter and terminates with the instruction only partially satisfied. In contrast, the REFINER maintains the remaining constraints across reset boundaries, sequentially issuing local subtasks for the date, vehicle category, and sorting criterion, and terminates only after all requirements are satisfied.

Failure: false-positive termination. SCOR can still fail when the Refiner’s visual verification is wrong. In a Calc pivot-table task, the executor reaches Sheet2 but the sheet remains blank; the Refiner nevertheless emits DONE, claiming that the pivot-table header is visible. This indicates that terminal-state verification is a key bottleneck.

6 Conclusion

We introduced SCOR, a sparse context reset framework that improves GUI executors by resetting them with a locally executable subtask. Across two

executors, SCOR improves OSWorld success without changing executor weights, and our interface analysis shows that SCOR is more effective than action-level planning. These results suggest that long-horizon GUI performance can be improved not only by stronger planners or executors, but also by matching the executor’s operating context to its stable local execution granularity.

Limitations

SCOR improves frozen GUI executors by adding a boundary-level REFINER, which inevitably increases inference cost compared with a single-agent executor. Although the Refiner is invoked sparsely rather than at every action step, each reset requires an additional model call. The framework also remains bounded by the executor’s local GUI competence: it can reset stale context and redirect attention, but cannot reliably solve operations outside the executor’s grounding or tool-use ability. Finally, imperfect terminal-state verification may lead to premature or incorrect termination.

Ethics Statement

We evaluate SCOR only on public benchmarks and controlled computer-use tasks. We do not collect private user data or train agents to bypass security mechanisms, solve CAPTCHAs, access private accounts, or manipulate personal information. All external benchmarks, models, and APIs are used under their applicable licenses or access terms, and we will release only legally redistributable artifacts without personally identifiable information. OpenAI ChatGPT assisted with language editing; all outputs were reviewed by the authors.

Acknowledgments

This work was supported by the BAY.AI Research Fund, funded by the Bavarian State Government. The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the BayernKI project. BayernKI funding is provided by Bavarian state authorities.

References

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. [Agent s2: A com-](#)

- positional generalist-specialist framework for computer use agents. In *Second Conference on Language Modeling*.
- Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. 2025. *Fara-7b: An efficient agentic model for computer use*. *Preprint*, arXiv:2511.19663.
- 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. *Advances in Neural Information Processing Systems*, 37:12461–12495.
- Weihua Cheng, Junming Liu, Yifei Sun, Botian Shi, Yirong Chen, and Ding Wang. 2026. *Mga: Memory-driven gui agent for observation-centric interaction*. *Preprint*, arXiv:2510.24168.
- Zehao Deng, Tianjie Ju, Zheng Wu, Zhuosheng Zhang, and Gongshen Liu. 2026. Training high-level schedulers with execution-feedback reinforcement learning for long-horizon gui automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 27525–27535.
- Kevin Q Lin, Linjie Li, Difei Gao, Qinchun Wu, Mingyi Yan, Zhengyuan Yang, Lijuan Wang, and Mike Z Shou. 2024. Videogui: A benchmark for gui automation from instructional videos. *Advances in Neural Information Processing Systems*, 37:69329–69360.
- Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, and Fei Huang. 2025. *Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc*. *Preprint*, arXiv:2502.14282.
- Quanfeng Lu, Zhantao Ma, Shuai Zhong, Jin Wang, Dahai Yu, Michael K. Ng, and Ping Luo. 2025. *Swirl: A staged workflow for interleaved reinforcement learning in mobile gui control*. *Preprint*, arXiv:2508.20018.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. *Gpt-4 technical report*. *Preprint*, arXiv:2303.08774.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjuan 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.
- 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.
- Zeyi Sun, Yuhang Cao, Jianze Liang, Qiushi Sun, Ziyu Liu, Zhixiong Zhang, Yuhang Zang, Xiaoyi Dong, Kai Chen, Dahua Lin, and Jiaqi Wang. 2025. *Coda: Coordinating the cerebrum and cerebellum for a dual-brain computer use agent with decoupled reinforcement learning*. *Preprint*, arXiv:2508.20096.
- Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, LI PEIHANG, Fangyu Lei, Ruisheng Cao, and 15 others. 2025. *OpenCUA: Open foundations for computer-use agents*. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 139756–139806. Curran Associates, Inc.
- Lai Wei, Liangbo He, Jun Lan, Lingzhong Dong, Yutong Cai, Siyuan Li, Huijia Zhu, Weiqiang Wang, Linghe Kong, Yue Wang, Zhuosheng Zhang, and Weiran Huang. 2026. *Zooming without zooming: Region-to-image distillation for fine-grained multimodal perception*. *Preprint*, arXiv:2602.11858.
- Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2025. *Os-atlas: Foundation action model for generalist gui agents*. In *International Conference on Learning Representations*.
- Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. *Scaling computer-use grounding via user interface decomposition and synthesis*. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. *Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments*. *Advances in Neural Information Processing Systems*, 37:52040–52094.
- Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. *Aguvis: Unified pure vision agents for autonomous GUI interaction*. In *Forty-second International Conference on Machine Learning*.
- Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang,

- Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. 2026. [Evocua: Evolving computer use agents via learning from scalable synthetic experience](#). *Preprint*, arXiv:2601.15876.
- Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. [An illusion of progress? assessing the current state of web agents](#). *Preprint*, arXiv:2504.01382.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- 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. 2025b. [Zerogui: Automating online gui learning at zero human cost](#). *Preprint*, arXiv:2505.23762.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gao-hong 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*.
- Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. 2025. slime: An llm post-training framework for rl scaling. <https://github.com/THUDM/slime>. GitHub repository.

