UI-MEM: STRATIFIED MEMORY-GUIDED ONLINE REINFORCEMENT LEARNING FOR MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Online Reinforcement Learning (RL) offers a promising paradigm for enhancing mobile GUI agents through direct environment interaction. However, its effectiveness is severely hindered by long horizons and sparse rewards: under group-based methods such as GRPO, sampled rollout groups often collapse into uniform failure, yielding near-zero advantage and little relative learning signal. We propose UI-Mem, a framework that repurposes past experience as a training-time scaffold for online RL. Instead of merely replaying successful trajectories or scoring rollouts after collection, UI-Mem uses stratified memory-guided group sampling: within each GRPO group, retrieved experience is injected at different strengths to produce trajectories with full, partial, or no guidance. These contrasted rollouts create informative reward differences and train the unguided policy to internalize the external experience. UI-Mem instantiates this scaffold with a hierarchical experience memory, where workflows, subtask skills, and failure patterns are stored as parameterized templates and updated online from newly collected successes and failures. Experimental results show that UI-Mem substantially outperforms vanilla RL, dense-reward training, and static experience reuse across model scales, with strong generalization to unseen applications.

1 INTRODUCTION

Recent advances in Multi-modal Large Language Models (MLLMs) (Liu et al., 2023; Bai et al., 2023; Achiam et al., 2023) have enabled their application as autonomous GUI agents (Deng et al., 2023; Zhang et al., 2025a; Qin et al., 2025), capable of interpreting screenshots and generating sequential actions to accomplish user goals. To further enhance the decision-making capabilities of these agents, Reinforcement Learning (RL) (Liu et al., 2025a; Lu et al., 2025b) has emerged as a promising paradigm. In particular, Online RL methods (Bai et al., 2024; Xu et al., 2025) enable agents to learn optimal policies through direct environment interaction, generating diverse rollouts and receiving outcome-based feedback via algorithms like Group Relative Policy Optimization (GRPO) (Shao et al., 2024).

However, the effectiveness of standard online RL (Figure 1(a)) in GUI environments is severely impeded by the combination of long horizons and sparse reward signals. In group-based algorithms such as GRPO, policy improvement depends on reward differences among sampled trajectories. Yet in long-horizon GUI tasks, unguided rollouts rarely reach meaningful intermediate states, causing sampled groups to collapse to uniform failure with near-zero reward variance and little useful gradient. Moreover, even when the agent occasionally discovers partial progress, standard online RL has no mechanism to retain and reuse such procedural experience across related tasks.

Existing approaches only partially address this issue. Data reuse methods (Figure 1(b)), such as Experience Replay (Xu et al., 2025; Lu et al., 2025a), stabilize training by injecting past successful trajectories when the on-policy batch contains mostly failures. However, they fail to generate meaningful trajectories when facing novel tasks, especially early in training when success is rare. Reward refinement methods (Figure 1(c)) (Wang et al., 2023; Feng et al., 2025) introduce step-level dense rewards to provide fine-grained supervision. While they identify which step was correct within the current rollout, they do not enable transfer of reusable experience across different tasks and applications.

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.

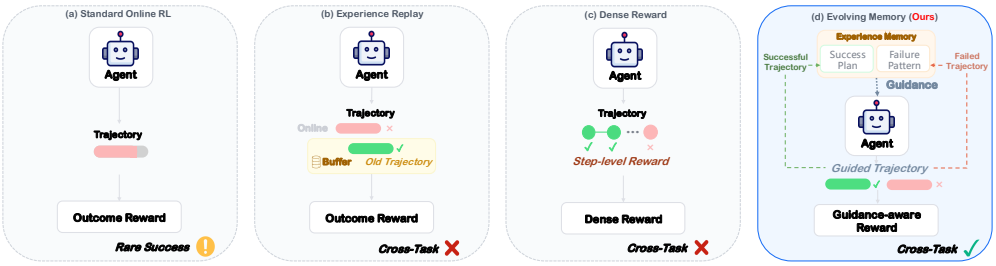


Figure 1: **Comparison of RL paradigms for GUI agents.** (a) Standard online RL suffers from sparse rewards. (b) Experience replay and (c) dense reward address sample efficiency and credit assignment respectively, but both lack mechanisms for cross-task transfer. (d) Our framework uses past experience as a training-time scaffold: retrieved guidance creates diverse rollout outcomes within each GRPO group, enabling effective relative policy optimization and cross-task experience transfer.

Our key insight is that *past experience can be converted into guided exploration signals for online RL*, rather than merely storing it for replay or scoring trajectories. By supplying structured guidance, past experience increases the chance of reaching meaningful intermediate states or successful completions, creating positive anchors in otherwise collapsed rollout groups. However, if such guidance is injected uniformly, the policy may simply follow external instructions without learning to act independently.

Motivated by this, we propose UI-Mem (Figure 1(d)), a framework that uses reusable GUI experience as a training-time scaffold for online RL. The core mechanism of UI-Mem is **stratified memory-guided group sampling**: within each GRPO group, retrieved experience is injected at varying strengths—full guidance, partial guidance, and no guidance. This deliberately creates within-group outcome diversity that trains the unguided policy toward behaviors discovered under guidance. As training progresses, we gradually reduce the amount of guidance through a dynamic curriculum, further shifting the agent from memory-assisted exploration toward independent task solving. To support multi-level guidance, we organize experience into a hierarchical memory of high-level workflows, subtask skills, and failure patterns, stored as parameterized templates that generalize across tasks (e.g., "Send email to `{{recipient}}`" instantiated with current task details). The memory is also updated through a self-evolving loop: successful trajectories are abstracted into new plans and failures yield error diagnoses, enabling the memory to co-evolve with the agent throughout training. Extensive evaluations on online Android GUI benchmarks demonstrate that our method significantly outperforms traditional RL baselines and static data reuse strategies across varying model scales. Furthermore, the generalization to unseen applications demonstrates that our method both internalizes reusable behavior and preserves experience as transferable test-time support.

2 Related Work

Multi-modal GUI Agents. Recent advances in Large Language Models (LLMs) (OpenAI, 2023a;b) and Multi-modal LLMs (MLLMs) (Liu et al., 2023; Bai et al., 2023) have significantly empowered agents to perceive and interact with Graphical User Interfaces (GUIs). Early efforts (Wang et al., 2024; Zhang et al., 2025a) build agent frameworks upon commercial MLLMs and demonstrate promising results in mobile and desktop environments. Recent research has shifted towards fine-tuning open-source MLLMs. For instance, OS-Atlas (Wu et al., 2024) and Aguvis (Xu et al., 2024b) focus on enhancing UI grounding and propose a unified action space for cross-platform generalization. UI-TARS (Qin et al., 2025) further scales up instruction tuning with massive datasets covering element-wise grounding and System-2 reasoning. Despite these successes in Supervised Fine-Tuning (SFT), these agents often struggle with multi-step reasoning and generalization to novel tasks outside their training distribution, which requires further optimization through environment interaction.

Reinforcement Learning for Language and Agents. Reinforcement Learning has emerged as a powerful paradigm for aligning language models with human preferences (Christiano et al., 2017) and improving reasoning capabilities (Schulman et al., 2017; Rafailov et al., 2023) beyond SFT. Recent advancements such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) offer efficient optimization by leveraging group-based advantage estimation without a critic network. In GUI domains, several works (Lu et al., 2025b; Liu et al., 2025a) have explored GRPO-based

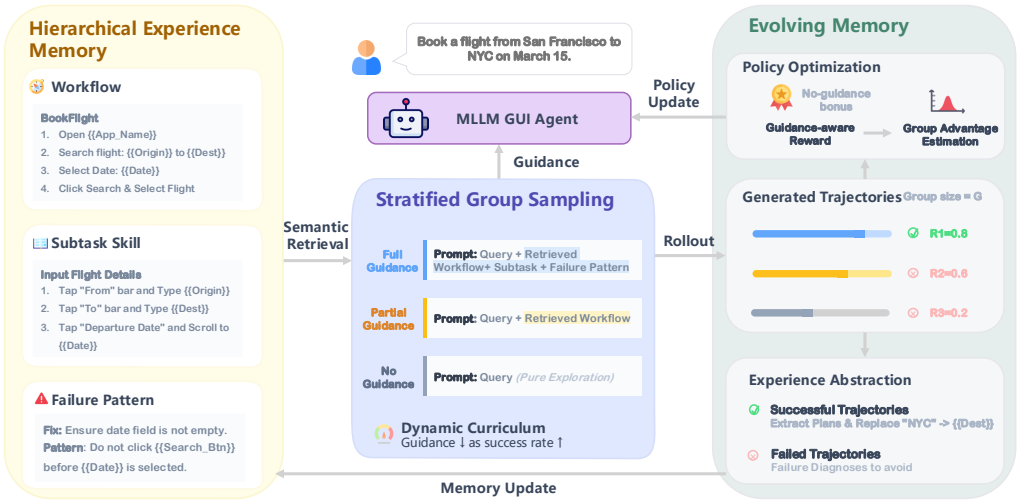


Figure 2: **Overview of UI-Mem.** Given a task instruction, UI-Mem retrieves abstract GUI experience from a hierarchical memory and instantiates it as rollout guidance. The core mechanism is stratified memory-guided group sampling, which constructs trajectories with full, partial, or no guidance within the same GRPO group to create informative reward differences. After policy optimization, newly collected successes and failures are abstracted back into the memory, keeping the training scaffold aligned with the evolving policy.

optimization with rule-based rewards in an offline manner. However, applying online RL to GUIs remains challenging due to extremely sparse rewards. To mitigate this, recent works improve reward design with step-level feedback (Wang et al., 2023; Feng et al., 2025), while others rely on Experience Replay to stabilize training (Bai et al., 2024; Xu et al., 2025; Lu et al., 2025a). Despite their effectiveness, these approaches lack mechanisms to transfer reusable experience across tasks, failing to address the fundamental bottleneck of generating high-quality trajectories in the first place. By contrast, our UI-Mem constructs a hierarchical, evolving experience memory that actively guides online sampling, significantly improving both trajectory quality and training efficiency.

3 METHOD

3.1 OVERVIEW

We formulate GUI interaction as an MDP and adopt GRPO (Shao et al., 2024) as the base optimizer, with formal definitions provided in Appendix A. In sparse-reward GUI tasks, standard GRPO often samples groups in which all trajectories fail, producing near-zero reward variance and little useful gradient signal. As shown in Figure 2, UI-Mem addresses this failure mode by using experience memory as a training-time scaffold. The core mechanism is **stratified memory-guided group sampling** (Section 3.2). By contrasting rollouts under different guidance strengths, the resulting reward differences provide informative relative advantages for policy optimization. To instantiate different guidance levels in GUI environments, we maintain a hierarchical and abstracted experience memory consisting of workflows, subtask skills, and failure patterns (Section 3.3). Finally, the memory is updated online from newly collected successes and failures, keeping the guidance aligned with the evolving policy (Section 3.4).

3.2 STRATIFIED MEMORY-GUIDED GROUP SAMPLING

We utilize experience memory as guidance to increase the chance of reaching meaningful intermediate states or successful completions for sampled trajectories. However, simply prepending the same retrieved guidance to every rollout prompt suffers from the risk of dependency overfitting, where the agent learns to follow external instructions rather than internalizing transferable skills. Furthermore, if strong guidance makes all trajectories in a group successful, the reward variance becomes zero, hindering optimization in GRPO.

