MAGNET: Towards Adaptive GUI Agents with Memory-Driven Knowledge Evolution

Libo Sun[1*], **Jiwen Zhang**[1*], **Siyuan Wang**[3], **Zhongyu Wei**[1,2†]
[1]Fudan University, Shanghai, China
[2]Shanghai Innovation Institute, Shanghai, China
[3]University of Southern California, Los Angeles, USA
{lbsun23,jiwenzhang21}@m.fudan.edu.cn;sw_641@usc.edu; zywei@fudan.edu.cn

Abstract

Mobile GUI agents powered by large foundation models enable autonomous task execution, but frequent updates altering UI appearance and reorganizing workflows cause agents trained on historical data to fail. Despite surface changes, functional semantics and task intents remain fundamentally stable. Building on this insight, we introduce **MAGNET**, a **m**emory-driven **a**daptive a**gent** framework with dual-level memory: **stationary memory** linking diverse visual features to stable functional semantics for robust action grounding and **procedural memory** capturing stable task intents across varying workflows. We propose a **dynamic memory evolution** mechanism continuously refining both memories by prioritizing frequently accessed knowledge. Online benchmark AndroidWorld evaluations show substantial improvements over baselines, while offline benchmarks confirm consistent gains under distribution shifts. These results validate that leveraging stable structures across interface changes improves agent performance and generalization in evolving software environments. Our code and dataset are available at https://github.com/sunlibo2390/MAGNET.

1 Introduction

Graphical user interfaces (GUIs) are the primary medium for operating mobile devices (Shneiderman, 2010), yet they require users to specify every low-level action (Hutchins et al., 1986), limiting efficiency in complex workflows (Sheridan and Parasuraman, 2005). GUI agents powered by multimodal large language models (MLLMs) (Wang et al., 2024b; Liu et al., 2024; Wu et al., 2024b; Qin et al., 2025; Wang et al., 2025b) address this by autonomously executing multi-step tasks from natural language instructions. However, ensuring their robustness in continuously evolving software environments remains a critical challenge (Yang et al., 2025; Hu et al., 2025; Sager et al., 2025).

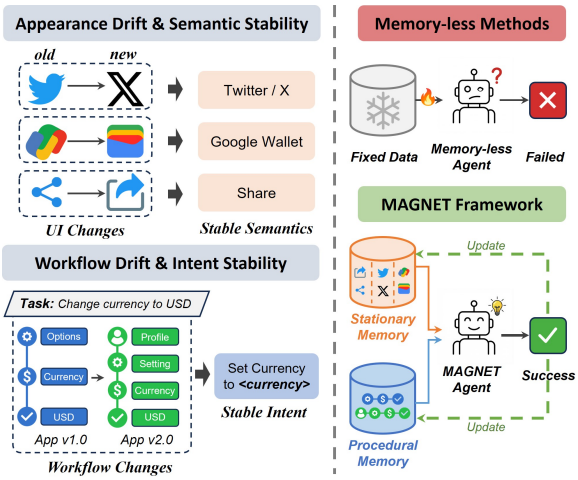


Figure 1: **Challenges and opportunities in evolving mobile interfaces.** (Left) Two types of drifts and their underlying stability: Appearance Drift vs. Semantic Stability, Workflow Drift vs. Intent Stability. (Right) MAGNET exploits these stable aspects to maintain effectiveness, while memory-less agents relying on frozen knowledge struggle to adapt.

Mobile ecosystems are inherently dynamic. We find that frequent version updates in commercial applications introduce two forms of drift, as illustrated in Figure 1 (left). Firstly, *appearance drift* occurs when UI elements are redesigned without altering their functions (e.g., the icon transition from Twitter to X). Secondly, *workflow drift* arises when operation logic is reorganized across app versions (e.g., the task of "changing the currency to USD"). However, existing specialized models (Hong et al., 2024; Zhang et al., 2024b; Chen et al., 2025) are trained on fixed datasets that will be outdated as applications evolve, which limits them to generalize to evolving interface states (Wang et al., 2024c). Although memory-augmented systems (Li et al.; Agashe et al., 2024; Gao et al., 2025) attempt to store reusable knowledge, they mainly focus on