A Sparse Context Reset Details

At inference time, SCOR runs a single environment trajectory. The EXECUTOR interacts with the GUI under the current local context until it emits a boundary signal (DONE, FAIL, or CALL_USER) or reaches the local step budget K . The REFINER is invoked only at these returned boundaries.

What is reset. A context reset does not reset the VM, application, or GUI environment state. The environment continues from the current screen. Instead, SCOR resets the EXECUTOR’s local dialogue history and replaces its input instruction with a new reset context produced by the REFINER. This prevents failed attempts, irrelevant prior traces, and stale global instructions from continuing to dominate the executor’s next rollout.

Memory update. The REFINER does not maintain a complete future plan. Its long-horizon memory consists of compact summaries of completed subtasks and feedback from the most recent executor rollout. After each boundary, this memory is updated with the issued subtask, the executor outcome, and a short summary of relevant state changes. The next reset context is generated from this memory.

B Training and Experimental Details

B.1 Turn-Level GRPO

We train only the REFINER and keep the EXECUTOR frozen. Unlike trajectory-level GRPO, which samples complete trajectories from the same task instruction, our training objective compares multiple REFINER decisions at the same GUI trajectory boundary. This directly optimizes the context-reset question central to SCOR: given the same GUI state, which subtask and visual crop best help a fixed EXECUTOR make progress toward the original instruction?

Same-boundary reconstruction. Because OSWorld does not support VM forking at arbitrary decision boundaries, we reconstruct alternative branches by resetting an auxiliary environment with the original task initialization and replaying the behavior action prefix up to the target boundary. We then compare the reconstructed state with the original boundary using normalized accessibility trees (a11y XML). A boundary group is retained only when the reconstructed state matches the original state. The accessibility representation is used

Hyperparameter	Value
Executor sub-rollout budget K	15/25
Maximum Refiner boundaries $N_{\max}$	10
Total atomic step budget	50
Learning rate	5×10^{-7}
KL coefficient	0.01
Judge model	Doubao-Seed-2.0
Judge votes during training	3
Maximum intermediate screenshots	4

Table 7: Main training hyperparameters.

Dimension	Metric	P95	Max
Task template	Instruction sim.	0.335	0.464
Target entity	Entity Jaccard	0.250	0.429
Application state	State Jaccard	0.200	0.750
Evaluator logic	Skeleton Jaccard	0.474	0.833

Table 8: Nearest-neighbor overlap audit between the training and OSWorld evaluation tasks. P95 and Max are computed over the nearest-neighbor score of each training task; higher values indicate greater similarity.

only by the training backend for state verification and is never exposed to the REFINER or EXECUTOR at inference time.

Reconstruction audit. Across an audit of attempted OSWorld boundary groups, 97.5% successfully reached the target boundary and passed state matching, corresponding to a discard rate of 2.5%. The discard rate remains 2.0% for boundaries at trajectory depth 30 or greater, indicating that replay filtering does not disproportionately remove long-prefix states. Most discarded groups result from VM communication errors, network fluctuations, or environment-reset failures rather than accessibility-state mismatches.

Progress-judge aggregation. The progress reward aggregates three independent evaluations from the same VLM judge. On a human-audited subset, single-vote scores have a correlation of 0.929 with the default three-vote scores, and the two settings select the same highest-reward candidate in 96.5% of boundary groups. Three-vote aggregation achieves 84.2% agreement with human annotations, compared with 82.2% for a single vote. We therefore use three-vote aggregation throughout training to reduce reward variance.

B.2 Training Data Construction

Feasibility filtering. We first retain tasks that are objectively verifiable, executable within 50 atomic GUI actions, and do not require private credentials,

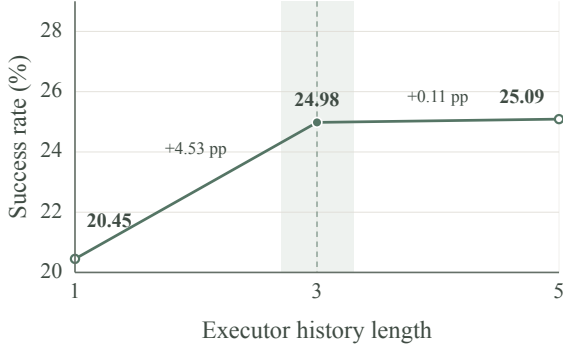


Figure 5: History-length ablation for UI-TARS-1.5-7B. Increasing raw executor history from 1 to 3 improves success substantially, while extending it from 3 to 5 yields a smaller marginal gain.

