# MEMGUI-AGENT: AN END-TO-END LONG-HORIZON MOBILE GUI AGENT WITH PROACTIVE CONTEXT MANAGEMENT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

MLLM-based mobile GUI agents have made substantial progress on short-horizon tasks, yet remain unreliable on long-horizon tasks that require retaining intermediate facts across many steps and app transitions. We attribute this limitation to ReAct-style prompting, which passively accumulates per-step records, leading to prompt explosion and dilution of critical cross-app facts. To address this, ***(i)*** we introduce **MemGUI-Agent**, an end-to-end long-horizon mobile GUI agent with proactive context management. MemGUI-Agent is built on **Con**text-as-**Act**ion (**CONACT**), which casts context management as first-class actions emitted by the same policy that selects UI actions. Instead of passively appending history, CONACT actively maintains three structured context fields: folded action history, folded UI state, and recent step record, enabling the agent to preserve critical UI facts while keeping context compact. To make proactive context management learnable across model scales, ***(ii)*** we construct **MemGUI-3K**, a 2,956-trajectory dataset with full CONACT annotations, enabling supervised training and analysis of proactive context management. ***(iii)*** Training an 8B model on MemGUI-3K produces **MemGUI-8B-SFT**, an 8B MemGUI-Agent that achieves the best open-data 8B performance on MemGUI-Bench and demonstrates strong out-of-distribution performance on MobileWorld. An **anonymized release** of the code, data, and trained models is publicly available at `https://memgui-agent-anonymous.github.io/`.

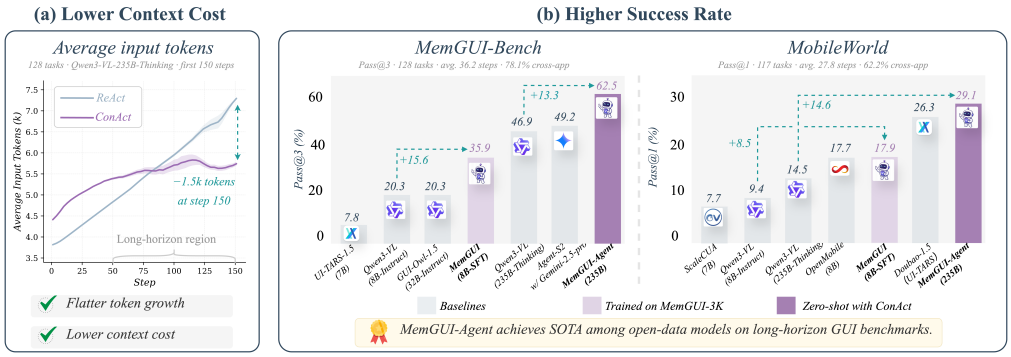



Figure 1: **Context efficiency and benchmark performance of MemGUI-Agent.** **(a)** Pass@3 token trajectories on 128 MemGUI-Bench tasks, truncated to 150 steps. Purple/blue lines denote CONACT/ReAct; translucent bands show per-step uncertainty across trajectories. CONACT saves ∼1.5k input tokens by step 150. **(b)** Benchmark results on MemGUI-Bench and MobileWorld, where MemGUI-Agent improves both 235B zero-shot and 8B-SFT settings, with MemGUI-8B-SFT setting the best open-data 8B performance.

## 1 INTRODUCTION

Recent MLLM-based mobile GUI agents can understand screenshots, reason over user goals, and control devices Gao et al. (2026); Zhou et al. (2025a); Hong et al. (2024); Xu et al. (2026a); Liu et al. (2025a). However, they remain unreliable in long-horizon settings that require retaining



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.

intermediate facts, tracking multi-step progress, and carrying information across app boundaries Liu et al. (2026); Kong et al. (2025). As task horizons grow, even the strongest 8B end-to-end model, GUI-Owl-1.5-8B Xu et al. (2026a), drops from 71.6% on AndroidWorld Rawles et al. (2025) (avg. 8.4 steps) to 38.2% on MobileWorld Kong et al. (2025) (avg. 27.8 steps), and further to 11.7% on MemGUI-Bench Liu et al. (2026) (avg. 36.2 steps).

We attribute this degradation to context management. *Agentic frameworks* couple a backbone MLLM with planning, memory, and grounding modules Agashe et al. (2025); Rawles et al. (2025); Wang et al. (2024; 2025), but rely on complex pipelines and often proprietary backbones. *End-to-end models* are simpler and more deployable Gao et al. (2026); Zhou et al. (2025a); Xu et al. (2026a), yet manage context through passive strategies (e.g., Action-Thought, Multi-turn, Rule-based, No History; see Figure 2 and Table 2) that append, truncate, or discard records mechanically rather than according to task needs. Consequently, prompts grow with task horizon (*prompt explosion*), while critical cross-app facts such as prices, identifiers, or copied text may be diluted, paraphrased, truncated, or forgotten (*information loss*) Liu et al. (2026). We review related mobile GUI agents and context-management methods in Appendix A.

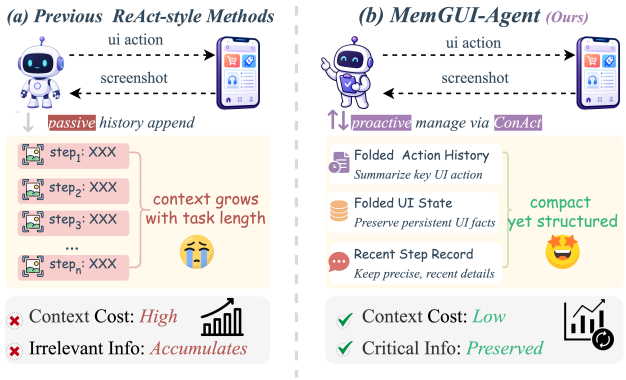



Figure 2: Context management in mobile GUI agents. **(a)** Previous ReAct-style methods passively append per-step records, making context grow with task length and accumulate noise. **(b)** MemGUI-Agent proactively manages context via CONACT's three structured fields: Folded Action History, Folded UI State, and Recent Step Record, keeping context compact while preserving critical UI facts.

Long-horizon mobile GUI control requires compact working context and persistent UI-derived facts across screens, steps, and app transitions. Existing strategies do not provide both: external memory moves context curation outside the end-to-end policy, while prompt-based or rule-based methods accumulate passive logs or discard old facts mechanically. This motivates a policy-level mechanism that decides what to compress, remember, and retain during interaction.

**Research Question.** *Can a single end-to-end mobile GUI agent jointly choose UI actions and manage its working context, so that long-horizon interaction stays compact while critical UI facts persist across steps and app transitions?*

We answer this question with **MemGUI-Agent**, an end-to-end long-horizon mobile GUI agent with proactive context management. MemGUI-Agent is built on **Con**text-as-**Act**ion (**CONACT**), which casts context management as first-class actions emitted alongside UI actions. Instead of passively appending history, CONACT maintains three structured context fields: *Folded Action History* $H_t$ for compressed trajectory summaries, *Folded UI State* $M_t$ for persistent UI-derived facts, and *Recent Step Record* $L_t$ for recent interaction details. All three are predicted in a single forward pass, keeping context curation inside the agent policy.

As shown in Figure 1(a), CONACT keeps prompt growth flatter than ReAct-style prompting. Figure 1(b) shows that this lower context cost translates into stronger long-horizon task success: applied zero-shot to Qwen3-VL-235B-Thinking, MemGUI-Agent reaches 62.5% Pass@3 on MemGUI-Bench, surpassing agentic workflows built on Gemini-2.5-Pro. Matched zero-shot comparisons further show positive gains across three fixed inference backbones (Table 6). To make proactive context management learnable across model scales, we construct **MemGUI-3K**, a 2,956-trajectory dataset with full CONACT annotations. Training an 8B model on MemGUI-3K produces **MemGUI-8B-SFT**, which achieves the best open-data 8B performance on MemGUI-Bench and demonstrates strong out-of-distribution performance on MobileWorld.

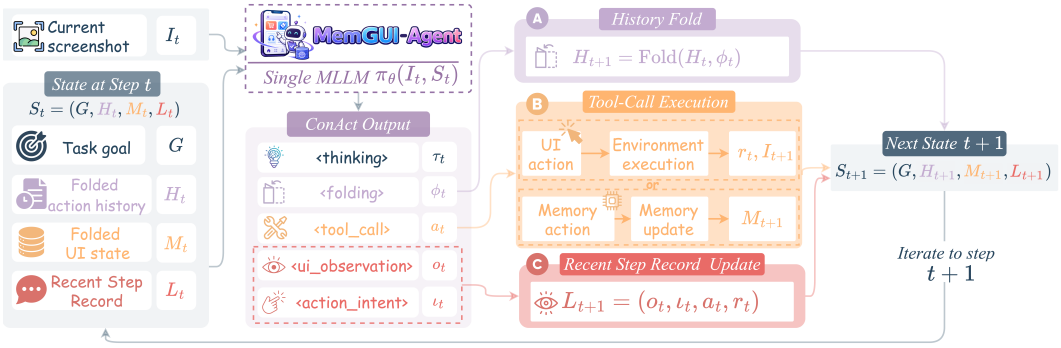



Figure 3: Single-step execution of MemGUI-Agent with ConAct. The MLLM $\pi_\theta$ takes screenshot $I_t$ and structured state $\mathcal{S}_t = (G, H_t, M_t, L_t)$ and emits five outputs $(\tau_t, \phi_t, a_t, o_t, \iota_t)$. Three parallel updates follow: **(A)** folding directive $\phi_t$ updates $H_{t+1}$; **(B)** tool call $a_t$ acts on the environment or updates $M_{t+1}$; **(C)** recent step record $L_{t+1}$ is assembled from $(o_t, \iota_t, a_t, r_t)$. The next state $\mathcal{S}_{t+1}$ feeds into step $t+1$.

**Contributions.** **(1)** We introduce **MemGUI-Agent**, an end-to-end long-horizon mobile GUI agent built on ConAct, a Context-as-Action paradigm that unifies history folding, UI memory, and self-describing step outputs within a single policy (§2). **(2)** We construct **MemGUI-3K**, a 2,956-trajectory dataset with full ConAct annotations, supporting supervised training and offline analysis of proactive context management across model scales (§3). **(3)** We train **MemGUI-8B-SFT**, an 8B MemGUI-Agent supervised on MemGUI-3K, achieving the best open-data 8B performance on MemGUI-Bench and demonstrating strong out-of-distribution performance on MobileWorld (§4).

## 2 MemGUI-Agent: End-to-End Mobile GUI Agent with ConAct

ConAct makes context management a first-class part of the action policy in an end-to-end mobile GUI agent. Instead of treating context as a passive log outside the model's control, the agent jointly decides *what UI or memory action to execute*, *what history to fold*, and *how to describe the current interaction*. This turns context maintenance into policy-level behavior. Figure 3 illustrates the single-step execution flow.

**Problem formulation.** We formulate mobile GUI automation as a sequential decision problem with a structured working context. Given task goal $G$ and screenshot $I_t$, the agent observes

$$\mathcal{S}_t = (G, H_t, M_t, L_t), \tag{1}$$

where $H_t$, $M_t$, and $L_t$ denote *Folded Action History*, *Folded UI State*, and *Recent Step Record*, storing compressed trajectory summaries, persistent UI-derived facts, and latest-step details. At each step, the MLLM policy emits a joint ConAct decision:

$$\begin{aligned} y_t &= (\tau_t, \phi_t, a_t, o_t, \iota_t) \\ &\sim \pi_\theta(\cdot \mid I_t, \mathcal{S}_t), \end{aligned} \tag{2}$$

where $\tau_t$ is reasoning, $\phi_t$ is a folding directive, $a_t$ is a UI or memory action, $o_t$ is the UI observation, and $\iota_t$ is the action intent. Unlike ReAct-style prompting, which appends a growing task-progress string, ConAct partitions context into three fields and updates them through model-emitted context actions.

**Step output protocol.** At each step, the model emits a **5-part structured output**. We build on the Qwen3-VL function-calling prompt format Bai et al. (2025), which asks for a concise thought and a JSON tool call enclosed by `<tool_call>` tags. ConAct adds context-management fields for folding and self-description, and extends the executable tool-call action space with memory operations. The fields are colored by their roles, consistent with Figure 3:

```
<thinking> reasoning </thinking>
```

```
<folding> {range, summary} </folding>
<tool_call> UI / memory action </tool_call>
<ui_observation> screen facts </ui_observation>
<action_intent> next-step plan </action_intent>
```

The <tool_call> field executes exactly one action per step from the extended action space

$$\mathcal{A} = \mathcal{A}_{ui} \cup \mathcal{A}_{mem}, \quad \mathcal{A}_{mem} = \{\texttt{add}, \texttt{update}, \texttt{delete}\}. \tag{3}$$

Here, $\mathcal{A}_{ui}$ contains UI actions such as clicking, typing, swiping, waiting, and terminating. We use add, update, and delete as shorthand for memory_add, memory_update, and memory_delete, which update $M_t$. Other context fields do not act on the environment; they specify how the working context is updated for future steps.

♣ **History folding (over $H_t$).** From step 2 onward, the agent emits a mandatory <folding> block. Formally, the folding directive is $\phi_t = ([s_t, t], z_t)$, where $[s_t, t]$ is the history span to compress and $z_t$ is the generated summary. *Step-level distillation* corresponds to $s_t = t$ and keeps the latest step as a compact record, while *span-level abstraction* uses $s_t < t$ to summarize a completed sub-task into one reusable record. The folded history is updated as

$$H_{t+1} = \text{Fold}(H_t, \phi_t), \tag{4}$$

which replaces uncontrolled <conclusion> accumulation with model-controlled folding, pushing context growth from linear accumulation toward sub-linear growth.

