Naive Visual Memory is Not Enough: A Failure-Mode Study of GUI Agents

Seoyoung Choi [1] **Minseok Ko** [2] **Hyunseok Lee** [3] **Kunwoong Kim** [3] **Woomin Song** [3] **Chanseok Jeon** [4] **Jinwoo Shin** [3]

Abstract

Graphical User Interface (GUI) agents are increasingly used to automate complex computer tasks across applications, websites, and operating systems. To improve their reliability, recent work has introduced experiential memory, where agents retrieve prior trajectories to guide decision-making in similar states. More recent approaches further extend this idea to visual memory by storing and retrieving screenshots from past interactions, providing agents with richer contextual information than text-only memories. However, the role of visual memory in GUI agents remains insufficiently understood: it is unclear which failures visual memory mitigates, or which failures it exacerbates. To systematically analyze the effect of visual memory, we introduce a taxonomy of four GUI agent failures (i.e., *cognitive failure*, *visual state misunderstanding*, *hidden operation blindness*, and *grounding error*) that map to distinct stages of the perception-reasoning-action pipeline. We find that prepending full-image memory has a divergent effect on the failure distribution: it reduces state-level failures but worsens action-level ones, and increases hidden operation blindness and grounding error. Motivated by this finding, we propose Action-Grounded Visual Memory (AGMem), an action-grounded memory framework for GUI agents. The core idea of AGMem is to store image crops that capture the local GUI region closely related to a successful action or a recovery, rather than storing full screenshots. Experiments on OSWorld show that AGMem improves task success rates relatively by 33.3% over full-image memory. These results demonstrate that AGMem is an effective representation for visual memory in GUI agents.

[1]Seoul National University [2]POSTECH [3]KAIST [4]Pion Corporation. Correspondence to: Seoyoung Choi <chlsy07@snu.ac.kr>, Jinwoo Shin <jinwoos@kaist.ac.kr>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Graphical User Interface (GUI) control is often determined by local visual states rather than by the entire screen. These local states recur across tasks and applications, providing reusable cues for deciding what action should be taken next. Recent advances in vision-language models (VLMs) have enabled GUI agents that operate directly from screenshots and natural-language instructions, and prior works have scaled them by training on massive decision-making trajectories (Xu et al., 2025; Wang et al., 2026; Wu et al., 2025). Despite this progress, recent VLM-based GUI agents still fail in non-trivial ways, and the structure of these failures should receive attention.

Recent works have proposed experiential working memory to tackle this problem. These methods log past decision trajectories and, at test time, retrieve the trajectories most relevant to the current state to guide action selection (Agashe et al., 2024; Kagaya et al., 2024). More recent approaches further introduce image-based memory systems, such as Wu et al. (2026), which provide agents with richer, domain-specific contextual information beyond text. These systems typically consume screenshots as monolithic images, without an explicit representation of the local regions that determine actions. Despite this progress, the role of visual memory in GUI agents remains insufficiently understood. Its effects have not been systematically characterized in terms of how it improves performance, which failure modes it mitigates, and where it may introduce regressions.

This leads to a question: *What are the core failure modes in GUI agents, and how does visual memory mitigate them?*

In this work, we categorize and analyze the common failure patterns of existing GUI agents. Specifically, we investigate the failure patterns in representative benchmarks, including AgentNetBench (Wang et al., 2026), OSWorld (Xie et al., 2024), and WebForge (Yuan et al., 2026). On these benchmarks, we identify four recurring failure modes that together account for the majority of agent errors: (i) Visual State Misunderstanding, (ii) Hidden Operation Blindness, (iii) Cognitive Failure, and (iv) Grounding Error.

Based on these patterns, we then analyze how introducing visual memory shifts the failure distribution. Following Wu