*Equal contribution.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28181–28206

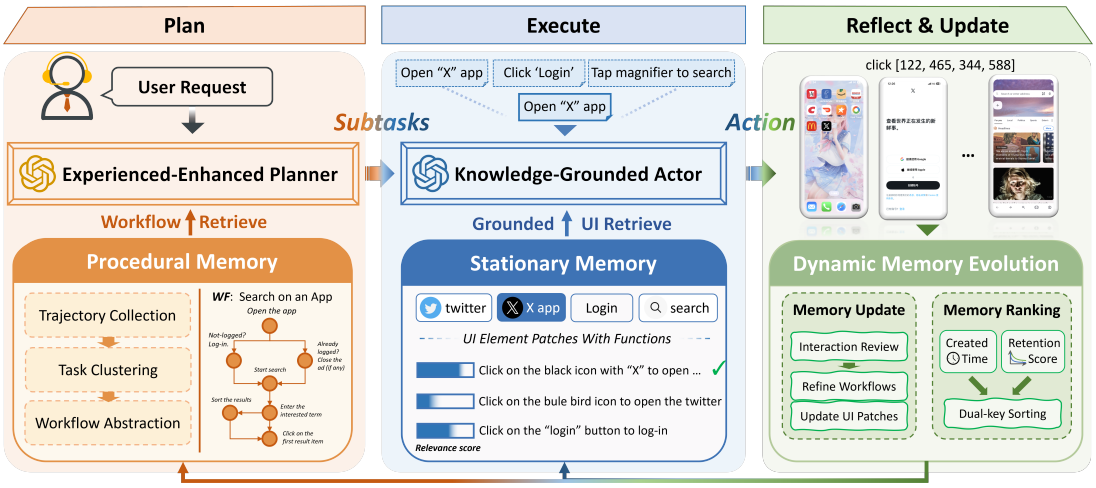


Figure 2: **MAGNET framework.** The planner leverages procedural memory to decompose user requests into subtasks, while the actor grounds each subtask with stationary memory of UI elements.

text-based workflow descriptions that lack multimodal knowledge (Hu et al., 2025), making them vulnerable to visual changes in UI elements.

Despite these drifts, we observe that certain properties remain consistent across application updates, as illustrated in Figure 1 (left). We summarize these observations as two forms of stability. *Semantic stability* refers to the preservation of functional meaning despite visual redesigns, where updated elements support the same actions. *Intent stability* captures the persistence of high-level task goals even when workflows are reorganized (e.g., changing currency to USD via different navigation paths). These stable properties provide opportunities for building GUI agents that can adapt its actions to interface evolution and continue to complete tasks, as illustrated in Figure 1 (right).

Motivated by this, we introduce a **m**emory-driven **a**daptive a**gent** framework, namely **MAGNET**, that maintains two memory modules to leverage these stabilities. Specifically, we use a **procedural memory** to capture intent stability by storing workflow variants for each task objective, enabling adaptation to reorganized logic. We further propose a **stationary memory** that captures semantic stability by linking diverse visual features to stable functional semantics for robust grounding under appearance changes. To support continuous knowledge evolution, we design a **dynamic memory evolution** mechanism that updates memory and prioritizes frequently accessed information. These updates are extracted by using an automated construction pipeline that distills element–function pairs and workflow templates from completed tasks.

We evaluate MAGNET on three representative offline benchmarks (Zhang et al., 2024a; Lu et al., 2024; Chai et al., 2024) to validate the reliability and generalization of the initialized memory. We further assess MAGNET in the online AndroidWorld environment (Rawles et al., 2024), where the memory is continuously updated through interaction to enable adaptation in dynamic settings. Experiments show that MAGNET outperforms zero-shot memory-augmented baselines while remaining competitive with specialized models.

Our primary contributions are:

- We identify appearance and workflow drift as key challenges in evolving applications, and design **MAGNET**, a dual-memory framework leveraging semantic and intent stability to address them.
- We propose a dynamic memory evolution mechanism together with an automated construction pipeline for continual memory optimization.
- We conduct extensive experiments and validate that MAGNET achieves superior adaptability and robust generalization.