♠ **UI memory actions (over $M_t$).** Each memory item is a structured triple $m = (\text{id}, d, c)$, where id is a unique identifier, $d$ is a short description, and $c$ is the complete content to preserve. Let $\mathcal{A}_{mem} = \{a^+, a^\circ, a^-\}$ denote memory_add, memory_update, and memory_delete. A memory action induces

$$M_{t+1} = \begin{cases} \text{Add}(M_t, m), & a_t = a^+, \\ \text{Update}(M_t, \text{id}, c), & a_t = a^\circ, \\ \text{Delete}(M_t, \text{id}), & a_t = a^-, \\ M_t, & a_t \in \mathcal{A}_{ui}. \end{cases} \tag{5}$$

Memory writes store complete task-relevant information rather than references or lossy summaries, e.g., a full copied price or phone number instead of "the value seen earlier." This is crucial when prices, codes, contact information, or copied text must survive screen changes, long delays, and app transitions.

♢ **Self-describing step output (fills $L_t$).** The two self-describing fields make each step reusable by future context actions. <ui_observation> provides a grounded screen description with exact visible text, numbers, names, and task-relevant UI facts. <action_intent> states what the current tool call is intended to accomplish. Together with the executed action and tool result, they form

$$L_{t+1} = (o_t, \iota_t, a_t, r_t). \tag{6}$$

This record supplies grounded content for memory writes into $M_t$ and future folding into $H_t$. Without these fields, folding and memory actions must infer step meaning only from raw <thinking> and <tool_call> text, which we find insufficient when context actions are layered onto the ReAct 3-part protocol (§4.3).

**Step execution loop.** At step $t$, the MLLM $\pi_\theta$ takes $(I_t, \mathcal{S}_t)$ and emits $y_t = (\tau_t, \phi_t, a_t, o_t, \iota_t)$. The environment/tool result is

$$(r_t, I_{t+1}) = \begin{cases} \text{Env}(I_t, a_t), & a_t \in \mathcal{A}_{ui}, \\ (\text{ok}, I_t), & a_t \in \mathcal{A}_{mem}, \end{cases} \tag{7}$$

where UI actions change the screen and memory actions update only the structured context. Combining Eq. 4, Eq. 5, and Eq. 6, the complete state transition is

$$\begin{aligned} \mathcal{S}_{t+1} &= \mathcal{T}(\mathcal{S}_t, y_t, r_t) \\ &= (G, H_{t+1}, M_{t+1}, L_{t+1}). \end{aligned} \tag{8}$$

The next state $\mathcal{S}_{t+1}$ is fed into step $t+1$. Because folding, memory, and action intent are predicted by the same multimodal policy in one forward pass, compression and memorization inherit the agent's task-level reasoning rather than being delegated to a separate summarizer, retriever, or memory agent.

# 3 MEMGUI-3K DATASET

## 3.1 MOTIVATION: CONTEXT MANAGEMENT MUST BE LEARNED

CONACT provides an end-to-end interface for folding history, writing UI memory, and describing recent steps, but using it well still requires policy decisions. We therefore ask whether the protocol alone is sufficient across model scales.

Table 1 applies the same 5-part CONACT protocol to Qwen3-VL backbones of different sizes on MemGUI-Bench-40. Gains emerge only on the strongest backbone we test, Qwen3-VL-235B-Thinking; on smaller open models and Qwen3-VL-235B-Instruct, the added protocol often reduces performance. This suggests that proactive context management is not merely a prompt-format change: models must learn when to fold history, when to write UI facts into memory, and how to produce reusable step descriptions. We therefore construct **MemGUI-3K**, a supervised dataset with full CONACT annotations, to support training and analysis across model scales.

| Model Scale | P@1 (%) | | P@2 (%) | | P@3 (%) | | IRR (%) | |
|---|---|---|---|---|---|---|---|---|
| | Base | CONACT | Base | CONACT | Base | CONACT | Base | CONACT |
| Qwen3-VL-2B | 10.0 | 5.0 | 12.5 | 5.0 | 17.5 | 10.0 | 10.8 | 1.6 |
| Qwen3-VL-4B | 20.0 | 12.5 | 25.0 | 15.0 | 27.5 | 15.0 | 19.0 | 10.7 |
| Qwen3-VL-8B | 12.5 | 7.5 | 17.5 | 10.0 | 22.5 | 15.0 | 15.0 | 10.7 |
| Qwen3-VL-235B-Instruct | 23.4 | 19.5 | 36.7 | 23.4 | 36.7 | 28.9 | 29.2 | 32.5 |
| **Qwen3-VL-235B-Thinking** | 5.0 | **40.0** | 20.0 | **52.5** | 27.5 | **62.5** | 19.5 | **51.0** |

Table 1: Zero-shot CONACT across Qwen3-VL sizes on MemGUI-Bench-40. **Base**: ReAct-style 3-part prompt; **CONACT**: 5-part context-as-action prompt. Green/red denote improvement/regression. Only 235B-Thinking benefits zero-shot, motivating MemGUI-3K supervision.

## 3.2 CONSTRUCTION PIPELINE

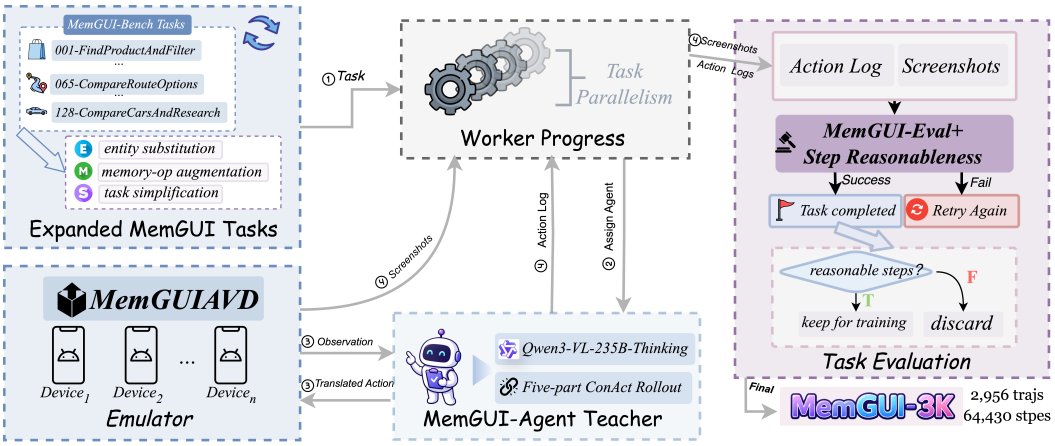



Figure 4: MemGUI-3K data collection and filtering pipeline. Expanded tasks are rolled out by a Qwen3-VL-235B-Thinking teacher, evaluated by MemGUI-Eval, filtered at trajectory and step levels, and converted into SFT samples.

**Expanding seed tasks.** MemGUI-3K is built from the 128 seed tasks of MemGUI-Bench Liu et al. (2026). We expand them with three complementary strategies: *entity substitution*, which preserves task structure while replacing task-specific entities; *memory-operation augmentation*, which increases coverage of `memory_update` and `memory_delete`; and *task simplification*, which decomposes complex multi-app tasks into shorter single-objective variants. This produces a 7,303-task pool, from which 5,293 tasks enter rollout. Detailed expansion statistics are provided in Appendix C.1.

**Collecting teacher rollouts.** Each task is executed once in the snapshot-based Android environment of MemGUI-Bench using Qwen3-VL-235B-Thinking as the teacher with the full 5-part CONACT protocol. Rollouts terminate when the agent emits `terminate` or reaches the task-specific budget $2.5g + 1$, where $g$ is the task's golden-step count. We evaluate trajectories with the MemGUI-Eval progressive scrutiny pipeline Liu et al. (2026) and additionally annotate every step as *reasonable* or

*unreasonable*. This filtering removes recovery loops, repeated errors, and counterproductive actions from SFT targets while preserving successful context-management behavior. Figure 4 visualizes the workflow; details and an annotated example are in Appendices C.2–C.4.

**Filtering into training samples.** After trajectory filtering and a 90/10 split, MemGUI-3K contains 2,956 successful trajectories across 26 apps. We find no exact task-identifier or raw-instruction matches against the 128 MemGUI-Bench evaluation tasks. Extracting reasonable steps yields 64,430 SFT samples, including 57,951 train and 6,479 test samples. Each sample consists of the shared system prompt, the user message with structured context state and screenshot, and the gold 5-part CONACT response. We train **MemGUI-8B-SFT** from Qwen3-VL-8B-Instruct using LoRA SFT with ms-swift Zhao et al. (2025); dataset format and training details are provided in Appendix C.6 and Appendix D.1.

### 3.3 STATISTICS

Figure 5 summarizes MemGUI-3K. The dataset covers 26 Android apps across 7 functional categories, with productivity, shopping, and information & news being the most frequent (Figure 5(a)). Trajectories average 28.8 steps with a median of 25, about 1.9× longer than GUIOdyssey Lu et al. (2025) (avg. 15.3), the previously longest mobile GUI SFT dataset (Figure 5(b)). Step-level reasonableness analysis marks 75.7% of steps as reasonable, yielding 21.8 trainable steps per trajectory on average. Memory actions appear in 65.1% of trajectories (Figure 5(c)). The teacher also performs nontrivial history abstraction: 23.8% of folds are span-level summaries, with an average span of 6.25 steps and 88.7% of trajectories containing at least one span-level fold (Figure 5(d)). Additional statistics on splits, task sources, trajectory lengths, and folding granularity are reported in Appendix C.5. These statistics show that MemGUI-3K supervises UI actions, memory writing, history compression, and reusable step descriptions.

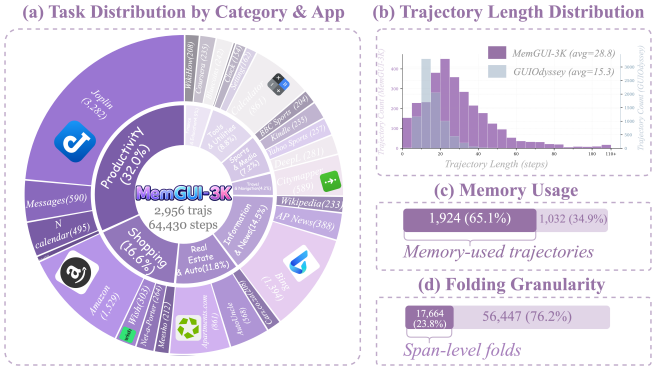



Figure 5: MemGUI-3K statistics. **(a)** Task distribution over 26 apps and 7 categories. **(b)** Trajectory length: avg. 28.8 vs. 15.3 for GUIOdyssey Lu et al. (2025) (1.9×). **(c)** Memory-action usage. **(d)** Folding granularity, separating span-level abstraction from step-level distillation.

## 4 EXPERIMENTS

**Evaluation setup.** We evaluate on MemGUI-Bench Liu et al. (2026), MobileWorld GUI-Only Kong et al. (2025), and MemGUI-Bench-40 for ablations and failure analysis. We report Pass@$k$ ($k \in \{1, 3\}$), IRR, and MTPR on MemGUI-Bench, and single-attempt success rate on MobileWorld. Baselines are listed in Tables 2 and 3; our two reported agents are **MemGUI-Agent-235B**, a zero-shot CONACT agent using Qwen3-VL-235B-Thinking, and **MemGUI-8B-SFT**, an SFT Qwen3-VL-8B-Instruct agent trained on MemGUI-3K. Unless otherwise stated, ablations, case studies, and failure analysis use zero-shot MemGUI-Agent-235B. Benchmark details and prompts are in Appendices B and H.

### 4.1 MAIN RESULTS

**Zero-shot CONACT sets a new SOTA on MemGUI-Bench.** On MemGUI-Bench (Table 2), MemGUI-Agent-235B reaches 37.5% Pass@1 and 62.5% Pass@3, surpassing the best agentic framework, M3A on Gemini-2.5-Pro (32.8% / 47.7%), and improving over the same backbone by +13.3 Pass@1 and +16.8 IRR. On MobileWorld (Table 3), MemGUI-Agent-235B achieves 29.1% SR, +14.6 over the Qwen3-VL-235B-Thinking baseline, showing that zero-shot CONACT also transfers to a different environment and app set. As Figure 1(a) shows, these gains are not explained

