EASE: EFFICIENT MOBILE AGENTS WITH SELF-MAINTAINED STATE AND EXECUTOR-CALIBRATED PLANNING FOR LONG-HORIZON TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graphical user interface (GUI) agents have achieved strong performance by integrating perception, reasoning, and action generation within a single model, yet still struggle with long-horizon and complex tasks. Hierarchical systems address these challenges through specialized modules for planning, memory, and execution, but incur substantial inference overhead. This creates a fundamental trade-off between efficient end-to-end execution and higher-overhead hierarchical assistance. We address this trade-off with EASE-Agent, a mobile GUI agent that preserves lightweight execution while leveraging sparse strong-model assistance to improve long-horizon task performance. To realize this design, we introduce two complementary mechanisms. First, we equip the executor EASE-UI with Self-Maintained State and introduce Token-level Advantage Policy Optimization (TAPO) to reduce cross-objective interference during joint training. Second, we introduce executor-calibrated Skill Memory for task decomposition, enabling the planner to adapt subgoal granularity to the executor's capabilities. Extensive experiments demonstrate the effectiveness and efficiency of our approach. On GUI-Odyssey, EASE-UI achieves the best performance among the compared methods, with TAPO outperforming GRPO and GDPO. On MemGUI-Bench, EASE-Agent achieves 31.3% pass@3, while stronger configurations further improve performance: EASE-Agent-Pro reaches 50.0%, and EASE-Agent-Max reaches 75.0%, compared with 49.2% for Agent-S2. EASE-Agent-Max incurs only 11.1% of Agent-S2's API cost per attempt.

1 INTRODUCTION

Graphical user interface (GUI) agents can follow natural-language instructions to operate software interfaces (Wu et al., 2025). Recent end-to-end GUI agents have achieved strong performance by integrating perception, reasoning, and action generation within a single model. However, reliable execution remains challenging for long-horizon and complex tasks, which often involve multi-step workflows, cross-application interactions, and long-range information dependencies. Hierarchical systems address these challenges by introducing specialized modules for planning, memory, reflection, and execution (Agashe et al., 2025a; Yi et al., 2026). This creates a fundamental trade-off: end-to-end agents are efficient but remain constrained on long-horizon execution, whereas hierarchical agents extend long-horizon capability through specialized assistance, but at substantially greater inference overhead.

This trade-off raises a question: **Can we retain the efficiency of end-to-end execution while selectively leveraging hierarchical assistance to excel at long-horizon tasks?** Achieving this goal exposes two coupled bottlenecks. From an execution perspective, long-horizon interaction demands organizing an ever-growing history, yet a lightweight end-to-end executor often struggles to maintain task-relevant state without extra memory modules. This, in turn, places demands on planning: from a planning perspective, decomposition must be calibrated to what the executor can reliably complete—overly coarse subgoals overwhelm it, while overly fine ones trigger excessive planner intervention.

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

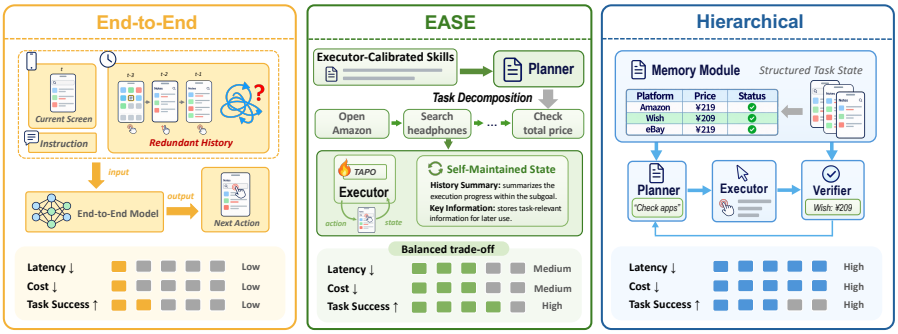


Figure 1: Comparison of end-to-end, EASE, and hierarchical GUI-agent architectures for long-horizon tasks. End-to-end agents are efficient but struggle with long histories, while hierarchical systems incur extra model and module overhead. EASE balances task performance and efficiency through self-maintained execution and sparse planner assistance.

To resolve this trade-off, we propose an efficient mobile GUI agent framework consisting of a lightweight executor for frequent environment interaction and a strong planner for task decomposition and subgoal completion verification. In this way, the framework preserves efficient end-to-end interaction while introducing strong-model assistance only at the higher task level. Specifically, we make two complementary improvements:

First, we internalize state maintenance into the executor. At each interaction step, it generates a GUI action and simultaneously updates a compact history summary within a single model invocation, retaining task-relevant information for subsequent execution. To jointly optimize these two capabilities, we introduce Token-level Advantage Policy Optimization (TAPO), which computes objective-specific advantages and assigns them only to the corresponding token regions. Using TAPO, we train EASE-UI, a 2B-parameter executor that jointly performs GUI interaction and maintains execution state.

Second, we introduce Skill Memory to provide the planner with executor-calibrated guidance for task decomposition. We construct Skill Memory by having the executor perform the same set of tasks under plans of different granularities and distilling the resulting trajectories into reusable planning skills. This process automatically derives executor-specific planning guidance from observed behavior, rather than requiring manually crafted capability descriptions for each executor.

Building on these two designs, we develop EASE-Agent for efficient long-horizon mobile GUI tasks. On GUI-Odyssey (Lu et al., 2025a), EASE-UI achieves 64.9% SR, surpassing GRPO and GDPO by 1.9 percentage points. On MemGUI-Bench (Liu et al., 2026b), EASE-Agent with EASE-UI achieves 31.3% pass@3, while EASE-Agent-Max, using a stronger model configuration, reaches 75.0%, outperforming Agent-S2 by 25.8 points at 88.9% lower cost per attempt. Ablations further show that multi-granularity skill memory raises EASE-Agent's pass@3 from 19.5% to 31.3%, highlighting the benefit of matching task decomposition to executor capabilities. Our contributions are threefold:

- We introduce EASE-Agent, a hierarchical mobile GUI agent combining a lightweight executor with a strong planner for efficient long-horizon task execution.
- We propose TAPO for jointly learning GUI actions and self-maintained state, alongside automated skill construction to calibrate task decomposition to executor capabilities.
- Extensive experiments demonstrate that EASE-Agent improves long-horizon task performance while substantially reducing inference overhead and monetary cost.

2 Related Work

2.1 Long-Horizon GUI Agents

Recent end-to-end GUI agents, including UI-TARS, GUI-Rise, and UI-S1, perform perception, reasoning, and action generation within a unified model (Qin et al., 2025; Liu et al., 2025; Lu et al., 2025b). Their simple execution pipeline avoids the coordination overhead of multi-component systems, but reliable performance remains difficult on long-horizon tasks. Hierarchical agents instead distribute responsibilities across specialized modules. Agent S2 combines generalist and specialist models and refines plans at multiple temporal scales (Agashe et al., 2025b), while EcoAgent separates cloud-based planning from lightweight device-side observation and execution (Yi et al., 2026). These architectures improve support for complex tasks, but require additional planning, communication, or model invocations during execution. Overall, achieving both reliable long-horizon execution and low inference overhead remains an important challenge for GUI agents. Our EASE-Agent targets this trade-off by retaining lightweight execution while invoking a strong planner only when high-level assistance is needed.