To address these challenges, we propose a stratified group sampling mechanism that injects memory guidance at varying strengths within each rollout group. We define three guidance levels $z \in \mathcal{Z} = \{\text{full}, \text{partial}, \text{none}\}$. For each task $q \sim \mathcal{D}$, we retrieve relevant experience $\mathcal{M}_q = \text{Retrieve}(q)$ from the memory and construct guidance prompts $\Gamma_z(q, \mathcal{M}_q)$ for each guidance level z. Given a GRPO group size G, we allocate trajectories according to proportions $\{\lambda_z\}_{z\in\mathcal{Z}}$, where $\sum_{z\in\mathcal{Z}} \lambda_z = 1$.

Let $n_z = \lfloor \lambda_z G \rfloor$ denote the number of trajectories sampled under guidance level z (with rounding adjusted so that $\sum_z n_z = G$). For each z, trajectories are sampled from the old policy under guidance:

$$\tau_i^z \sim \pi_{\theta_{\text{old}}}(\cdot \mid q, \Gamma_z(q, \mathcal{M}_q)), \quad i = 1, \ldots, n_z. \tag{1}$$

The resulting group contains full-guidance trajectories that expose feasible solution paths, partial-guidance trajectories that require the policy to complete execution details, and no-guidance trajectories that estimate the policy's independent competence. Rewards are normalized *across the entire mixed-guidance group*, rather than separately within each guidance level. Let $\mathcal{B}(q) = \{\tau_i^z \mid z \in \mathcal{Z}, i = 1, \ldots, n_z\}$ denote all trajectories sampled for task q, and let $R(\tau_i^z)$ be the trajectory reward. We compute the group-relative advantage as

$$A_i^z = \frac{R(\tau_i^z) - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}} + \epsilon}, \quad \mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\tau\in\mathcal{B}} R(\tau), \quad \sigma_{\mathcal{B}} = \text{Std}_{\tau\in\mathcal{B}}[R(\tau)]. \tag{2}$$

Thus, successful guided rollouts can create positive anchors, while failed or low-progress rollouts receive lower relative advantages. This converts external experience into an optimization signal for group-based RL, with the standard clipped GRPO objective over the stratified group:

$$\mathcal{J}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\, \mathcal{B}(q)\sim\pi_{\theta_{\text{old}}}} \Bigg[\frac{1}{N_{\mathcal{B}}} \sum_{z\in\mathcal{Z}} \sum_{i=1}^{n_z} \sum_{t=1}^{|\tau_i^z|} \min\left(\rho_{i,t}^z(\theta) A_i^z, \text{clip}\left(\rho_{i,t}^z(\theta), 1-\epsilon, 1+\epsilon\right) A_i^z\right) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \Bigg], \tag{3}$$

where $\rho_{i,t}^z(\theta) = \frac{\pi_\theta((\tau_i^z)_t | q, \Gamma_z(q,\mathcal{M}_q), (\tau_i^z)_{<t})}{\pi_{\theta_{\text{old}}}((\tau_i^z)_t | q, \Gamma_z(q,\mathcal{M}_q), (\tau_i^z)_{<t})}$ is the importance ratio, and $N_{\mathcal{B}} = \sum_{z\in\mathcal{Z}} \sum_{i=1}^{n_z} |\tau_i^z|$ denotes the total trajectory length in the stratified group. The key effect of stratification lies in advantage assignment. Under vanilla GRPO, an all-failure group has zero reward variance, so unguided failures receive zero advantage and the unguided policy receives no gradient. When the same group contains a guided success, unguided failures receive negative advantages and occasional unguided successes receive positive ones, so the unguided policy is trained directly with informative signals. Mixing guidance levels also prevents the opposite collapse, in which uniformly guided groups all succeed as the policy learns to follow the guidance (Section 4). As training progresses and task success rates improve, we reduce λ_{full} and increase λ_{none}, shifting optimization from memory-assisted exploration toward independent task solving.

Guidance-Aware Reward. The reward used in the group-relative advantage should reflect not only task success, but also how much external guidance was required to obtain it. We therefore augment the outcome reward with a progress term and a small bonus for unguided success. For trajectory τ_i^z, let $r_{\text{out}}(\tau_i^z) \in \{0, 1\}$ denote the final task success. We compute subtask progress from the reference workflow as

$$r_{\text{prog}}(\tau_i^z) = \frac{|\mathcal{C}_{\text{completed}}(\tau_i^z)|}{|\mathcal{C}_{\text{total}}|}, \tag{4}$$

where $\mathcal{C}_{\text{total}}$ is the set of subtasks in the reference workflow and $\mathcal{C}_{\text{completed}}(\tau_i^z)$ denotes the subset completed by the trajectory. The trajectory reward used in Eq. 2 is then

$$R(\tau_i^z) = \lambda_o r_{\text{out}}(\tau_i^z) + \lambda_p r_{\text{prog}}(\tau_i^z) + \alpha \mathbb{I}[z = \text{none}] r_{\text{out}}(\tau_i^z), \tag{5}$$

where λ_o and λ_p weight outcome and progress rewards, and α controls the unguided-success bonus.

3.3 Hierarchical Experience Memory

To support multi-level guidance injection, we construct a structured, hierarchical, and abstracted experience memory. This design allows the agent to retrieve relevant past experience at different granularities and instantiate it to form specific plans when facing novel tasks.

Memory Representation. Our hierarchical memory contains three types of reusable experience:

- **Workflows ($\mathcal{W}$):** task-level plans that decompose an instruction into ordered subtasks. For example, "Send an email" may correspond to "[Open Mail App, Select Recipient, Type Content, Send]". Workflows provide abstract planning guidance.
- **Subtask Skills (Σ):** execution-level procedures for recurring GUI operations, such as searching for an item, filling a form field, or selecting a file. A skill stores concise action guidance, e.g., "Tap the search icon, type `{{query}}`, then select the top result". Subtask skills provide low-level execution guidance.
- **Failure Patterns ($\mathcal{F}$):** common mistakes extracted from failed trajectories, such as acting before a required field is filled or ignoring a confirmation popup. Failure patterns provide corrective guidance to avoid repeated errors.

Abstraction into Templates. A key observation is that many GUI task instructions share a similar structure but differ only in concrete values such as filenames, dates, phone numbers, or UI element names. We therefore abstract experience into parameterized templates by replacing task-specific values with semantic placeholders. For example, "Click the 'Save' button to confirm creating report.txt" becomes "Click the `{{confirm_button}}` button to confirm creating `{{filename}}`". At rollout time, these placeholders are filled with variables extracted from the current instruction, allowing the same experience to support different tasks and applications.

Guidance Construction. The memory is indexed by embeddings of task and subtask descriptions for semantic retrieval. Given a task instruction q, UI-Mem retrieves relevant workflows, subtask skills, and failure patterns, and instantiates their templates with variables from q. We denote the retrieved experience as

$$\mathcal{M}_q = \{\mathcal{W}_q, \Sigma_q, \mathcal{F}_q\}. \tag{6}$$

When multiple candidates match, we rank them using historical success and usage statistics; details are provided in Appendix F. The retrieved experience $\mathcal{M}_q$ is used to instantiate the guidance constructor $\Gamma_z(q, \mathcal{M}_q)$ introduced in Section 3.2. Specifically, full guidance provides the complete scaffold, including the task-level workflow, executable subtask skills, and relevant failure warnings: $\Gamma_{\text{full}}(q, \mathcal{M}_q) = \text{Prompt}(q, \mathcal{W}_q, \Sigma_q, \mathcal{F}_q)$. Partial guidance removes execution-level details and keeps only the abstract workflow: $\Gamma_{\text{partial}}(q, \mathcal{M}_q) = \text{Prompt}(q, \mathcal{W}_q)$. No guidance removes the external scaffold entirely: $\Gamma_{\text{none}}(q, \mathcal{M}_q) = \text{None}$. Thus, the same memory supports controlled guidance strengths: full guidance exposes feasible solution paths, partial guidance tests whether the policy can execute from an abstract plan, and no guidance measures policy-only competence.

Handling Low-Confidence Retrieval. Memory retrieval provides soft rollout guidance rather than hard constraints. If no exact template match exists, we retrieve the nearest experiences in the embedding space and use them only as optional references in guided rollouts. Because stratified group sampling preserves partial-guidance and no-guidance trajectories in each group, low-confidence retrieval cannot fully determine the training batch. The final update is still governed by environment-derived outcome and progress rewards, allowing irrelevant guidance to receive low advantage while successful unguided or partially guided behavior is reinforced. This design lets UI-Mem learns to ignore guidance noise and degrade toward independent exploration under limited memory coverage, while still enabling analogical transfer from partially related GUI patterns.

3.4 Evolving the Experience Memory

To ensure that the experience memory aligns with the policy as training progresses, UI-Mem updates the memory using trajectories collected by stratified sampling after each training iteration. The policy model is still optimized only through the GRPO objective in Eq. 3.

Experience Extraction. We first use the reward verification module to determine task success and completed subtasks for each trajectory. For successful trajectories, an LLM-based extractor summarizes the task workflow and successful subtask execution plans. For failed trajectories, it retains valid plans for completed subtasks and extracts a failure diagnosis for the first failed subtask. All extracted entries are abstracted into parameterized templates by replacing task-specific values with semantic placeholders, following Section 3.3.

Memory Update. The abstracted entries are merged into the hierarchical memory by semantic similarity. If a matching entry already exists, we update its statistics, such as success count, usage count, or failure recency; otherwise, we create a new template entry. This continual update allows

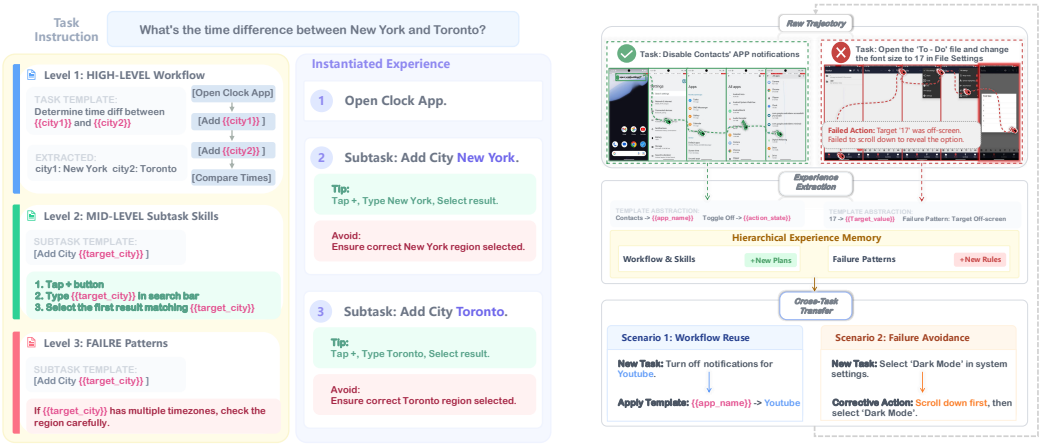


Figure 3: **Memory retrieval and update in UI-Mem.** Left: Retrieved abstract experience is instantiated into task-specific rollout guidance. Right: Newly collected trajectories are abstracted into reusable successes and failure patterns to update the memory.

the memory to incorporate newly discovered strategies and recurring errors without storing raw trajectories.