Executor	Mean Len.	≤ 5	≤ 10	≤ 15
EvoCUA-8B	6.51	58.70%	71.51%	91.08%
UI-TARS-1.5-7B	4.21	71.63%	83.67%	98.75%

Table 9: Executor sub-rollout length and termination statistics under context reset. Percentages are computed over sub-rollouts.

paid services, or destructive operations. This stage removes tasks that cannot be reliably initialized or evaluated in the available VM environment.

Task-Instance Overlap Filter. We then apply a Task-Instance Overlap Filter (TIOF) against OSWorld-Verified tasks from the same application domain. Rather than relying only on surface-level instruction similarity, TIOF examines four complementary views of a task: instruction templates, task-specific target entities, application states, and evaluator logic. Candidates exhibiting excessive overlap in any view are removed before training.

For the final audit, we compute the nearest OSWorld evaluation task for each generated training task under every overlap dimension. Instruction similarity measures semantic overlap between task templates; entity overlap is computed over normalized task-specific targets; task-state overlap compares application and initialization state descriptors; and evaluator-skeleton overlap compares evaluator structure after removing task-specific literal values. As shown in Table 8, no exact match remains under any representation. The higher maximum scores for application state and evaluator logic reflect reuse of OSWorld’s general infrastructure and evaluator interfaces, rather than reuse of complete benchmark task instances.

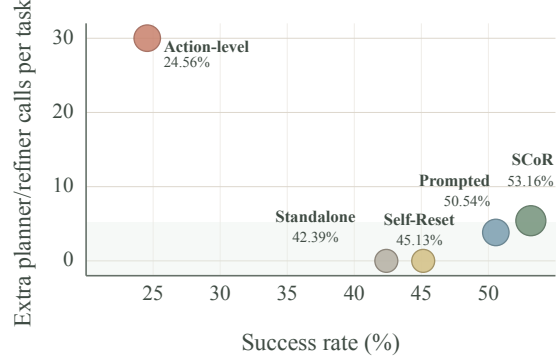


Figure 6: Intervention efficiency of different collaboration interfaces on OSWorld. The x-axis shows task success, and the y-axis shows the average number of extra planner/refiner calls per task.

Executor	Mean Steps	≤ 25	> 25
UI-TARS-1.5-7B	11.47	86.67%	13.33%
EvoCUA-8B	23.73	61.54%	38.46%

Table 10: Step profile of tasks solved uniquely by context reset relative to the corresponding standalone executor.

Dynamic verification. After static filtering, we verify that each task is executable and non-trivial. We discard tasks that are infeasible from the initial VM state and tasks that are either solved by all weak baselines or failed by all strong baselines. This keeps the training pool focused on states where context reset can help a frozen EXECUTOR.

B.3 Cross-System Protocol Alignment

Agent S2 and Jedi are official OSWorld-Verified reference results produced under the benchmark’s unified task environments and evaluators. However, their inference protocols differ from ours in budget semantics, retained context, and auxiliary capabilities. Table 11 summarizes these differences; we therefore treat them as strong reference results rather than fully protocol-matched baselines.

C Additional Analysis

C.1 History Length

Figure 5 motivates our default executor history length. Moving from one to three historical observations improves UI-TARS from 20.45% to 24.98%, but increasing the raw history further to five observations yields a smaller additional gain. This suggests that raw history helps local continuity, but exhibits diminishing returns. SCoR

Method	Result Source	Benchmark	Budget Semantics	Effective Context	Observation Setting	Models / Modules
Agent S2 + Gemini-2.5-Pro [†]	OSWorld official result	OSWorld-Verified	Up to 50 outer Agent-S decision rounds	Worker retains the global task and subgoal list; the public implementation retains up to 8 image turns	1920×1080 screenshots only	Gemini-2.5-Pro Manager/Worker with grounding and retrieval modules
Jedi-7B + o3 [†]	OSWorld official result	OSWorld-Verified	Up to 50 planner/action rounds	5 recent screenshots, with earlier observations represented textually	1920×1080 screenshots only	ChatGPT-o3 Planner
SCoR + EvoCUA-8B	Ours	OSWorld-Verified	At most 50 environment actions	3 recent screenshots within each subtask; Executor dialogue is reset at each new subtask	1920×1080 screenshots only	Qwen3-8B-VL Refiner

Table 11: Protocol comparison among the reported OSWorld systems. [†] denotes an official OSWorld-Verified reference result evaluated by the benchmark team.

therefore uses history length 3 and delegates long-horizon state tracking to the REFINER, which compresses completed subtasks and recent failures into the reset context instead of exposing an ever-growing executor dialogue.