2.2 Memory Maintenance for GUI Agents

Long-horizon interaction requires preserving task-relevant history. MemGUI-Agent learns context management through supervised fine-tuning (Liu et al., 2026a), while MementoGUI trains a separate memory controller with supervised and preference optimization (Zeng et al., 2026). STAMP combines supervised training with online reinforcement learning in synthetic environments (Wang et al., 2026). Reinforcement learning also improves history summarization in GUI-Rise (Liu et al., 2025) and conversational memory in Memory-R1, which uses PPO and GRPO (Yan et al., 2026). GUI-Rise combines multiple rewards into a shared GRPO advantage, leaving action and summary tokens without objective-specific credit assignment. Our EASE-UI addresses this issue by jointly learning GUI actions and self-maintained state with TAPO.

2.3 Skills and Experience for GUI Agents

GUI agents can externalize reusable knowledge acquired from demonstrations, exploration, or prior task execution. AppAgent explores applications or observes demonstrations to construct reference documents about interface usage (Zhang et al., 2025). Mobile-Agent-E distills prior experience into general tips and executable shortcuts (Wang et al., 2025), while CUA-Skill represents computer-use knowledge as parameterized skills with execution and composition graphs (Chen et al., 2026). These approaches primarily capture application knowledge, procedural experience, or reusable operations that inform what should be done and how to perform it. Comparatively less attention has been paid to experience representations that explicitly capture differences in agent capabilities. Our skill memory instead captures how task decomposition should adapt to the executor's capabilities by distilling comparisons across multi-granularity execution trajectories.

3 Method

3.1 EASE-Agent Overview

EASE-Agent adopts a hierarchical planner–executor system. The planner π_P is a strong general-purpose model responsible for task decomposition, subgoal verification, and replanning. The executor π_E is a lightweight GUI model, performing frequent interactions with the environment. The system incorporates two complementary mechanisms: skill memory M_{skill} for executor-calibrated task decomposition and state memory M_{state} maintained by the executor during interaction.

Before execution begins, the planner generates an initial plan P of N subgoals based on the user instruction and skill memory:

$$P = \pi_P(I, M_{\text{skill}}) = [g_1, g_2, \ldots, g_N], \tag{1}$$

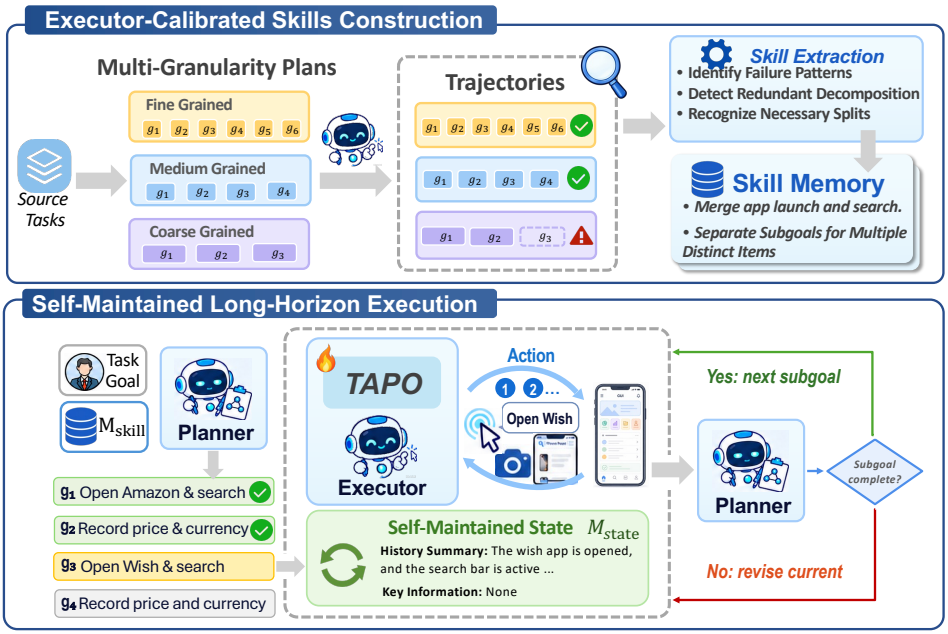


Figure 2: Overview of EASE-Agent with executor-calibrated planning and self-maintained state. Executor-calibrated skills are distilled from multi-granularity execution trajectories to guide task decomposition. During execution, the TAPO-trained executor maintains compact task state, with the planner providing sparse assistance for decomposition, verification, and replanning.

For each subgoal g_i, the executor jointly generates a GUI action a_t and updates a compact history summary h_t based on the current screenshot o_t and the previous summary h_{t-1}:

$$(a_t, h_t) = \pi_E(g_i, o_t, h_{t-1}), \tag{2}$$

The history summary tracks execution progress within the current subgoal, while the executor uses a REMEMBER(text) action to explicitly retain task-relevant information for later use in K^i_{sub}. Together, they constitute the state memory of subgoal g_i, $M^i_{\text{state}} = (h^i_{\text{final}}, K^i_{\text{sub}})$. Both the history summary and K^i_{sub} are reset at the start of each new subgoal.

When the executor declares the current subgoal complete or unable to proceed, the planner receives this state memory together with the task information store K_{task}. It verifies subgoal completion and determines what to execute next:

$$(c_i, g_{\text{next}}) = \pi_P(I, o_t, g_i, M^i_{\text{state}}, K_{\text{task}}), \tag{3}$$

where $c_i \in \{\text{COMPLETE}, \text{IN-PROGRESS}, \text{FAILED}\}$ and g_{next} is the next execution target. The key information collected during the attempt is then merged into the task information store for use in subsequent subgoals as $K_{\text{task}} \leftarrow K_{\text{task}} \cup K^i_{\text{sub}}$. If g_i is verified as complete, the planner advances to the next pending subgoal in P. Otherwise, it revises the current subgoal based on the execution state. This execution–verification–replanning loop continues until the complete instruction is satisfied.

3.2 SKILL MEMORY FOR EXECUTOR-CALIBRATED PLANNING

We introduce skill memory to calibrate task decomposition to the executor's capabilities, enabling subgoals of appropriate granularity while minimizing unnecessary planner intervention. Constructing such memory requires identifying how different planning decisions affect the executor's performance. However, naturally collected trajectories provide limited guidance, as execution outcomes are usually influenced by factors beyond planning decisions. Therefore, we construct skill memory by contrasting execution trajectories under different planning strategies and distilling their differences into reusable planning skills.