| Agent | Memory Type | Easy | | | Medium | | | Hard | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P@1 | P@3 | IRR | P@1 | P@3 | IRR | P@1 | P@3 | IRR | P@1 | P@3 | IRR |
| ***Agentic Frameworks*** *(w/ Gemini-2.5-Pro backbone)* | | | | | | | | | | | | | |
| Agent-S2 Agashe et al. (2025) | *Memory Agent* | **41.7** | 64.6 | 38.7 | 19.0 | 42.9 | 42.5 | 18.4 | 36.8 | 36.9 | 27.3 | 49.2 | 39.5 |
| M3A Rawles et al. (2025) | *Memory Agent* | 39.6 | 47.9 | 31.7 | **35.7** | 50.0 | 48.7 | 21.1 | 44.7 | 36.4 | 32.8 | 47.7 | 39.3 |
| T3A Rawles et al. (2025) | *Memory Agent* | 31.2 | 45.8 | – | 16.7 | 45.2 | – | 18.4 | 34.2 | – | 22.7 | 42.2 | 29.6 |
| Mobile-Agent-E Wang et al. (2025) | *Memory Agent* | 12.5 | 22.9 | 0.9 | 2.4 | 2.4 | 4.4 | 0.0 | 2.6 | 1.8 | 5.5 | 10.2 | 2.4 |
| Mobile-Agent-V2 Wang et al. (2024) | *Memory Agent* | 8.3 | 10.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.1 | 3.9 | 0.0 |
| SeeAct Zheng et al. (2024) | *Rule-based* | 6.2 | 12.5 | 0.0 | 0.0 | 2.4 | 0.5 | 0.0 | 0.0 | 0.0 | 2.3 | 5.5 | 0.2 |
| AppAgent Zhang et al. (2025) | *Action-Thought* | 8.3 | 22.9 | 4.6 | 0.0 | 2.4 | 0.0 | 0.0 | 0.0 | 0.0 | 3.1 | 9.4 | 1.5 |
| ***End-to-End Models*** | | | | | | | | | | | | | |
| Qwen3-VL-235B-Thinking Bai et al. (2025) | *Action-Thought* | 18.8 | 47.9 | 21.8 | 28.6 | 47.6 | 38.8 | 26.3 | 44.7 | 28.2 | 24.2 | 46.9 | 30.0 |
| Qwen3-VL-235B-Instruct Bai et al. (2025) | *Action-Thought* | 27.1 | 39.6 | 26.5 | 21.4 | 40.5 | 28.8 | 21.1 | 28.9 | 32.2 | 23.4 | 36.7 | 29.2 |
| Qwen3-VL-32B-Instruct Bai et al. (2025) | *Action-Thought* | 35.4 | 47.9 | 39.7 | 14.3 | 23.8 | 22.4 | 7.9 | 21.1 | 13.8 | 20.3 | 32.0 | 25.0 |
| GUI-Owl-1.5-32B-Instruct Xu et al. (2026a) | *Action-Thought* | 22.9 | 37.5 | 18.4 | 7.1 | 14.3 | 21.3 | 0.0 | 5.3 | 15.3 | 10.9 | 20.3 | 18.4 |
| Qwen3-VL-8B-Instruct Bai et al. (2025) | *Action-Thought* | 18.8 | 35.4 | 20.1 | 4.8 | 14.3 | 16.0 | 2.6 | 7.9 | 9.5 | 9.4 | 20.3 | 15.1 |
| GUI-Owl-1.5-8B-Instruct Xu et al. (2026a) | *Action-Thought* | 22.9 | 31.2 | 15.7 | 2.4 | 2.4 | 16.5 | 7.9 | 10.5 | 14.2 | 11.7 | 15.6 | 15.5 |
| GUI-Owl-7B Ye et al. (2025a) | *Action-Thought* | 14.6 | 22.9 | 9.9 | 0.0 | 2.4 | 5.0 | 2.6 | 2.6 | 2.6 | 6.2 | 10.2 | 5.7 |
| UI-TARS-1.5-7B Qin et al. (2025) | *Multi-turn* | 8.3 | 16.7 | 9.3 | 0.0 | 0.0 | 1.7 | 0.0 | 0.0 | 0.9 | 3.1 | 6.2 | 3.8 |
| UI-Venus-7B Gu et al. (2025) | *Action-Thought* | 14.6 | 20.8 | 6.9 | 0.0 | 0.0 | 1.2 | 0.0 | 0.0 | 0.0 | 5.5 | 7.8 | 2.6 |
| CogAgent Hong et al. (2024) | *No History* | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **MemGUI-Agent-235B** *(Ours, zero-shot)* | CONACT | **41.7** | **68.8** | **40.8** | **35.7** | **61.9** | **52.3** | **34.2** | **55.3** | **46.5** | **37.5** | **62.5** | **46.8** |
| Δ *vs. Qwen3-VL-235B-Thinking* | | +22.9 | +20.9 | +19.0 | +7.1 | +14.3 | +13.5 | +7.9 | +10.6 | +18.3 | +13.3 | +15.6 | +16.8 |
| **MemGUI-8B-SFT** *(Ours)* | CONACT | 25.0 | 39.6 | 20.5 | 23.8 | 33.3 | 36.5 | 21.1 | 34.2 | 32.6 | 23.4 | 35.9 | 30.2 |
| Δ *vs. Qwen3-VL-8B-Instruct* | | +6.2 | +4.2 | +0.4 | +19.0 | +19.0 | +20.5 | +18.5 | +26.3 | +23.1 | +14.0 | +15.6 | +15.1 |

Table 2: Performance breakdown on MemGUI-Bench by difficulty: Easy (1–20 steps), Medium (21–40), Hard (41+). Both of our agents use a Qwen3-VL backbone with CONACT: MemGUI-Agent-235B applies CONACT zero-shot to Qwen3-VL-235B-Thinking (weights unchanged); MemGUI-8B-SFT is fine-tuned on MemGUI-3K from Qwen3-VL-8B-Instruct. **Bold**/underline = best/second-best (ties share bold); "–" = unavailable.

by larger prompts alone: CONACT keeps prompt growth substantially flatter than the ReAct baseline over 150 steps.

**MemGUI-3K enables an 8B agent with strong OOD generalization.** MemGUI-8B-SFT reaches 23.4% Pass@1 on MemGUI-Bench versus 9.4% for the Qwen3-VL-8B-Instruct baseline, with gains concentrated on harder tasks (Medium +19.0, Hard +18.5 vs. Easy +6.2). This difficulty trend indicates that MemGUI-3K does not merely teach a new output format; it improves behavior in the regime where long-horizon context drift is most severe. On MobileWorld, where the environment, apps, and evaluation protocol all differ, MemGUI-8B-SFT reaches 17.9% SR (+8.5 over backbone) and ranks among the best open-data 8B models. These results demonstrate that the learned context-management skills generalize out of distribution beyond MemGUI-Bench.

| Agent | Memory Type | GUI-Only SR (%) |
|---|---|---|
| ***Open-Weight Models*** | | |
| GUI-Owl-1.5-32B-Instruct Xu et al. (2026a) | *Action-Thought* | 43.9 |
| MAI-UI-235B-A22B Zhou et al. (2025a) | *Action-Thought* | 39.7 |
| GUI-Owl-1.5-8B-Instruct Xu et al. (2026a) | *Action-Thought* | 38.2 |
| MAI-UI-32B Zhou et al. (2025a) | *Action-Thought* | 36.2 |
| GUI-Owl-1.5-2B-Instruct Xu et al. (2026a) | *Action-Thought* | 32.2 |
| MAI-UI-8B Zhou et al. (2025a) | *Action-Thought* | 27.5 |
| Qwen3-VL-235B-Thinking Bai et al. (2025) | *Action-Thought* | 14.5 |
| Qwen3-VL-235B-Instruct Bai et al. (2025) | *Action-Thought* | 12.8 |
| Qwen3-VL-32B-Instruct Bai et al. (2025) | *Action-Thought* | 11.9 |
| Qwen3-VL-8B-Instruct Bai et al. (2025) | *Action-Thought* | 9.4 |
| ***Open-Data Models*** | | |
| OpenMobile-8B Cheng et al. (2026) | *Action-Thought* | 17.7 |
| ClawGUI-2B Tang et al. (2026) | *Action-Thought* | 17.1 |
| OpenMobile-7B Cheng et al. (2026) | *Action-Thought* | 14.8 |
| ScaleCUA-7B Liu et al. (2025b) | *Action-Thought* | 7.7 |
| **MemGUI-Agent-235B** *(Ours, zero-shot)* | CONACT | **29.1** |
| Δ *vs. Qwen3-VL-235B-Thinking* | | +14.6 |
| **MemGUI-8B-SFT** *(Ours)* | CONACT | 17.9 |
| Δ *vs. Qwen3-VL-8B-Instruct* | | +8.5 |

Table 3: Success rate (%) on MobileWorld GUI-Only.

### 4.2 OFFLINE SKILL ANALYSIS

Online benchmarks measure end-to-end success but obscure subskills. We conduct step-level offline evaluation on the MemGUI-3K test set (295 trajectories, 6,479 steps): for each reasonable step, the evaluator feeds the same SFT input, consisting of the system prompt, structured context state, and

screenshot, and compares the generated response with the gold response. This isolates whether the model learns action, memory, folding, and formatting decisions.

Table 4 reports core metrics, with details in Table 13. *UI actions* improve moderately: match accuracy rises from 29.2% to 36.3%, with larger gains on click and type. *Memory timing* improves most: trigger F1 rises from 19.9% to 48.0%, exceeding zero-shot MemGUI-Agent-235B (34.3%). *Deep folding* also improves: MemGUI-8B-SFT reaches 26.1% deep ratio and 58.9% deep range accuracy, compared with 8.8% deep ratio for zero-shot 8B and 23.2% gold. *Format compliance* reaches 99.9%, confirming reliable structured generation. Metric definitions are in Appendix E.

| Agent | UI Action | Memory | Folding | | Format |
|---|---|---|---|---|---|
| | Match | Trigger F1 | Deep Ratio | Range Acc. | Tags |
| 8B-Inst. *(ZS)* | 29.2 | 19.9 | 8.8 | 45.2 | 94.9 |
| **MemGUI-8B-SFT** *(Ours)* | **36.3** | **48.0** | **26.1** | **58.9** | **99.9** |
| Δ *vs. 8B-Inst.* | +7.1 | +28.1 | +17.3 | +13.7 | +5.0 |
| MemGUI-Agent-235B *(ZS)* | 33.9 | 34.3 | 18.7 | 56.7 | 99.2 |

Table 4: Core gold-context step-level offline metrics on the MemGUI-3K test set. Each agent receives the gold SFT input for a single step and is compared with the gold assistant response.8B-Inst. is Qwen3-VL-8B-Instruct; ZS denotes zero-shot. Detailed metrics are in Table 13.

> **INSIGHT 1.** *MemGUI-3K teaches context control decisions, not just formatting: the largest gain comes from promoting transient observations to durable memory.*

### 4.3 ABLATION STUDY

**Component ablation.** We ablate the three components of CONACT on MemGUI-Bench-40 with zero-shot Qwen3-VL-235B-Thinking (Table 5). Each targets a different failure mode, and none suffices alone. UI memory actions triple Pass@1 from 5.0% to 17.5%, showing the value of persistent state. History folding reaches 22.5% Pass@1 but only 32.5% Pass@3, confirming that compression alone cannot preserve task-critical facts. Self-describing step outputs reach 25.0% Pass@1, indicating that grounded observations and action intents help context reuse. Full CONACT reaches 40.0% Pass@1 / 62.5% Pass@3, far exceeding any single-component variant.

| Variant | Success Rate | | | Memory Metrics | |
|---|---|---|---|---|---|
| | P@1 | P@2 | P@3 | MTPR$^{\dagger}$ | IRR$^{\ddagger}$ |
| ReAct baseline | 5.0 | 20.0 | 27.5 | 0.143 | 19.5 |
| + UI memory actions | 17.5 | 35.0 | 42.5 | 0.357 | 39.1 |
| + history folding | 22.5 | 25.0 | 32.5 | 0.179 | 25.7 |
| + self-describing step | 25.0 | 40.0 | 45.0 | 0.214 | 33.1 |
| **Full CONACT** *(Ours)* | **40.0** | **52.5** | **62.5** | **0.429** | **51.0** |
| Δ *vs. ReAct baseline* | +35.0 | +32.5 | +35.0 | +0.286 | +31.5 |

Table 5: Component ablation of CONACT on MemGUI-Bench-40 (Qwen3-VL-235B-Thinking, zero-shot).

> **INSIGHT 2.** *The three components are complementary: folding controls context growth, memory preserves exact facts, and self-description grounds both.*

**Backbone robustness.** Table 6 evaluates zero-shot CONACT on all 40 MemGUI-Bench-40 tasks with three fixed-weight backbones. Only the context protocol changes: Qwen3-VL and Gemini use Action-Thought baselines, while Seed uses Multi-turn plus Action-Thought; all runs use three attempts, temperature 0, identical task-dependent budgets and initial states, and the same MemGUI-Eval success protocol. CONACT improves Pass@1/Pass@3 on every backbone by 12.5–35.0/10.0–35.0 points.

| Backbone | Base Protocol | P@1 (%) | P@3 (%) |
|---|---|---|---|
| Qwen3-VL-235B-Thinking | *Action-Thought* | 5.0 → **40.0** (+35.0) | 27.5 → **62.5** (+35.0) |
| Gemini-3.1-Pro-Preview | *Action-Thought* | 50.0 → **62.5** (+12.5) | 65.0 → **77.5** (+12.5) |
| Seed-2.0-Pro | *Multi-turn* + *Action-Thought* | 42.5 → **60.0** (+17.5) | 62.5 → **72.5** (+10.0) |

Table 6: Zero-shot robustness of CONACT across fixed inference backbones on MemGUI-Bench-40. Each metric reports Base → CONACT, with the absolute gain in parentheses.

> **INSIGHT 3.** CONACT *gains are robust across backbones and baseline context protocols.*

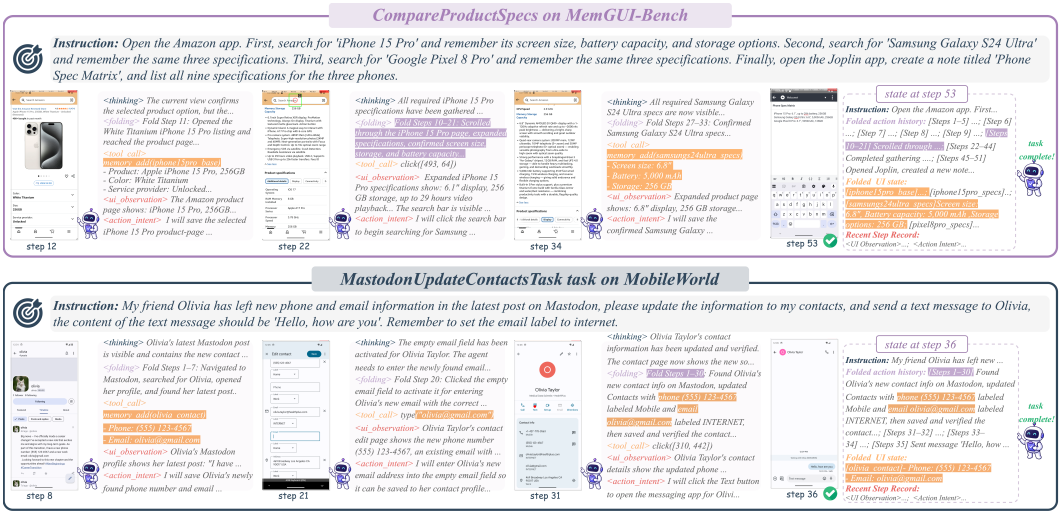



Figure 6: Successful zero-shot MemGUI-Agent-235B trajectories on **MemGUI-Bench** and **MobileWorld**. Rows pair screenshots and CONACT outputs; *folded action history*/*memory writes* mark compressed steps/saved facts, and dashed boxes show compact state.