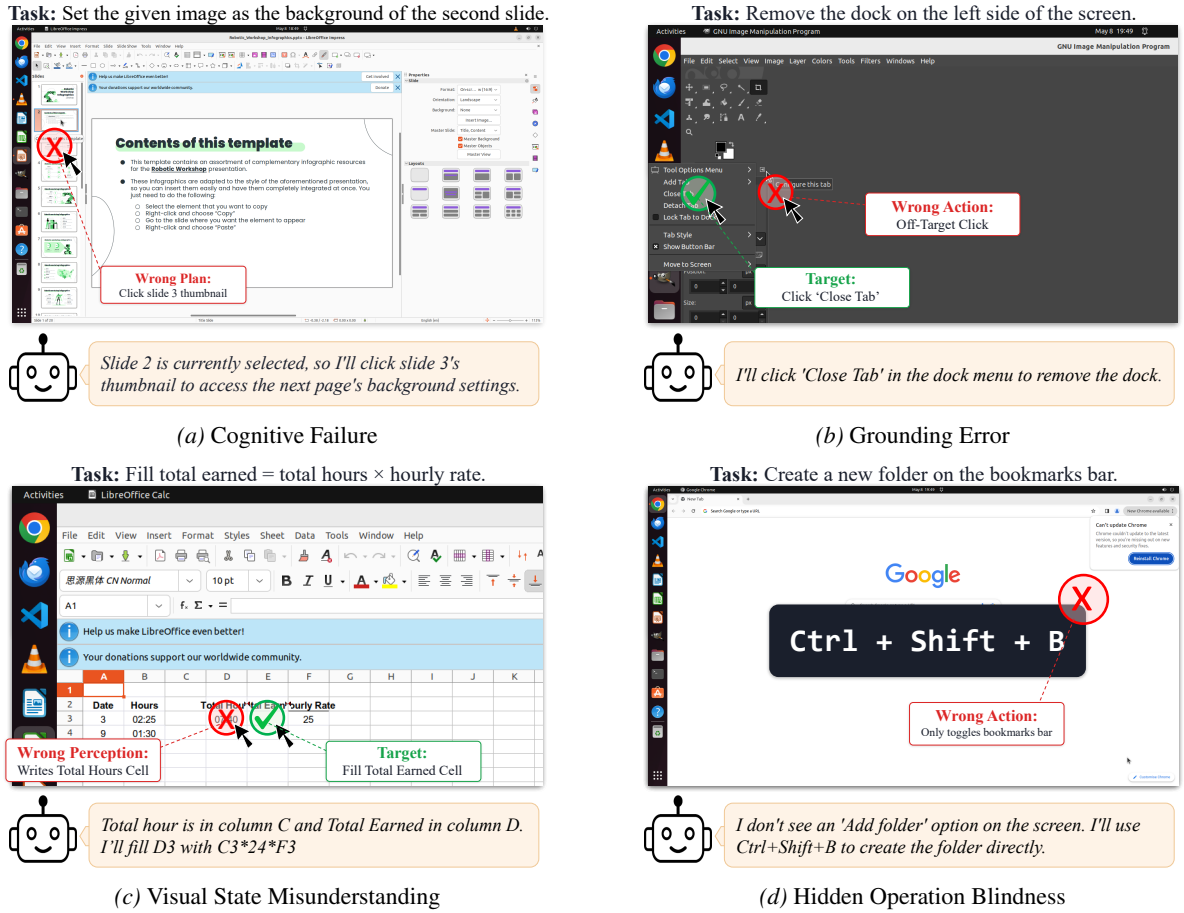

Figure 1. **Four failure modes of GUI agents.** We observe four recurring patterns in agent failures. In (a), the agent should be working on the second page, but instead clicks the thumbnail of the third page. In (b), the agent's target is correct, but the click is slightly off and misses the menu item. In (c), the agent misreads a cell, entering the formula into the Total Hours column instead of Total Earned. In (d), creating a folder requires an operation that is not visible on the screen, but the agent only presses `Ctrl+Shift+B`, a shortcut that toggles the bookmarks bar instead of creating the folder. Red and green markers denote the agent's action and the correct one, respectively.

et al. (2026), we prepend full screenshots of relevant history entries to the agent context. For example, on OSWorld with GPT-5.4-mini (OpenAI, 2026), full-image memory reduces state-level failures such as Visual State Misunderstanding from 73.1% to 69.6% (−3.5 pp), but worsens action-level failures, increasing Hidden Operation Blindness by 11.7 pp. That is, full visual memory injects useful global context for state-level confusion but simultaneously crowds the visual context with task-irrelevant cues that distract from invisible operations and degrade coordinate grounding. This suggests that visual memory is valuable in principle, but *which pixels are stored* matters as much as whether memory is used at all.

To further mitigate this problem, we introduce Action-Grounded Visual Memory (AGMem), an action-grounded information compacting framework for managing GUI agentic memory. The core idea of AGMem is to enable agents to access focused GUI observations through a working memory and a domain-compacting retrieval system. Specifically, we introduce two focusing strategies: (i) a visual memory consisting of cropped images that capture the elements that should be focused on, and (ii) memory narrowing related to the given state. To construct the memory system, we propose an automatic memory processing pipeline that can simply extract working memory trajectories from existing GUI action trajectories. At test-time, the agent retrieves relevant crop-view memories and uses them during planning, allowing it to access decision-critical visual evidence while avoiding the cost of repeatedly processing full-screen images, as depicted in Figure 5.

In experiments on OSWorld with GPT-5.4-mini, AGMem mitigates the failure modes amplified by full-image memory, reducing Visual State Misunderstanding by 37.3 pp and Hidden Operation Blindness by 26.3 pp, while improving end-to-end task accuracy by 6.8 pp.

We highlight the main contributions of this work as follows:

- We introduce a four-class taxonomy of GUI agent failure

Table 1. **The four representative failure modes that can be observed in GUI agents.**

Failure mode	Description
Cognitive failure	The model misunderstands the task goal, current subtask, required next step, success condition, or recovery strategy (e.g., following the wrong plan step, assuming a failed action succeeded, or terminating too early).
Visual state misunderstanding	The model misreads the visible screen state (e.g., believing a dialog or dropdown is open when it is not, misidentifying the selected object, or incorrectly inferring that a sort order has changed).
Hidden operation blindness	The correct action requires an operation not directly visible until another UI action reveals it (e.g., opening a menu, context menu, or overflow menu; using a keyboard shortcut; performing a non-obvious selection).
Grounding error	The model intends the correct visible target and predicts a compatible action type, but the executed coordinate or low-level action misses the target.

modes and an LLM-as-Judge protocol for labeling them across online and offline benchmarks.