Multi-granularity trajectory collection. For each task I in a skill-construction set, we use three planning prompts p_f, p_m, and p_c to induce fine-, medium-, and coarse-grained plans P^z =

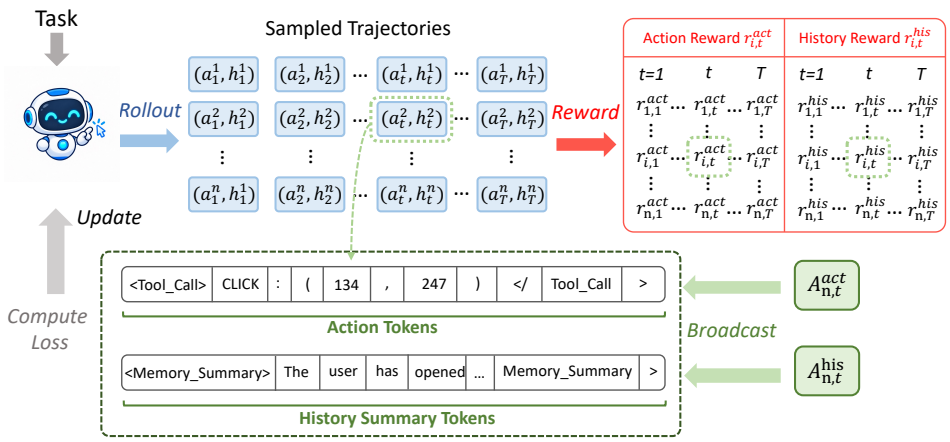


Figure 3: Overview of TAPO for learning self-maintained state. TAPO computes objective-specific advantages for action generation and history-summary maintenance and assigns them to the corresponding token spans during joint optimization.

$\pi_P(I, p_z)$ for $z \in \{f, m, c\}$. Only the prompt controlling plan granularity changes, and all other experimental settings are held constant across the three conditions. The executor follows each plan, producing trajectories τ^f, τ^m, and τ^c. This controlled construction makes differences across trajectories more directly attributable to subgoal granularity and instruction specificity.

For example, a medium-grained plan may use two subgoals, *open the Amazon app* and *enter "iPhone 16 Pro" in the search bar and search*, whereas a coarse-grained plan combines them into a single subgoal. If the executor reliably completes the task under both plans, the separate planner intervention is unnecessary. The comparison then yields a skill recommending that opening an application and initiating an in-app search be combined, thereby reducing planner calls.

Trajectory compression and contrastive skill extraction. For efficient comparison, we compress each raw trajectory into a subgoal-level summary containing the subgoal description, action intentions, verification outcome, and interface states. An LLM-based extraction function then compares the three compressed trajectories and produces candidate planning skills $S = F_{\text{skill}}(I, \hat{\tau}^f, \hat{\tau}^m, \hat{\tau}^c)$, where $\hat{\tau}$ denotes a compressed trajectory. Candidate skills from all construction tasks are further consolidated into the final skill memory $M_{\text{skill}} = \{s_1, s_2, \ldots, s_K\}$ by merging similar entries, reconciling conflicting recommendations, and removing redundant information.

3.3 TAPO for Learning Self-Maintained State

Given the planner's subgoals, the executor must generate executable GUI actions while maintaining a compact execution state. We parameterize the executor π_E as π_θ and jointly train these two coupled behaviors with reinforcement learning. However, conventional GRPO assigns a shared sequence-level advantage to all output tokens, causing interference between action generation and state maintenance. GDPO (Liu et al., 2026c) normalizes rewards separately but combines their advantages before policy optimization. We therefore introduce Token-level Advantage Policy Optimization (TAPO), which assigns objective-specific advantages to their corresponding token spans. Figure 3 illustrates the TAPO training workflow.

Semi-online trajectory collection. Following UI-S1 (Lu et al., 2025b), we sample a group of G rollouts from $\pi_{\theta_{\text{old}}}$ for each training task using offline reference trajectories, while carrying forward the executor's generated history summaries. Each rollout terminates at the end of the reference trajectory or after accumulating four action mismatches, determined by the action-matching criterion defined below. We denote the resulting length of rollout j by T_j.

Structured executor output and rewards. At step t of rollout j, the executor generates a response $y_{j,t}$ containing a history-summary span and an action span. Let $\Gamma_{\text{his}}(y_{j,t})$ and $\Gamma_{\text{act}}(y_{j,t})$ denote the token positions belonging to these spans. We compute the action reward at step t of rollout j as

$r_{j,t}^{\text{act}} = \text{CheckAction}(a_{j,t}, a_t^\star)$, where $a_t^\star$ is the reference action, and CheckAction returns 1 if the predicted action type and required parameters satisfy the matching criteria, and 0 otherwise.

The utility of a history summary is determined by whether it supports correct subsequent decisions. We therefore define its reward using discounted future action correctness. For rollout j and $t < T_j$,

$$r_{j,t}^{\text{his}} = \frac{\sum_{k=t+1}^{T_j} \gamma^{k-t-1} r_{j,k}^{\text{act}}}{\sum_{k=t+1}^{T_j} \gamma^{k-t-1}}, \tag{4}$$

where $\gamma \in (0, 1]$ controls discounting. This reward is a weighted average of subsequent action correctness, favoring nearer steps when $\gamma < 1$.

Objective-specific group advantages. Using the collected rollout group, TAPO computes group-relative advantages for the two rewards independently, rather than combining them before normalization. For objective $q \in \{\text{act}, \text{his}\}$,

$$A_{j,t}^q = \frac{r_{j,t}^q - \mu_t^q}{\sigma_t^q + \delta}, \tag{5}$$

where μ_t^q and σ_t^q are the group mean and standard deviation for objective q, and $\delta > 0$ ensures numerical stability. Separate normalization preserves the relative quality signal for each objective.

Token-level advantage assignment. Standard GRPO (Shao et al., 2024) broadcasts one sequence-level advantage to every token in a response. For the structured executor output, this can assign summary feedback to action tokens and action feedback to summary tokens. TAPO instead constructs a token-level advantage for token position n:

$$\tilde{A}_{j,t,n} = \begin{cases} A_{j,t}^{\text{act}}, & n \in \Gamma_{\text{act}}(y_{j,t}), \\ A_{j,t}^{\text{his}}, & n \in \Gamma_{\text{his}}(y_{j,t}), \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Let

$$\rho_{j,t,n}(\theta) = \frac{\pi_\theta(y_{j,t,n} \mid x_{j,t}, y_{j,t,<n})}{\pi_{\theta_{\text{old}}}(y_{j,t,n} \mid x_{j,t}, y_{j,t,<n})} \tag{7}$$

be the token-level probability ratio between the current and rollout policies, where $x_{j,t}$ denotes the model input. The clipped TAPO objective is

$$\mathcal{J}_{\text{TAPO}}(\theta) = \mathop{\text{E}}_{j,t,n}\Big[\min\big(\rho_{j,t,n}(\theta)\tilde{A}_{j,t,n}, \text{clip}(\rho_{j,t,n}(\theta), 1-\varepsilon, 1+\varepsilon)\tilde{A}_{j,t,n}\big)\Big] - \beta\, \mathcal{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{8}$$

where ε is the clipping range, π_{ref} is a fixed reference policy, and β controls KL regularization.