4 Experiments

4.1 Implementation Details

Training Settings. We integrate our framework with the Qwen3-VL family (Bai et al., 2025a), specifically the 4B and 8B parameter variants. We implement a parallelized online Reinforcement Learning framework. Specifically, we deploy distributed Android emulator instances across CPU clusters to perform asynchronous rollouts. During the sampling phase, the agent interacts with the environment guided by adaptively retrieved experiences to generate trajectories. We employ the Group Relative Policy Optimization algorithm based on our stratified group sampling. For each query, we sample $G = 4$ outputs. We train both models for five epochs with a learning rate of 1×10^{-6}.

Dataset Construction. To construct the training dataset, we collect task instructions from AMEX (Chai et al., 2024), AndroidLab-Instruct (Xu et al., 2024a), and UI-Genie (Xiao et al., 2025), filtering for tasks compatible with core apps in our emulator environment. To increase task diversity, we employ GPT-4o to augment these seed instructions, resulting in a total of $N_{train} = 256$ training queries. Notably, the original datasets provide only high-level task instructions without explicit subtask definitions. For seed tasks with ground-truth trajectories, we generate textual history from screenshots and actions, then prompt DeepSeek-v3 to summarize subtasks. For augmented tasks, we retrieve top-5 similar annotated tasks and use their subtask definitions as few-shot examples. Our manual evaluation (Appendix B) shows this subtask annotation method achieves high accuracy for both task types. We verify that no test task in either benchmark is identical to, or shares a task template with, any training query (Appendix J).

Reward Evaluation and Experience Extraction. A key limitation of prior MLLM-as-a-Judge approaches is that directly processing screenshot sequences often induces severe visual hallucinations. To address this, we adopt a two-stage text-based verification pipeline: we first use Qwen2.5-VL-72B-Instruct (Bai et al., 2025b) to convert each screen state and action into textual descriptions, and then employ DeepSeek-V3 (Liu et al., 2024) to perform rule-based state verification on the resulting text history. This decoupling of visual grounding from logical reasoning significantly improves evaluation reliability (Accuracy: 0.900 vs. 0.724 for direct MLLM scoring). We then use an experience extraction module (Seed1.8 (Seed, 2025a)) to obtain successful plans and failure diagnoses. Detailed designs and validation results are provided in Appendices C and D.

4.2 Evaluation Benchmarks

We evaluate UI-Mem on two challenging online benchmarks that require dynamic interaction:

Table 1: **Performance comparison on the AndroidWorld benchmark.** The table presents closed-source API models, open-source foundation models, and our approach. * denotes inference-time memory retrieval. Best results are in **bold**, and the second-best results are underlined.

Model	Params.	Success Rate
Seed1.5-VL (Guo et al., 2025)	-	62.1
UI-Tars-1.5 (Seed, 2025b)	-	64.2
Gemini-2.5-Pro (Comanici et al., 2025)	-	69.7
Seed1.8 (Seed, 2025a)	-	**70.7**
Qwen3-VL-2B (Bai et al., 2025a)	2B	36.4
MAI-UI-2B (Zhou et al., 2025)	2B	49.1
ScaleCUA-3B (Liu et al., 2025b)	3B	23.7
Ferret-UI Lite-3B (Yang et al., 2025)	3B	28.0
Qwen3-VL-4B (Bai et al., 2025a)	4B	45.3
UI-Mem-4B	4B	58.2
UI-Mem-4B*	4B	**62.5**
UI-Venus-7B (Gu et al., 2025)	7B	49.1
GUI-Owl-7B (Ye et al., 2025)	7B	66.4
Step-GUI-8B (Yan et al., 2025)	8B	67.7
UI-Tars-1.5-7B (Seed, 2025b)	7B	30.0
Qwen3-VL-8B (Bai et al., 2025a)	8B	47.6
UI-Mem-8B	8B	66.8
UI-Mem-8B*	8B	**71.1**

AndroidWorld (Rawles et al., 2024) evaluates agents on 116 programmatic tasks across 20 real-world apps. It is characterized by long-horizon dependencies (often > 10 steps) and dynamic task parameterization, rigorously testing the agent's reasoning and generalization capabilities. Rewards are derived directly from system states, ensuring robust evaluation.

AndroidLab (Xu et al., 2024a) focuses on nine commonly used offline applications. It provides fine-grained metrics beyond binary success, making it ideal for analyzing subtask completion and operation efficiency. The benchmark comprises 138 tasks simulating daily user interactions such as managing events, editing notes, and retrieving information.

4.3 Main Results

Performance on AndroidWorld Table 1 presents the success rates on AndroidWorld. UI-Mem demonstrates consistent improvements across model scales. The UI-Mem-4B model achieves a success rate of 58.2%, significantly outperforming the vanilla Qwen3-VL-4B (45.3%) and even surpassing the larger UI-Venus-7B (49.1%). With inference-time memory retrieval, UI-Mem-4B* further improves to 62.5%, indicating that UI-Mem also learns to utilize external memory. On the 8B scale, UI-Mem-8B* reaches 71.1%; note that proprietary models are evaluated zero-shot, whereas UI-Mem is RL-adapted to the mobile app domain.

Performance on AndroidLab We utilize AndroidLab to analyze fine-grained capabilities using Sub-Goal Success Rate (Sub-SR), Reversed Redundancy Ratio (RRR), Reasonable Operation Ratio (ROR), and overall Success Rate (SR). Results are shown in Table 2. UI-Mem shows a substantial improvement in Sub-SR compared to the vanilla baseline, demonstrating the effectiveness of our hierarchical memory. By explicitly retrieving workflows and subtask plans, the agent can decompose complex long-horizon tasks into manageable segments, thereby reducing error propagation. Notably, UI-Mem-8B outperforms MobileRL (42.5%), a strong baseline that utilizes static experience replay. While MobileRL relies on the reuse of successful trajectories, UI-Mem employs active memory to guide the generation of high-quality new trajectories with diverse progress rewards.

4.4 Ablation Study

Comparison with RL Training Paradigms. We compare UI-Mem with several RL baselines in Table 3: (1) Vanilla GRPO without external memory; (2) GRPO + Progress Reward with subtask-level rewards; (3) Inference-time Prompting using our memory as context; (4) GRPO + Experience Replay mixing past successful trajectories into training; and (5) GRPO + Inference-time Prompting. Vanilla

Table 2: **Performance comparison on the AndroidLab benchmark.** The table presents closed-source API models, open-source foundation models and our approach. ⋆ denotes inference-time memory retrieval. The best results are in **bold**, and the second-best are underlined.

Model	Params	Sub-Goal Success Rate	Reversed Redundancy Ratio	Reasonable Operation Ratio	Success Rate
Gemini-1.0	–	12.6	72.5	76.7	10.9
Claude-3-Opus	–	15.1	81.4	83.9	13.0
Gemini-1.5-Pro	–	18.5	106.0	**91.5**	16.7
Claude-3.5-Sonnet	–	32.7	**113.4**	81.2	29.0
GPT-4o	–	**35.0**	87.3	85.4	31.2
AutoGLM	–	–	–	–	**36.2**
UI-Genie-Agent	3B	35.4	91.3	90.6	28.8
Qwen3-VL-4B	4B	48.2	**93.0**	90.5	37.0
UI-Mem-4B	4B	49.5	**93.0**	93.5	37.7
UI-Mem-4B⋆	4B	**51.9**	89.2	**94.6**	**39.9**
LLaMA3.2-11B-Vision	11B	13.0	61.7	87.9	10.1
CogVLM2-ft	19B	16.1	57.4	85.6	11.6
Qwen2.5-VL	7B	18.7	70.6	76.8	14.9
UI-Genie-Agent	7B	46.3	92.8	91.4	38.7
Qwen3-VL-8B	8B	45.3	**97.7**	91.8	34.8
UI-TARS-1.5-7B	7B	49.4	84.2	92.5	40.6
MobileRL	7B	-	-	-	42.5
UI-Mem-8B	8B	52.7	93.9	90.9	43.5
UI-Mem-8B⋆	8B	**56.0**	89.8	**94.9**	**44.9**

Table 3: **Comparison of different RL training paradigms on AndroidWorld.**

Method	**Mechanism**	**SR (%)**
Qwen3-VL-8B	Baseline	47.6
Vanilla GRPO	Online RL	53.0
GRPO + Progress Reward	Online RL	57.3
Inference-time Prompting	RAG (No Training)	52.6
GRPO + Experience Replay	Data Reuse	56.5
GRPO + Inference-time Prompting	Online RL+RAG	55.2
UI-Mem (Ours)	**Memory-Guided RL**	**66.8**

GRPO struggles with exploration in the vast GUI state space. Dense rewards improve performance (57.3%) but cannot transfer experience to novel tasks. Experience Replay improves training stability (56.5%) but only replays static data without guiding exploration. Inference-time Prompting yields limited gains for both the baseline and GRPO-trained models, suggesting that context alone is insufficient without training the model to utilize it. In contrast, UI-Mem achieves the highest success rate by combining memory-guided exploration during training with adaptive retrieval.

Computational Overhead. We measure the wall-clock overhead introduced by memory-related operations and find it to be modest. Under the same rollout budget, UI-Mem increases the per-iteration time from 8.4 to 8.8 minutes; memory retrieval, experience extraction, abstraction, and memory update account for 9.1% of the total time, with environment interaction remaining the dominant cost. The full breakdown is provided in Appendix I.

Ablation of Stratified Memory-Guided Training. Figure 4(a) evaluates the contribution of core mechanisms in UI-Mem. Removing the hierarchical experience organization drops the success rate to 59.5%, showing that structured memory is important for composing reusable task knowledge. Replacing abstracted memory with raw experience degrades performance to 58.2%, proving that abstraction is essential for generalization. Disabling the self-evolving loop reduces scores to 62.9%, highlighting the need for continuous memory refinement. Finally, removing stratified sampling causes the largest drop to 48.3% due to over-reliance on memory; our strategy effectively balances guidance with internalized reasoning.

Cross-Application Generalization. While our main benchmarks already evaluate out-of-domain tasks, we conduct a stricter generalization test on five held-out apps from AndroidLab unseen during training: Bluecoins, Cantook, Maps.me, Pi-Music, and Zoom. Figure 4(b) compares the baseline, our

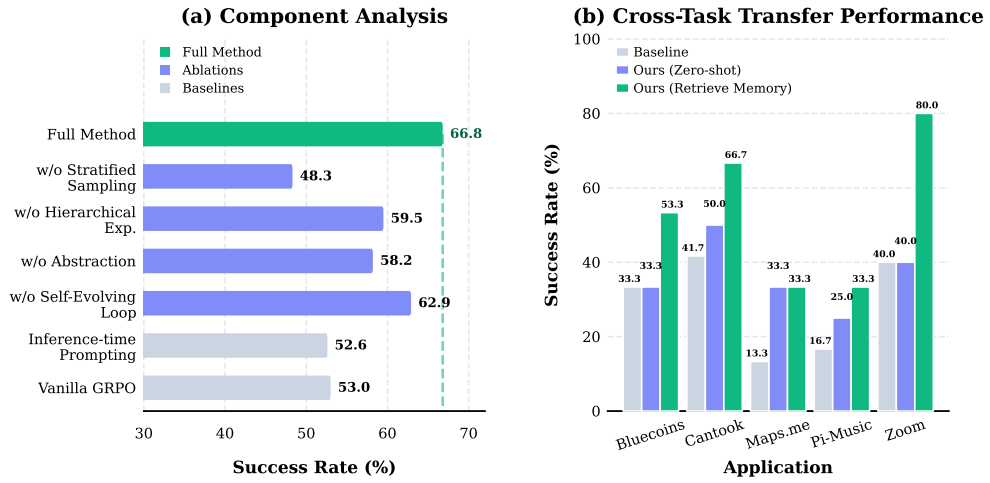


Figure 4: **Analysis of UI-Mem's design choices and cross-task transfer ability.** (a) Component analysis of core algorithmic mechanisms. (b) Cross-task transfer performance across applications.