- We provide the first empirical analysis of how visual memory shifts the failure distribution, showing that full-image memory cuts state-level failures but amplifies hidden-operation and grounding failures.
- We propose AGMem, an action-grounded visual working memory framework for GUI agents with an automatic memory construction pipeline, which effectively mitigates the failure modes in GUI agents.

2. Failure Modes of GUI Agents

In this section, we provide a comprehensive analysis of the trajectories of GUI agents and classify their failure modes. First, we define a taxonomy of failure modes observed in the GUI agents (Section 2.1). Then, we explain our setups (Section 2.2). Finally, we demonstrate main results of observed errors classified based on our taxonomy (Section 2.3).

2.1. Taxonomy: Four Failure Modes

We identify four recurring failure modes across benchmarks, as illustrated in Figure 1. Each mode corresponds to a different stage of the perception-reasoning-action pipeline that a VLM-based GUI agent executes at every step: planning (*cognitive failure*), perception (*visual state misunderstanding*), action-space inference (*hidden operation blindness*), and execution (*grounding error*). Because these stages require different kinds of evidence to fix (i.e., evidence about state for perception, about affordances for action-space inference, and about pixel-level alignment for execution), a memory mechanism that helps one stage need not help the other stages, which we exploit in the remainder of this section as a measurement instrument.

- **Cognitive failure.** This failure occurs when the model misunderstands the task goal, the current subtask, the required next step, the success condition, or the recovery strategy. It includes choosing the wrong action despite sufficient visible information, following the wrong plan step, assuming a failed action succeeded, terminating too early, or solving a semantically different task.
- **Visual state misunderstanding.** This is when the model misreads the current visible screen. For example, the model may believe it is on the wrong page, think a dialog or dropdown is open when it is not, misidentify the selected object, or incorrectly infer that a form, filter, sort order, or progress state has already changed.
- **Hidden operation blindness.** The correct action requires an operation that is not directly visible until another UI action reveals it. Examples include opening a menu, toolbar, context menu, or overflow menu; using a shortcut; or performing a non-obvious selection or drag.
- **Grounding error.** This failure is observed when the model intends the correct visible target and predicts a compatible action type, but the executed coordinate or low-level action misses the target. We assign this type only when the raw reasoning indicates the same intended target as the ground truth; if the intended target is different, the failure is classified as cognitive failure or visual state misunderstanding instead.

2.2. Setup

Evaluation setup. We evaluate GUI agents on three benchmarks: two online benchmarks, OSWorld (Xie et al., 2024) and WebForge (Yuan et al., 2026), and one offline benchmark AgentNetBench (Wang et al., 2026). Specifically, OSWorld contains 369 tasks across 10 application domains that use a 1920×1080 viewport. We exclude Google Drive tasks and proxy-dependent tasks, resulting in a 316-task evaluation subset. Unless otherwise stated, OSWorld results are reported on this filtered subset. WebForge is evaluated on a randomly sampled subset of 50 tasks due to the cost of online browser-agent evaluation. Each episode is limited to 50 steps unless explicitly stated. Also, we evaluate AgentNetBench on pre-captured screenshot sequences and a given task instruction. We report the average task score, computed from the benchmark's composite action-matching metric aggregated over all tasks.

Labeling setup. Based on our taxonomy in Table 1, we use a Codex-based LLM-as-Judge to annotate failure modes from full trajectory evidence, rather than detector outputs or benchmark summaries. Labels are assigned only to failed

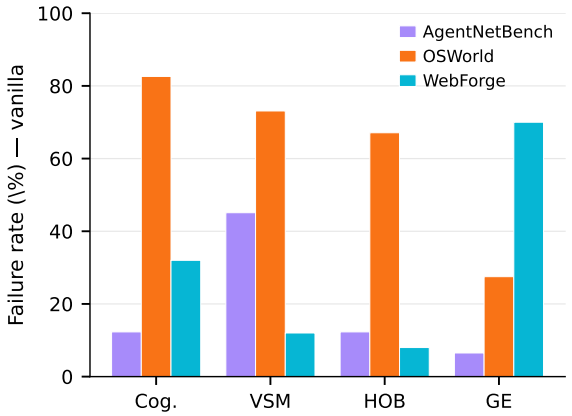

Figure 2. **Failure-mode distributions.** Failure-mode distribution of GPT-5.4-mini across OSWorld, AgentNetBench, and WebForge. Each benchmark exhibits a distinct dominant mode: hidden operation blindness on OSWorld, grounding error on WebForge, and visual state misunderstanding on AgentNetBench.