4 Experiments

4.1 Experimental Setup

Models and baselines. EASE-UI is initialized from Qwen3-VL-2B-Instruct (Bai et al., 2025a). EASE-Agent pairs EASE-UI as the executor with Qwen3-VL-32B-Instruct as the planner. We further instantiate two stronger variants with Gemini-2.5-Pro (Comanici et al., 2025) as the planner: EASE-Agent-Pro uses Qwen3-VL-32B-Instruct as the executor, while EASE-Agent-Max further scales the executor to Qwen3.8-27B (Qwen Team, 2026). To evaluate both the step-level action prediction and end-to-end execution capabilities of EASE-UI (Table 1), we compare it with Qwen vision-language models (Bai et al., 2025b;a) and GUI-specialized models (Ye et al., 2025; Lu et al., 2026a; 2025b). To evaluate long-horizon task execution (Table 2), we compare three EASE-Agent variants with two categories of baselines: end-to-end models (Xu et al., 2026; Gu et al., 2025) and agentic workflows (Zheng et al., 2024; Rawles et al., 2024; Agashe et al., 2025b; Lu et al., 2026b).

Benchmarks and metrics. We evaluate action prediction on GUI-Odyssey (Lu et al., 2025a) using TR, GR, and SR, and long-horizon task completion on MemGUI-Bench (Liu et al., 2026b) using

Table 1: Standalone executor performance of EASE-UI and baseline models across step-level action prediction and long-horizon task completion (%).

	GUI-Odyssey			MemGUI-Bench (pass@3)			
Model	TR	GR	SR	Easy	Medium	Hard	Ave
Qwen2.5-VL-7B	67.4	56.3	52.4	12.5	0.0	0.0	4.7
Qwen3-VL-2B-Instruct	67.4	58.8	46.2	18.8	0.0	0.0	7.0
Qwen3-VL-8B-Instruct	71.5	60.0	53.5	29.2	**16.7**	**5.3**	**18.0**
GUI-Owl-7B	78.7	–	61.3	22.9	2.4	2.6	10.1
UI-R1-7B	67.1	41.3	43.5	–	–	–	–
UI-S1-7B	76.3	61.7	59.5	14.6	2.6	0.0	6.2
EASE-UI	**80.6**	**69.9**	**64.9**	**31.3**	0.0	0.0	11.7

pass@1 and pass@3. We additionally report response time, model calls, execution steps, and API cost to evaluate efficiency. Detailed benchmark descriptions, metric definitions, and evaluation protocols are provided in Appendix A.

Implementation details. We train EASE-UI on 500 tasks from the GUI-Odyssey high-random training partition, using their high-level task instructions. Training runs for three epochs, with eight rollouts sampled per task. Further training details are provided in Appendix C.

Skill memory is constructed from 30 manually authored tasks. These tasks are disjoint from the evaluation tasks. We execute the tasks under fine-, medium-, and coarse-grained plans and extract planning skills as described in the skill-memory construction procedure. We use Gemini 3.1 Pro (Google DeepMind, 2026) to generate the skills.

4.2 Main Results

Action prediction on GUI-Odyssey. Table 1 shows that EASE-UI achieves the highest reported TR, GR, and SR among the compared models on GUI-Odyssey. Its SR reaches 64.9%, improving over its Qwen3-VL-2B-Instruct initialization by 18.7 percentage points and over UI-S1-7B by 5.4 points. The gains extend beyond action-type selection: grounding accuracy improves from 58.8% to 69.9% relative to the base model. Joint action and summary training therefore retains strong action-prediction capability in a 2B executor.

Task completion on MemGUI-Bench. Table 2 evaluates the complete system across task difficulties and numbers of apps. EASE-Agent achieves 19.5% pass@1 and 31.3% pass@3. Using a stronger planner–executor configuration, EASE-Agent-Pro reaches 28.9% pass@1 and 50.0% pass@3, while EASE-Agent-Max further improves pass@1 from M3A's 32.8% to 51.6% and pass@3 from Agent-S2's 49.2% to 75.0%, achieving the best overall performance among the compared agentic systems on MemGUI-Bench. These results show that the framework supports different model configurations and benefits from stronger backbones on long-horizon tasks.

Response-time efficiency. Table 3 compares task completion and cumulative API response time on MemGUI-Bench. The results show that EASE-Agent outperforms all three end-to-end Qwen3-VL baselines while requiring substantially less response time than the 8B and 32B models. Compared with Qwen3-VL-32B, EASE-Agent reduces response time from 307.14 to 130.52 seconds under pass@1 and from 322.28 to 144.95 seconds under pass@3, while improving task completion by 3.1 and 3.2 percentage points, respectively. These results show that EASE improves long-horizon task performance without incurring the high inference latency of large end-to-end models.

API-cost efficiency. Table 4 compares task success and API cost on MemGUI-Bench. Under pass@3, EASE-Agent-Pro achieves 50.0% success at USD 0.1499 per attempt, comparable to Agent-S2's 49.2% while using only 9.4% of its cost. Scaling to EASE-Agent-Max further increases pass@3 to 75.0% at USD 0.1760 per attempt, while remaining substantially less expensive than T3A, M3A, and Agent-S2. Despite requiring more execution steps, both variants maintain low cost

Table 2: Long-horizon task completion of EASE-Agent variants and baseline agents across task difficulty and number of involved applications on MemGUI-Bench (%); p@1 and p@3 denote pass@1 and pass@3.

	Number of Apps								Task Difficulty						Average	
	1 App		2 Apps		3 Apps		4 Apps		Easy		Medium		Hard			
Model	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3
Qwen3.8-27B	64.3	85.7	**32.1**	**58.9**	38.2	70.6	40.0	**70.0**	52.1	72.9	31.0	59.5	39.5	**73.7**	41.4	68.8
Qwen3-VL-32B	46.4	67.9	7.1	17.9	11.8	20.6	0.0	0.0	35.4	54.2	4.8	11.9	5.3	13.2	16.4	28.1
Qwen3-VL-8B	21.4	46.4	3.6	14.3	2.9	5.9	0.0	0.0	14.6	29.2	2.4	16.7	2.6	5.3	7.0	18.0
GUI-Owl-1.5-32B	35.7	60.7	1.8	7.1	5.9	11.8	10.0	10.0	22.9	37.5	7.1	14.3	0.0	5.3	10.9	20.3
UI-Venus-7B	21.4	28.6	1.8	1.8	0.0	2.9	0.0	0.0	14.6	20.8	0.0	0.0	0.0	0.0	5.5	7.8
UI-TARS-1.5-7B	14.3	21.4	0.0	1.8	0.0	2.9	0.0	0.0	8.3	16.7	0.0	0.0	0.0	0.0	3.1	6.2
SeeAct	10.7	25.0	0.0	0.0	0.0	0.0	0.0	0.0	6.2	12.5	0.0	2.4	0.0	0.0	2.3	5.5
Mobile-Agent-E	25.0	42.9	0.0	1.8	0.0	0.0	0.0	0.0	12.5	22.9	2.4	2.4	0.0	2.6	5.5	10.2
AppAgent	14.3	42.9	0.0	0.0	0.0	0.0	0.0	0.0	8.3	22.9	0.0	2.4	0.0	0.0	3.1	9.4
UI-Copilot-7B	42.9	50.0	12.5	16.1	5.9	8.8	0.0	10.0	29.2	33.3	13.2	18.4	4.8	7.1	16.4	20.3
T3A	42.9	60.7	16.1	37.5	23.5	38.2	0.0	30.0	31.2	45.8	16.7	45.2	18.4	34.2	22.7	42.2
M3A	46.4	64.3	28.6	41.1	29.4	44.1	30.0	50.0	39.6	47.9	35.7	50.0	21.1	44.7	32.8	47.7
Agent-S2	50.0	78.6	19.6	35.7	26.5	52.9	10.0	30.0	41.7	64.6	19.0	42.9	18.4	36.8	27.3	49.2
EASE-Agent	42.9	57.1	8.9	12.5	23.5	50.0	0.0	0.0	33.3	54.2	14.3	21.4	7.9	13.2	19.5	31.3
EASE-Agent-Pro	50.0	71.4	23.2	33.9	23.5	61.8	20.0	40.0	39.6	60.4	23.8	47.6	21.1	39.5	28.9	50.0
EASE-Agent-Max	**75.0**	**92.9**	30.4	**58.9**	**67.6**	**88.2**	**50.0**	**70.0**	**64.6**	**87.5**	**45.2**	**66.7**	**42.1**	68.4	**51.6**	**75.0**