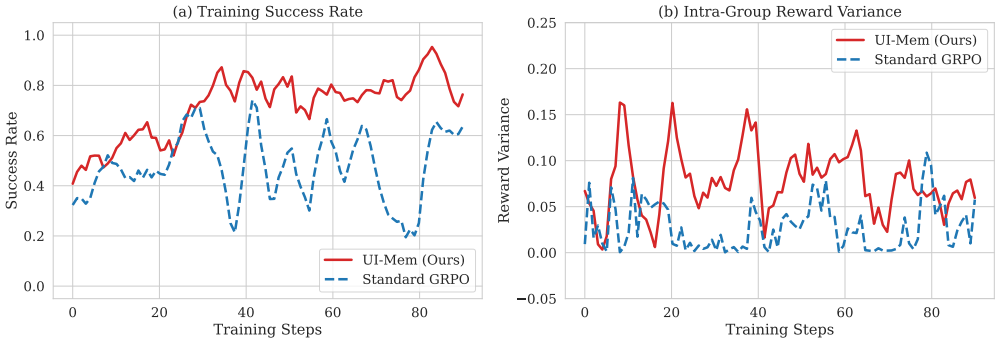


Figure 5: **Training dynamics comparison between UI-Mem and standard GRPO.** (a) Success rate over training steps. (b) Mean intra-group reward variance, which directly impacts the effectiveness of advantage estimation.

method in a zero-shot manner (without memory retrieval), and full method with retrieved memory. Our zero-shot model already outperforms or matches the baseline across all five apps, demonstrating improved generalization ability. Notably, incorporating retrieved memory at inference provides consistent additional gains, particularly on Bluecoins and Zoom, indicating that experiences from training apps effectively transfer to novel applications through our retrieval mechanism.

Mitigating Reward Collapse. We analyze whether stratified sampling alleviates the reward-collapse failure mode in Figure 5. As shown in Figure 5(a), standard GRPO shows unstable progress, while UI-Mem improves more steadily and converges faster. Figure 5(b) measures intra-group reward variance, which determines whether GRPO can compute meaningful relative advantages. Under sparse GUI rewards, vanilla GRPO frequently samples groups where all trajectories fail and receive identical rewards, causing the variance to collapse to zero. In contrast, stratified memory-guided sampling mixes full-, partial-, and no-guidance trajectories within each group, reducing zero-variance collapse and preserving informative reward differences for policy optimization.

5 Conclusion

We introduced UI-Mem, which turns reusable GUI experience into a training-time scaffold for online RL of mobile GUI agents. Stratified memory-guided group sampling mixes full-, partial-, and no-guidance rollouts within each GRPO group to restore informative relative advantages, while a hierarchical, self-evolving template memory supplies the guidance. Experiments on two online Android benchmarks show substantial gains over vanilla RL baselines and generalization to unseen applications.

AI USE STATEMENT

We use LLMs as part of the methodology for reward verification, experience extraction, task augmentation, and memory abstraction. We describe these uses in Section 4.1 and provide additional details in Appendices C and D. Additionally, we use LLMs to check grammar and polish our phrasing of our paper.

REPRODUCIBILITY STATEMENT

Section 4.1 and Table 7 list the main training, rollout, and evaluation configuration, with compute cost and multi-seed results in Appendix I. Section 3.2 and Appendix A give the training algorithm, and Algorithms 1 and 2 specify memory retrieval and update. Appendices C and D provide all prompts and validation results for the auxiliary modules, together with open-source replacements in Appendix J. Section 4.1 and Appendices B and J describe the training-query construction and the train–test decontamination. Code, training queries, and the memory pool will be publicly released.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *ArXiv*, abs/2308.12966, 2023.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, 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.

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

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, et al. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*, 2025.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025a.

Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*, 2025b.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025a.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025b.

OpenAI. Chatgpt. `https://chat.openai.com`, 2023a.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023b.

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.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

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

Bytedance Seed. Seed1. 8 model card: Towards generalized real-world agency. 2025a.

ByteDance Seed. Ui-tars-1.5. `https://seed-tars.com/1.5`, 2025b.

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

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

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

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, et al. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. *arXiv preprint arXiv:2505.21496*, 2025.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*, 2024a.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *arXiv preprint arXiv:2509.18119*, 2025.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024b.

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, et al. Step-gui technical report. *arXiv preprint arXiv:2512.15431*, 2025.

Zhen Yang, Zi-Yi Dou, Di Feng, Forrest Huang, Anh Nguyen, Keen You, Omar Attia, Yuhao Yang, Michael Feng, Haotian Zhang, et al. Ferret-ui lite: Lessons from building small on-device gui agents. *arXiv preprint arXiv:2509.26539*, 2025.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

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*, pp. 1–20, 2025a.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

A PRELIMINARIES

GUI Task Formulation. We formalize the interaction between the MLLM agent and the Graphical User Interface (GUI) as a finite-horizon Markov Decision Process (MDP), represented by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, H \rangle$. The state space $\mathcal{S}$ captures the multi-modal context; a state $s_t \in \mathcal{S}$ comprises the pixel-level screenshot I_t and the high-level user instruction q. The action space $\mathcal{A}$ consists of atomic GUI operations (e.g., `Click`, `Swipe`, `Type`, `Home`). At timestep t, the agent policy $\pi_\theta(a_t|s_t, q)$ generates an action a_t, transitioning the environment to s_{t+1} according to the system dynamics $\mathcal{P}(s_{t+1}|s_t, a_t)$. Reflecting the sparse nature of real-world GUI tasks, the reward function is defined as $\mathcal{R}(\tau) = r_{outcome} \in \{0, 1\}$, provided only at the termination of the trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T)$, where $T \leq H$.

Group Relative Policy Optimization (GRPO). We adopt GRPO as our base optimization algorithm, as it eliminates the need for a separate critic model. GRPO samples a group of G trajectories $\{\tau_i\}_{i=1}^G$ for a task instruction q from the current policy $\pi_{\theta_{old}}$. The advantage A_i for each trajectory is computed by normalizing rewards within the group:

$$A_i = \frac{\mathcal{R}(\tau_i) - \text{mean}(\{\mathcal{R}(\tau_j)\}_{j=1}^G)}{\text{std}(\{\mathcal{R}(\tau_j)\}_{j=1}^G) + \epsilon} \tag{7}$$

where ϵ is a small constant for numerical stability. The policy is updated by maximizing the following objective:

$$\begin{aligned} \mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{c\sim\mathcal{D},\{\tau_i\}\sim\pi_{\theta_{old}}} \Bigg[\frac{1}{G}\sum_{i=1}^{G} \min\Bigg(\frac{\pi_\theta(\tau_i)}{\pi_{\theta_{old}}(\tau_i)} A_i, \\ \text{clip}\left(\frac{\pi_\theta(\tau_i)}{\pi_{\theta_{old}}(\tau_i)}, 1-\delta, 1+\delta \right) A_i \Bigg) - \beta \mathbb{D}_{KL}(\pi_\theta || \pi_{ref}) \Bigg] \end{aligned} \tag{8}$$

where δ is the clipping hyperparameter and β controls the KL-divergence penalty from the reference model π_{ref} to prevent policy collapse.

B HIERARCHICAL MEMORY DETAILS

Accuracy of subtask definition. Our training queries include seed tasks (with GT trajectories) and augmented tasks (without GT trajectories). For seed tasks, we generate textual history from screenshots and actions, then prompt DeepSeek-V3 to summarize subtasks. For augmented tasks, we retrieve top-5 similar annotated tasks and use their subtask definitions as few-shot examples. To evaluate the accuracy of this annotation method, we perform manual check on 200 samples of the generated subtask definitions. As presented in Table 4, our subtask definitions achieve high accuracy for both task types, confirming effective generalization even without GT trajectories.

Table 4: Manual evaluation of subtask definition accuracy for seed and augmented tasks.

Task Type	Source	Has GT Traj.?	Subtask Acc.
Seed tasks	AMEX, AndroidLab-Instruct, UI-Genie	Yes	**96.5%**
Augmented tasks	Seed tasks	No	94.0%

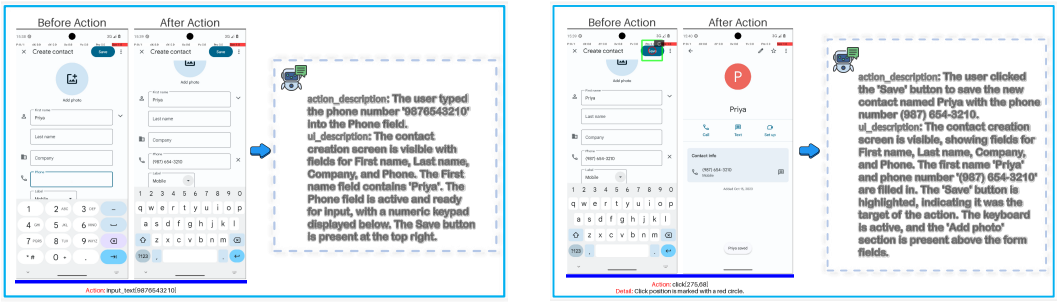
Example. We provide a detailed example of our hierarchical memory representation structure in Figure 6. Beyond standard task instructions, we introduce an `essential_states_template`, which defines the key intermediate states that characterize the valid completion of the overall task, thereby facilitating our reward evaluation based on rule-based state verification.

```
[
  {
    "task_id": "AutoGenerated_Task_75",
    "package_name": ["net.gsantner.markor"],
    "task_template": {
      "content": "Rename file {{current_filename}} to {{new_filename}} in Markor.",
      "parameter_config": {
        "fixed_parameters": {
          "app_package": "net.gsantner.markor",
          "rename_icon_name": "A",
          "confirm_button_name": "OK"
        },
        "variable_parameters": ["current_filename", "new_filename"]
      }
    },
    "essential_states_template": {
      "S1": { "content": "Markor app is open displaying the file list." },
      "S2": {
        "content": "File '{{current_filename}}' is selected with context options
            visible.",
        "variable_mapping": { "current_filename": "current_filename" }
      },
      "S3": { "content": "Rename dialog is active." },
      "S4": {
        "content": "File is renamed to '{{new_filename}}'.",
        "variable_mapping": { "new_filename": "new_filename" }
      }
    },
    "subtask_template": {
      "T1": {
        "subtask_label": "Select Note via Long Press",
        "content": "Long press the file named '{{current_filename}}'."
      },
      "T2": {
        "subtask_label": "Tap Rename Icon",
        "content": "Tap the '{{icon_name}}' icon to open rename options."
      },
      "T3": {
        "subtask_label": "Enter Text and Confirm",
        "content": "Enter '{{new_filename}}' and tap '{{button_name}}' to confirm."
      }
    }
  }
]
```

Figure 6: **Example of Hierarchical Data Structure.** The `essential_states_template` defines the critical milestones of the task, providing a mechanism for state-based verification rather than directly judging subtask or task completion.