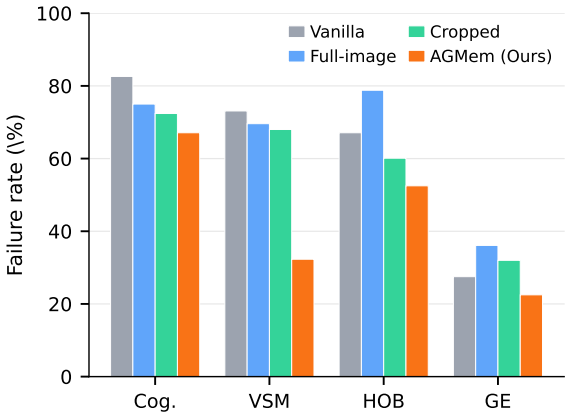

Figure 3. **Per-mode failure rates on OSWorld with GPT-5.4-mini.** Full-image memory *reduces* state-level failures (cognitive failure, visual state misunderstanding) but *worsens* action-level failures (hidden operation blindness, grounding error). AGMem is the only configuration that consistently reduces all failure modes.

or partially failed units and must be directly supported by the trace. The taxonomy is multi-label, so a single unit may receive multiple labels and per-mode rates need not sum to 100%. We annotate OSWorld and WebForge at the task level and AgentNetBench at the action level. All failure-mode percentages are computed over all evaluated units, not only failed units.

GUI agent. A GUI agent is a large language model (LLM)-based autonomous agent that perceives and interacts with on-screen interface elements, such as screenshots, through human-like actions such as clicking and typing (Nguyen et al., 2025). We follow the standard screenshot-based GUI agent setup (Xie et al., 2024; Cheng et al., 2024). Specifically, at each step t, an LLM-based policy receives the instruction I, a short history h_t of recent screenshot-action pairs, the current screenshot o_t, and a set of retrieved memory steps $\mathcal{R}_t$ from a memory bank $\mathcal{M}$, and emits an action a_t: $a_t \sim \pi\big(\cdot \mid I,\, h_t,\, o_t,\, \mathcal{R}_t\big)$.

2.3. Main results

GUI agents frequently fall into four failure modes. Figure 2 shows the vanilla failure distribution across the three benchmarks with GPT-5.4-mini. Each environment has a distinct dominant mode: hidden operation blindness on OSWorld (67.1%), grounding error on WebForge (70.0%), and visual state misunderstanding on AgentNetBench (45.1%). Three modes—(i) cognitive failure, (ii) visual state misunderstanding, and (iii) grounding error-appear across all benchmarks, while hidden operation blindness is desktop-specific.

Per-benchmark trends reflect environment characteristics. The dominant failure modes differ across benchmarks because each environment stresses a different part of the perception-reasoning-action pipeline. OSWorld's high cognitive failure (82.6%) and hidden operation blindness (67.1%) reflect the long-horizon, menu-heavy nature of desktop applications, where essential commands are routinely buried in menus, context menus, ribbons, and keyboard shortcuts. WebForge's grounding error dominance (70.0%) reflects the dense layout of web interfaces with many small click targets, making pixel-level grounding more brittle than in larger desktop UIs; correspondingly, hidden operation blindness is rare (8.0%) because web controls are typically exposed directly on the page. AgentNetBench's lower absolute rates and the relative prominence of visual state misunderstanding (45.1%) follow from its per-action evaluation, which isolates each step's state-recognition requirement and sidesteps long-horizon planning. These various aspects of the benchmarks reflect the diverse modes of GUI agent usage and suggest where each evaluation is the most useful.

Full-image memory has a divergent effect. As shown in Figure 3, adding full-image memory has a divergent effect on the failure distribution. On the one hand, it reduces state-level failures by injecting global context that helps the agent re-orient to the task and the visible state: cognitive failure drops from 82.6% to 75.0% (-7.6 pp) and visual state misunderstanding drops from 73.1% to 69.6% (-3.5 pp). On the other hand, the same full-screen retrieved screenshots substantially expand the visual context with task-irrelevant elements that distract the model from less salient operations. This shows up as an increase in two action-level failures: hidden operation blindness rises from 67.1% to 78.8% ($+11.7$ pp) and grounding error rises from 27.5% to 36.1% ($+8.6$ pp). The net effect is that full-image memory trades state-level errors for action-level errors rather than consistent improvement.