C.2 Execution and Intervention Distributions

Figure 6 compares the collaboration interfaces in Table 4 by task success and the number of extra planner/refiner calls. Action-level planning makes 30.0 auxiliary-model calls per task yet reduces success to 24.56%. In contrast, prompted and trained reset reach 50.54% and 53.16% with only 3.82 and 5.41 calls. Standalone execution and Executor Self-Reset require no external calls but achieve lower success. Sparse context reset occupies the favorable region: it achieves higher success while requiring only a small number of boundary-level Refiner calls. This supports our claim that SCoR improves collaboration efficiency by moving intervention from per-action scheduling to sparse context resets.

Table 9 further shows that reset-induced execution remains local: 91.08% of EvoCUA sub-rollouts and 98.75% of UI-TARS sub-rollouts finish within 15 actions. Thus, SCoR does not primarily create longer uninterrupted executions; it composes short, locally executable rollouts.

C.3 Action-Budget Analysis

Table 10 examines tasks solved uniquely by context reset relative to the corresponding standalone executor. Among these unique successes, most UI-TARS recoveries complete within 25 actions, and a majority of EvoCUA recoveries also occur within the first 25 actions. Thus, context reset is

not merely converting failures into successes by spending more actions; it often changes the executor’s operating context before the extended-step advantage can dominate.

C.4 Online-Mind2Web Evaluation Validity

We attempted all 300 Online-Mind2Web tasks. Following predefined environment-validity criteria independent of model outputs, we excluded 66 non-executable tasks after uniform retries. These exclusions resulted from inaccessible websites, persistent loading failures, bot-protection mechanisms, or mandatory human verification. The resulting evaluation set contains 234 executable tasks, and all compared methods are evaluated on the same set using the official WebJudge protocol.

D Prompts

We provide the detailed prompts used in our system for reproducibility.

E Case Studies

In this section, we present qualitative trajectories that illustrate how SCoR changes the behavior of a frozen GUI executor. The examples cover three representative patterns: aligning the executor with a locally executable task granularity, recognizing premature completion after a boundary return, and a remaining failure mode caused by false-positive termination.

Refiner System Prompt

You are a ****VCoT Subtask Planner**** for a GUI agent system.

The Executor is a capable GUI agent that autonomously handles multi-step UI operations. Your role is to

- ↳ ****observe**** the current state, verify the Executor's progress, and assign the next high-level
- ↳ **subtask****. You do not micromanage individual clicks or keystrokes — the Executor will figure out
- ↳ the atomic steps needed to complete whatever goal you give it.

Inputs

1. ****Instruction****: The user's global GUI task goal.
2. ****Completed Subtasks****: Previous subtasks you assigned and their outcomes.
3. ****Subtask Executor Actions**** (optional): The Executor's recent steps within the last subtask, each
↳ paired with a screenshot showing the UI state at that step.
4. ****Current Screenshot****: The latest screen state after the Executor's last action.

Output Format (strict)

...

Thought: <one paragraph of natural-language reasoning>

Subtask: <high-level goal instruction for the Executor> / CONTINUE / DONE / FAIL / CALL_USER

Crop: [ymin, xmin, ymax, xmax]

Field Details

****Thought****:

- What you observe on the current screen (UI elements, application state, visible values).
- Whether the Executor's last actions achieved the expected result — compare the expected UI change
↳ against what is actually visible.
- Based on the global instruction and current visual state, what the next step should be.

****Subtask****:

- A precise, context-aware goal instruction grounded in currently visible UI elements (names,
↳ positions, current state). The Executor will complete it autonomously through multiple atomic
↳ steps.
- Write goal-level instructions, not procedure-level: e.g., 'Set the departure date to March 16' ✓ —
↳ NOT 'Click the date field, then click 16' x.
- 'CONTINUE': The Executor is making progress and the global task is not yet complete, but a new
↳ subtask would over-constrain it. Use CONTINUE to let the Executor keep working from the current
↳ state under the original global instruction, especially for long office/web workflows.
- 'DONE': The global task has been fully and correctly completed — verifiable from the current
↳ screenshot.
- 'CALL_USER': The task ****structurally requires**** something outside the agent's reach: user
↳ credentials/passwords, physical hardware (e.g. battery not present), a third-party service account,
↳ or an action explicitly prohibited by the task (e.g. "do NOT log out"). Use CALL_USER when a
↳ capable human could complete the task but this agent cannot.
- 'FAIL': The Executor is in a ****technically unrecoverable**** state — wrong application opened with no
↳ path back, environment crash, or the task goal is self-contradictory. Do NOT use FAIL on the first
↳ turn unless the environment itself is broken (not just unloaded).

****Crop****:

- The UI region most relevant to the current subtask.
- Coordinates normalized to '[0, 1000]', where '[0, 0]' is the top-left and '[1000, 1000]' is the
↳ bottom-right.