### 4.4 CASE STUDY AND ERROR ANALYSIS

**Case study.** Figure 6 shows two successful zero-shot MemGUI-Agent-235B trajectories. On MemGUI-Bench, the agent compares Amazon products and records specifications in Joplin; on MobileWorld, it carries contact information across Mastodon, Contacts, and Messages. Both demonstrate cross-app CONACT: writing facts before transitions, folding completed subtasks, and preserving recent details. Full base-vs.-MemGUI trajectories are in Appendix F.

**Error analysis.** Following the MemGUI-Bench taxonomy Liu et al. (2026), Figure 7 groups failures into process hallucination, output hallucination, knowledge deficiency, intent misunderstanding, and other errors. Full CONACT reduces total failures by 41% (99→58), mainly through process hallucination (−22, −42%) and output hallucination (−17, −57%). Knowledge deficiency, intent misunderstanding, and other errors remain roughly stable, suggesting that CONACT mainly prevents progress loss and fabricated UI facts, while remaining errors require stronger knowledge, intent understanding, and environment robustness. Details and examples are in Appendix G.

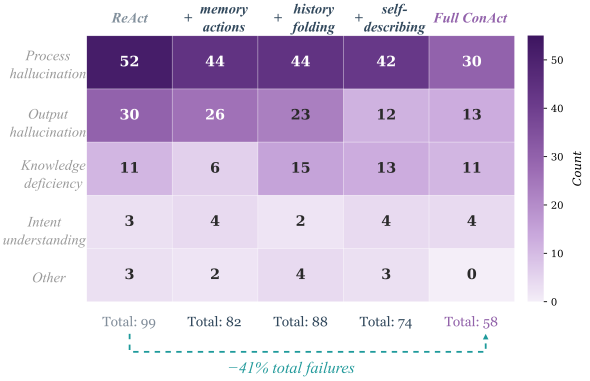



Figure 7: Failure category counts for zero-shot Qwen3-VL-235B-Thinking across five ablation variants on MemGUI-Bench-40 Liu et al. (2026).

> **INSIGHT 4.** CONACT *mainly reduces context-induced hallucinations; the remaining ceiling lies in knowledge, intent understanding, and environment robustness.*

## 5 CONCLUSION

We presented MemGUI-Agent, a long-horizon mobile GUI agent that manages context through CONACT. Zero-shot CONACT achieves state-of-the-art MemGUI-Bench performance across three backbones. MemGUI-3K enables MemGUI-8B-SFT to deliver the best open-data 8B results.

### AI USE STATEMENT

Generative AI tools were used in two roles. In the research workflow, Qwen3-VL-235B-Thinking was used to generate the MemGUI-3K rollout trajectories, while LLM-based evaluation supported trajectory filtering and step-level reasonableness annotation, as described in Section 3 and Appendix C. Generative AI assistants also supported research code development, data processing and analysis, and manuscript editing and LaTeX formatting. The authors reviewed all AI-assisted work, independently verified the data-processing pipeline, code, experimental results, citations, and final manuscript, and take full responsibility for the complete submission.

## REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Kanzhi Cheng, Zehao Li, Zheng Ma, Nuo Chen, Jialin Cao, Qiushi Sun, Zichen Ding, Fangzhi Xu, Hang Yan, Jiajun Chen, et al. OpenMobile: Building open mobile agents with task and trajectory synthesis. *arXiv preprint arXiv:2604.15093*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

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.

Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, Beitong Zhou, et al. Ui-venus-1.5 technical report. *arXiv e-prints*, pp. arXiv–2602, 2026.

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.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14281–14290, 2024.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, et al. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*, 2025a.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, et al. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *arXiv preprint arXiv:2602.06075*, 2026.

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.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22404–22414, 2025.

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.

Chris 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. In *International Conference on Learning Representations*, volume 2025, pp. 406–441, 2025.

Fei Tang, Zhiqiong Lu, Boxuan Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. ClawGUI: A unified framework for training, evaluating, and deploying gui agents. *arXiv preprint arXiv:2604.11784*, 2026.

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. *Advances in Neural Information Processing Systems*, 37:2686–2710, 2024.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026a.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026b.

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*, 2025a.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025b.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 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, 2025.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *Advances in Neural Information Processing Systems*, 38:12988–13018, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 29733–29735, 2025.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, 2024.

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*, 2025a.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025b.

# APPENDIX CONTENTS

**A Related Work** **14**

**B Benchmark and Online Evaluation Protocol** **14**

**C MemGUI-3K Dataset Details** **15**

C.1 Task Expansion . . . . . 15

C.2 Teacher Rollout and Trajectory Filtering . . . . . 15

C.3 Step-Level Reasonableness Filtering . . . . . 16

C.4 Training Data Example . . . . . 16

C.5 Dataset Statistics . . . . . 16

C.6 Dataset Format . . . . . 18

**D Training Setup and Dynamics** **19**

D.1 Training Setup . . . . . 19

D.2 Training Dynamics . . . . . 19

**E Offline Skill Evaluation Protocol** **20**

**F Trajectory Completion Comparisons** **21**

**G Failure Taxonomy and Annotation** **24**

**H Full Prompt Templates** **29**

H.1 ReAct-Style Qwen3-VL Prompt . . . . . 29

H.2 MemGUI-Agent Prompt . . . . . 30

# A Related Work

**Mobile GUI agents.** Mobile GUI agents built on multimodal models generally follow two paradigms. *Agentic frameworks* combine a backbone MLLM with planning, memory, and grounding modules Rawles et al. (2025); Wang et al. (2024; 2025); Agashe et al. (2025), often around proprietary backbones such as Gemini-2.5-Pro Comanici et al. (2025); they can perform strongly but are complex and difficult to reproduce. *End-to-end models* Gao et al. (2026); Zhou et al. (2025a); Hong et al. (2024); Xu et al. (2026a) directly map screenshots to actions, offering simplicity and deployability, but typically rely on passive context mechanisms: Action-Thought traces Zhang et al. (2025); Gu et al. (2025); Xu et al. (2026a); Bai et al. (2025), multi-turn prompts Qin et al. (2025), rule-based aggregation Zheng et al. (2024), or no-history prompting Hong et al. (2024). These mechanisms either let prompts grow with task horizon or lose UI-derived facts needed across screens and apps Liu et al. (2026); Kong et al. (2025). MemGUI-Agent brings proactive context management into the end-to-end paradigm itself.

**Context management in LLM agents.** Long-context LLM agents manage histories through external memory or prompt-internal curation. External memory maintains a separate store, such as retrieval-augmented databases Lewis et al. (2020); Chhikara et al. (2025); Xu et al. (2026b), hierarchical memory Zhang et al. (2026), or experiential replay Zhao et al. (2024). Prompt-internal curation edits the in-task context through truncation, summarization, or model-controlled compression, as in recent web-agent methods such as MEM1 Zhou et al. (2025b), MemAgent Yu et al. (2025), and AgentFold Ye et al. (2025b). Unlike text-only web settings, mobile GUI control must preserve UI-derived facts, such as prices, codes, and copied text, verbatim across screen and app transitions. CoNAct addresses this by treating context curation as joint action over structured history, UI state, and recent-step fields within the same end-to-end policy.

# B Benchmark and Online Evaluation Protocol

**Benchmark details.** MemGUI-Bench Liu et al. (2026) contains 128 tasks across 26 Android apps, of which 89.8% are explicitly memory-intensive. Tasks are partitioned by golden-step count into Easy (1–20, 48 tasks), Medium (21–40, 42 tasks), and Hard (41+, 38 tasks); 78.1% are cross-app, with an average length of 36.2 steps. The 40-task subset MemGUI-Bench-40 is reserved for ablations. MobileWorld Kong et al. (2025) GUI-Only contains 117 tasks in a containerized environment with disjoint apps (Mattermost, Mastodon, Mall4Uni), averaging 27.8 steps per task with 62.2% cross-app.

**Compared agents.** On MemGUI-Bench, we compare against agentic frameworks powered by Gemini-2.5-Pro: Agent-S2 Agashe et al. (2025), M3A/T3A Rawles et al. (2025), Mobile-Agent-E Wang et al. (2025), Mobile-Agent-V2 Wang et al. (2024), SeeAct Zheng et al. (2024), and AppAgent Zhang et al. (2025); and end-to-end models including Qwen3-VL 8B/32B/235B variants Bai et al. (2025), GUI-Owl-1.5 8B/32B Xu et al. (2026a), UI-TARS-1.5-7B Qin et al. (2025), and UI-Venus-7B Gu et al. (2025). On MobileWorld, we additionally include ScaleCUA-7B Liu et al. (2025b), OpenMobile 7B/8B Cheng et al. (2026), and ClawGUI-2B Tang et al. (2026).

**MemGUI-Bench evaluation.** For MemGUI-Bench and MemGUI-Bench-40, each task is evaluated with up to 3 independent attempts. For a task with golden-step count $g$, the maximum interaction budget is

$$B_{\mathrm{MemGUI}}(g) = 2.5g + 1,$$

rounded to an integer in implementation. This adaptive budget gives longer tasks proportionally more room while preventing unlimited exploration. We report Pass@$k$ for $k \in \{1, 3\}$, where a task is counted as successful if any of the first $k$ attempts is judged successful by MemGUI-Eval.

**MobileWorld evaluation.** For MobileWorld GUI-Only, we follow the benchmark protocol and run a single attempt per task with a 50-step limit. We report Pass@1 success rate (SR), where task success is determined by the MobileWorld backend verifier.

**Memory-oriented metrics.** Besides success rate, MemGUI-Bench reports memory-oriented metrics. IRR (Information Retention Rate) measures whether task-critical information units are retained

and correctly used. MTPR (Memory-Task Proficiency Ratio) measures success on explicitly memory-intensive tasks. These metrics are computed by the MemGUI-Eval pipeline Liu et al. (2026).

# C MemGUI-3K Dataset Details

This appendix provides additional details for the MemGUI-3K construction pipeline summarized in §3.

## C.1 Task Expansion

We start from the 128 seed tasks of MemGUI-Bench Liu et al. (2026), which are long-horizon mobile GUI tasks with strong cross-step dependencies. We expand them through three complementary strategies.

- **Entity substitution** preserves the original task pipeline but replaces task-specific entities, such as product names, search queries, contacts, notes, locations, or app-specific targets. This yields 456 expanded tasks.
- **Memory-operation augmentation** explicitly constructs tasks requiring not only `memory_add`, but also `memory_update` and `memory_delete`. This yields 289 memory-operation-focused tasks.
- **Task simplification** decomposes complex expanded tasks into shorter, single-objective variants while preserving the need for structured context management. This yields 6,558 simplified tasks.

The three strategies produce a combined pool of 7,303 tasks, of which 5,293 are selected for rollout. The final dataset contains three source types: simplified tasks, entity-substitution tasks, and memory-operation augmentation tasks.

| Source prefix | Total | Train | Test |
|---|---|---|---|
| S: simplified | 2,566 (86.8%) | 2,309 | 257 |
| E: entity substitution | 234 (7.9%) | 208 | 26 |
| M: memory operations | 156 (5.3%) | 144 | 12 |

Table 7: Task-source composition of MemGUI-3K.

## C.2 Teacher Rollout and Trajectory Filtering

All rollouts are collected with Qwen3-VL-235B-Thinking using the full 5-part ConAct protocol described in §2. Each generated task is executed once in the snapshot-based Android environment of MemGUI-Bench. The environment is reset before each task and supports UI actions, such as `click`, `swipe`, `type`, `wait`, and `system_button`, as well as memory actions, i.e., `memory_add`, `memory_update`, and `memory_delete`. For a generated task with golden-step count $g$, the teacher rollout budget is also set to $2.5g + 1$ steps, following the MemGUI-Bench evaluation protocol. Each rollout stops when the agent emits `terminate` or reaches this task-specific budget.

| Filtering stage | Removed | Remaining |
|---|---|---|
| Successful trajectories | – | 2,959 |
| Remove abnormal 321-step outlier | 1 | 2,958 |
| Remove low-frequency-app tasks | 2 | 2,956 |

Table 8: Trajectory-level filtering for MemGUI-3K.

We first retain only successful trajectories according to the final decision of MemGUI-Eval Liu et al. (2026). MemGUI-Eval follows a progressive scrutiny pipeline: it first screens task completion from the goal, action logs, and final screenshots; then performs semantic analysis with generated step descriptions and memory-related checks; and finally verifies critical visual information against requested screenshots. Among 5,293 rollout trajectories, 2,959 receive a positive final decision. We then remove one abnormal 321-step outlier from an early collection run and two successful

trajectories involving low-frequency apps with at most 8 occurrences. This produces the final 2,956 MemGUI-3K trajectories.

The final trajectories are split 90%/10% at the trajectory level with seed 42. We verify no exact task-identifier or raw-instruction matches against the 128-task MemGUI-Bench evaluation set. Since MemGUI-Bench-40 is a subset of MemGUI-Bench, it has no exact matches in MemGUI-3K either.

| Split | Trajectories | SFT samples | Avg. reasonable steps |
|---|---|---|---|
| Train | 2,661 | 57,951 | 21.8 |
| Test | 295 | 6,479 | 22.0 |
| Total | 2,956 | 64,430 | 21.8 |

Table 9: Train/test split and reasonable-step SFT samples.

### C.3 Step-Level Reasonableness Filtering

Successful trajectories may still contain redundant, exploratory, or counterproductive steps that are later corrected. To avoid training the student on such actions, we extend MemGUI-Eval with step-level reasonableness analysis. The evaluator labels each step as *reasonable* or *unreasonable*, attaches a natural-language explanation, and assigns an impact label (*positive*, *negative*, or *neutral*). Only steps labeled as reasonable are converted into SFT samples.