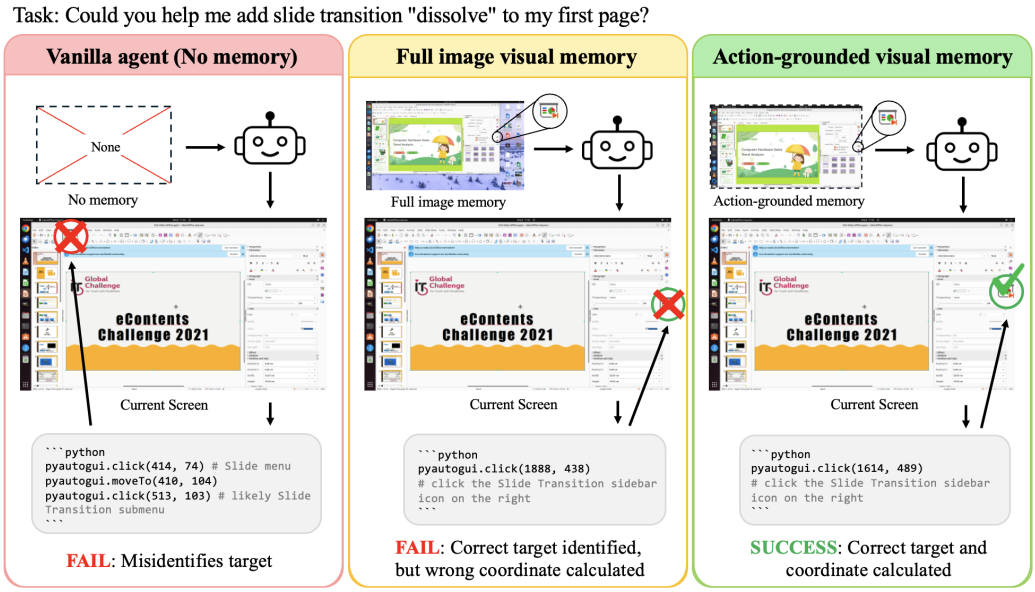

Figure 4. **Illustration of action-grounded visual memory.** Without memory, an agent may select an incorrect UI target. Full-image visual memory can provide useful prior experience, but the action-relevant cue may remain small or spatially ambiguous within the full screenshot, causing incorrect grounding. Action-grounded visual memory focuses the retrieved example on the region associated with the demonstrated action, making the relevant target easier to identify and localize.

Directions for improvement. This divergent effect points to a clear improvement target. To preserve the gains in state-level failures while avoiding the costs on action-level failures, visual memory should be *smaller* (containing only action-relevant pixels rather than full screenshots) and *more selective* (narrowed to memory steps aligned with the current subtask). We translate these two requirements into the design of AGMem in the next section (Section 3).

3. AGMem: mitigating the side effects of visual memory

In this section, we present our proposed framework Action-Grounded Visual Memory (AGMem), which addresses the two directions for improvement identified in Section 2.3. First, we formalize the problem setup and outline AGMem in Section 3.1. Then, we introduce action-grounded memory entries that compact each memory entry to an action-relevant view in Section 3.2. Finally, we describe action-grounded memory candidates that narrow retrieval and provide a separate recovery memory for erroneous states in Section 3.3. An overview of AGMem is provided in Figure 5.

3.1. Overview

To guide the agent through a given long-horizon instruction I, AGMem first decomposes I into a list of *subtasks* (Gao et al., 2024):

$$\mathbf{s} = (s_1, s_2, \ldots, s_p).$$

Here, each s_i is a natural-language subtask derived from the instruction I alone before any action is executed. Note that the number of subtasks p is generally different from the number of steps T, since one subtask may span multiple steps. The pair $(I, \mathbf{s})$ is given to the agent throughout the trajectory. Before emitting each action, the agent self-reports its current subtask, so that action execution is aligned with the current subtask.

We then keep the memory salient in two core components. First, an action-relevant view (Section 3.2). Second, a narrowed retrieval pipeline with a separate recovery memory for erroneous states (Section 3.3).

3.2. Action-grounded visual memory

A desktop screenshot often contains content unrelated to the current action or state, such as background windows, system trays, and idle panels, which can consume memory and occupy the agent's visual context (Lin et al., 2025). Storing all such screenshots would unnecessarily expand the memory bank and burden the agent's visual context with task-irrelevant cues. We therefore reduce each memory entry to its salient form, called the *action-relevant view*. For each step t, we denote this view by C_t, which is a tightly cropped image on which action a_t has important effects.

Construction from trajectories. Starting from screenshot o_t at time t, the agent executes action a_t, and the environment transitions to the next screenshot o_{t+1}.

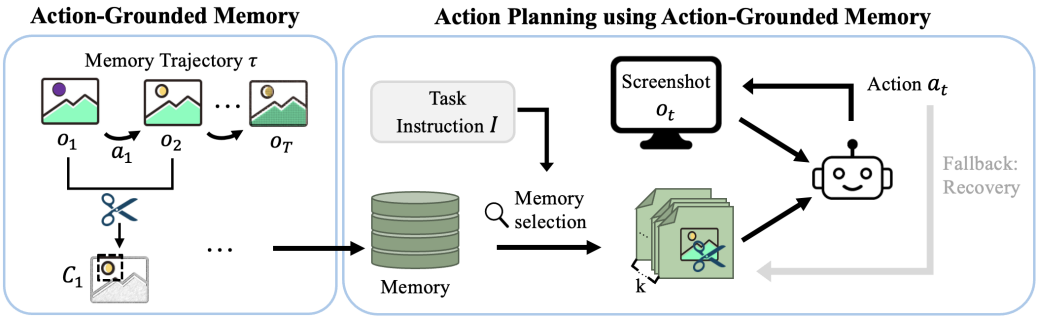

Figure 5. **Overview of AGMem.** AGMem is a compact GUI trajectory memory framework based on action-relevant visual views, salient memory candidates from trajectory- and step-level retrieval, and recovery memory for erroneous states.

Memory structure. We construct the memory bank $\mathcal{M}$ from AgentNetBench dataset (Wang et al., 2026), $\mathcal{D}_{\text{AGNet}}$, where we start from retaining only non-redundant and correct trajectories τ.