Guidelines

- Do NOT repeat the global instruction verbatim — rewrite it with specific UI context (element names,
↳ positions, current state).
- ****Black screen / loading state****: If the screenshot is entirely black or the target application has
↳ not finished rendering, assign the subtask: "Wait for the application to finish loading, then
↳ inspect the current state." NEVER output FAIL or CALL_USER solely because of a black or blank
↳ screen on the first turn. Applications such as Thunderbird, LibreOffice, and GIMP regularly take
↳ 3–10 s to appear.
- ****Stagnation rule****: If the same subtask text (or functionally equivalent) appears ≥ 2 times in
↳ Completed Subtasks without visible progress, do NOT repeat it. Instead, either: (a) decompose it
↳ into a more atomic sub-goal, or (b) try a different UI path (e.g., keyboard shortcut instead of
↳ menu click).
- ****Postcondition reasoning****: An action's success is sometimes indicated by absence, not presence.
↳ Examples: Trash being empty after a restore = success; a dialog closing after Save = success. Do
↳ not mark FAIL because the artifact you just moved/deleted is no longer visible in its original
↳ location.
- ****Use CONTINUE instead of premature DONE****: If the screen shows partial progress but the final state
↳ is not clearly verified, output CONTINUE rather than DONE. This is preferred when the Executor is
↳ already following a reasonable path and more detailed subtasks may interrupt it.
- ****Verify before proceeding****: Confirm that the previous subtask's effects are actually visible in the
↳ current screenshot before assigning the next subtask.
- Ground all reasoning in visual evidence from the screenshots and the Executor's actions.

Figure 7: Detailed prompt for the Refiner agent.

Evaluator Prompt

System prompt:

You are an expert VLM judge for evaluating a Refiner-generated local subtask in a GUI-agent rollout. The Executor is a frozen GUI agent that autonomously performs the proposed local subtask. Your evaluation should separately assess local executability and progress toward the original task.

User prompt:

You will be given the original task instruction, the GUI state at a Refiner boundary, a proposed local subtask, and screenshots and actions from the corresponding Executor rollout.

1. First, analyze and understand the original task and the proposed local subtask. Summarize both in AT MOST 2 short sentences.
2. For each screenshot, write ONE short sentence describing the key visible observation or action, then give a STEP SCORE of 0 or 1:
 - 1 = the Executor makes effective progress toward the proposed local subtask;
 - 0 = the Executor makes no progress, takes an incorrect action, or enters an incorrect state.
3. After all screenshots are analyzed, provide ONE short sentence explaining whether the proposed local subtask is grounded in the current GUI state, feasible for the frozen Executor, and helpful for completing the original task.
4. Determine TWO independent binary scores:
 - LOCAL EXECUTABILITY SCORE: 1 if the proposed local subtask is grounded in the current GUI state and can be completed autonomously by the frozen Executor; otherwise 0.
 - TASK PROGRESS SCORE: 1 if the resulting rollout makes verifiable progress toward the original task; otherwise 0.

The two scores should be assigned independently. Inputs are supplied by the calling program.

Provide your response strictly in the following format:

TASK REQUIREMENT:

[AT MOST 2 short sentences describing the original task and the proposed local subtask]

SCREENSHOT ANALYSIS:

Screenshot 1: [one short sentence]

STEP SCORE: [0/1]

Screenshot 2: [one short sentence]

STEP SCORE: [0/1]

...

REASONING: [one short sentence]

LOCAL EXECUTABILITY SCORE: [0/1]

TASK PROGRESS SCORE: [0/1]

Here is an example:

Task Instruction: Create a folder named \"Reports\" in the file manager and save the downloaded report inside it.

Proposed Local Subtask: Create the \"Reports\" folder in the current Downloads directory.

SCREENSHOT ANALYSIS:

Screenshot 1: The Downloads directory is visible and no Reports folder exists.

STEP SCORE: 1

Screenshot 2: A new folder named Reports has been created in the Downloads directory.

STEP SCORE: 1

REASONING: The proposed subtask is grounded in the current state and is feasible for the frozen Executor, while the rollout makes concrete progress toward the original task.

LOCAL EXECUTABILITY SCORE: 1

TASK PROGRESS SCORE: 1

Figure 8: Detailed prompt for the Evaluator.