2 MAGNET Framework

MAGNET (Figure 2) is a memory-driven agent framework for adaptation in dynamic mobile environments. Following prior planner–actor frameworks (Zheng et al., 2024; Zhang et al., 2024a; Gou et al., 2024), we adopt a multi-agent architecture with a *planner* for task decomposition and an *actor* for grounded execution. Distinct from traditional frameworks, MAGNET augments the planner with a **procedural memory** (§2.1) to retrieve abstract workflows, and the actor with a **sta-**

tionary memory (§2.2) to ground actions using UI element exemplars. Upon receiving a user request, the planner queries procedural memory to retrieve relevant workflows and decomposes the objective into subtasks. For each subtask involving grounding operations, the actor queries stationary memoryto retrieve UI element patches as visual exemplars, enabling reliable action grounding despite interface changes. The actor iteratively executes actions until subtask completion. Finally, we utilize a **dynamic memory evolution** mechanism (§2.3) to incorporate new workflows and UI elements from successful trajectories while adjusting memory priorities through retention-based ranking, enabling autonomous adaptation to environment updates.

2.1 Experience-enhanced Planner with Procedural Memory

To mitigate workflow drift caused by application updates, we equip the planner with a procedural memory that stores abstract workflows distilled from completed tasks. Each workflow consists of a task category name (e.g., "Search and Install an App") and a sequence of high-level steps with categorical placeholders (e.g., [AppName], [SearchQuery]), enabling reuse across different contexts and interface variations.

When receiving a new instruction I_{new}, the planner computes cosine similarity with workflow names in the memory base and retrieves the most similar workflows based on the ranking mechanism (§2.3.2). The retrieved workflows, including their names and step sequences, are concatenated into the planner's context, which comprises the current screenshot, instruction, screen description, and historical actions. By integrating these workflows, the planner generates subtasks that decompose the high-level objective into executable steps.

Memory Construction. To initialize the procedural memory, we propose a three-stage automatic construction pipeline, detailed as follows:

- **Trajectory Collection**: We collect instruction-trajectory pairs $\{(I_j, \pi_j)\}_{j=1}^{M}$ from human demonstrations or curated datasets, where $\pi_j = \langle a_1^j, \ldots, a_{k_j}^j \rangle$ represents action sequences that successfully complete tasks.
- **Task Clustering**: To group tasks with similar workflows, we embed instructions using MiniCPM-Embedding (Hu et al., 2024b) and construct a similarity graph $\mathcal{G}$ by connecting instruction pairs whose cosine similarity exceeds a threshold τ. We then extract maximal cliques $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ (Sun et al., 2019) as task clusters (Algorithm 1), where each cluster c_i groups instructions with similar task patterns.
- **Workflow Abstraction**: For each cluster $c_i \in \mathcal{C}$, associated trajectories are synthesized into abstract workflows by prompting the planner to distill common patterns (Wu et al., 2024a; Yang et al., 2024) (Appendix L.1).

2.2 Knowledge-grounded Actor with Stationary Memory

To address appearance drift caused by UI updates, we ground the actor with a stationary memory that associates visual representations of UI elements with their functional intents. The memory stores pairs $\langle d_i, v_i \rangle$, where d_i describes an element's function (e.g., "click the search icon to start searching") and v_i is the corresponding visual patch. This representation enables the actor to generalize across interface variations by reasoning about functionality rather than exact visual appearance.

When the planner generates subtasks involving UI element localization, the subtask description is used as a query to compute cosine similarity with functional descriptions d_i in the memory base. The most similar entries are retrieved using the ranking mechanism (§2.3.2), and their visual patches v_i are extracted as references. For grounding based on general MLLMs, these patches are concatenated into the actor's context (screenshot, instruction, description, historical actions, subtask), enabling robust identification of target UI elements despite appearance variations and interface changes. For specialized grounding models with fixed input formats, stationary memory is injected via a lightweight adaptation: retrieved patches are template-matched to the current screenshot to locate a center point, around which a bounding box enclosing the nearest k icons is drawn as a visual hint. This introduces stationary memory without modifying the model's input format, demonstrating the adaptability of our stationary memory.