The following instruction block is appended to the MemGUI-Eval final-decision prompt. The evaluator receives the task description, action logs, generated step descriptions, and screenshots, and returns both the trajectory-level judgment and per-step reasonableness annotations in a single JSON response.

```
STEP REASONABLENESS ANALYSIS

In addition to the overall task success/failure judgment, provide a step-level
reasonableness analysis for every step in the trajectory.

Return:
- reasonable_steps: an array of step numbers that are reasonable and
  contribute to task completion.
- unreasonable_steps: an array of step numbers that are incorrect, redundant,
  or counterproductive.
- step_analysis: an object keyed by step number.
  For each step, include:
    - reasonableness: "reasonable" or "unreasonable"
    - explanation: why the step is reasonable or unreasonable
    - impact: "positive", "negative", or "neutral"

Notes:
- Successful trajectories may still contain unreasonable steps that were later corrected.
- Failed trajectories may still contain reasonable steps that move toward the goal.
```

### C.4 Training Data Example

Figure 8 shows one MemGUI-3K trajectory with step-level reasonableness annotations. Reasonable steps are converted into SFT targets, while unreasonable steps are retained in the trajectory-level record for analysis but excluded from the step-level training split.

### C.5 Dataset Statistics

MemGUI-3K contains long-horizon trajectories with an average of 28.8 steps and a median of 25 steps. After step-level filtering, the average number of reasonable steps is 21.8, corresponding to a 75.7% reasonable-step ratio.

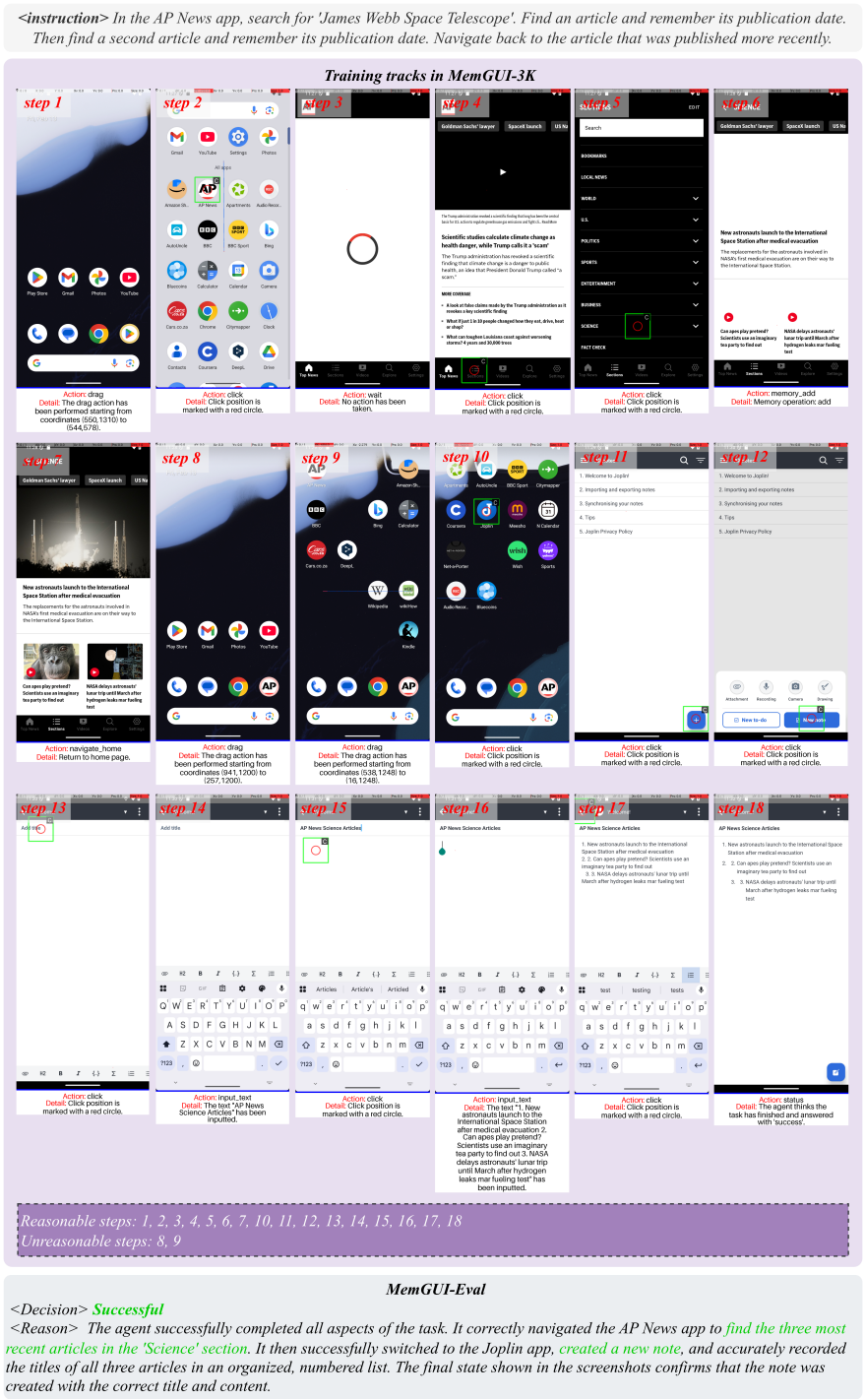



Figure 8: Training-data example from MemGUI-3K. The trajectory contains step-level reasonableness labels, explanations, and impact annotations. Reasonable steps provide supervised CONACT targets; unreasonable steps illustrate redundant, incorrect, or counterproductive behavior filtered out of SFT samples.

MemGUI-3K contains both step-level and span-level folding annotations. Step-level folds distill the latest step, while span-level folds compress a completed multi-step subtask into a reusable record.

| Metric | Mean | Median |
|---|---|---|
| All steps | 28.8 | 25 |
| Reasonable steps | 21.8 | 20 |

Table 10: Trajectory length statistics under all-step and reasonable-step views.

Context actions appear frequently in MemGUI-3K. On average, each trajectory contains 1.17 `memory_add` actions, 0.03 `memory_update` actions, and 0.04 `memory_delete` actions. At the trajectory level, 65.1% of trajectories invoke at least one memory action.

| Metric | Value |
|---|---|
| Step-level folds | 56,447 |
| Span-level folds | 17,664 |
| Span-level fold ratio | 23.8% |
| Average span length | 6.25 steps |
| Median span length | 4 steps |
| 95th percentile span length | 18 steps |
| Maximum span length | 118 steps |
| Trajectories with $\geq$1 span-level fold | 88.7% |

Table 11: Folding-granularity statistics in MemGUI-3K.

MemGUI-3K spans 26 Android apps across 7 functional categories. The most frequent apps include Joplin, Amazon, Bing, Calculator, Apartments.com, Messages, Citymapper, AutoUncle, Calendar, and AP News. We normalize app-name variants before counting, e.g., `joplin`/`Joplin`, `bing`/`Bing`, and `messages`/`Messages`.

### C.6 Dataset Format

MemGUI-3K is released in two complementary formats. The trajectory-level format preserves complete rollouts, evaluation metadata, IRR statistics, token usage, screenshots, and step-level reasonableness annotations, supporting offline analysis of memory operations, folding granularity, token cost, and recovery patterns. The step-level format converts all reasonable steps into ms-swift-compatible SFT samples. Each sample contains the shared system prompt, the user message with structured context state and current screenshot, and the gold 5-part ConAct response. Only steps labeled as reasonable are included in the SFT split.

The SFT data follows the standard chat-style multimodal format. A schematic example is shown below:

```
{
  "messages": [
    {"role": "system", "content": "..."},
    {"role": "user",
     "content": "<image>### User Query\n...\n"
                "### Folded Action History\n...\n"
                "### Folded UI State\n...\n"
                "### Recent Step Record\n..."},
    {"role": "assistant",
     "content": "<thinking>...</thinking>\n"
                "<folding>...</folding>\n"
                "<tool_call>...</tool_call>\n"
                "<ui_observation>..."
                "</ui_observation>\n"
                "<action_intent>...</action_intent>"}
  ],
  "images": ["..."]
}
```

# D TRAINING SETUP AND DYNAMICS

## D.1 TRAINING SETUP

MemGUI-8B-SFT is initialized from Qwen3-VL-8B-Instruct and fine-tuned on the 57,951 reasonable-step training samples of MemGUI-3K using LoRA SFT with ms-swift Zhao et al. (2025). Each sample contains the shared system prompt, the user message consisting of the structured context state and current screenshot, and the gold 5-part CONACT response. Training is conducted on eight 80GB GPUs and takes 535 minutes.

We use bfloat16 precision and train for one epoch with AdamW fused optimizer, cosine learning-rate scheduling, and a warmup ratio of 0.05. The per-device training batch size is 2 with gradient accumulation of 8, giving an effective batch size of 128 across 8 GPUs. We set the learning rate to $1 \times 10^{-4}$, weight decay to 0.1, maximum gradient norm to 1.0, and maximum sequence length to 32,768. LoRA uses rank 8, alpha 32, dropout 0.05, and is applied to all linear language-model projection modules. We use seed 42, four dataloader workers, and report training logs to Weights & Biases.

| Hyperparameter | Value |
|---|---|
| Base model | Qwen3-VL-8B-Instruct |
| Trainable parameters | LoRA adapters |
| Training samples | 57,951 reasonable steps |
| Epochs | 1 |
| Precision | bfloat16 |
| Optimizer | AdamW fused |
| LR scheduler | cosine |
| Learning rate | $1 \times 10^{-4}$ |
| Warmup ratio | 0.05 |
| Weight decay | 0.1 |
| Max grad norm | 1.0 |
| Max sequence length | 32,768 |
| Per-device train batch size | 2 |
| Gradient accumulation steps | 8 |
| Effective batch size | 128 |
| LoRA rank / alpha / dropout | 8 / 32 / 0.05 |
| LoRA target modules | all linear LM projections |
| Dataloader workers | 4 |
| Seed | 42 |
| Hardware | 8×80GB |
| Training time | 535 minutes |
| Software | ms-swift, Transformers 4.57.0, Python 3.10 |

Table 12: Training hyperparameters for MemGUI-8B-SFT.

## D.2 TRAINING DYNAMICS

Figure 9 shows the training dynamics of MemGUI-8B-SFT. The token accuracy steadily increases and the training loss decreases during the single-epoch LoRA SFT run, while the learning rate follows the configured warmup and cosine decay schedule.

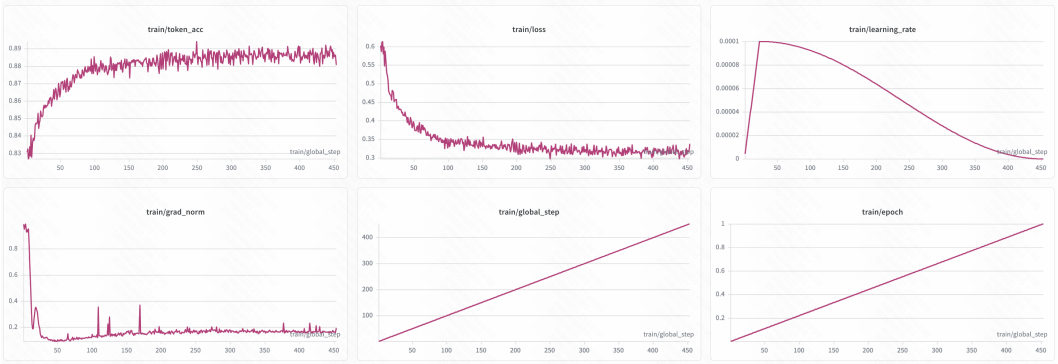



Figure 9: Training dynamics of MemGUI-8B-SFT. We report token accuracy, training loss, learning rate, gradient norm, global step, and epoch during LoRA SFT.

## E Offline Skill Evaluation Protocol

This section details the offline skill evaluation protocol used in Table 4. The implementation is provided in `memgui3k_test_eval/` and has three stages: `run_offline_eval.py` performs gold-context step-level inference, `evaluate_predictions.py` computes automatic metrics, and `compare_results.py` builds the model-comparison summary. The reported run evaluates three models on the MemGUI-3K test split, which contains 295 trajectories and 6,479 reasonable-step samples.

**Offline inference protocol.** For each test sample, the inference script sends only the first two chat messages from `test_sft.jsonl`, namely the system prompt and user message containing the task, structured context state, and screenshot. The gold assistant message is held out for evaluation. We use an OpenAI-compatible chat-completion API with temperature 0.01, maximum generation length 16,384 tokens, 32 concurrent workers, resumable JSONL output, and retry logic for transient API errors. The predictions are written as `{task_id, step, prediction, gold, usage}` records, and the evaluator writes both an aggregate JSON report and per-step details.

**Action matching.** The evaluator parses JSON from the `<tool_call>` block and follows a ClawGUI-style two-level protocol Tang et al. (2026). Type accuracy checks whether the predicted action type matches the gold action type. Match accuracy additionally checks action arguments. For `click` and `long_press`, the predicted coordinate must fall within 14% of the $1000\times1000$ screen diagonal from the gold coordinate. For `swipe`, both start and end coordinates must fall within 1.5 times this threshold. For `type` and `answer`, the text must match exactly after stripping whitespace or achieve token-level F1 $> 0.5$. For `system_button` and `terminate`, the button/status argument must match exactly; `wait` succeeds whenever the action type matches.

**Memory-action metrics.** For memory actions, type accuracy evaluates whether the predicted operation is `memory_add`, `memory_update`, or `memory_delete` when the gold output requires the same operation. For `memory_delete`, match success requires exact `memory_id` match. For `memory_add` and `memory_update`, match success requires either description token F1 $> 0.5$ or content token F1 $> 0.5$. The description criterion is used because numeric contents such as prices may have low token overlap even when the predicted memory intent is correct. Trigger precision, recall, and F1 treat any memory action as a binary event over all steps and evaluate whether the model invokes memory at the same steps as the gold response.

**Folding metrics.** The evaluator parses JSON from the `<folding>` block. Presence accuracy checks whether folding is present or absent consistently with gold. Range accuracy checks whether both predicted range endpoints are within $\pm2$ steps of the gold endpoints. A fold is considered span-level, or deep, if its range covers more than one step; otherwise it is shallow. We report predicted deep-fold ratio and separate range accuracy for shallow and deep folds, because a model can obtain a high overall range score by emitting mostly shallow folds.