Table 3: Performance–latency comparison between EASE-Agent and end-to-end Qwen3-VL baselines on MemGUI-Bench.

	Success rate (%)		Time/Attempt (s)	
Model	pass@1	pass@3	pass@1	pass@3
Qwen3-VL-32B-Instruct	16.4	28.1	307.14	322.28
Qwen3-VL-8B-Instruct	7.0	18.0	166.71	164.84
Qwen3-VL-2B-Instruct	3.9	7.0	103.46	102.76
EASE-Agent	19.5	31.3	130.52	144.95

per attempt because frequent GUI interactions are handled by the executor, while the more expensive planner is invoked sparsely for planning and verification. These results show that EASE enables substantial gains in task completion with only modest increases in API cost.

4.3 Ablation Studies

TAPO for joint action and state learning. Under identical experimental settings, TAPO achieves 64.9% SR on GUI-Odyssey, outperforming GRPO (Shao et al., 2024) and GDPO (Liu et al., 2026c) by 1.9 percentage points (Table 5). To isolate its effect on state maintenance, Figure 4 fixes the executor to Qwen3-VL-2B and varies only the history-summary source. TAPO-generated summaries yield 48.3% SR, compared with 43.8% without summaries and 46.2% with summaries from the original model, showing that TAPO improves execution performance and maintained-state utility.

Skill-memory and multi-granularity ablation. We compare no skill memory, skill memory from naturally collected trajectories, and our multi-granularity construction under the same planner, executor, and evaluation settings; both construction methods use the same 30 tasks. As shown in Table 6, skill memory raises pass@3 from 19.5% to 28.1%, while multi-granularity construction further reaches 19.5%/31.3% pass@1/pass@3. This confirms the benefit of planning skills and con-

Table 4: Performance–cost comparison of EASE-Agent-Pro and different agents on MemGUI-Bench.

Model	Success rate (%)		Cost/Step (USD)		Avg. Steps/ Attempt		Cost/Attempt (USD)	
	pass@1 ↑	pass@3 ↑	pass@1 ↓	pass@3 ↓	pass@1 ↓	pass@3 ↓	pass@1 ↓	pass@3 ↓
Qwen3.8-27B	41.4	68.8	0.002111	0.002003	37.66	40.58	0.0795	0.0813
Qwen3-VL-8B-Instruct	7.0	18.0	0.000374	0.000373	49.01	48.46	0.0183	0.0181
Qwen3-VL-32B-Instruct	16.4	28.1	0.001595	0.001598	54.93	57.65	0.0876	0.0921
T3A	22.7	42.2	0.0176	0.0175	33.87	34.59	0.5788	0.5871
M3A	32.8	47.7	0.0165	0.0162	34.38	36.24	0.5509	0.5703
Agent-S2	27.3	49.2	0.0510	0.0522	30.69	31.45	1.5139	1.5897
EASE-Agent-Pro	28.9	50.0	0.0032	0.0031	46.12	47.72	0.1489	0.1499
EASE-Agent-Max	51.6	75.0	0.0037	0.0034	44.78	51.19	0.1677	0.1760

Table 5: Ablation of policy optimization methods for EASE-UI action prediction on GUI-Odyssey (%).

Model	TR	GR	SR
Qwen3-VL-2B-Instruct	67.4	58.8	46.2
+GRPO	79.9	69.8	63.0
+GDPO	79.8	69.8	63.0
+TAPO	**80.6**	**69.9**	**64.9**

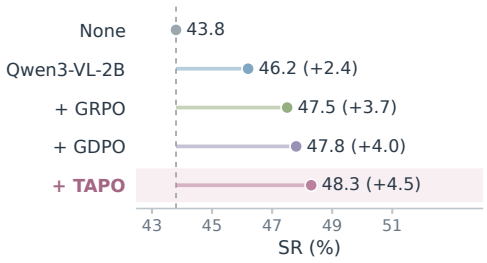


Figure 4: GUI-Odyssey SR across history-summary sources with a fixed Qwen3-VL-2B executor. Labels give SR and gains over no summary; the dashed line marks the baseline.

Table 6: Ablation of skill memory and multi-granularity skill construction on MemGUI-Bench task completion (%).

Method	pass@1	pass@3
w/o skills	12.5	19.5
w/o multi-granularity	14.1	28.1
ours	**19.5**	**31.3**

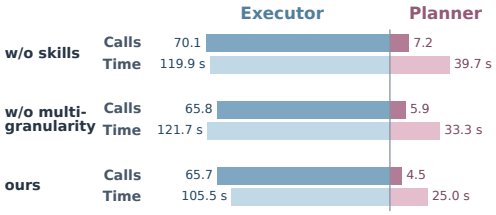


Figure 5: Planner and executor overhead under different skill-memory ablations, measured by model calls and API response time per attempt under pass@1.

trolled multi-granularity construction for executor-calibrated planning. Figure 5 further shows that our construction reduces planner calls from 7.2 to 4.5 and cumulative planner response time from 39.7 to 25.0 seconds per attempt, indicating fewer unnecessary planner interventions.

5 Conclusion

We presented EASE-Agent, an efficient mobile GUI agent for long-horizon tasks that combines self-maintained state in a lightweight executor with executor-calibrated planning through skill memory. TAPO trains the executor to predict actions and maintain history summaries using token-level, objective-specific advantages, while the skill memory adapts subgoal granularity to the executor. The executor reaches 64.9% SR on GUI-Odyssey; on MemGUI-Bench, the agent reaches 31.3% pass@3 and a stronger configuration reaches 75.0% pass@3. Ablations support the contributions of TAPO and skill construction, and the efficiency comparisons show that selective high-level planning can improve long-horizon task completion with lower planner overhead and API cost.