Memory Construction. To initialize the stationary memory, we design an automatic construction pipeline composed of three stages illustrated in Figure 3. Specifically, it involves:

- **Triplet Collection**: Screen-action-screen triplets $\langle o_t, a_t, o_{t+1} \rangle$ are extracted from various sources

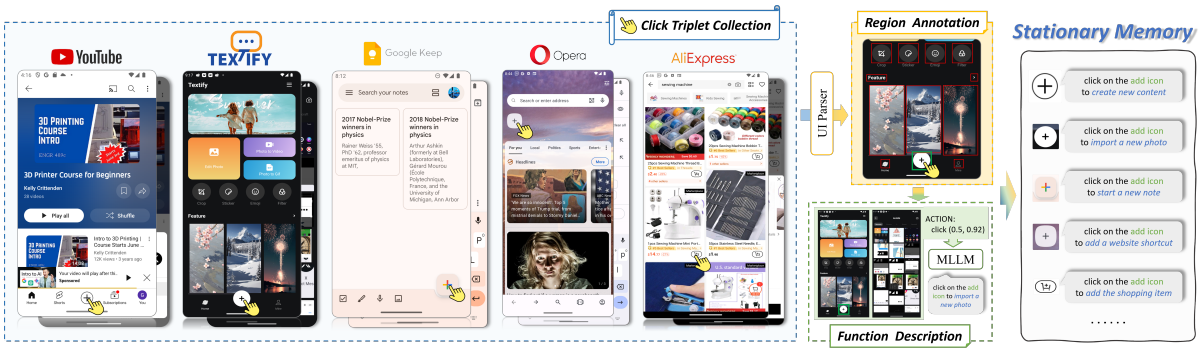


Figure 3: **Construction pipeline of stationary memory.** Each newly generated entry $\langle d_i, v_i \rangle$ is checked against the stationary memory by retrieving similar functional descriptions and corresponding UI element patches. If a duplicate entry is detected, it is discarded to avoid redundancy.

including offline episodic GUI datasets and online interaction trajectories. Each triplet comprises consecutive screenshots captured before and after a click action, enabling inference of both the visual target and functional outcome.

- **Region Annotation**: We employ OmniParserV2 (Yu et al., 2025) to parse screenshots into candidate regions with bounding boxes. The clicked element is identified by selecting the candidate region whose spatial layout best aligns with the click position, and the resulting bounding box crops the element patch v_k.
- **Function Description**: We apply Qwen2.5-VL-32B (Bai et al., 2025) to infer functional semantics d_i in a consistent format (Appendix L.1). This ensures each visual patch v_i is consistently paired with its functional intention d_i.

The UI-40K Dataset. Applying this pipeline to AITZ (Zhang et al., 2024a), GUI-Odyssey (Lu et al., 2024), and Amex (Chai et al., 2024), we construct the **UI-40K** dataset containing 41,009 multimodal entries $\langle o_t, a_t, d_t, o_{t+1} \rangle$ across 20,618 unique functional intents. Beyond serving as stationary memory in MAGNET, UI-40K has broader potential applications including GUI grounding for instruction-tuning and offline reinforcement learning. We confirm UI-40K with high quality through both automatic validation using SOTA MLLMs and manual verification (detailed in Appendix A).

2.3 Dynamic Memory Evolution

To ensure memories remain effective as applications evolve, both memory modules employ content update and adaptive ranking mechanisms.

2.3.1 Memory Content Update

Procedural Memory When successful task trajectories are obtained, the workflow abstraction process (§2.1) extracts structured workflows. Each extracted workflow is stored as an independent memory entry with a creation timestamp τ_i and retrieval count $n_i = 0$, preserving multiple execution paths for similar tasks. During retrieval, the ranking mechanism (§2.3.2) balances usage frequency and recency to prioritize reliable workflows while down-weighting outdated ones, enabling procedural memory to adapt to interface changes.