C Reward Model Details

Reward Model Design. We identify a significant limitation in previous MLLM-as-a-Judge approaches: directly feeding the entire history of screenshots often leads to severe hallucination issues. To address this, we propose a text-based verification approach. Our method operates in two stages. First, we utilize Qwen2.5-VL-72B-Instruct to generate textual descriptions for each screen state and the corresponding executed actions in the trajectory. As illustrated in Figure 7, by analyzing paired screenshots captured before and after an action, the model provides precise descriptions of the UI elements relevant to the task instruction, as well as the specific operation performed. These descriptions are then concatenated to form a textual history of the agent's actual actions. After that, we feed this textual history into an open-source LLM, DeepSeek-V3, which is prompted to identify which essential states are completed based on the UI descriptions and action descriptions. This effectively converts the previous visual-based evaluation into a robust rule-based verification process, which we have found to be significantly more accurate than direct MLLM evaluation. Based on the model response, we calculate a precise progress reward $\mathcal{R}_{progress} = |\mathcal{S}_{completed}|/|\mathcal{S}_{total}|$, offering a dense signal for optimization. By grounding reasoning in verified state changes rather than raw pixels, this approach significantly reduces the rate of hallucination common in pure MLLM-based evaluators. We provide our prompt for the LLM verification model in Figure 8.



(a) Text Input Operation (b) Button Click Operation

Figure 7: Textual Description Examples of UI states and executed actions. By comparing *Before Action* and *After Action* screens, the MLLM provides (1) `ui_description`, capturing the visual state and element visibility relevant to the current task, and (2) `action_description`, specifying the actual operation executed (e.g., typing text, clicking buttons). (a) shows a **Text Input** scenario where the user types a phone number, capturing the input field state change. (b) illustrates a **Button Click** operation (saving a contact), identifying the interactable element. These structured descriptions serve as grounded inputs for the Reward Computation.

Table 5: Reward Model Accuracy Comparison. UI-Mem Text-based Verification based on open-source models (Qwen2.5-VL-72B-Instruct + DeepSeek-V3) achieves competitive performance compared with strong closed-source API models across all metrics.

	Model	Accuracy	Precision	Recall	F1 Score
MLLM-based	Gemini 2.5 Pro	0.776	0.845	0.908	0.837
	Seed1.5-VL	0.639	0.879	0.770	0.775
	Qwen2.5-VL-72B-Instruct	0.724	0.841	0.845	0.811
UI-Mem (Text-based)	Gemini 2.5 Pro + DeepSeek-V3	**0.922**	0.912	0.967	**0.916**
	Seed1.5-VL + DeepSeek-V3	0.880	0.907	0.921	0.887
	Qwen2.5-VL-72B-Instruct + DeepSeek-V3	<u>0.900</u>	0.909	0.943	<u>0.902</u>

Reward Model Evaluation. We quantitatively assess the accuracy of our reward computation method and compare it against standard MLLM-based evaluators. Table 5 reports the prediction accuracy, Precision, Recall, and F1 Score across different backbone models. The baseline MLLM-based approaches often struggle with visual hallucinations, resulting in suboptimal F1 scores (e.g., 0.837 for Gemini 2.5 Pro). In contrast, our proposed UI-Mem framework, which employs a two-stage text-based verification pipeline, consistently achieves superior performance. Notably, the Qwen2.5-VL-72B-Instruct+DeepSeek-V3 setting reaches an F1 score of 0.902, significantly outperforming the closed-source API model Gemini 2.5 Pro. This demonstrates that decoupling visual grounding from logical verification effectively mitigates hallucinations, providing a robust and reliable reward signal for reinforcement learning. We adopt this setting for our reward evaluation during online training.

D Experience Extraction Details

We employ an experience extraction model, specifically Seed1.8, due to its strong reasoning abilities, to derive both success plans and failure patterns from newly generated trajectories. Based on the evaluation results provided by our reward model, this extraction process is divided into two distinct phases: Subtask-Level Experience Extraction and High-level Workflow Extraction.

Subtask-Level Experience Extraction. The model first generates experience for each subtask by analyzing the trajectory. For each completed subtask, the model abstracts detailed success plans. A critical constraint imposed is *Entity Preservation*: specific details (e.g., "call 1234") are preserved to ensure subsequent experience abstraction can accurately map them to template variables. For failed

System instruction

You are an expert in evaluating mobile UI automation tasks. Your goal is to determine if a user's task was successfully completed based on a trajectory of actions and a list of expected essential states.
Evaluation Guidelines
1. **Primary Goal - Task Success**: Your most important judgment is whether the final goal described in *Task Description* was achieved. **If the final outcome in the trajectory shows the task is done, is_task_completed MUST be true, even if some intermediate essential states were skipped or not explicitly captured.**
2. **Essential States Analysis**: Identify which *essential_states* were visited. Note that a user might take shortcuts or the trajectory might miss capturing a specific intermediate state. This is acceptable as long as the final result is correct.
3. **Evidence Based**: Base your decision on the provided UI description and action summary.
4. **Output Logic**:
- If the task goal is achieved → is_task_completed: true (regardless of whether all essential states list is full).
- If the task goal is NOT achieved →is_task_completed: false.

User prompt

Task Description: {task_description}
The expected essential states (standard path) are: {essential_states}.
Total Steps in Full Trajectory: {total_steps}
Here is a summary of all the {total_steps} actions taken:
{formatted_steps_str}

Your output MUST be a strict JSON object containing the following keys:
- completed_essential_state: A list of keys representing the essential states that **were actually observed** in the trajectory.
- is_task_completed: A boolean (true/false). Set this to true if the Task Description implies the user's goal is met at the end of the trajectory, even if the list of completed states is incomplete.
- reasoning: Explain your judgment. If is_task_completed is true but some states are missing, explain that the goal was achieved via an alternative path or the state was skipped.
Example Output format

```json
{{
  "completed_essential_state": ["S1", "S5"],
  "is_task_completed": true,
  "reasoning": "Although intermediate states S2-S4 were not explicitly observed, the final screenshot shows the settings were successfully changed as requested, so the task is completed."
}}
```

Figure 8: Prompt design for the LLM verification Model.

subtasks (derived from the first uncompleted subtask id), the model gives analysis about the error root cause and targeted correction guidelines. We provide the used prompt in Figure 9).

High-level Workflow Extraction. For successful trajectories, we additionally extract the high-level workflow which captures global planning. This process maps the raw agent trajectory to a sequence of subtasks. We provide the used prompt in Figure 10).

E Experience Abstraction Details

Experience Parameterization. To transition from task-specific experience to generalizable knowledge, we implement an Experience Parameterization module using **DeepSeekV3**. This module

System instruction

You are an expert in analyzing mobile UI automation agents.
Your goal is to extract 'Rules of Experience' based on an execution trajectory.
You will be given the results of a trajectory analysis, indicating which subtasks succeeded and which failed.

User prompt

Task Context
Goal: '{task_description}'
Subtasks: {all_subtasks}

Trajectory Summary
{formatted_steps_str}

Analysis Results (Pre-computed)
Completed subtasks: {completed_subtasks}
Failed subtasks: {failed_subtasks}

Your Task
Generate 'Experience' for the successful subtasks and the failed subtasks.
1. **Success Experience**: For every completed subtask, extract a clear, step-by-step execution plan.
- **CRITICAL**: If the step involves interacting with a specific name, number, or text found in the 'Goal', you MUST include that exact value in your summary (e.g., 'Click on contact Alice' instead of 'Click on the contact').
2. **Failure Analysis**: For the *first* failed subtask encountered (the bottleneck), explain WHY it failed and HOW to avoid it.

Output Format
Return a JSON object:
```json
{
  "success_experiences": [
    { "subtask_id": "T1", "plan_summary": "Locate the conversation entry corresponding to '+123456' on the main screen and tap it..." },
    { "subtask_id": "T2", "plan_summary": "Type 'Hello World' into the text box..." }
  ],
  "failure_diagnosis": {
    "first_failed_subtask_id": "S3",
    "root_cause": "...",
    "correction_guideline": "..."
  }
}
```

Figure 9: Prompt design for Experience Extraction for subtask-level skills and failure analysis. This module identifies the critical `success_experiences` for completing a subtask or analyzes the `first_failure_subtask_id` to generate targeted error diagnosis.

transforms obtained concrete experiences into abstracted experiences containing variables, enabling the transfer of learned strategies to new tasks. The input prompt is presented in Figure 11.

Experience Ranking Mechanism. To facilitate the selection of the most effective abstract knowledge, we implement a scoring strategy. For workflows or subtask skills, we employ an Upper Confidence Bound (UCB) algorithm: $Score = \frac{N_{success}}{N_{total}} + c\sqrt{\frac{\ln N_{global}}{N_{used}}}$. This balances the *exploitation* of high-success strategies with the *exploration* of newer, less-tested plans. For *Failure Diagnoses*, we employ a time-decay function $R(t) = \frac{1}{1+\lambda\Delta t}$ derived from its last updated time. This ensures

System instruction

You are an expert GUI agent analyst. Your goal is to reverse-engineer the
High-Level Execution Plan from a successful agent trajectory.

The Concept
An 'Execution Plan' is strictly a sequence of Essential State IDs (e.g., ['S1', 'S3', 'S2'])
that represents the logical order in which key milestones were achieved.

Inputs provided to you
1. **Task Instruction**: What the agent was trying to do.
2. **Essential State Definitions**: A dictionary mapping IDs (S1, S2...) to state descriptions.
3. **Agent Trajectory**: The actual actions taken.
4. **Verified States**: Which states the Reward Model confirms were triggered.

Your Task
Analyze the connection between the Trajectory and the Essential State Definitions.
Determine the exact order in which the Essential States were achieved.

Guidelines
- **Strict Vocabulary**: You must ONLY use the provided State IDs (S1, S2, etc.). Do not invent new steps.
- **Order Matters**: The list must reflect the chronological order of achievement.
- **Noise Filtering**: If the agent performed useless actions between S1 and S2, ignore them. Only capture the state transitions.
- **Missing States**: If a state was defined in the template but NOT achieved in the trajectory, do NOT include it.
- **Rationale**: Provide a brief explanation of why this order was chosen (e.g., 'S3 had to be done before S2 because...').

Output Format
Respond with a JSON object:
```json
{
  "execution_plan": ["S1", "S3", "S2"],
  "rationale": "The agent first opened the dialog (S1), then immediately solved the puzzle (S3) within the dialog, and finally confirmed the app selection (S2).
}
```

User prompt

Task Instruction
'{instruction}'

Essential State Definitions (Vocabulary)
{states_desc_str}

Verified States (from Reward Model)
{completed_states_sequence}

Agent Trajectory
{history_block}

Perform Analysis
Based on the trajectory and verified states, reconstruct the sequence of Essential States.

Figure 10: Prompt design for High-Level Workflow Extraction. The system extracts a concrete execution plan from successful traces.

the agent prioritizes recent error, which is essential for adapting to dynamic UI updates and policy evolving.

Experience Extraction Evaluation. We provide a quantitative evaluation of the accuracy and reliability of LLM-based experience extraction. We manually check the correctness of extracted workflows, subtask skills and failure patterns, each of 100 samples respectively. As shown in Table 6, our LLM-based extraction achieves 92% accuracy for workflows, 86% for subtask skills, and 88% for failure patterns. This high accuracy is achieved because extraction operates on already-verified task/subtask completion signals from our two-stage verification, rather than on raw trajectories.