Reproducibility Statement

The Method section specifies the agent architecture, TAPO objective, trajectory collection, and skill construction. The Experiments section reports model configurations, training settings, evaluation metrics, baselines, and ablations on the publicly available GUI-Odyssey and MemGUI-Bench benchmarks.

AI Use Statement

Gemini 3.1 Pro was used as part of the experimental pipeline to generate the skill-memory entries described in the paper. Generative AI tools were also used for language polishing and minor manuscript editing. All AI-assisted content was reviewed and verified by the authors.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent S: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/394c7c30ea87b5c3521b4d9e9d419071-Abstract-Conference.html.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A compositional generalist-specialist framework for computer use agents. In *Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=zg5is4GJ3R.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025a. URL https://arxiv.org/abs/2511.21631.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025b. URL https://arxiv.org/abs/2502.13923.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. CUA-Skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*, 2026. URL https://arxiv.org/abs/2601.21123.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. URL https://arxiv.org/abs/2507.06261.

Google DeepMind. Gemini 3.1 Pro model card. Model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, et al. UI-Venus technical report: Building high-performance UI agents with RFT. *arXiv preprint arXiv:2508.10833*, 2025. URL https://arxiv.org/abs/2508.10833.

Guangyi Liu, Gao Wu, Congxiao Liu, Pengxiang Zhao, Liang Liu, Mading Li, Qi Zhang, Mengyan Wang, Liang Guo, and Yong Liu. MemGUI-Agent: An end-to-end long-horizon mobile GUI agent with proactive context management. *arXiv preprint arXiv:2606.19926*, 2026a. URL https://arxiv.org/abs/2606.19926.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, Zhengxi Lu, Gao Wu, Hao Wang, Liang Liu, and Yong Liu. MemGUI-Bench: Benchmarking memory of mobile GUI agents in dynamic environments. *arXiv preprint arXiv:2602.06075*, 2026b. URL https://arxiv.org/abs/2602.06075.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. GDPO: Group reward-decoupled normalization policy optimization for multi-reward RL optimization. *arXiv preprint arXiv:2601.05242*, 2026c. URL https://arxiv.org/abs/2601.05242.

Tao Liu, Chongyu Wang, Rongjie Li, Yingchen Yu, Xuming He, and Bai Song. GUI-Rise: Structured reasoning and history summarization for GUI navigation. *arXiv preprint arXiv:2510.27210*, 2025. URL https://arxiv.org/abs/2510.27210.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. GUIOdyssey: A comprehensive dataset for cross-app GUI navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22404–22414, 2025a. URL https://openaccess.thecvf.com/content/ICCV2025/papers/Lu_GUIOdyssey_A_Comprehensive_Dataset_for_Cross-App_GUI_Navigation_on_Mobile_ICCV_2025_paper.pdf.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, and Yueting Zhuang. UI-S1: Advancing GUI automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*, 2025b. URL https://arxiv.org/abs/2509.11543.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Pengxiang Zhao, Guangyi Liu, Guanjing Xiong, and Hongsheng Li. UI-R1: Enhancing efficient action prediction of GUI agents by reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 17608–17616, 2026a. doi: 10.1609/aaai.v40i21.38816. URL https://ojs.aaai.org/index.php/AAAI/article/view/38816.

Zhengxi Lu, Fei Tang, Guangyi Liu, Jin Ma, Kaitao Song, Xu Tan, Wenqi Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. UI-Copilot: Advancing long-horizon GUI automation via tool-integrated policy optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19741–19762, 2026b. doi: 10.18653/v1/2026.acl-long.904. URL https://aclanthology.org/2026.acl-long.904/.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.8-27B model card. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024. URL https://arxiv.org/abs/2405.14573.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Junyang Wang, Haiyang Xu, Xi Zhang, Zhaoqing Zhu, Ming Yan, Jieping Ye, and Jitao Sang. STAMP: Training explicit memory for mobile GUI agents in controllable and scalable virtual environments. *arXiv preprint arXiv:2605.29324*, 2026. URL https://arxiv.org/abs/2605.29324.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-Agent-E: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025. URL https://arxiv.org/abs/2501.11733.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: Foundation action model for generalist GUI agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0faa4bc5f522076947a030273629d4fe-Abstract-Conference.html.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, et al. Mobile-Agent-v3.5: Multi-platform fundamental GUI agents. *arXiv preprint arXiv:2602.16855*, 2026. URL https://arxiv.org/abs/2602.16855.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, et al. Mobile-Agent-v3: Fundamental agents for GUI automation. *arXiv preprint arXiv:2508.15144*, 2025. URL https://arxiv.org/abs/2508.15144.

Biao Yi, Xueyu Hu, Yurun Chen, Shengyu Zhang, Hongxia Yang, and Fan Wu. EcoAgent: An efficient device-cloud collaborative multi-agent framework for mobile automation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29838–29846, 2026. doi: 10.1609/aaai.v40i35.40230. URL https://ojs.aaai.org/index.php/AAAI/article/view/40230.

Ziyun Zeng, Hang Hua, Bocheng Zou, Mu Cai, Rogerio Feris, and Jiebo Luo. MementoGUI: Learning agentic multimodal memory control for long-horizon GUI agents. *arXiv preprint arXiv:2605.18652*, 2026. URL https://arxiv.org/abs/2605.18652.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. AppAgent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025. URL https://arxiv.org/abs/2312.13771.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024. URL https://arxiv.org/abs/2401.01614.

A Benchmarks and Evaluation Protocols

A.1 Benchmark Characteristics

GUI-Odyssey. GUI-Odyssey (Lu et al., 2025a) is an offline dataset of recorded mobile-GUI demonstrations for cross-application navigation. It contains 8,334 episodes across 212 applications and 1,357 application combinations, with an average of 15.3 actions per episode. Each episode pairs a high-level task instruction with reference screenshots and actions; the dataset also provides lower-level instructions and semantic annotations. Its cross-application workflows and relatively long action sequences make action prediction sensitive to information carried forward from earlier steps. We use high-level instructions for the evaluations in this paper.

MemGUI-Bench. MemGUI-Bench (Liu et al., 2026b) is an interactive, memory-centric benchmark for mobile agents. Its full task set comprises 128 tasks across 26 applications. Of these, 115 are designed to require information retention across steps or applications. The reference tasks span 3–160 steps, with an average of 36.2, and cover tasks involving one to four applications. The benchmark groups 48 tasks as easy, 42 as medium, and 38 as hard; the numbers of tasks involving one, two, three, and four applications are 28, 56, 34, and 10, respectively. It provides snapshot-based environment resets, multi-attempt evaluation, and a staged evaluator for task completion. We evaluate the complete 128-task set.