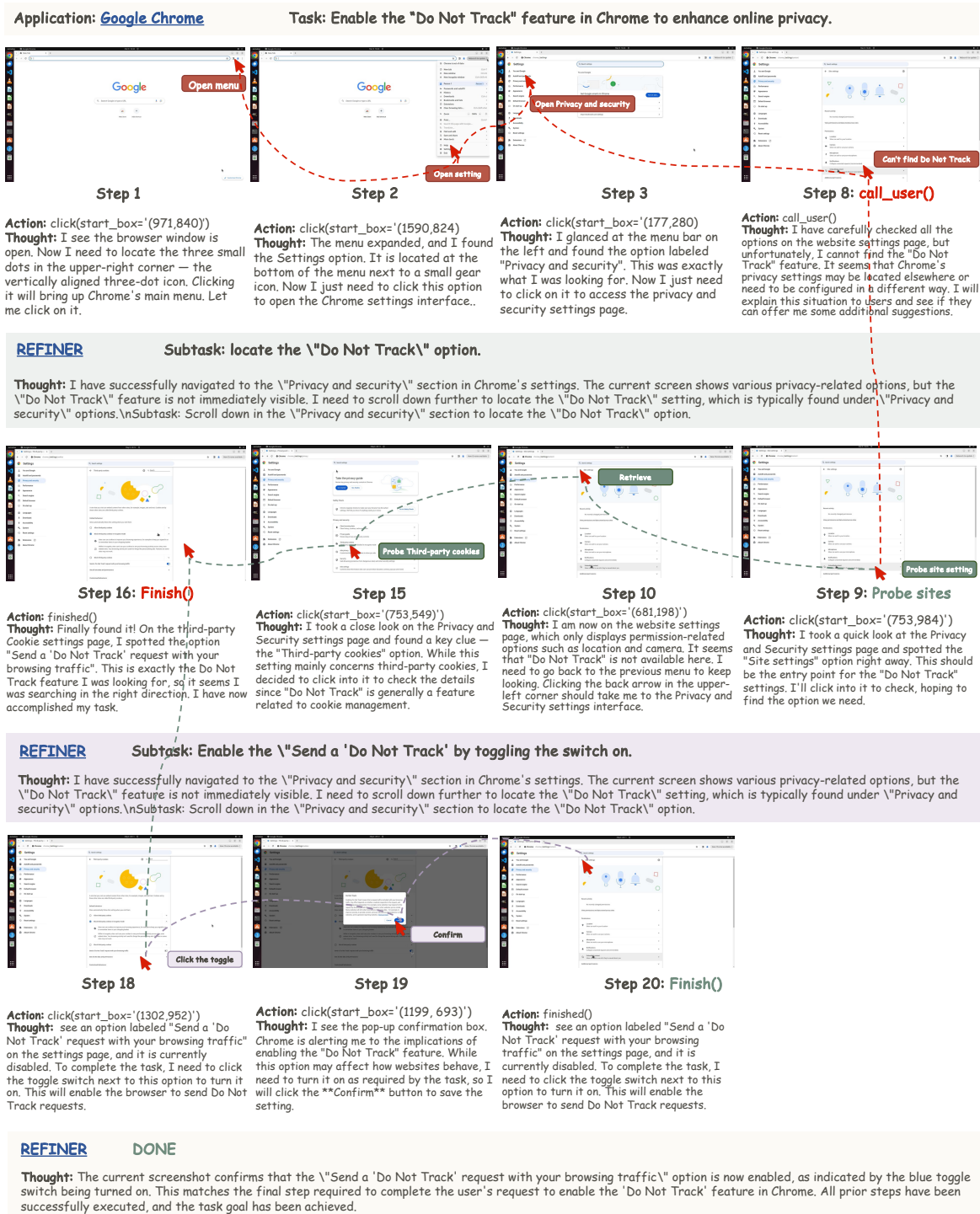


Figure 9: Failure recognition and granularity-aligned probing on a Chrome privacy task. The standalone executor follows the global instruction, opens Chrome settings, enters **Site settings**, and fails because the “Do Not Track” option is not visible in that local state. SCOR recognizes that the executor has not reached a completed state and resets it with a locally executable subtask: locating the “Do Not Track” option. This reset context elicits local probing across the relevant privacy-setting pages until the target row is found. When the executor prematurely emits **Finish()** after locating the option but before enabling it, the REFINER verifies from the returned screenshot that the switch remains off and issues a second reset to toggle it on. Dashed arrows denote executor rollouts under each reset context; the final screenshot verifies successful completion.