System instruction

You represent a system that generalizes specific automation rules into template rules.
Your task is to replace specific values with their corresponding variable placeholders based on the provided bindings.
You must output valid JSON.

User prompt

Input Data
1. **Specific Plan**: "{extracted_plan_summary}"
2. **Variable Bindings**: {json.dumps(slot_bindings)}

Instruction
Rewrite the 'Specific Plan' by replacing any occurrence of the values found in 'Variable Bindings' with their dictionary keys wrapped in double curly braces (e.g., {{{{key}}}}).
Do not change the structure of the sentence, only swap the values for variables.

Output Format
Return a single JSON object with the key \"generalized_plan\". Do not include markdown formatting (like ```json).

Example
Input Plan: "Search for 'Starbucks' in the map bar and select the one on '5th Avenue'.
Bindings: {{ "search_term": "Starbucks", "target_street": "5th Avenue" }}
Output: {{ "generalized_plan": "Search for {{{{search_term}}}} in the map bar and select the one on {{{{target_street}}}}." }}

Your Output

Figure 11: Prompt design for Experience Abstraction. This prompt aims to map concrete entities (e.g., specific cities) to template slots without modifying the original semantic logic.

Table 6: Accuracy and robustness of our LLM-based experience extraction.

Method	Workflow Acc. (%)	Skill Acc. (%)	Failure Acc. (%)	Task SR (%)	Task SR (+retrieval)
Baseline	-	-	-	47.6	52.6
UI-Mem (full)	92	86	88	**66.8**	**71.1**
+10% noise	-	-	-	65.1	68.9
+20% noise	-	-	-	64.7	67.2

Moreover, to test its robustness to extraction errors, we use an LLM to inject controlled noise into 10% and 20% of the extracted experiences. As shown in Table 6, task success rate degrades only marginally, demonstrating the framework's robustness. This stems from our experience abstraction and stratified sampling, which prevent over-reliance on any single extracted experience.

F THE MEMORY RETRIEVAL ALGORITHM

As presented in **Algorithm 1**, our memory retrieval procedure consists of several hierarchical phases:

Task Matching and Variable Extraction. We first encode the raw instruction into semantic embeddings, using an embedding model Qwen3-Embedding-8B (Zhang et al., 2025b). We use the embedding model to select top-k similar task templates, which are further processed by an LLM (DeepSeek-V3) to determine the best match template and extract the task-specific variables $\mathcal{V}$.

High-Level Workflow Retrieval and Instantiation. Upon identifying the template, the system retrieves a corresponding abstract workflow from High-level Memory. Since the raw experience is

Algorithm 1 Hierarchical Experience Retrieval & Instantiation

Require: Current instruction I, Task Meta-info $\mathcal{T}_{task}$, High-level Memory $\mathcal{M}_{high}$, Mid-level Memory $\mathcal{M}_{mid}$, Embedding Model $E(\cdot)$.
Ensure: Structured Guidance G (Plan + Tips/Warnings).

PHASE 1: TASK MATCHING & VARIABLE EXTRACTION
$e_I \leftarrow E(I)$
Retrieve top-k candidates $\mathcal{C} \subset \mathcal{T}_{task}$ via $\text{COSINESIM}(e_I, E(\tau))$ for $\tau \in \mathcal{T}_{task}$
$match, \tau^*, V \leftarrow \text{LLM_DECIDEMATCH}(I, \mathcal{C})$ ▷ Return matched template τ^* & variables V
if not $match$ **then**
  $\tau^* \leftarrow \arg\max_{\tau \in \mathcal{C}} \text{SCORE}(\tau)$ ▷ Fallback to best analogy
  $V \leftarrow \text{EXTRACTVARS}(I, \tau^*)$
end if

PHASE 2: WORKFLOW RETRIEVAL
$P_{raw} \leftarrow \text{GETBESTPLAN}(\mathcal{M}_{high}, \tau^*.id)$ ▷ Retrieve execution plan with highest success rate
$P_{filled} \leftarrow \emptyset$
for each subtask step $s_{raw} \in P_{raw}$ **do**
  $s_{filled} \leftarrow \text{INSTANTIATE}(s_{raw}, V)$ ▷ Fill placeholders (e.g., {{app}} → "Gmail")
  P_{filled}.append(s_{filled})
end for

PHASE 3: SUBTASK-LEVEL EXPERIENCE RETRIEVAL
$G \leftarrow$ InitializeGuidance(P_{filled})
for each subtask $s \in P_{filled}$ **do**
  if $s.label \in \mathcal{M}_{mid}$ **then**
    $E \leftarrow \mathcal{M}_{mid}[s.label]$
  else
    $E \leftarrow$ best match via $\text{COSSIM}(E(s.content), \mathcal{M}_{mid}.keys)$
  end if
  $E_{tips} \leftarrow \text{UCB_RANK}(E.\text{plan_summary})$ ▷ Prioritize using success rate + exploration
  $E_{warn} \leftarrow \text{TIMEDECAY_RANK}(E.\text{failure_diagnosis})$ ▷ Prioritize recent failures
  G.append($\{s, E_{tips}, E_{warn}\}$)
end for
return G

parameterized, the system instantiates it by injecting the variable set $\mathcal{V}$ into predefined placeholders, transforming the workflow into a context-specific, executable plan.

Mid-Level Experience Retrieval. After that, the instantiated plan is enriched with subtask-level guidance and failure correction guidelines. For each pending subtask, the retrieval module queries the memory pool; using semantic similarity, similar to the task template match. This step aggregates historical success plans and failure diagnoses into the final structured guidance.

A retrieved task template may be associated with multiple experience entries accumulated from past explorations. To balance exploitation and exploration, we employ a scoring mechanism inspired by the Upper Confidence Bound (UCB) algorithm. For a candidate plan p, its retrieval score $S(p)$ is:

$$S(p) = \frac{N_{succ}(p)}{\sum_{p' \in \mathcal{P}} N_{succ}(p')} + \lambda_{ucb} \sqrt{\frac{\ln(\sum_{p' \in \mathcal{P}} N_{used}(p'))}{N_{used}(p) + 1}} \tag{9}$$

where N_{succ} and N_{used} denote the success count and usage count respectively, and λ_{ucb} is the exploration weight. The first term encourages the selection of plans with high historical success rates, while the second term provides an exploration bonus for plans that have been tried fewer times. For failure patterns, we use a recency bias to prioritize recent error diagnoses, ensuring that the guidance reflects the agent's latest policy behavior. This adaptive retrieval mechanism ensures that the guidance provided to the agent evolves dynamically as the memory is refined through training.

G THE MEMORY UPDATE ALGORITHM

The memory update mechanism (Figure 3 right) is the core of our self-evolving loop, deriving structured, long-term knowledge from raw trajectories. As demonstrated in **Algorithm 2**, this process ensures that the agent learns effectively from past successful and failed trajectories. The procedure consists of the following phases:

Algorithm 2 Self-Evolving Experience Abstraction & Update

Require: Instruction I, Trace $\mathcal{T} = \{(S_i, \text{bindings}_i)\}$, Feedback $\mathcal{F}$ (Success/Correction), Global Rate SR.

Input: Executed trace with instantiated templates and variable bindings.

PHASE 1: GLOBAL STATISTIC UPDATE (EMA)

$SR(I) \leftarrow \gamma \cdot SR(I) + (1 - \gamma) \cdot \text{Score}(\mathcal{T})$ ▷ Update moving avg success rate

PHASE 2: MID-LEVEL MEMORY UPDATE (SUBTASK EXPERIENCE)

for each subtask $S_i \in \mathcal{T}$ containing feedback in $\mathcal{F}$ **do**
  $key \leftarrow (S_i.\text{pkg}, S_i.\text{label})$
  $exp_{raw} \leftarrow \mathcal{F}.\text{get_content}(S_i)$ ▷ Get summary if success, diagnosis if fail
  $exp_{abs} \leftarrow \text{LLM}(exp_{raw}, S_i.\text{bindings})$ ▷ Parameterize: replace values with slots
  $Target \leftarrow \mathcal{M}_{mid}[key].\text{select_list}(S_i.\text{status})$ ▷ Select PlanSummary or FailureDiagnosis list
  $sim \leftarrow \max_{e \in Target} \text{CosSim}(exp_{abs}, e)$ ▷ Check for semantic redundancy
  if $sim \geq \delta_{dup}$ **then**
    Update counts $(N_{success}, N_{used})$ ▷ Update metrics for UCB/Decay
    if new experience is more detailed **then**
      Update content ▷ Keep higher quality experience
    end if
  else
    $Target.\text{append}(exp_{abs})$ ▷ Register novel subtask experience
  end if
end for

PHASE 3: HIGH-LEVEL MEMORY UPDATE (TASK PLANNING)

if Task is Success **then**
  $Seq \leftarrow [S_1.id, \ldots, S_T.id]$ ▷ Extract abstract execution flow
  $\mathcal{P} \leftarrow \mathcal{M}_{high}[I].\text{plans}$
  if $Seq \in \mathcal{P}$ **then** ▷ Exact workflow matching
    Update success count & avg steps for Seq
    Update rationale if new one is better
  else
    $\mathcal{P}.\text{add}(\{Seq, \text{count} : 1\})$ ▷ Record new successful workflow
  end if
  Sort $\mathcal{P}$ by UCB algorithm.
end if

Global Statistic Update. The system updates instruction difficulty metrics via an Exponential Moving Average (EMA) of task success rate. This mechanism dynamically tracks the agent's competence for each task.

Mid-Level Experience Abstraction and De-duplication. The system abstracts subtask feedback by replacing specific parameters with generic placeholders, followed by semantic deduplication. This prevents overfitting to instance-specific values (e.g., filenames) and minimizes the memory redundancy, ensuring a compact and high-quality knowledge base.

High-Level Workflow Consolidation. Successful trajectories are compressed into workflows and merged into the high-level workflow library. The statistical information like success counts are also updated, ensuring that the system effectively prioritizes robust workflows and discards unstable strategies over time.

H QUALITATIVE EXAMPLES

To qualitatively evaluate the effectiveness of the proposed UI-Mem framework, we present visualized examples illustrating successful long-horizon planning, error correction via failure diagnosis, and remaining challenges in visual grounding.