For each trajectory $\tau \in \mathcal{D}_{\text{AGNet}}$, an LLM produces a list of post-hoc subtask labels $\mathbf{s}_\tau$ that contain information about what actually happened on the screen during τ, and we segment τ along $\mathbf{s}_\tau$ into sub-trajectories. Consequently, the atomic unit of memory is a *memory step*:

$$m_t = (s_j, a_t, C_t).$$

where $s_j \in \mathbf{s}_\tau$ is the subtask label that the step realizes, a_t is the executed action, and C_t is the action-relevant view.

3.3. Recovery-aware memory retrieval

Searching the entire memory bank at every step or action is both costly and noisy, since most trajectories are irrelevant to the current task and individual steps may produce spurious visual memories. We therefore narrow the search space in two stages. We first select a small pool of trajectories at the task level, and then retrieve individual memory steps within that pool. In addition, when the agent flags an error, we redirect retrieval to a separate *recovery memory*.

Two-stage retrieval. Given the current subtask list $\mathbf{s}$, a trajectory-level retriever first selects a size-k pool $\mathcal{M}_{\text{sub}} \subseteq \mathcal{M}$ of memory trajectories whose post-hoc subtask labels $\mathbf{s}_\tau$ are close to $\mathbf{s}$ in a Sentence-Transformer (Reimers & Gurevych, 2019) embedding space, greedily covering each subtask in $\mathbf{s}$. A step-level retriever then encodes each memory step's subtask label and action-relevant view with the CLIP encoders (Radford et al., 2021), computes a fixed-weight similarity against the current state – the agent's self-reported subtask s_t and the previous step's view C_{t-1} – and returns the top-5 memory steps as $\mathcal{R}_t$.

Recovery-aware memory. A common reason behind the failures of GUI agents is error propagation, where the agent enters an erroneous state and the subsequent action proposals are made under a faulty premise. To address this issue, we add recovery-aware verification memory which teaches the agent *how to recover*. Specifically, an LLM-based *recovery-state detector* identifies erroneous steps during operation. When an error is detected, the recovery memory provides examples of correct behaviors that bring the agent back from a faulty state to a valid one. The recovery memory is built from AgentNetBench sub-trajectories that were excluded from the main bank (redundant or incorrect) but were later corrected within the same trajectory. Specifically, each entry pairs a faulty sub-trajectory with the correct step, labeled with its failure mode so retrieval can match the flagged pattern.

4. AGMem experiments

We provide empirical evidence of AGMem by analyzing how it improves the failure-mode distribution that full-image memory induces (Section 4.1), and end-to-end task performance against existing baselines (Section 4.2).

4.1. Failure-mode analysis

We use the failure-mode taxonomy of Section 2.1 to measure how AGMem changes the failure distribution on OSWorld with GPT-5.4-mini.

AGMem reduces all failure modes. As shown in Table 2, AGMem consistently reduces all four failure modes against the vanilla agent. For example, visual state misunderstanding decreases by 40.8 pp. This suggests that compact action-relevant crops are particularly effective at processing the small subset of pixels that determine state-level decisions. Notably, while hidden operation blindness and grounding error are the two failure modes that are worsened by full-image memory, AGMem also reduces them. This indicates the gains from memory persist even when the visual context is restricted to action-relevant regions.

Cropping alone is insufficient. The cropped configura-

Table 2. **Failure-Mode Analysis on OSWorld.** Comparison of overall Success Rate (Acc.) and the distribution of specific failure modes across different visual memory configurations.

Method	Overall	Failure Modes (Root Cause %)			
	Acc. (%)	*Cognitive*	*Visual Mis.*	*Hidden Op.*	*Grounding*
Vanilla Agent	18.3	82.6	73.1	67.1	27.5
+ Visual Memory	20.4	75.0	69.6	78.8	36.1
+ Crop	25.8	72.4	68.3	60.3	32.1
AGMem (Ours)	**27.2**	**67.1**	**32.3**	**52.5**	**22.5**

* *Cognitive*: Cognitive Failure / *Visual Mis.*: Visual State Misunderstanding / *Hidden Op.*: Hidden Operation Blindness / *Grounding*: Grounding Error

Table 3. **AGMem improves performance by effectively leveraging visual memory.** Accuracy (%) and Milestone Accuracy (M. Acc.) of AGMem (Ours) compared with baselines across three benchmarks, for GPT-5.4-mini. For OSWorld and WebForge, we report overall success rate; for AgentNetBench, we report step-level accuracy.

Method	OSWorld	WebForge	AgentNetBench	
	Acc.	Acc.	Acc.	M. Acc.
Vanilla agent	18.3	2.0	25.8	24.2
+ visual memory	20.4	2.0	28.1	33.8
AGMem (Ours)	**27.2**	**2.0**	**28.8**	**34.6**

tion crops memory screenshots but performs retrieval over the full memory bank. Compared to full-image memory, cropping alone partially recovers performance on hidden operation blindness (78.8% to 60.3%) and grounding error (36.1% to 32.1%), but still falls short of AGMem in every mode and especially on visual state misunderstanding (68.3% vs. 32.3%). The remaining gap is closed by subtask-aligned retrieval narrowing, which restricts retrieval to memory steps relevant to the current subtask and prevents off-task crops from re-injecting irrelevant cues.