A.2 Evaluation Protocols and Metrics

Offline action prediction. For GUI-Odyssey, we report action-type accuracy (TR), grounding accuracy (GR), and step success rate (SR), following the step-level reporting convention used by UI-S1 (Lu et al., 2025b). TR measures whether the predicted action type matches the reference type. GR measures click-point grounding on reference click actions: a prediction is considered correctly grounded only if it is also a click and the Euclidean distance between the predicted and reference click coordinates, both normalized by dividing by 1000, is at most 0.14. A step counts toward SR only when the action type and all required parameters satisfy the action-matching rule.

Interactive task completion. On MemGUI-Bench, we run each task for at most three attempts. For $k \in \{1, 3\}$, pass@k is the percentage of the 128 tasks that succeed at least once within their first k attempts; pass@1 uses only the first attempt, whereas pass@3 allows success on any of the first three. Results by difficulty or number of applications use the same definition with the corresponding task group as the denominator.

Reported baseline results. The UI-R1-7B results on GUI-Odyssey and the UI-S1-7B results on MemGUI-Bench in Table 1, together with the UI-Copilot-7B results in Table 2, are taken from UI-Copilot (Lu et al., 2026b). We retain the source-reported group scores and overall averages rather than recomputing them from rounded subgroup percentages.

Efficiency measures. We include all valid attempts in the efficiency analysis, whether successful or failed. Time/Attempt is the total API response time of planner and executor calls divided by the number of valid attempts; Calls/Attempt uses the same denominator for each role's call count. Avg. Steps/Attempt is the total number of execution steps divided by the number of valid attempts. Cost/Step is the total API cost divided by the total number of execution steps. For T3A, M3A, and Agent-S2, total API expenditure is unavailable, so Cost/Attempt is estimated as Cost/Step multiplied by Avg. Steps/Attempt. For methods with recorded total expenditure, Cost/Attempt is computed directly by dividing that expenditure by the number of valid attempts. Each execution step is an interaction with the environment. These measures are computed separately for the pass@1 and pass@3 settings where indicated. Offline training and skill construction are excluded from the reported response time and API cost.

B EXECUTOR ACTION SPACE

The executor selects an action from the space $\mathcal{A}$ defined in Table 7. Coordinates use the normalized range $[0, 999]$ on each axis, and the scroll direction is one of up, down, left, or right. Actions without a parameter are marked with a dash.

Table 7: Executor action types, required parameters, and their functions.

Action type	Parameter	Function
`CLICK`	`coordinate`	Tap the specified screen location.
`LONG_PRESS`	`coordinate`	Press and hold at the specified location.
`SCROLL`	`direction`	Scroll the screen in the specified direction.
`TYPE`	`text`	Enter the specified text using the keyboard.
`ANSWER`	`text`	Return a final text answer.
`OPEN_APP`	`app_name`	Open the named application.
`WAIT`	–	Wait for a screen transition or loading.
`COMPLETE`	–	Signal completion of the current subgoal.
`IMPOSSIBLE`	–	Signal that the current subgoal cannot be completed.
`PRESS_HOME`	–	Return to the device's home screen.
`PRESS_BACK`	–	Return to the previous screen.
`REMEMBER`	`text`	Store task-relevant information for later use.

C EASE-UI TRAINING DETAILS

Figure 6 shows the training rewards and optimization losses over training steps. The action and history rewards generally increase, while the policy-gradient loss fluctuates around zero. The KL loss rises over the run; these curves describe training dynamics rather than held-out benchmark performance.



Figure 6: EASE-UI training curves over optimization steps. Top: action and history rewards. Bottom: policy-gradient and KL losses.

Table 8 summarizes the data, rollout, and optimization settings used to train EASE-UI. The training set contains 500 tasks with high-level instructions from the GUI-Odyssey high-random training partition.

Table 8: Training configuration for EASE-UI.

Setting	Value
Initial model	Qwen3-VL-2B-Instruct
Training tasks	500
Training epochs	3
Rollouts per task (G)	8
Action-mismatch termination threshold	4
Future-reward discount (γ)	0.5
Learning rate	10^{-6}
Training batch size / PPO mini-batch size	8 / 8
Maximum prompt / response length (tokens)	12,288 / 512
KL coefficient (β)	10^{-4}
KL loss type	`low_var_kl`

D EASE-AGENT TASK EXECUTION CASE STUDY

User Instruction

In the Meesho app, find the first search result for three sarees: ""Banarasi Silk"", ""Kanjivaram Silk"", and ""Paithani Silk"". For each, remember its star rating and price. Then, navigate to the product page of the saree with the best value (highest rating-to-price ratio).

Task Decomposition

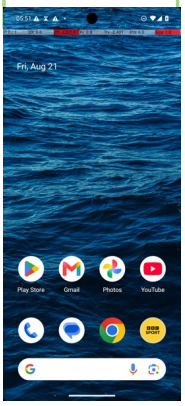
Skills: 1. Separate Subgoals for Multiple Distinct Items: Create a separate subgoal for each distinct item, entity, or event. Don't combine multiple independent items into a single subgoal. 2. Combine App Opening with Initial Substantive Action: Combine opening the application with the first substantive action (e.g., entering a search query or linear navigation) into a single subgoal. ……

Planner Output: {'subgoals': [{'id': 1, 'worker': 'executor', 'subgoal': 'Open the Meesho. app and use the search bar to search for 'Banarasi Silk', then extract and remember the star rating and price of the first search result.'}, {'id': 2, 'worker': 'executor', 'subgoal': 'In the Meesho app, use the search bar to search for 'Kanjivaram Silk', then extract and remember the star rating and price of the first search result.'}, {'id': 3, 'worker': 'executor', 'subgoal': 'In the Meesho app, use the search bar to search for 'Paithani Silk', then extract and remember the star rating and price of the first search result.'}, {'id': 4, 'worker': 'planner': 'Calculate the rating-to-price ratio for each of the three sarees using the remembered star ratings and prices, and determine which saree has the highest ratio.'}, {'id': 5, 'worker': 'executor', 'subgoal': 'Navigate to the product page of the saree with the highest rating-to-price ratio by searching for its name ('Banarasi Silk', 'Kanjivaram Silk', or 'Paithani Silk') in the Meesho app and selecting the first result.'}]}

Task Execution

Step 1

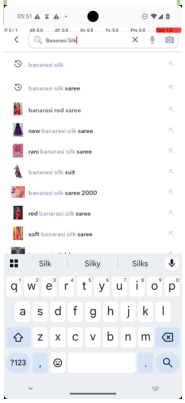
g_1: Open the Meesho app and use the search bar to search for ...



<Memory Summary> The task has just started. The user wants to search for 'Banarasi Silk' and ... </Memory Summary> <tool_call>{"action": "SCROLL", "direction": "UP"}</tool_call>

Step 8

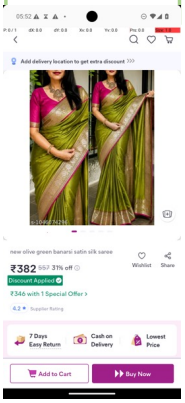
g_1: Open the Meesho app and use the search bar to search for ...