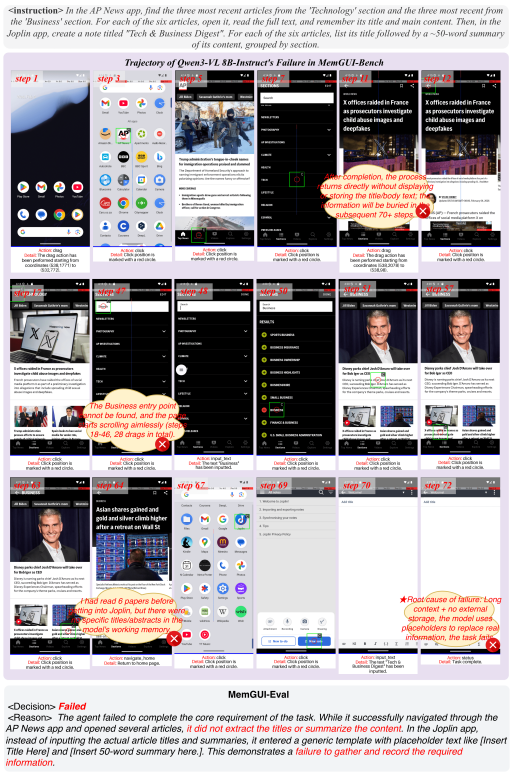
| Metric | Model 8B-Inst. | 8B-SFT | 235B-Th. |
|---|---|---|---|
| ***Overall Action Matching*** | | | |
| Type Accuracy | 52.7 | **62.6** | 58.7 |
| Match Accuracy | 28.2 | **35.6** | 33.1 |
| ***UI Actions*** | | | |
| Type Accuracy | 55.1 | **65.8** | 61.1 |
| Match Accuracy | 29.2 | **36.3** | 33.9 |
| click | 18.7 | **26.7** | 24.2 |
| type | 53.1 | **69.5** | 67.3 |
| swipe | 41.2 | **45.4** | 44.5 |
| system_button | 63.6 | **69.0** | 65.6 |
| ***Memory Actions*** | | | |
| Type Accuracy | 10.9 | **56.4** | 23.0 |
| Match Accuracy | 10.6 | **51.5** | 22.4 |
| Trigger F1 | 19.9 | **48.0** | 34.3 |
| ***Folding*** | | | |
| Deep Fold Ratio$^{\dagger}$ | 8.8 | **26.1** | 18.7 |
| Range Acc (deep) | 45.2 | **58.9** | 56.7 |
| Range Acc (shallow) | **98.2** | 86.2 | 92.3 |
| ***Format Compliance*** | | | |
| All Tags Correct | 94.9 | **99.9** | 99.2 |

Table 13: Detailed aggregate metrics for gold-context step-level offline evaluation on the MemGUI-3K test set (295 trajectories, 6,479 steps). **8B-Inst.** = Qwen3-VL-8B-Instruct (zero-shot); **8B-SFT** = MemGUI-8B-SFT; **235B-Th.** = Qwen3-VL-235B-Thinking (zero-shot rerun). Match accuracy follows the implementation in Appendix E: click/long-press within 14% of the screen diagonal, type/answer exact match or token F1 $> 0.5$, and memory add/update matching by description or content F1 $> 0.5$. $^{\dagger}$Gold deep fold ratio is 23.2%. **Bold**: best per row.

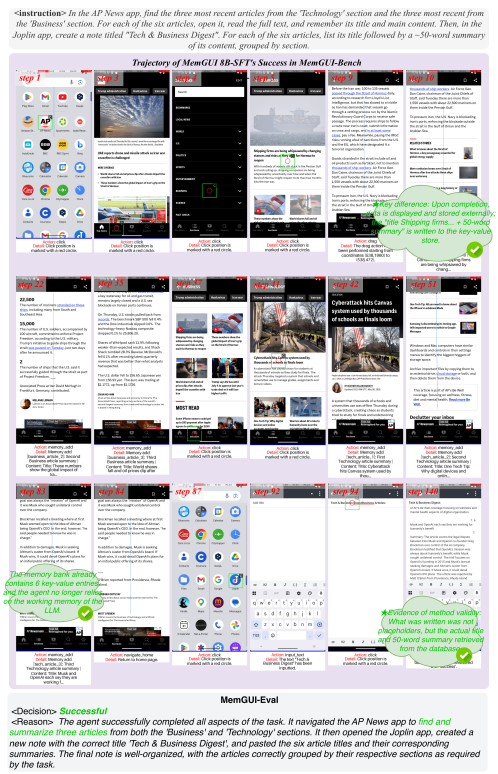
**Format compliance.** Format compliance checks whether the output contains matched opening and closing tags for `<thinking>`, `<folding>`, `<tool_call>`, `<ui_observation>`, and `<action_intent>`. The `<folding>` tag is treated as optional for step 1, matching the CONACT protocol. The evaluator also computes a trajectory-level full-match rate, defined as the fraction of trajectories for which every evaluated step achieves action match success; we use this only as a diagnostic because the offline setting fixes gold context at each step rather than rolling out model-induced states.

## F TRAJECTORY COMPLETION COMPARISONS

Figures 10–13 show paired executions on the same task under a ReAct-style base agent and the corresponding MemGUI-Agent agent. These examples complement the aggregate results by showing how explicit context actions help preserve task-relevant facts across long trajectories.

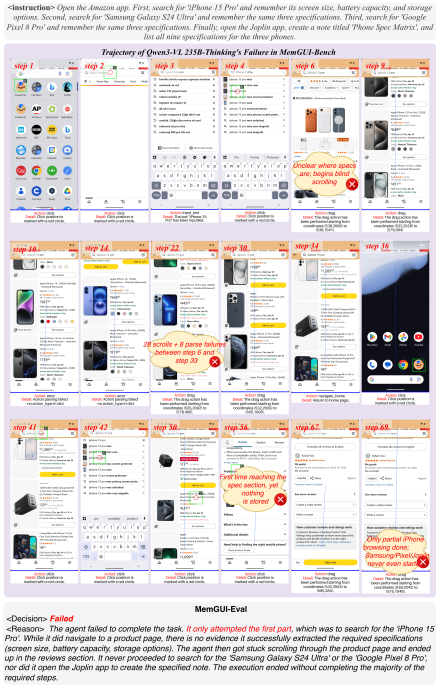



(a) Qwen3-VL-8B-Instruct base.

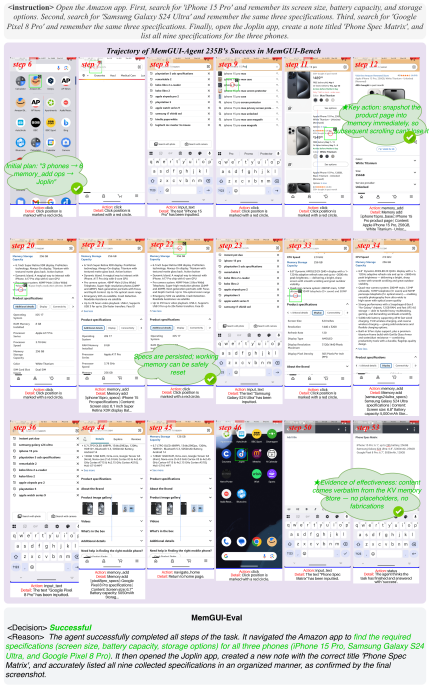



(b) MemGUI-8B-SFT.

Figure 10: MemGUI-Bench trajectory comparison on the same task with an 8B backbone. The base agent fails, while MemGUI-8B-SFT completes the task after learning CONACT behavior from MemGUI-3K.

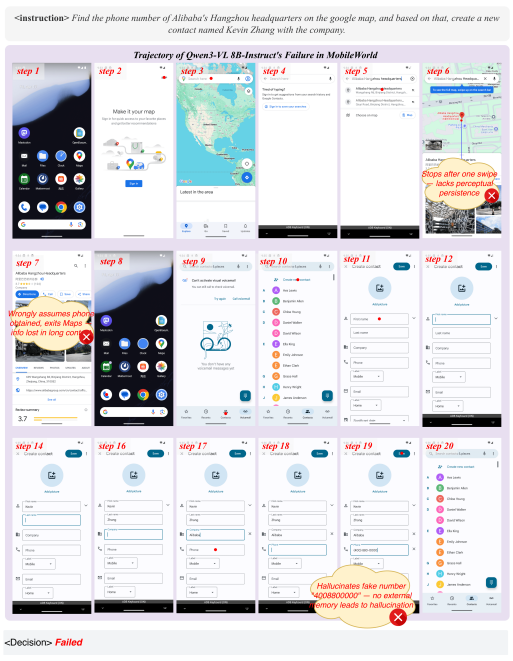



(a) Qwen3-VL-235B-Thinking with ReAct-style prompting.

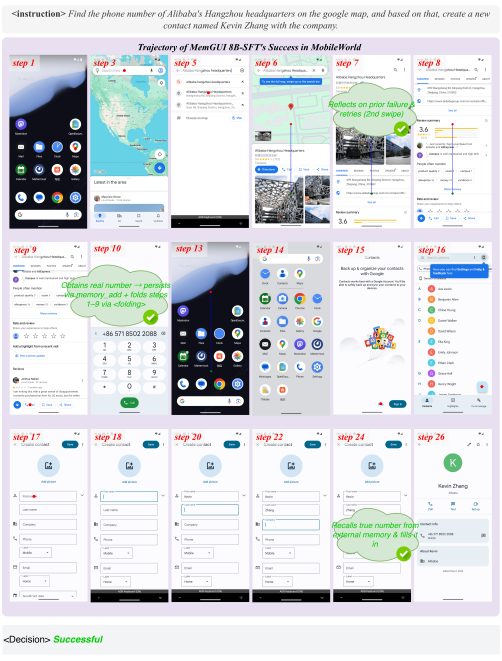



(b) MemGUI-Agent-235B with zero-shot CONACT.

Figure 11: MemGUI-Bench trajectory comparison on the same task with Qwen3-VL-235B-Thinking. MemGUI-Agent-235B keeps the weights unchanged and changes only the prompting/action protocol.

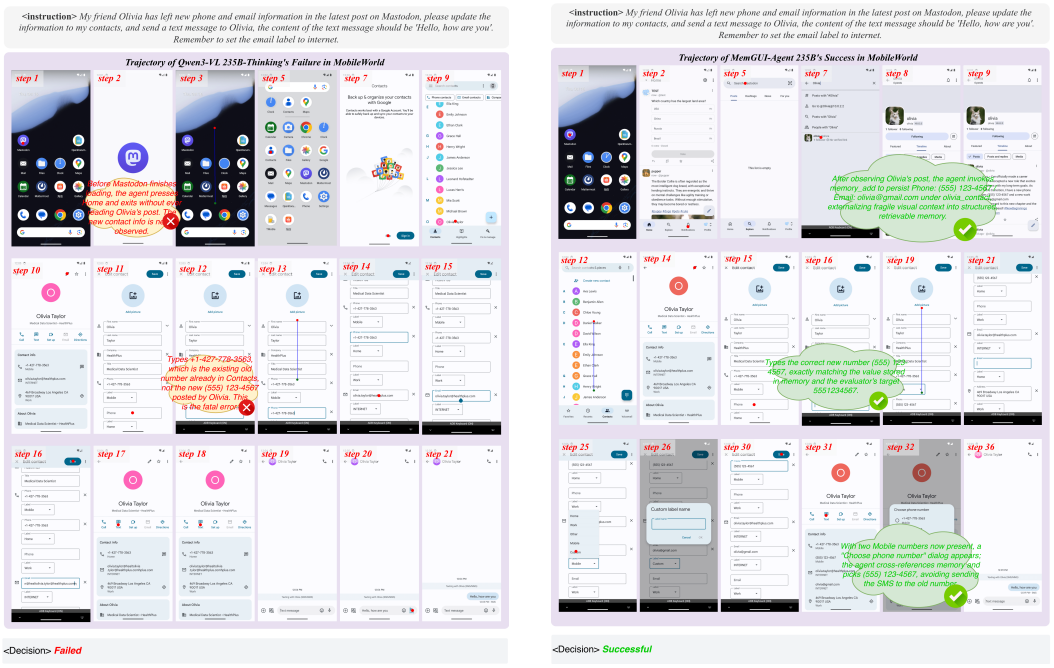



(a) Qwen3-VL-8B-Instruct base.




(b) MemGUI-8B-SFT.

Figure 12: MobileWorld trajectory comparison on the same task with an 8B backbone. The example illustrates that learned context-management behavior transfers beyond the MemGUI-Bench source environment.




(a) Qwen3-VL-235B-Thinking with ReAct-style prompting.

(b) MemGUI-Agent-235B with zero-shot CONACT.

Figure 13: MobileWorld trajectory comparison on the same task with Qwen3-VL-235B-Thinking. Zero-shot CONACT helps the agent maintain compact task state across app transitions without changing the backbone weights.

## G FAILURE TAXONOMY AND ANNOTATION

Figure 7 categorizes failed attempts from the zero-shot Qwen3-VL-235B-Thinking ablation setting on MemGUI-Bench-40 into five dominant failure types. The annotation protocol follows the MemGUI-Bench failure taxonomy Liu et al. (2026). Each failed attempt is assigned to one primary category according to its main cause of failure.

**Process hallucination.** The agent loses track of the task objective or procedural workflow during execution, leading to goal drift, premature termination, irrelevant actions, skipped required steps, or a false belief that an intermediate operation has already been completed.

**Output hallucination.** The agent has observed task-relevant UI information, and often follows the correct workflow, but later stores, recalls, writes, calculates, or reports incomplete or incorrect facts. It includes selective retention failures over multi-item information and final-output transcription errors.