4.2. Main results

We present the performance of AGMem by comparing GUI agent performance with other baselines. Here we mainly compare with one offline benchmark (i.e., AgentNetBench) and two online benchmarks (i.e., OSWorld and WebForge).

Performance improvement. As shown in Table 3, AGMem improves performance on OSWorld and AgentNetBench, while matching the baselines on WebForge. For instance, on OSWorld, AGMem improves task accuracy by 8.9 pp (from 18.3% to 27.2%) with GPT-5.4-mini. On AgentNetBench, AGMem further improves step accuracy from 25.8% to 28.8% with GPT-5.4-mini.

These results indicate that AGMem is effective beyond a single evaluation setting, with gains on both online desktop-control tasks and offline action-prediction tasks. They further suggest that compact and task-relevant visual memory can provide useful state information for GUI agents without relying on full visual screenshots.

Effectively leveraging visual memory. Notably, simply adding visual memory does not always improve GUI agent performance. For example, on OSWorld, the full-image memory baseline increases the hidden operation blindness rate from 67.1% to 78.8%, compared to the vanilla agent, indicating that storing full screenshots can introduce redundant or distracting context. In contrast, cropped visual memory reduces the hidden operation blindness rate from 67.1% to 60.3% by restricting memory to action-relevant visual regions. Furthermore, AGMem lowers the rate to 52.5% through subtask-aligned retrieval, outperforming both the vanilla agent and the full-image memory baseline. This demonstrates that AGMem does not merely add more history or visual information, but more effectively leverages visual memory by compacting and preserving task-relevant visual evidence.

5. Case studies

Figure 4 shows a representative *hidden operation blindness* case in LibreOffice Impress. The task is to apply the *Dissolve* transition to the first slide. Although the first slide is already selected, the transition settings are not directly visible in the main canvas and must be opened through a right-sidebar tab. The vanilla agent understands the high-level goal but guesses an incorrect route, clicking the top *Slide* menu and a presumed transition submenu. Full-image memory provides a related prior state, but the useful cue is still embedded in a cluttered whole-screen observation, where the relevant sidebar icon is small and competes with many unrelated UI elements. As a result, full-image memory can suggest the correct route but does not reliably ground the action to the precise affordance. In contrast, the paired crop in action-grounded memory suppresses irrelevant visual context by isolating the sidebar tab or icon region. This changes the agent's next action from a menu-based guess to clicking the transition icon in the right sidebar in the current screenshot. Thus, the memory does not provide the final transition value directly, but supplies the hidden UI route. Through action-grounding, it makes the relevant visual affordance salient enough to produce the correct action.

Although action-grounded memory helps the agent interpret retrieved examples, it does not by itself guarantee recovery once the agent has entered an erroneous state. In an OSWorld VLC task, the agent must stream an HLS video by opening VLC's network-stream dialog and entering a *.m3u8* URL. The vanilla agent follows the correct high-level route and reaches the target dialog but fails at the final low-level interaction by misclicking the *Play* button and incorrectly assuming task completion. Action-grounded memory provides more interpretable UI regions and example actions, but it does not correct the faulty premise that the stream was closed without playback. AGMem resolves this error propagation through recovery-aware verification memory. When the detector flags the trajectory as erroneous, the recovery memory supplies examples of corrective behavior from similar faulty states, allowing the verifier to replace the unstable menu-click trajectory with the task-relevant recovery action *Ctrl+N*, which directly reopens VLC's network-stream dialog, and succeeds the task. This case illustrates that action-grounded memory is useful for selecting and grounding normal actions, but recovery-aware memory is necessary when an earlier miss creates a misleading state premise that ordinary retrieval cannot correct.

6. Related work

This section provides a comprehensive review of works related to AGMem, including agentic memory, GUI agents, and compaction methods for multimodal agents.

Agentic memory. Prior memory-augmented agents retrieve past trajectories, contextual memories, or multimodal GUI memories to guide current decisions (Agashe et al., 2024; Kagaya et al., 2024; Wu et al., 2026). These methods show that memory can improve agent decision-making, but they typically store experience as text, trajectory context, or full-screen visual examples. For GUI tasks, such representations can miss or dilute local cues such as icons, layouts, selected regions, input fields, and error messages. AGMem instead stores action-relevant cropped images and uses recovery memory for erroneous states.

GUI agents. GUI agents control graphical interfaces from screenshots and natural-language instructions. Recent work has improved GUI agents through stronger action models and large-scale computer-use datasets (Wu et al., 2025; Xu et al., 2025; Wang et al., 2026). Memory-augmented GUI agents further retrieve prior trajectories or visual memories to guide planning (Agashe et al., 2024; Wu et al., 2026). These methods are close to AGMem because they reuse previous GUI trajectories, but they usually rely on text summaries, full screenshots, trajectory-level memories, or learned embeddings. AGMem instead stores and retrieves action-grounded visual crops tied to successful actions and recoveries.