<Memory Summary> The Meesho app is open, and the search bar contains 'Banarasi Silk'. The user is ready to execute the search. </Memory Summary> <tool_call>{"action": "CLICK", "coordinate": [915, 915]} </tool_call>

Step 10

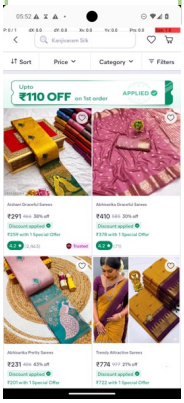
g_1: Open the Meesho app and use the search bar to search for ...



<Memory Summary> The search for 'Banarasi Silk' has been completed. The first result is ... </Memory Summary> <tool_call>{"action": "REMEMBER", "text": "Banarasi Silk: 4.2 star rating, price: ₹382"} </tool_call>

Step 13

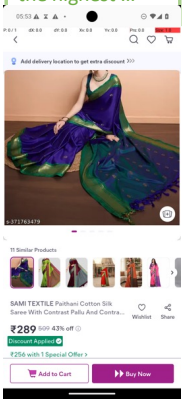
g_2: ... search for 'Kanjivaram Silk', then extract and remember ...



<Memory Summary> The user has entered 'Kanjivaram Silk' in the search bar of the Meesho app. </Memory Summary> <tool_call>{"action": "CLICK", "coordinate": [919, 918]}</tool_call>

Step 21

g_5: Navigate to the product page of the saree with the highest ...



<Memory Summary> The user has searched for 'Paithani Silk' in the Meesho app and is now viewing the product details of the first search result. </Memory Summary> <tool_call>{"action": "COMPLETE"}</tool_call>

E EXECUTOR SYSTEM PROMPT

The following is the author-provided system prompt for the interactive executor, with long lines wrapped for typesetting. The tool schema specifies the available actions; task instructions and screenshots are supplied separately at each step.

```
You are a mobile device control assistant.

You can call the `mobile_use` function within <tool_call>...</tool_call> tags:
<tool_call>
{"type": "function", "function": {
"name": "mobile_use",
"description": "Use a touchscreen to interact with a mobile device and take
  screenshots.",
"parameters": {
    "type": "object",
    "properties": {
    "action": {
        "description": "The action to perform.",
        "enum": ["CLICK", "LONG_PRESS", "SCROLL", "TYPE", "ANSWER", "OPEN_APP",
          "WAIT", "REMEMBER", "COMPLETE", "IMPOSSIBLE", "PRESS_HOME", "PRESS_BACK"],
        "type": "string"
    },
    "coordinate": {
        "description": "(x, y): Required only by CLICK and LONG_PRESS.",
        "type": "array",
        "items": {"type": "number"}
    },
    "text": {
        "description": "Required by TYPE (the text to input), ANSWER (the final
          answer text), and REMEMBER (the content to remember).",
        "type": "string"
    },
    "app_name": {
        "description": "Required only by OPEN_APP (the app name to open).",
        "type": "string"
    },
    "direction": {
        "description": "Scroll direction. Required only by SCROLL. Valid values: UP,
          DOWN, LEFT, RIGHT.",
        "enum": ["UP", "DOWN", "LEFT", "RIGHT"],
        "type": "string"
    }
    },
    "required": ["action"]
}
}}
</tool_call>


Your output must strictly follow this format:

1) <Reasoning>...</Reasoning>
- Contains exactly one or two concise sentence explaining the next move.
- Do not include multi-step reasoning or extra text.

2) <Memory Summary>...</Memory Summary>
- Contains a concise, information-dense summary of prior actions and GUI states.
- The History Summary should briefly describe and capture the key progress made in
  this segment of the task.As you review the screenshots, extract any details that
  may be relevant for accomplishing the task later.
- The summary must not speculate about future actions. Do NOT predict the next step.
  Do NOT include phrases such as "will", "next", "going to", or any forward-looking
  reasoning. The summary should ONLY describe what has already happened.

3) <tool_call>...</tool_call>
- A json object with function name and arguments within <tool_call></tool_call> XML
  tags:
- Do not include extra text or comments.
- Example:
    <tool_call>{"action": "", "coordinate": [100, 200]}</tool_call>


# Guidelines
- Output exactly in the order: <Reasoning>...</Reasoning>,
  <Memory Summary>...</Memory Summary>, <tool_call>...</tool_call>.
- Do not output anything else outside those three parts.
- Strictly follow the 999x999 coordinate system.
- Use `WAIT` if the UI is loading or transitioning.
```

```
- Use COMPLETE once the current subgoal is finished.
- Be brief. No multi-step reasoning. Only one thought/action per step.


# Remember Action
- Use REMEMBER when the current screenshot contains task-critical information that
  must be saved for later, such as prices, dates, names, ratings, addresses or
  selected options.
- REMEMBER does NOT change the screen state. It only records the provided `text` as
  a persistent note for later steps.
- Only output REMEMBER when there is truly new information worth saving; avoid
  repeating the same memory across steps.
- For Example:
  <tool_call>{"action": "REMEMBER", "text": "Hotel A: $129/night, free cancellation
    until May 20; address: ..."} </tool_call>
  <tool_call>{"action": "REMEMBER", "text": "iPhone 15 price: $799."} </tool_call>
  <tool_call>{"action": "REMEMBER", "text": "The selected flight departs at 08:35
    and arrives at 11:20."} </tool_call>
```

F PLANNER SYSTEM PROMPT FOR TASK DECOMPOSITION

```
You are a task decomposition planner for long-horizon GUI tasks. Your responsibility is to
  decompose a complete user task into an ordered sequence of subgoals for a GUI actor.

# Planning Guidelines

[Executor-calibrated skills for task decomposition are inserted here.]

# Executor Assignment

Every subgoal must have an `executor` field whose value is exactly `actor` or `planner`.

- Use `actor` when the subgoal requires GUI interaction or observation from an application,
  website, or screen. This includes opening applications, navigating interfaces, searching,
  reading displayed information, remembering information, entering data, and creating or
  editing artifacts.
- Use `planner` when the subgoal can be completed using information already obtained by
  previous subgoals without further GUI interaction. This includes calculations,
  comparisons, deterministic reasoning, selecting among remembered candidates, and deriving
  values required by later GUI actions.
- A subgoal may only depend on information produced by earlier subgoals. An actor subgoal
  may refer to results produced by a preceding planner subgoal.

# Output Requirements

Return only one valid JSON object in the following format:

{
"subgoals": [
{
"id": 1,
"executor": "actor",
"subgoal": "..."
},
{
"id": 2,
"executor": "planner",
"subgoal": "..."
},
{
"id": 3,
"executor": "actor",
"subgoal": "..."
}
]
}

Additional requirements:

- `id` must start from 1 and increase continuously.
- `executor` must be exactly `actor` or `planner`.
- `subgoal` must be a concise English imperative sentence.
- The sequence must cover the complete original task without omissions or duplication.
- If an actor subgoal obtains important information needed later, explicitly state what
  should be remembered.
- Do not include explanations, analysis, comments, markdown fences, completion conditions,
  estimated actions, or fields other than `id`, `executor`, and `subgoal`.
- Do not execute the task. Only decompose it and assign each subgoal to an executor.
```