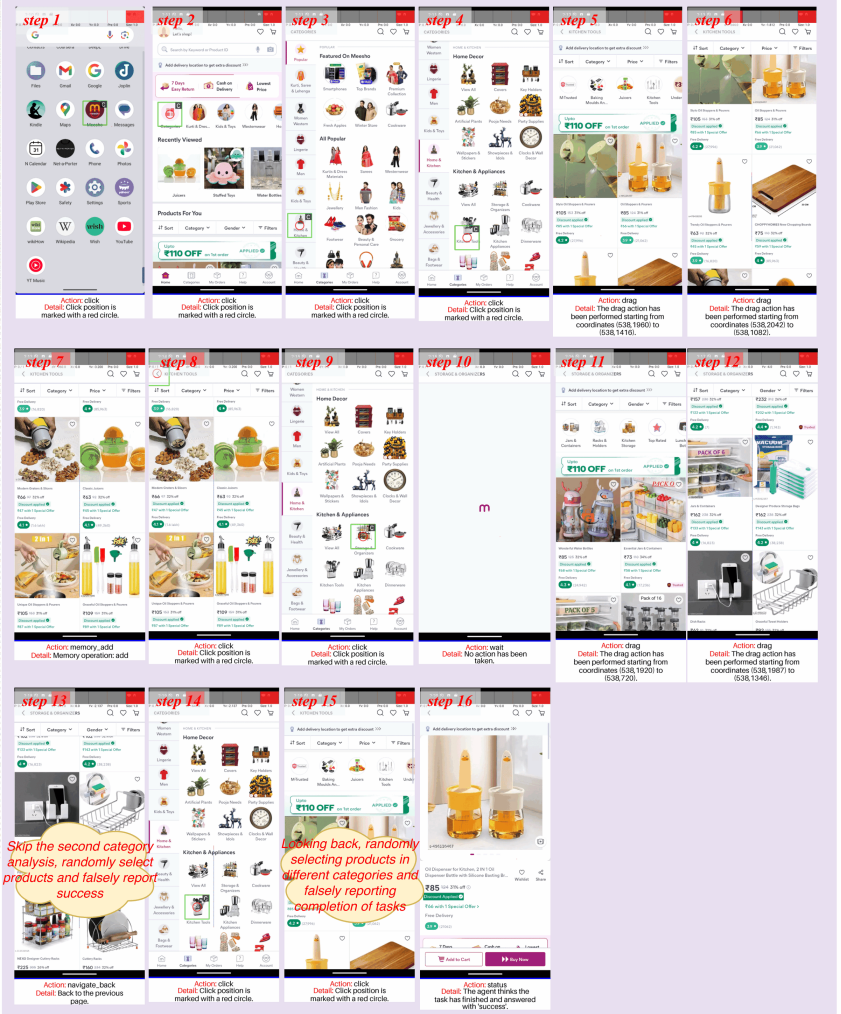
**Knowledge deficiency.** The agent lacks fundamental app, domain, or operational knowledge required for task completion, independent of whether it retained the needed information. Typical cases include misidentifying an app, misunderstanding a common UI convention, or not knowing the skill needed to continue.

**Intent misunderstanding.** The agent misinterprets the user instruction or task condition, then executes an action sequence for the wrong objective. The agent may correctly observe and retain facts but still fail because it optimizes for the wrong sub-goal, comparison criterion, destination, or target information.

**Other.** This category includes failures that do not fit the above classes, such as action-space limitations, environment instability, parsing errors, inefficient execution, or rare system-level failures.

***<instruction>*** *In the DeepL Translate app, translate the following paragraph into five languages: German, French, Korean, Hindi, and Italian: "Please be advised that due to a system upgrade, network services will be unavailable from 11:00 PM on Friday until 5:00 AM on Saturday. The expected downtime is approximately 6 hours. We apologize for any inconvenience this may cause." After remembering all five translations, open the messages app. Important: Do not use the copy-paste function for the translations. Send each translation as a separate message to a different recipient: the German translation to +8613200001111, French to +8613200002222, Korean to +8613200003333, Hindi to +8613200004444, and Italian to +8613200005555.*

***Process Hallucination Bad Case***




***MemGUI-Eval***

*<Decision>* ***Failed***

*<Reason> The agent demonstrated a clear failure in maintaining its procedural memory during a multi-step task. It correctly executed the first major part of the task: navigating to the 'Kitchen Tools' category and performing an analysis (indicated by the `memory_add` operation in Step 7). It then correctly proceeded to the second part by navigating to the 'Storage & Organizers' category. However, at this point, it completely abandoned the required sub-task. Instead of analyzing the items in the second category, it immediately navigated away (Step 13), returned to the first category (Step 14), and selected an item. This behavior shows the agent 'lost track' of its current objective (analyzing the second category) and fell back to a previous, now irrelevant, state. This is a classic example of process-type memory hallucination, where the agent forgets its place in a sequence of operations and begins to perform illogical or repetitive actions.*

Figure 14: Representative process-hallucination failure. The agent deviates from the required workflow or falsely assumes that a necessary intermediate operation has been completed, causing progress loss even when the task remains feasible.

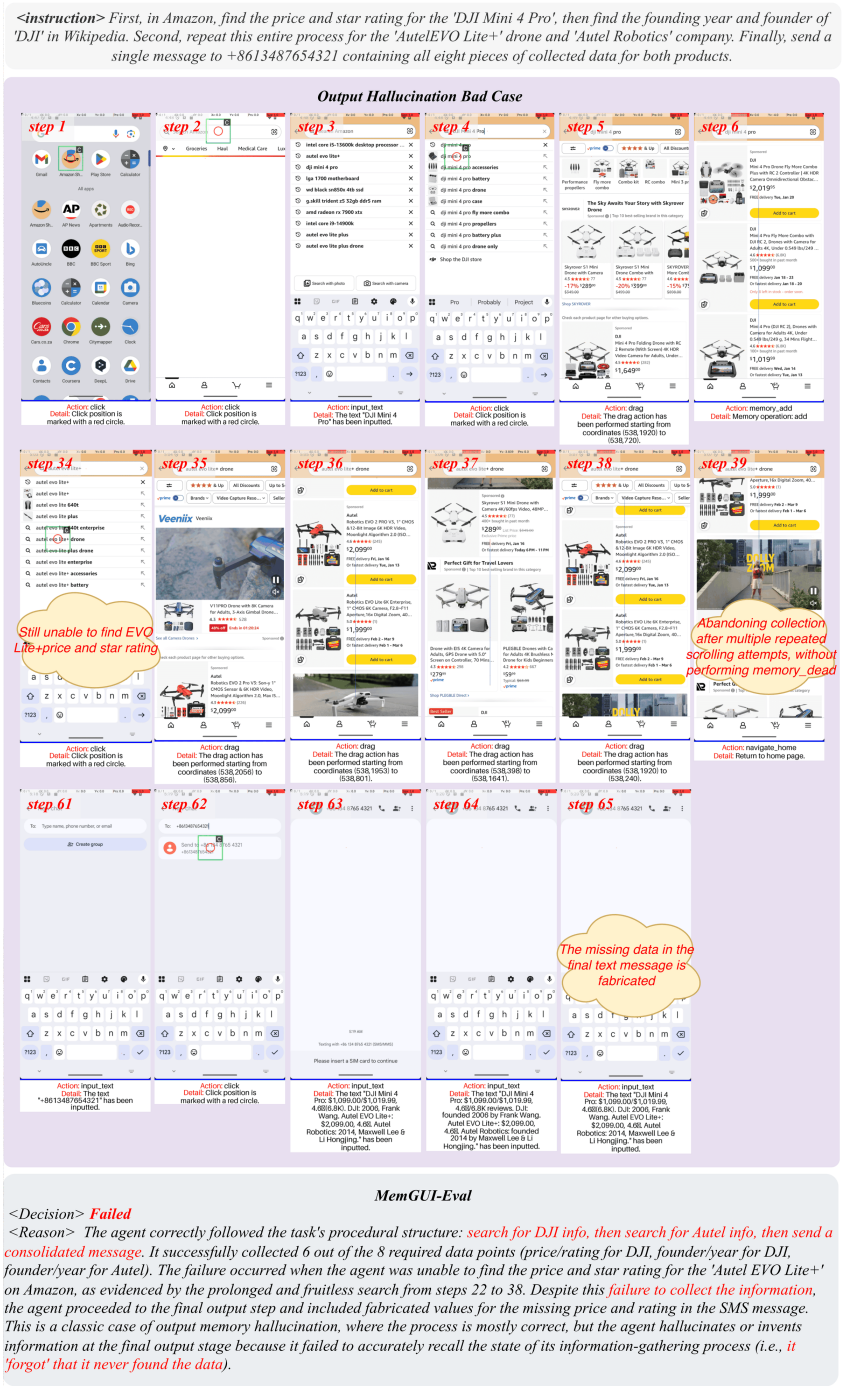
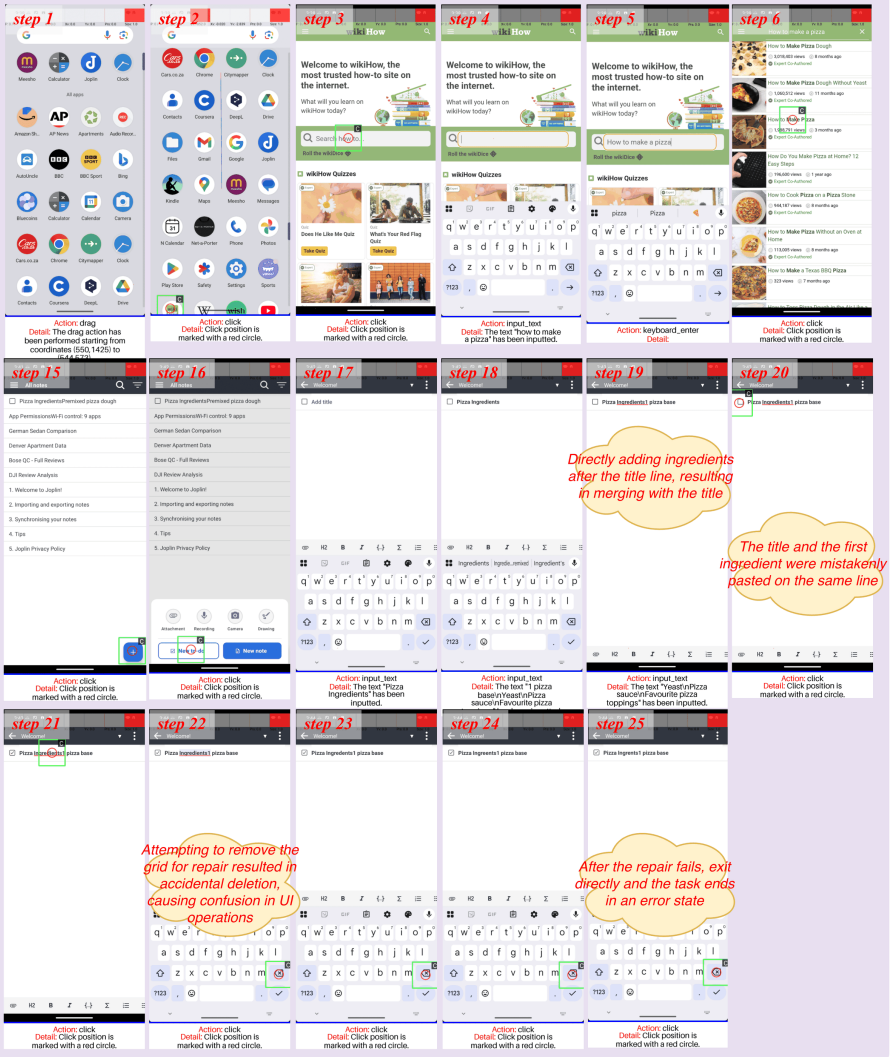



Figure 15: Representative output-hallucination failure. The agent reaches relevant UI evidence but later reports, stores, or transcribes an incorrect fact, showing a failure in retaining exact task-critical information.

***<instruction>*** *Open the wikiHow app and search for "how to make a pizza". Create a checklist in the Joplin app named "Pizza Ingredients" with the first four ingredients listed. Then, open the Calculator app and calculate the total cost, assuming each of the four ingredients costs $4.25.*

***Knowledge Deficiency Bad Case***




***MemGUI-Eval***

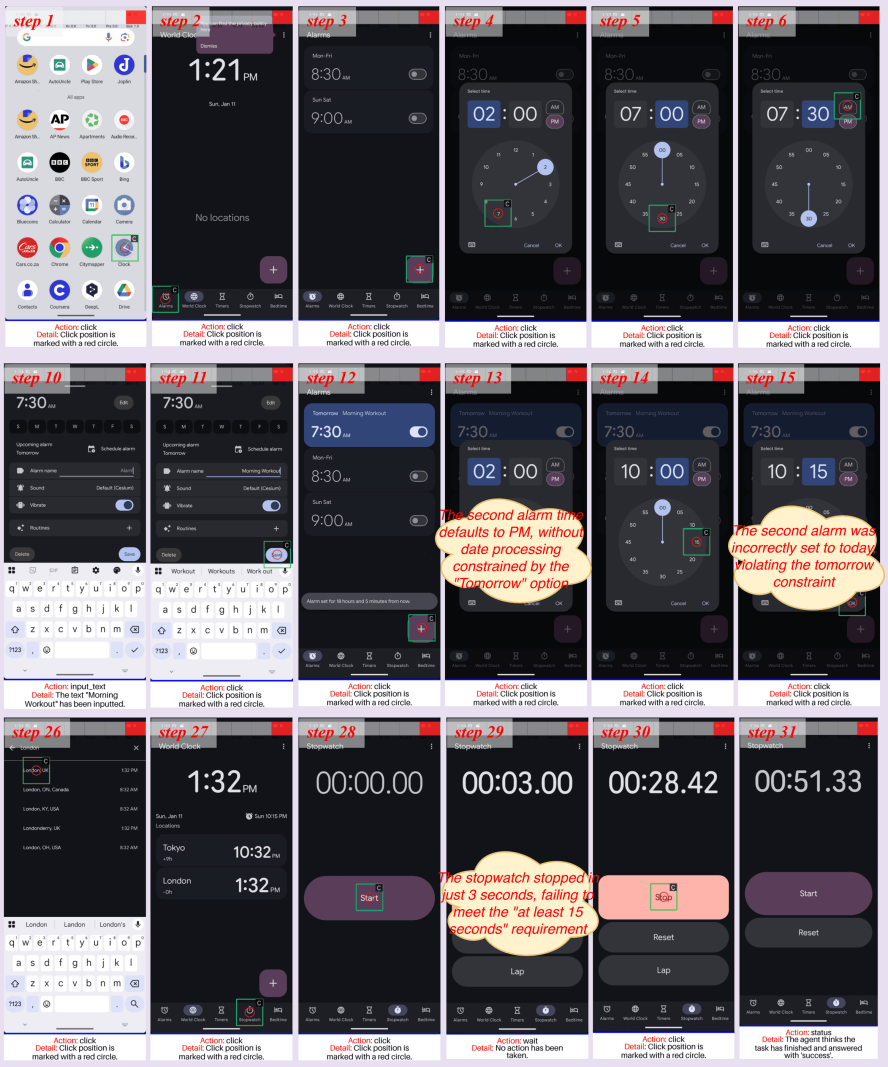
*<Decision>* ***Failed***

*<Reason> The agent successfully executed the first part of the task: finding the ingredients in wikiHow and storing them in memory. It also correctly identified the next steps: opening Joplin and creating a checklist. The failure occurred specifically during the interaction with the Joplin app. The agent incorrectly merged the checklist title ('Pizza Ingredients') and the first ingredient ('1 pizza base') into a single input field, creating a malformed entry. This was not a memory failure, as it correctly recalled both pieces of information. Instead, it was a failure to understand the correct UI operation sequence for creating a titled checklist in this specific application. The agent's subsequent failed attempts to edit and fix the entry further demonstrate its lack of knowledge on how to operate the Joplin UI effectively. It knew *what* to do but not *how* to do it within the given app.*

Figure 16: Representative knowledge-deficiency failure. The agent lacks an app-level, domain-level, or operation-level prerequisite needed to continue, so better context retention alone is insufficient.