Visual focusing. Visual focusing methods reduce irrelevant visual context for GUI agents by selecting instruction-relevant regions or tokens from the current observation (Ouyang et al., 2026). In contrast, AGMem focuses on memories: it extracts crop-view memories from past trajectory transitions and retrieves visual evidence related to the current subtask and state.

7. Conclusion and discussion

We took an analysis-first view of memory in GUI agents and showed that screenshot-based agents can fail in benchmark-specific ways. For example, with GPT-5.4-mini, OSWorld is dominated by *cognitive failure* (82.6%) and *hidden operation blindness* (67.1%), WebForge by *grounding error* (70.0%), and AgentNetBench by *visual state misunderstanding* (45.1%). Adding naïve full-image visual memory does not move these dominant modes uniformly, while it helps where the bottleneck is recognition but amplifies template-style *memory over-following* where the bottleneck is action-space reasoning. AGMem targets exactly these side effects: action-relevant crops keep memory references focused on the relevant UI region, and the recovery-state detector intervenes in residual failures, jointly yielding an 8.9 pp OSWorld accuracy gain over the vanilla agent. We see this analysis-design-reanalysis loop as a useful lens for future work on memory-augmented GUI agents.

Impact statement

The main goal of this work is to advance the field of Machine Learning, in particular the analysis and design of memory for GUI agents. As GUI agents observe whatever appears on a user's screen, deployments should filter sensitive UI content from any persistent memory and retain human oversight to prevent automated misuse. Beyond these standard considerations, we do not identify additional societal consequences that require special discussion.

References

Agashe, S., Han, J., Gan, S., Yang, J., Li, A., and Wang, X. E. Agent S: An open agentic framework that uses computers like a human, 2024. URL https://arxiv.org/abs/2410.08164.

Cheng, K., Sun, Q., Chu, Y., Xu, F., YanTao, L., Zhang, J., and Wu, Z. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Asso-

ciation for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.505.

Gao, D., Ji, L., Bai, Z., Ouyang, M., Li, P., Mao, D., Wu, Q., Zhang, W., Wang, P., Guo, X., Wang, H., Zhou, L., and Shou, M. Z. ASSISTGUI: Task-oriented desktop graphical user interface automation, 2024. URL https://arxiv.org/abs/2312.13108.

Kagaya, T., Yuan, T. J., Lou, Y., Karlekar, J., Pranata, S., Kinose, A., Oguri, K., Wick, F., and You, Y. RAP: Retrieval-augmented planning with contextual memory for multimodal LLM agents, 2024.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, S. W., Wang, L., and Shou, M. Z. ShowUI: One vision-language-action model for GUI visual agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19498–19508, June 2025.

Nguyen, D., Chen, J., Wang, Y., Wu, G., Park, N., Hu, Z., Lyu, H., Wu, J., Aponte, R., Xia, Y., Li, X., Shi, J., Chen, H., Lai, V. D., Xie, Z., Kim, S., Zhang, R., Yu, T., Tanjim, M., Ahmed, N. K., Mathur, P., Yoon, S., Yao, L., Kveton, B., Kil, J., Nguyen, T. H., Bui, T., Zhou, T., Rossi, R. A., and Dernoncourt, F. GUI agents: A survey. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 22522–22538, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1158. URL https://aclanthology.org/2025.findings-acl.1158/.

OpenAI. Introducing GPT-5.4 mini and nano, March 2026. URL https://openai.com/index/introducing-gpt-5-4-mini-and-nano/. Accessed: 2026-05-02.

Ouyang, M., Lin, K. Q., Shou, M. Z., and Ng, H. T. FocusUI: Efficient ui grounding via position-preserving visual token selection. *arXiv preprint arXiv:2601.03928*, 2026.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Reimers, N. and Gurevych, I. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL http://arxiv.org/abs/1908.10084.

Wang, X., Wang, B., Lu, D., Yang, J., Xie, T., Wang, J., Deng, J., Guo, X., Xu, Y., Wu, C. H., Shen, Z., Li, Z., Li, R., Li, X., Chen, J., Zheng, B., Li, P., Lei, F., Cao, R., Fu, Y., Shin, D., Shin, M., Hu, J., Wang, Y., Chen, J., Ye, Y., Zhang, D., Wang, Y., Wang, H., Yang, D., Zhong, V., Charles, Y., Yang, Z., and Yu, T. OpenCUA: Open foundations for computer-use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=6iRZvJiC9Q.

Wu, W., Zhou, K., Yuan, R., Yu, V., Wang, S., Hu, Z., and Huang, B. Auto-scaling continuous memory for GUI agent, 2026. URL https://openreview.net/forum?id=cvAZKWA1HZ.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., and Qiao, Y. OS-ATLAS: Foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=n9PDaFNi8t.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

Xu, Y., Wang, Z., Wang, J., Lu, D., Xie, T., Saha, A., Sahoo, D., Yu, T., and Xiong, C. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=PlihOwfx4r.

Yuan, P., Yin, Y., Cai, Y., and Wei, Z. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *arXiv preprint arXiv:2604.10988*, 2026.