Impact of Memory Guidance. We demonstrate how different strengths of guidance contribute to the progress of task completion. We provide visualizations of the agent's rollout trajectories for the specific task instruction *"Create a new contact for Chen Yu, with the organization listed as Xiaomi."* As shown in the top row of Figure 12, full memory guidance leads to **High Task**

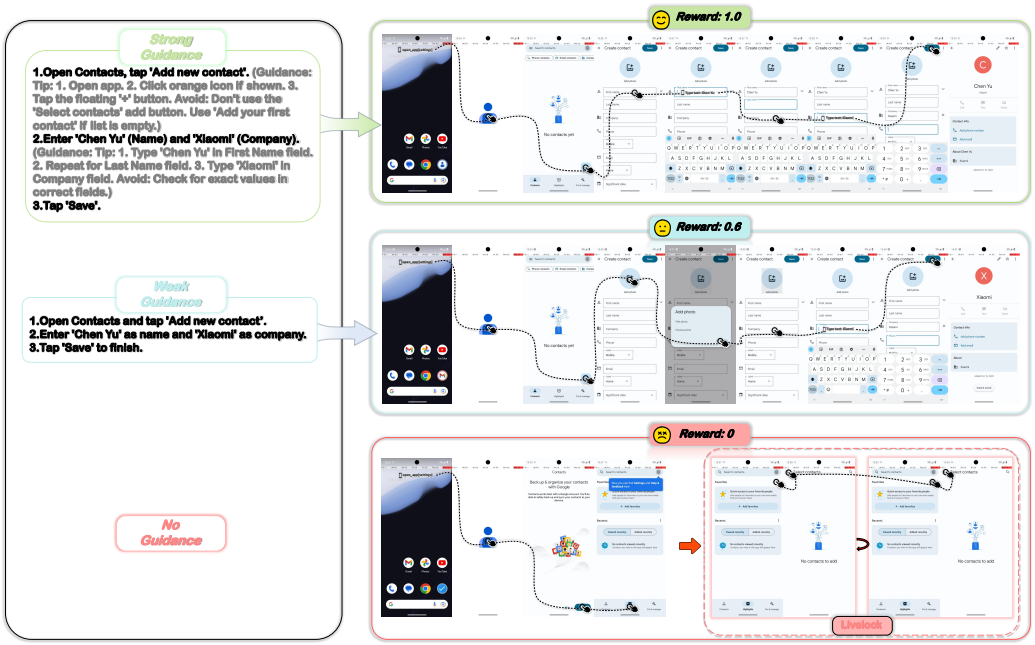


Figure 12: Trajectory analysis on the task "Create a new contact for Chen Yu, with the organization listed as Xiaomi." **Top (Reward 1.0):** Full memory guidance yields a perfect execution. **Middle (Reward 0.6):** Incomplete execution where the agent correctly fills the organization but misses the name. **Bottom (Reward 0):** Lack of memory results in a total failure.

Progress (Reward 1.0), where the agent executes a flawless trajectory, mapping "Chen Yu" to the Name field and "Xiaomi" to the Company field before saving. In contrast, the middle row illustrates **Partial Completion (Reward 0.6)** under partial guidance (only workflow). Here, the agent exhibits incomplete task execution: while it correctly inputs the organization "Xiaomi," it fails to input the name "Chen Yu" and deviates into irrelevant actions (e.g., adding a photo), resulting in a contact saved with missing information. Finally, without any prior guidance, the agent fails to achieve any progress in completing the task, repeatedly toggling between the search bar and the empty list view. This comparison validates our design that injecting different levels of guidance increases the outcome diversity within a rollout group given the same task instruction.

Error Correction via Diagnosis. We demonstrate the effectiveness of integrating failure pattern into our memory using another Audio Recorder task: *"Enter the list and sort the files by name (A-Z)."* Initially, as shown in the top of Figure 13 (First Rollout), the agent fails by misinterpreting the welcome screen's navigation, erroneously exiting the app to an unrelated "Files" application. After this rollout, our memory incorporates the **Failure Diagnosis** identifying the root cause: *"The user opened the Audio Recorder but navigated back... instead of proceeding to the main recording list."* Equipped with this insight, during the second attempt (**w/ Failure Diagnosis**), the agent retrieves a **Correction Guideline** instructing it to *"Tap the 'Get started' button... avoid switching to other apps."* Given this guidance, the agent correctly enters the main list view and successfully performs the sorting operation.

Plan-Execution Gap in Failure Cases. Despite the overall effectiveness of our memory guidance, there are remaining failure modes due to unexpected grounding errors. In Figure 14, the agent is required to complete the task *"Create a new incognito tab and visit www.baidu.com. (If the current screen displays 'No thanks' or 'Use without an account', click that first.)"* The agent retrieves a fine-grained guidance to handle the distinctive "Welcome" popup and navigates the Chrome menu to select "New Incognito tab." However, the rollout fails at the final step due to a visual grounding error. Although the planner intends to visit the URL, the agent fails to correctly select the address bar (Omnibox) to type "www.baidu.com."

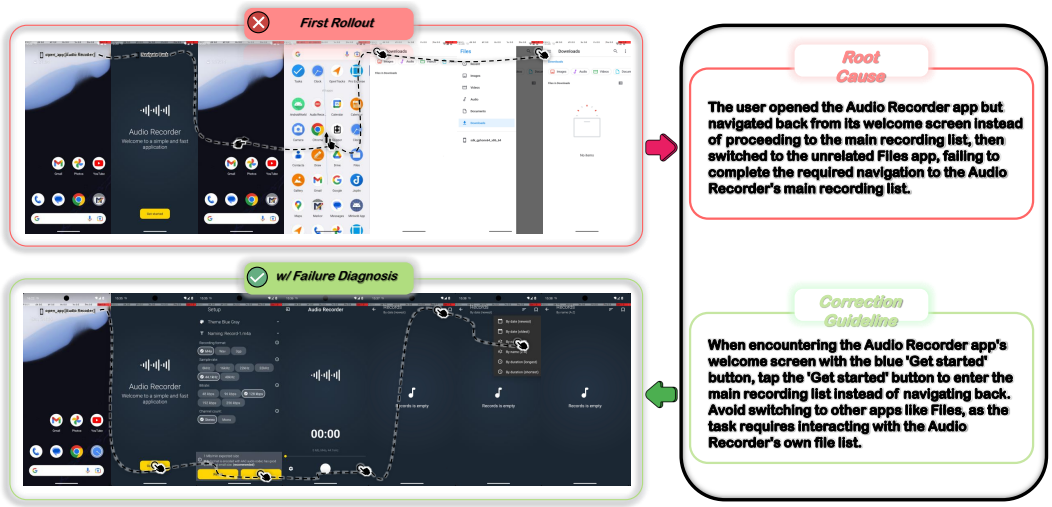


Figure 13: Visualizing the Failure Diagnosis mechanism on the task "Enter the file list and sort the files by name (A-Z)." The system identifies the navigation error in the first rollout and generates a specific **Correction Guideline**, enabling success in the second-round attempt.

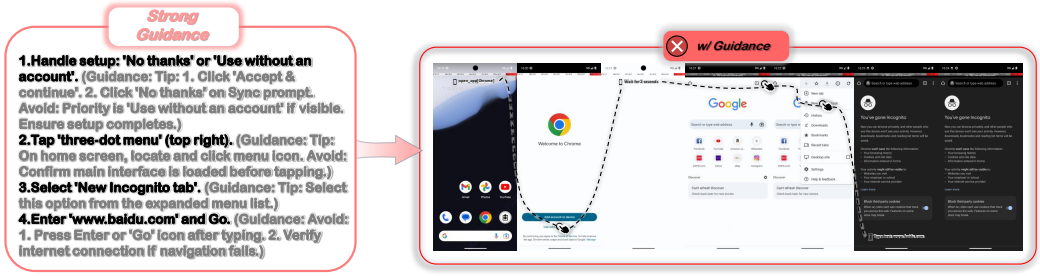


Figure 14: A failure case in the Browser task. Despite fine-grained guidance (opening Incognito), the agent fails to visit the URL due to a specific visual grounding error during the final action step.

I Additional Experimental Details

Dynamic Dropout Curriculum. While stratified group sampling improves within-batch diversity, using fixed guidance proportions throughout training is suboptimal. In the early stages of training, full guidance is crucial for generating positive rewards. As training progresses, continued reliance can hinder generalization. To ensure that the agent eventually internalizes experience, we introduce a dynamic curriculum that gradually withdraws the guidance. Specifically, we track the Exponential Moving Average (EMA) of the success rate $S_t \in [0, 1]$ for each task. We define two thresholds: θ_{start}, below which the agent is considered to struggle with the task, and θ_{end}, above which the agent has achieved reliable competence. The sampling probabilities λ_{full} and λ_{none} are modeled as linear functions of S_t:

$$\lambda_{\text{full}}(S_t) = \text{clip}\left(\lambda_{\text{full}}^{\max} - \phi(S_t)\Delta\lambda_{\text{full}}, \lambda_{\text{full}}^{\min}, \lambda_{\text{full}}^{\max}\right) \tag{10}$$

$$\lambda_{\text{none}}(S_t) = \text{clip}\left(\lambda_{\text{none}}^{\min} + \phi(S_t)\Delta\lambda_{\text{none}}, \lambda_{\text{none}}^{\min}, \lambda_{\text{none}}^{\max}\right) \tag{11}$$

where $\phi(S_t) = \frac{S_t - \theta_{\text{start}}}{\theta_{\text{end}} - \theta_{\text{start}}}$ represents the normalized progress, and $\Delta\lambda_{\text{full}}$, $\Delta\lambda_{\text{none}}$ represent the respective ranges. As the agent's capabilities improve ($S_t \uparrow$), λ_{full} decreases and λ_{none} increases, shifting the focus from imitation to independent task solving.

Hyperparameters. Table 7 summarizes the main hyperparameters used in our online RL experiments. We do not perform benchmark-specific or model-specific grid search. Instead, we select these values through a few short pilot runs on a subset of training tasks by varying one group of

hyperparameters within heuristic ranges and comparing training stability and success-rate trends. The selected values are then fixed across all main experiments.

Table 7: **Hyperparameters used in online RL training.**

Hyperparameter	Value
Base optimizer	GRPO
Model backbones	Qwen3-VL-4B / Qwen3-VL-8B
Number of training seed queries	256
Number of training epochs	5
Group size G	4
Learning rate	1×10^{-6}
Curriculum threshold $\theta_{\text{start}}/\theta_{\text{end}}$	$0.2/0.8$
Guidance-aware reward bonus α	0.2
UCB retrieval weight λ_{ucb}	0.5
full guidance ratio $\lambda_{\text{full}}^{\max}/\lambda_{\text{full}}^{\min}$	$0.7/0.05$
No-guidance ratio $\lambda_{\text{none}}^{\max}/\lambda_{\text{none}}^{\min}$	$0.8/0.25$
partial guidance ratio λ_{partial}	$1 - \lambda_{\text{full}} - \lambda_{\text{none}}$

Compute Resources and Wall-Clock Overhead. All online RL experiments are conducted on a distributed training setup with 32 NVIDIA L40S GPUs. Android emulator workers are deployed in parallel for environment interaction, while auxiliary LLM modules are used for reward verification, experience extraction, abstraction, retrieval, and memory update during training. These auxiliary modules are not required for policy-only inference, except for lightweight memory retrieval when inference-time retrieval is enabled.

Table 8 reports the wall-clock time breakdown per training iteration for vanilla GRPO and UI-Mem. Memory-related operations, including retrieval, experience extraction, abstraction, and memory update, account for only 9.1% of the total wall-clock time per iteration in UI-Mem, while environment interaction remains the dominant cost. The additional financial cost of external model calls is approximately $\$0.013$ per training iteration in our setup. Despite this modest overhead, UI-Mem improves the final task success rate from 53.0% to 66.8%.

Table 8: **Wall-clock time and cost breakdown per training iteration.** Time is measured under the same rollout budget.

Component	Vanilla GRPO	UI-Mem	Cost
Environment reset	1.1 min	1.1 min	–
Memory retrieval	–	0.2 min	\$0
Trajectory rollout	6.7 min	6.3 min	–
Reward verification	0.6 min	0.6 min	\$0
Experience extraction & abstraction	–	0.5 min	\$0.013
Memory update	–	0.1 min	\$0
Total	8.4 min	8.8 min	\$0.013
Final Task SR	53.0%	66.8%	–

Statistical Significance To assess the stability of online RL training, we repeat the key experiments with three independent random seeds. The random seed affects rollout sampling, trajectory generation, and the order of online interactions, while all model, data, and hyperparameter settings are kept fixed across runs. As shown in Table 9, UI-Mem-8B achieves consistent improvements over vanilla GRPO on both AndroidWorld and AndroidLab. The improvements are substantially larger than the run-to-run standard deviation, suggesting that the observed gains are not due to a single favorable run.