***<instruction>*** *Open the Clock app. Add two new alarms: one for 7:30 AM tomorrow labeled "Morning Workout", and another for 10:15 PM tomorrow labeled "Read Book". Then, navigate to the world clock and add "Tokyo, Japan" and "London, UK". Finally, switch to the stopwatch and let it run for at least 15 seconds before stopping (but not resetting) it.*

***Intent Understanding Bad Case***




***MemGUI-Eval***

*<Decision>* ***Failed***

*<Reason> The agent successfully completed the first part of a complex instruction (setting an alarm for 'tomorrow') but failed to apply the same constraint to the second, similar part of the instruction. It set the first alarm for 7:30 AM 'tomorrow' correctly, but then set the second alarm for 10:15 PM 'today'. This indicates that while the agent possessed the capability, it failed to consistently apply all key requirements from the user's prompt across all relevant sub-tasks. The failure is not a lack of knowledge or a complete memory loss of the overall goal, but rather an incomplete application of the user's specified intent, specifically ignoring a key temporal constraint for the second alarm.*

Figure 17: Representative intent-understanding failure. The agent follows a coherent interaction path but optimizes for the wrong interpretation of the user request, causing failure despite available UI evidence.

Full CONACT mainly reduces process hallucination and output hallucination, while knowledge deficiency and intent misunderstanding remain comparatively stable. This supports the interpretation that the primary gains come from improved context management rather than unrelated model capabilities.

# H FULL PROMPT TEMPLATES

This section reproduces the rollout prompts used by the experimental agents. We present the system prompt and the per-step user prompt for the ReAct-style baseline and MemGUI-Agent.

## H.1 REACT-STYLE QWEN3-VL PROMPT

Prompt for ReAct-Style Qwen3-VL (system prompt).

```
You are a helpful assistant that can help with tasks on a mobile device.

# Tools

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

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{
    "type": "function",
    "function": {
        "name": "mobile_use",
        "description": "Use a touchscreen to interact with a mobile device, and take screenshots.
* This is an interface to a mobile device with touchscreen. You can perform actions like clicking, typing,
↪  swiping, etc.
* Some applications may take time to start or process actions, so you may need to wait and take successive
↪  screenshots to see the results of your actions.
* The screen's resolution is 1000x1000 (coordinates range from 0 to 1000).
* Make sure to click any buttons, links, icons, etc with the cursor tip in the center of the element. Don't
↪  click boxes on their edges unless asked.",
        "parameters": {
            "type": "object",
            "properties": {
                "action": {
                    "type": "string",
                    "description": "The action to perform. The available actions are:
* `click`: Click the point on the screen with coordinate (x, y).
* `long_press`: Press the point on the screen with coordinate (x, y) for specified seconds.
* `swipe`: Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).
* `type`: Input the specified text into the activated input box.
* `answer`: Output the answer.
* `system_button`: Press the system button.
* `wait`: Wait specified seconds for the change to happen.
* `terminate`: Terminate the current task and report its completion status.",
                    "enum": [
                        "click",
                        "long_press",
                        "swipe",
                        "type",
                        "answer",
                        "system_button",
                        "wait",
                        "terminate"
                    ]
                },
                "coordinate": {
                    "type": "array",
                    "description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge)
                    ↪  coordinates to move the mouse to. Required only by `action=click`,
                    ↪  `action=long_press`, and `action=swipe`."
                },
                "coordinate2": {
                    "type": "array",
                    "description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge)
                    ↪  coordinates to move the mouse to. Required only by `action=swipe`."
                },
                "text": {
                    "type": "string",
                    "description": "Required only by `action=type` and `action=answer`."
                },
                "time": {
                    "type": "number",
                    "description": "The seconds to wait. Required only by `action=long_press` and
                    ↪  `action=wait`."
                },
                "button": {
```

```
                    "type": "string",
                    "description": "Back means returning to the previous interface, Home means returning to the
                    ↪  desktop, Menu means opening the application background menu, and Enter means pressing
                    ↪  the enter. Required only by `action=system_button`",
                    "enum": [
                        "Back",
                        "Home",
                        "Menu",
                        "Enter"
                    ]
                },
                "status": {
                    "type": "string",
                    "description": "The status of the task. Required only by `action=terminate`.",
                    "enum": [
                        "success",
                        "failure"
                    ]
                }
            },
            "required": [
                "action"
            ]
        }
    }
}
</tools>

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

# Response format

Response format for every step:
1) Thinking: a <thinking>...</thinking> block explaining the next move (no multi-step reasoning).
2) Tool call: a <tool_call>...</tool_call> block containing only the JSON: {"name": <function-name>,
↪  "arguments": <args-json-object>}.
3) Conclusion: a short <conclusion>...</conclusion> block describing what to do in the UI.


Rules:
- Output exactly in the order: <thinking>,<tool_call>,<conclusion>.
- Be brief: one sentence for <thinking>, one for <conclusion>.
- Do not output anything else outside those three parts.
- **Task Feasibility**: If you determine the task is INFEASIBLE (e.g., required resources don't exist,
↪  prerequisites are missing, or the task is impossible to complete), use `action=terminate` with
↪  `status="failure"` immediately.
- If task is successfully completed, use `action=terminate` with `status="success"`.
- **Search Tip**: When search suggestions appear after typing, **click the suggestion directly** - most mobile
↪  apps do NOT respond to Enter key.
```

Prompt for ReAct-Style Qwen3-VL (per-step user prompt).

```
The user query: {goal}
Task progress (You have done the following operation on the current device): {action_history_desc_str}<image>.
↪  Before answering, explain your reasoning step-by-step in <thinking></thinking> tags, and insert them
↪  before the <tool_call></tool_call> XML tags. After answering, summarize your action in
↪  <conclusion></conclusion> tags, and insert them after the <tool_call></tool_call> XML tags.
```

### H.2 MemGUI-Agent Prompt

Prompt for MemGUI-Agent (system prompt).

```
You are MemGUI-Agent, an end-to-end GUI agent instantiated with CONACT, a context-as-action design for
↪  explicit UI memory and proactive context folding.

## Overview
Your task is to analyze a given user task, review current screenshot, compressed history, recent step record,
↪  and memory state, then determine the next UI action AND the context actions needed to manage your history
↪  and memory.

Your context consists of:
1. **Folded UI State**: Explicitly stored critical information extracted from UI
2. **Folded Action History**: Compressed records of past actions
3. **Recent Step Record**: Full details of your most recent step (to be folded this turn)
```

```
Under CONACT, these three fields form the structured context state, and the model may emit both UI actions and
↪  context actions (history folding or UI memory operations).

# Tools

You may call ONE function per step.

<tools>
[
    {
        "type": "function",
        "function": {
            "name": "mobile_use",
            "description": "Use a touchscreen to interact with a mobile device, manage memory, and take
            ↪  screenshots.
* This is an interface to a mobile device with touchscreen. You can perform UI actions (clicking, typing,
↪  swiping, etc.) OR memory operations (add, update, delete).
* **IMPORTANT**: You can only perform ONE action per step - either a UI action OR a memory operation.
* Some applications may take time to start or process actions, so you may need to wait and take successive
↪  screenshots to see the results of your actions.
* The screen's resolution is 1000x1000 (coordinates range from 0 to 1000).
* Make sure to click any buttons, links, icons, etc with the cursor tip in the center of the element.",
            "parameters": {
                "type": "object",
                "properties": {
                    "action": {
                        "type": "string",
                        "description": "The action to perform. Available actions:

**UI Operations:**
* `click`: Click the point on the screen with coordinate (x, y).
* `long_press`: Press the point on the screen with coordinate (x, y) for specified seconds.
* `swipe`: Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).
* `type`: Input the specified text into the activated input box.
* `answer`: Output the answer.
* `system_button`: Press the system button.
* `wait`: Wait specified seconds for the change to happen.
* `terminate`: Terminate the current task and report its completion status.

**Memory Operations:**
* `memory_add`: Store new information to memory.
* `memory_update`: Update existing memory item.
* `memory_delete`: Delete unnecessary memory items.",
                        "enum": [
                            "click", "long_press", "swipe", "type", "answer",
                            "system_button", "wait", "terminate",
                            "memory_add", "memory_update", "memory_delete"
                        ]
                    },
                    "coordinate": {
                        "type": "array",
                        "description": "(x, y): Coordinates for click, long_press, swipe start."
                    },
                    "coordinate2": {
                        "type": "array",
                        "description": "(x, y): End coordinates for swipe."
                    },
                    "text": {
                        "type": "string",
                        "description": "Text for type and answer actions."
                    },
                    "time": {
                        "type": "number",
                        "description": "Seconds to wait for long_press and wait."
                    },
                    "button": {
                        "type": "string",
                        "description": "System button: Back, Home, Menu, Enter.",
                        "enum": ["Back", "Home", "Menu", "Enter"]
                    },
                    "status": {
                        "type": "string",
                        "description": "Task completion status for terminate.",
                        "enum": ["success", "failure"]
                    },
                    "memory_id": {
                        "type": "string",
                        "description": "Unique identifier for memory operations."
                    },
                    "description": {
                        "type": "string",
                        "description": "Brief summary of memory content."
                    },
                    "content": {
                        "type": "string",
```

```
                    "description": "COMPLETE content to store (not summaries)."
                }
            },
            "required": ["action"]
        }
    }
}
]
</tools>

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

### Response Format (5 parts in order)

1) **Thinking**: `<thinking>...</thinking>` - Your reasoning for next action AND folding decision.

2) **Folding Directive**: `<folding>...</folding>` - JSON object specifying how to compress history:
   ```json
   {"range": [start_step, current_step], "summary": "Compressed description"}
   ```
   - **Step-level Distillation** (start_step == current_step): Distill only the latest step into a compact record
     Example: `{"range": [5, 5], "summary": "[Step 5] Opened Settings app and navigated to Wi-Fi."}`
   - **Span-level Abstraction** (start_step < current_step): Abstract a multi-step span into one reusable summary
     Example: `{"range": [3, 7], "summary": "[Steps 3-7] Searched for target app, failed multiple times due to network error, finally found it via alternative route."}`

   **PREFER Span-level Abstraction** when a sub-task has a clear outcome and intermediate details are irrelevant

   **Only use Step-level Distillation** for steps where the specific action matters for future reference

3) **Tool Call**: `<tool_call>...</tool_call>` - Your action (UI or memory operation).

4) **UI Observation**: `<ui_observation>...</ui_observation>` - **DETAILED** screen description. Include exact text, numbers, prices, names, counts visible. Quote task-relevant info verbatim.

5) **Action Intent**: `<action_intent>...</action_intent>` - What you INTEND to do next.

### Rules:
- Output exactly in order: <thinking>, <folding>, <tool_call>, <ui_observation>, <action_intent>
- First step (step 1): Skip <folding> as there's no history to fold
- ALWAYS include <folding> from step 2 onwards
- In <folding>, "range" must include the current step being folded
- **Aggressively use Span-level Abstraction** to keep history compact:
  - Merge related steps whenever a sub-task completes
  - Only use Step-level Distillation for steps with critical standalone findings
- ONE ACTION PER STEP
- **Task Feasibility**: If task is INFEASIBLE, use `terminate` with `status="failure"`
- **Memory**: Store COMPLETE information, not summaries or references
- **Search Tip**: When search suggestions appear after typing, **click the suggestion directly** - most mobile apps do NOT respond to Enter key
```

Prompt for MemGUI-Agent (per-step user prompt).

```
### User Query
{goal}

### Folded Action History
{self._format_state_summaries()}

### Recent Step Record (To be folded this turn)
{self._format_latest_interaction()}

### Folded UI State
{self._format_memory_state()}

### Current Screenshot
[Screenshot attached]

Please analyze the screenshot and determine your next action. Remember to:
1. Output <thinking> with your reasoning
2. {"Skip <folding> for the first step" if self.current_step == 1 else "Output <folding> to compress your previous step(s)"}
3. Output <tool_call> with your action
4. Output <ui_observation> with **DETAILED** screen description (include ALL task-relevant info: exact text, numbers, prices, names, counts visible on screen)
5. Output <action_intent> describing your planned action
```