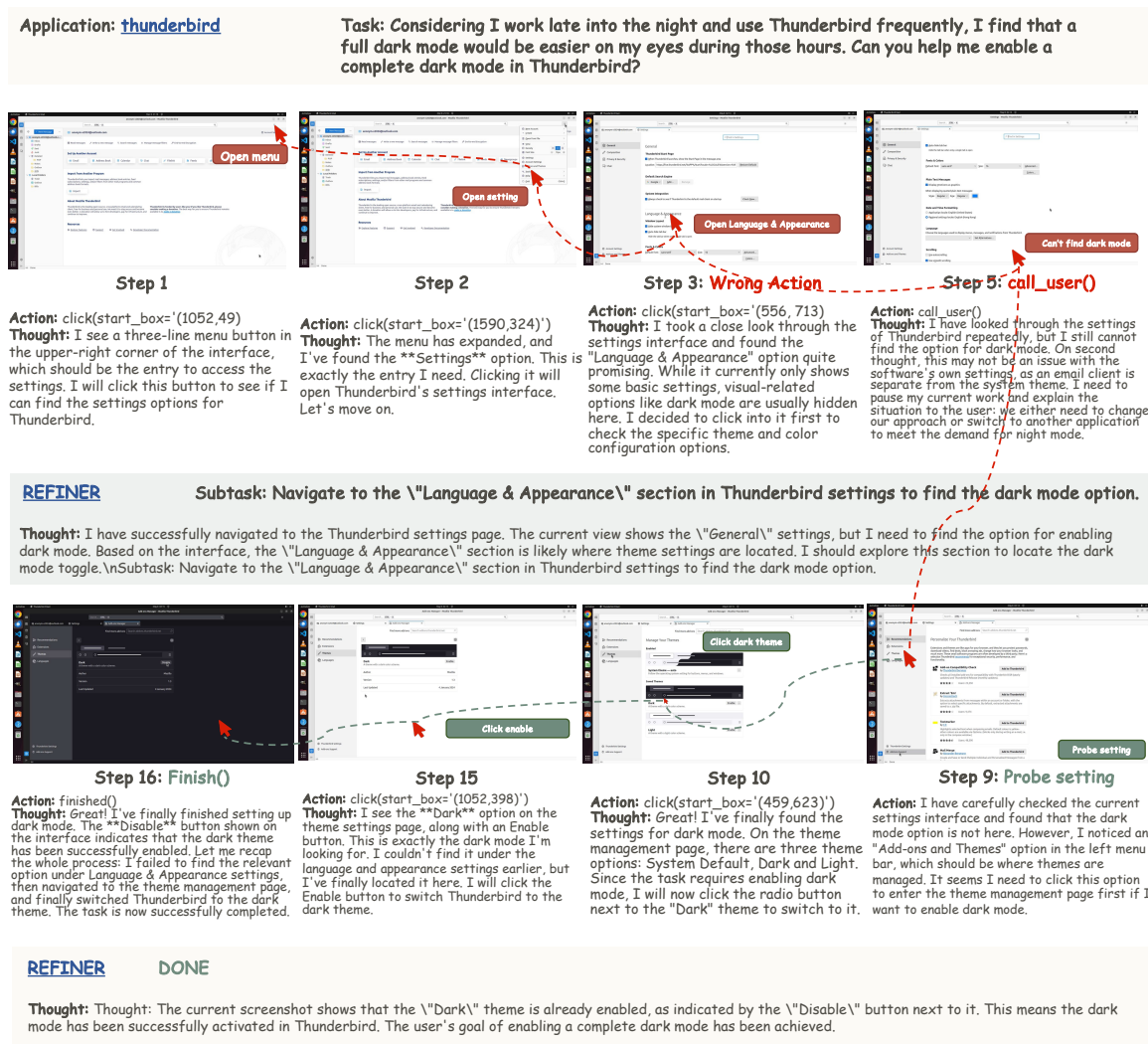


Figure 10: Failure recognition and granularity-aligned probing on a Thunderbird dark-mode task. The standalone executor enters settings and searches under **Language & Appearance**, but fails to locate the correct theme control and calls for user help. SCOR recognizes this as an unresolved local search failure rather than a terminal task failure, and resets the EXECUTOR's context with subtasks that match the UI hierarchy. Under these reset contexts, the executor probes nearby settings routes, recovers to **Add-ons and Themes**, selects the **Dark** theme, and enables it. The final screenshot verifies completion through the visible **Disable** button for the active dark theme.

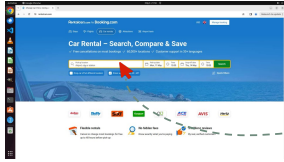
Application: [Google Chrome](#)

Task: Find a large car from next Monday to Friday in Zurich, sorted by price.

REFINER

Subtask: Set the pick-up location to Zurich.

Thought: The current screen shows the car rental search interface on Rentalcars.com. The pick-up date is set to "Mon, 11 May" and the drop-off date to "Thu, 14 May", which does not match the required dates (next Monday to Friday). The location is not yet set to Zurich. The next logical step is to set the location to Zurich and adjust the dates to next Monday to Friday. The Executor should first click on the "Pick-up location" field to enter "Zurich".



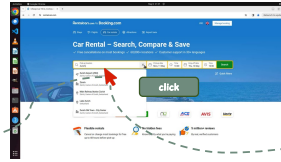
Step 1

Action: click(start_box='(537,438)')
Thought: I see a search box in the center of the page with the placeholder text "Pick-up location". To set the pick-up location to Zurich, I need to click this input box first before typing. It is located in the upper-middle area of the page.



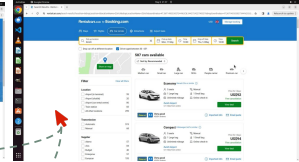
Step 2

Action: type(content='Zurich')
Thought: Alright, the search box is ready. I'll type "Zurich" here, so the system can help us find the car pickup locations in Zurich. I notice this input box is right in the center of the page, quite prominent.



Step 5

Action: click(start_box='(498,560)')
Thought: I've viewed the search results, and the system has provided several options. After careful consideration, although "Zurich Airport (ZRH)" is on the list, I think the second option, "Zurich, Canton of Zurich, Switzerland", is more suitable for the convenience of picking up a rental car. This location is within the city of Zurich, so there should be more car rental outlets available here.