Group Size. We use $G = 4$ in the main experiments for a favorable cost-performance trade-off. Increasing the group size to $G = 8$ improves AndroidWorld success rate from 66.8% to 69.0%, but increases the training time per iteration from 8.8 minutes to 24.5 minutes. This suggests that stratified group sampling can provide effective intra-group outcome diversity even under a small practical group size.

Table 9: Results of 3 independent runs with different random seeds. We report mean ± standard deviation of success rate (%).

Model	AndroidWorld SR (%)	AndroidLab SR (%)
Baseline (Vanilla GRPO)	49.57 ± 2.46	35.90 ± 1.33
UI-Mem-8B	**66.97 ± 0.54**	**44.70 ± 0.92**

Table 10: **Effect of group size and stratification.** Stratified sampling remains effective with $G = 4$, while increasing to $G = 8$ yields modest gains at substantially higher training cost.

Group Size	Method	Task SR	Training Time / Iter.
4	UI-Mem	66.8%	8.8 min
4	w/o Stratified Sampling	48.3%	–
8	UI-Mem	69.0%	24.5 min

J Additional Analyses and Clarifications

Train–test overlap and disjoint-split results. Our 256 training queries are curated from AMEX, AndroidLab-Instruct, and UI-Genie, with near-duplicate instructions (same template, different variables) removed. None of the 116 AndroidWorld tasks or their instruction sources are used in training. For AndroidLab, we cross-check all 138 test tasks against all training queries: no test task is identical to a training query or shares a full task template with one. Overlap appears only at the subtask level, where 27 test tasks share at least one subtask primitive (e.g., editing a contact's company field) with training. Training covers 4 of the 9 AndroidLab apps (Clock, Settings, Contacts, Calendar); the other 5 apps are entirely unseen. Table 11 shows that the gain is consistent across splits.

Table 11: **AndroidLab success rate (%) on disjoint splits.**

Split	#Tasks	Qwen3-VL-8B	UI-Mem-8B	Δ
Full AndroidLab	138	34.8	43.5	+8.7
Seen apps, subtask-overlapping	27	44.4	55.6	+11.2
Seen apps, subtask-disjoint	52	38.5	46.2	+7.7
Unseen apps (strictly disjoint)	59	27.1	35.6	+8.5

Held-out applications over repeated evaluations. Table 12 reports results on the five held-out apps over three evaluation runs. Given the small number of tasks per app, these results should be read as complementary to the AndroidWorld evaluation.

Disentangling stratification from the unguided-success bonus. Table 13 sweeps α. The sweep mainly affects the unguided policy and barely changes the retrieval setting; even at $\alpha = 0$, UI-Mem reaches 63.8%, well above vanilla GRPO (53.0%) and the w/o-stratification variant (48.3%). Stratified grouping thus accounts for most of the gain, while α further refines it. For the w/o-stratification variant, every rollout in every group receives full guidance. The fraction of zero-variance groups at early/mid/late training rises from 0.34/0.26/0.22 under UI-Mem to 0.41/0.52/0.46, as groups degenerate into all-success. Moreover, this variant reaches 56.5% with inference-time retrieval but only 48.3% without it, indicating that the policy becomes dependent on guidance.

Multi-seed ablations. Table 14 repeats the key ablations over three seeds. All deltas lie well outside the run-to-run standard deviation of UI-Mem.

Gains by task horizon. We bucket AndroidWorld tasks by their official optimal step count (Table 15). The improvement grows with horizon, consistent with our motivation that reward collapse is most severe on long-horizon tasks.

Table 12: **Success rate (%) on held-out AndroidLab apps** (mean ± std over 3 evaluation runs).

App	#Tasks	Baseline	UI-Mem (policy-only)	UI-Mem (+retrieval)
Bluecoins	15	33.3	33.3 ± 2.2	53.3 ± 3.8
Cantook	12	41.7	50.0 ± 4.8	66.7 ± 4.8
Maps.me	15	13.3	33.3 ± 3.8	33.3 ± 3.8
Pi-Music	12	16.7	25.0 ± 4.8	33.3 ± 4.8
Zoom	5	40.0	40.0 ± 0.0	80.0 ± 6.7
Task-weighted	59	27.1	35.6 ± 1.2	49.2 ± 1.6

Table 13: **Effect of the unguided-success bonus** α (AndroidWorld SR, %).

α	0	0.1	0.2 (default)
Policy-only	63.8	65.5	66.8
With retrieval	69.8	70.7	71.1

Hyperparameter sensitivity. Around the selected defaults, we vary the curriculum thresholds and reward weights with all other settings fixed (Table 16). The spread is at most 1.7 points, largely within the seed variation.

Table 16: **Sensitivity to curriculum thresholds and reward weights** (AndroidWorld SR, mean ± std over 3 seeds).

$\theta_{\text{start}}/\theta_{\text{end}}$	0.1/0.7	0.2/0.8 (default)	0.3/0.9
SR (%)	65.52 ± 0.71	66.97 ± 0.54	67.15 ± 0.63
(λ_o, λ_p)	(1.0, 0.3)	(1.0, 0.5) (default)	(1.0, 0.7)
SR (%)	65.81 ± 0.68	66.97 ± 0.54	66.12 ± 0.75

Role and replaceability of auxiliary models. The auxiliary models never generate actions or trajectories; the policy is trained exclusively on its own rollouts, and the auxiliary models only verify them and compress them into experience templates. Hence, no action-level supervision is distilled from stronger models. Moreover, only the subtask decompositions and essential states are constructed before training, and they are shared with the GRPO + Progress Reward baseline (Table 3), which still trails UI-Mem by 9.5 points. Table 17 replaces the auxiliary modules with fully open and smaller, locally deployable models. Replacing the proprietary Seed1.8 with DeepSeek-V3 costs only 0.9 points; with smaller verifiers, performance drops due to lower verification quality, but UI-Mem still outperforms vanilla GRPO trained with the same verifier by 8.6 points.

Table 17: **Replacing auxiliary models** (AndroidWorld SR, %).

Setting	Training	Verifier	Extraction	Verifier F1	SR
Default	UI-Mem	Qwen2.5-VL-72B + DeepSeek-V3	Seed1.8	0.902	66.8
Fully open	UI-Mem	Qwen2.5-VL-72B + DeepSeek-V3	DeepSeek-V3	0.902	65.9
Smaller models	UI-Mem	Qwen2.5-VL-32B + Qwen3-8B	Qwen3-8B	0.831	60.4
Smaller models	Vanilla GRPO	Qwen2.5-VL-32B + Qwen3-8B	Qwen3-8B	0.831	51.8

Robustness to annotation and verification errors. Beyond noisy extracted experience (Table 6), we perturb essential-state definitions and flip subtask-completion judgments (Table 18). Performance degrades gracefully.

Table 14: **Ablations over 3 random seeds** (AndroidWorld SR, mean ± std).

Variant	SR (%)	Δ vs. full
UI-Mem (full)	66.97 ± 0.54	–
w/o Self-Evolving Loop	63.08 ± 0.61	−3.89
w/o Hierarchical Memory	59.72 ± 0.83	−7.25
w/o Stratified Sampling	48.55 ± 1.97	−18.42
Vanilla GRPO	49.57 ± 2.46	−17.40

Table 15: **AndroidWorld SR (%) by task horizon.**

Horizon (optimal steps)	#Tasks	Vanilla GRPO	UI-Mem-8B	Δ
Short (≤ 10)	89	61.2	71.3	+10.1
Medium (11–20)	20	30.0	55.0	+25.0
Long (> 20)	7	14.3	42.9	+28.6

Table 18: **Robustness to perturbed annotations and verification** (AndroidWorld SR, %).

Perturbation	SR (%)
None (default)	66.8
10% / 20% noisy extracted experience	65.1 / 64.7
10% / 20% corrupted essential states	64.9 / 62.8
5% / 10% flipped verification labels	63.5 / 60.2

Initial memory and memory statistics. The initial memory contains only automatically constructed task templates, subtask definitions, and essential states (Appendix B); workflows, subtask skills, and failure patterns start empty and are built from the agent's own exploration. The 256 training queries serve as seeds for online RL rather than static supervision: they yield about 5k rollouts during training, from which the memory accumulates 329 workflows, 1,972 subtask skills, and 1,209 failure patterns.

K Limitations and Broader Impact

Limitations. Although UI-Mem improves online RL efficiency for mobile GUI agents, several limitations remain. First, the method utilizes auxiliary LLM modules for reward verification and experience extraction during training. Despite overall robustness, these modules may occasionally produce inaccurate state descriptions, incomplete subtask judgments, or noisy experience abstractions, which can introduce suboptimal guidance into the memory. Our default pipeline also uses a proprietary model (Seed1.8) for experience extraction, although replacing it with open-source models only slightly reduces performance (Appendix J). While stratified sampling reduces over-reliance on any single memory item and the self-evolving loop can update memory entries as new trajectories are collected, the framework cannot guarantee fully correct behavior across all GUI tasks. Second, our online RL training starts from 256 seed queries rather than a large-scale task corpus. These queries are obtained by filtering public GUI task sources to remove redundant instructions, maximize task diversity, and retain tasks that are compatible with our controlled emulator environment. They serve as starting points for online rollouts rather than static supervised examples, generating a larger set of trajectories and memory entries during training. Nevertheless, the initial seed distribution may still limit the diversity of experience discovered by the agent. Moreover, part of our training queries are drawn from AndroidLab-Instruct, so the AndroidLab evaluation is partially in-distribution at the app level; our out-of-distribution evidence therefore rests mainly on AndroidWorld and the held-out AndroidLab apps. Finally, our experiments are restricted to Android; scaling UI-Mem to broader task distributions and to other platforms such as web, desktop, and iOS remains an important direction for future work.

Broader Impact. This work aims to improve the efficiency and generalization of autonomous GUI agents. If deployed responsibly, such agents could make mobile devices more accessible, especially for users with disabilities, elderly users, or users with limited technical expertise. By learning reusable

interaction patterns, GUI agents may reduce the effort required to operate complex applications through natural language instructions.

However, autonomous GUI agents also introduce potential safety and security risks. During online exploration or real-world execution, an agent may perform unintended actions such as deleting files or initiating financial transactions. Our experiments are conducted in sandboxed emulator environments, and robust safe exploration protocols are needed before real-world deployment. Moreover, while our memory abstraction mechanism replaces sensitive text with placeholders such as "`{{password}}`", processing screenshots containing personal data still raises privacy concerns that require strict handling procedures. There is also a possibility of misuse. More capable GUI agents could be used to automate deceptive or harmful interactions, including spam, unauthorized account manipulation, or fraudulent transactions. We therefore recommend restricting deployment to authorized user-controlled environments and incorporating monitoring, rate limits, and policy-based constraints on sensitive actions. Finally, training online GUI agents can require substantial computation due to repeated environment rollouts and auxiliary model calls. Although our method improves sample efficiency, such training may still increase energy consumption and associated carbon emissions.