Stationary Memory When new grounding actions are observed, the system constructs $\langle d_i, v_i \rangle$ pairs via the §2.2 pipeline. For each pair, the system queries for matching functional descriptions. If found, the new visual patch v_i is appended to the existing entry's image list with independent timestamp τ_i and retrieval metadata (n_i, t_i), consolidating visual variants under shared functions where each image independently tracks its usage history. If no match exists, a new entry is created. During retrieval, images are ranked by individual retention scores and updated time. Importantly, unlike procedural memory which requires complete successful trajectories for workflow abstraction, stationary memory only requires individual valid click actions—any click that triggers a meaningful UI state transition yields a usable $\langle d_i, v_i \rangle$ entry, regardless of whether the overall task succeeds. This substantially lowers the cold-start barrier for stationary memory compared to procedural memory.

2.3.2 Memory Ranking for Retrieval

Memory retrieval in dynamic environments must balance contextual relevance with avoiding outdated knowledge as interfaces evolve. Therefore, we introduce a memory ranking mechanism that prioritizes reliable entries by considering two signals: knowledge creation time (timestamp τ_i), which favors entries aligned with current appli-

Methods	AITZ		GUI-Odyssey		Amex	
	SR (%)	Grd. (%)	SR (%)	Grd. (%)	SR (%)	Grd. (%)
Specialized Models (Parameter-tuned)						
Atlas-Pro-7B	**66.64**	62.18	58.82	67.00	**67.45**	67.78
GUI-R1-7B	44.22	57.21	47.70	59.20	49.36	59.42
UI-Venus-Navi-7B	60.27	**70.23**	**65.93**	**74.52**	63.58	**79.67**
InfiGUI-R1-3B	49.49	55.34	55.51	64.98	62.00	69.98
Agentic Frameworks (Training-free / Inference-only)						
COAT (Qwen2.5-VL-32B)	41.09	39.28	48.13	49.38	59.68	67.69
Agent-S† (Qwen2.5-VL-32B)	42.98	42.87	49.21	50.74	58.29	69.85
MAGNET† (Qwen2.5-VL-32B)	43.50	43.78	**50.16**	51.91	**62.84**	71.53
MAGNET† (Gemini-2.5-Pro)	**52.77**	**57.35**	49.74	**57.31**	62.23	**75.54**

Table 1: **Offline evaluation results.** We report Success Rate (SR) and Grounding Accuracy (Grd.). Methods are categorized into: (1) **Specialized Models** (parameter-tuned via SFT or RL) and (2) **Agentic Frameworks** (inference-only or retrieval-augmented based on frozen models). Methods marked with † denote memory-augmented frameworks. For agentic frameworks, **bold** indicates the best performance within the group.

cation versions, and usage history, which reflects empirical utility through retrieval frequency and recency. To quantify usage history, the system maintains a global counter C_{global} and per-entry metadata (last access time t_i and total retrieval count n_i), from which an inactivity gap $g_i = C_{\text{global}} - t_i$ is computed. Inspired by the Ebbinghaus forgetting curve (Ebbinghaus, 1885), we derive a retention score $R_i = \exp(-g_i/n_i)$, where frequently accessed entries decay more slowly.

Based on these signals, we adopt a two-stage retrieval strategy: the first stage uses semantic similarity to select the top-N candidates relevant to the query; the second stage ranks these candidates using the retention score R_i and creation timestamp τ_i to produce the final top-K entries. Complete algorithmic details are provided in Algorithm 2.

3 Experiments

3.1 Experimental Setup

Evaluation Benchmarks We evaluate MAGNET on both offline and online settings. We conduct offline evaluations using AITZ (Zhang et al., 2024a), GUI-Odyssey (Lu et al., 2024), and Amex (Chai et al., 2024). To assess distribution shift generalization, we create custom splits for GUI-Odyssey and Amex, yielding in-distribution (ID), template-shifted (TS), app-shifted (AS), and domain-shifted (DS) subsets (Details in Appendix B). Building on the validated offline results, we further evaluate MAGNET in an online environment AndroidWorld (Rawles et al., 2024). Following the official protocol, all methods share the same action budget and public exploration scripts. Detailed settings are provided in Appendix C.2.