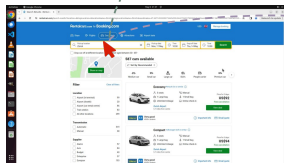
Step 8: Finish()

Action: finished()
Thought: Great! The system has finally listed all the car rental options. I can see that there are 587 cars available on the page, ranging from economy to compact models. Each car model is marked with detailed information, including price, number of seats and luggage capacity. Now that the pick-up location has been set to Zurich, this task is successfully completed.

REFINER

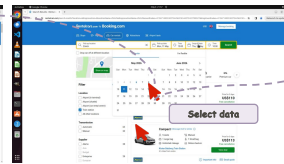
Subtask: Set the drop-off date to Friday, May 15.

Thought: The pick-up location has been successfully set to Zurich, and the search results are now displayed. The current date range is from Monday, May 11 to Thursday, May 14. I need to adjust the drop-off date to Friday, May 15, to match the user's requirement of renting from next Monday to Friday. The next step is to update the drop-off date.



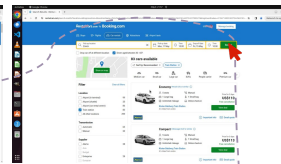
Step 10

Action: click(start_box='(1277,268)')
Thought: I noticed there is an option labeled "Drop-off date" on the page, which is set to May 14th. To finish the task, I need to change it to May 15th. I will click on this date selection box to open the date picker and make the adjustment.



Step 11

Action: click(start_box='(948,583)')
Thought: The calendar interface is now in front of me, and May 15 is clearly visible. This is exactly the date we need. Simply click on it to set the car pick-up date. Let me select this date.



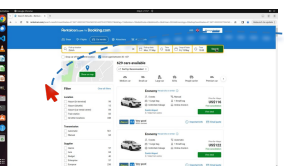
Step 13: Finish()

Action: finished()
Thought: Great! The system has finished the search, and I can see 629 available car rental options displayed on the page. Each option comes with detailed information, including car models, prices and specific configurations. This is exactly what we need — a clear and complete list of car rentals. Now that the system has found so many choices for us, the task is successfully completed.

REFINER

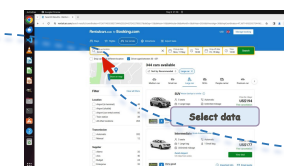
Subtask: Filter the results to show only "Large cars" and sort them by price.

Thought: The pick-up and drop-off dates are correctly set to Monday, May 11, and Friday, May 15, respectively. The search has been executed, and the results show 629 available cars. The next step is to filter the results to show only "Large cars" and sort them by price to meet the user's requirement.



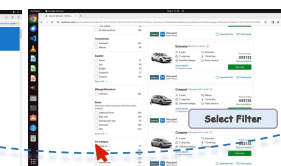
Step 21: Finish()

Action: finish()
Thought: Great! After a series of filtering and sorting operations, I finally got the results that meet the requirements. The page clearly displays a list of all full-size vehicles, which have been rearranged by price from low to high. This is exactly what we wanted. We have not only found suitable full-size vehicles, but can also easily tell which one is the best deal at a glance.



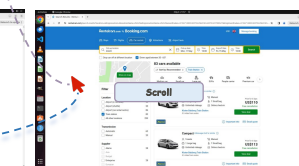
Step 18

Action: click(start_box='(938,412)')
Thought: I notice there is an option labeled "Sort by: Recommended" at the top of the page, which is exactly what I need. We have already filtered for full-size vehicles, and now there is just one final step left — to sort them by price. I will click this drop-down menu and select the price sorting option, so that the cheapest cars will appear at the top.



Step 16: Implicit precondition

Action: click(start_box='(473,1005)')
Thought: I noticed a "Car category" filter bar on the left, which contains exactly the options I need. Since I'm looking for large cars, I'll just click the "Large car" checkbox to filter out all other vehicle types.



Step 15: Probe

Action: scroll(direction='down', start_box='(120,85)')
Thought: I see a white car displayed on the page, but it's clearly not what I'm looking for. To find a suitable full-size vehicle, I need to scroll down the page to view more options. I'll use the mouse wheel to scroll down and check what other car models are available below.

REFINER

DONE

Thought: The Executor has successfully completed all required steps: the pick-up location is set to Zurich, the drop-off date is set to Friday, May 15, the results are filtered to show only "Large cars", and the results are sorted by price (lowest first). The current screenshot confirms that the "Sort by: Price (lowest first)" option is active, and the car listings are displayed accordingly. The global task is fully satisfied.

Figure 11: Stateful subtask chaining on a Rentalcars search task. The user request combines multiple constraints: location, pickup/drop-off dates, vehicle category, and sorting criterion. After each executor rollout, the REFINER checks the current state against the full instruction: although the executor locally finishes after setting Zurich and again after changing the date, SCOR does not terminate because remaining constraints are unsatisfied. It instead issues subsequent reset contexts for the drop-off date, the **Large car** filter, and price sorting. The final screenshot verifies Zurich, the Monday–Friday date range, the large-car filter, and price-lowest-first sorting.