Baselines We evaluate MAGNET against two categories of baselines. For offline evaluation, we compare with end-to-end specialized models, including UI-Venus-Navi-7B (Gu et al., 2025), Atlas-Pro-7B (Wu et al., 2024b), GUI-R1-7B (Luo et al., 2025), and InfiGUI-R1-3B (Liu et al., 2025), as well as the memory-free COAT framework (Zhang et al., 2024a) and the memory-augmented Agent-S (Agashe et al., 2024). We evaluate MAGNET with two representative backbones, Qwen2.5-VL-32B and Gemini-2.5-Pro, covering both open- and closed-source models with different capability levels. Detailed configurations are provided in Appendix C.3.1. For online evaluation, we compare MAGNET with M3A (Rawles et al., 2024), memory-augmented AppAgent (Zhang et al., 2025), and Agent-S (Agashe et al., 2024). All online agents use Qwen2.5-VL-32B as the backbone (Bai et al., 2025). In the online evaluation, memory-augmented agents first perform self-exploration or task execution to initialize their memory, and then leverage the accumulated memory during the evaluation phase. Implementation details are provided in Appendix C.3.2.

Evaluation Metrics For offline evaluation, we use two step-level metrics: Success Rate (SR), which measures the accuracy of predicted actions at each step, and Grounding accuracy (Grd.), which measures the accuracy of coordinate prediction for click actions. Details are in Appendix C.4. For online evaluation, we measure task completion

Method	SR (%)
M3A (Rawles et al., 2024)	32.78
AppAgent (Zhang et al., 2025)	34.43
Agent-S (Agashe et al., 2024)	40.98
MAGNET (Ours)	**42.62**

Table 2: **Comparison with other memory-augmented baselines on the online AndroidWorld environment.**

Variants		GUI-Odyssey		Amex	
Stat.	Proc.	SR	Grd.	SR	Grd.
		48.13	49.38	59.68	67.69
✓		48.62	49.98	60.20	68.57
	✓	49.72	51.28	62.34	70.86
✓	✓	**50.16**	**51.91**	**62.84**	**71.53**

(a) Results with Qwen2.5-VL-32B.

Variants		GUI-Odyssey		Amex	
Stat.	Proc.	SR	Grd.	SR	Grd.
		48.92	55.95	60.35	73.34
✓		49.35	56.55	60.52	73.60
	✓	49.65	57.18	62.11	75.40
✓	✓	**49.74**	**57.31**	**62.23**	**75.54**

(b) Results with Gemini-2.5-Pro.

Table 3: **Ablation study on memory components of MAGNET.** Results are reported on the ID subsets of GUI-Odyssey and Amex, where the same MLLM is used as both planner and actor. *Stat.* and *Proc.* denote stationary and procedural memory, respectively.

rate (SR) using AndroidWorld's official evaluation scripts, which assess whether agents successfully complete end-to-end tasks in the live environment.

3.2 Main Results

Offline Evaluation Table 1 reports results on the in-distribution subsets across three datasets. With Qwen2.5-VL-32B as backbone, MAGNET consistently outperforms COAT and Agent-S across all three datasets, demonstrating the effectiveness of the proposed dual-memory design under a fixed foundation model. While specialized models achieve higher absolute scores, MAGNET with Gemini-2.5-Pro as backbone further improves performance without task-specific training , indicating that the proposed framework can effectively benefit from stronger foundation models.

Online Evaluation Table 2 presents online evaluation results on AndroidWorld. All agents employ self-exploration to initialize their memory using three task execution episodes (detailed in Appendix C.3.2). AppAgent stores UI elements and their functions using element identifiers from XML page structures, while Agent-S emphasizes workflow-level memory but lacks grounded UI representations. In contrast, MAGNET integrates stationary memory with multimodal grounding of UI elements and procedural memory with reusable workflows, enabling the agent to reason jointly over interface structure and action-level experience. As a result, MAGNET achieves a 42.62% task completion rate, outperforming AppAgent (34.43%) by +8.2 points and Agent-S (40.98%) by +1.6 points. This performance gap underscores the importance of decoupling memory representations from page-specific identifiers while jointly modeling UI-level and workflow-level knowledge for robust task completion in dynamic environments.

Backbones		Avg. Performance	
Planner	Actor	ΔSR (%)	ΔGrd. (%)
Homogeneous Configurations			
QwenVL	QwenVL	+2.5	+3.6
Gemini	Gemini	+1.5	+2.3
Heterogeneous Configurations			
GPT-4o	OS-Atlas	+1.5	+1.5
QwenVL	OS-Atlas	+4.2	+3.4
Gemini	OS-Atlas	+1.7	+2.6
Average		**+2.3**	**+2.7**

Table 4: **Effectiveness of MAGNET across architectures.** Average improvements (Δ) with MAGNET across five planner-actor configurations, categorized into homogeneous (same model for planner and actor) and heterogeneous (different models). Results are averaged over AITZ, GUI-Odyssey, and Amex datasets.

3.3 Ablation Study

Memory component effectiveness. Table 3 isolates the effects of each memory component on ID subsets. We evaluate four configurations: baseline, procedural only, stationary only, and both combined. Results show both memories provide consistent gains, with their combination achieving optimal performance. Procedural memory yields larger SR improvements (+2.66% on AMEX with Qwen) than stationary memory (+0.52%), because stationary memory mainly benefits cases in which models lack prior coverage of specific UI icons, a condition that is underrepresented in current static offline benchmarks. However, stationary memory shows clear value in Grd improvements (ranging from +0.26% to +0.88%) and in cases with novel icons (detailed in Appendix C.5).

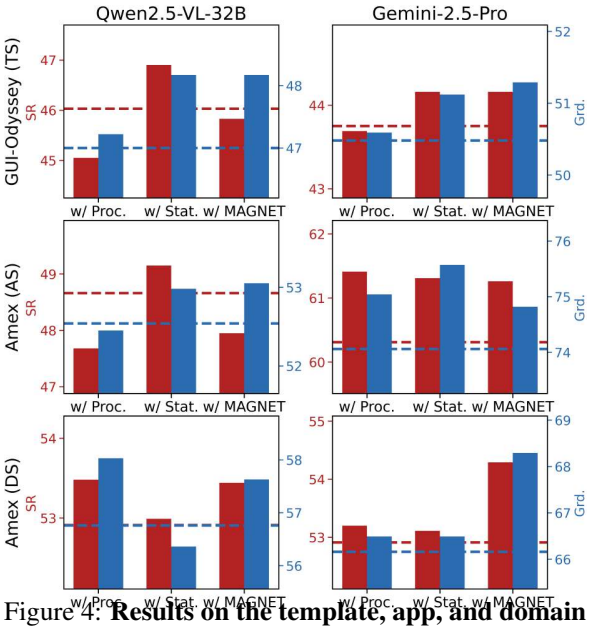


Figure 4: **Results on the template, app, and domain shifted subsets.** The MLLM serves as both the planner and the actor. The dashed lines in the figure indicate the baseline performance.

Robustness across architectures. Table 4 demonstrates the average effectiveness of MAGNET among three datasets (detailed in C.6) across five planner–actor configurations, categorized into *Homogeneous* (same model for planner and actor) and *Heterogeneous* (different models) types. MAGNET consistently improves performance with average gains of +2.3% SR and +2.7% Grd across all configurations. Improvements are particularly pronounced with heterogeneous pairings (e.g., QwenVL + OS-Atlas: +4.2% SR), where the actor is a specialized grounding model that incorporates retrieved stationary memory using the injection strategy described in §2.2, suggesting that memory can effectively compensate for limitations or mismatches between planning and execution modules by providing reusable intermediate knowledge. Even with strong homogeneous setups, memory still offers clear benefits (Gemini + Gemini: +1.5% SR), indicating that its gains are not limited to weaker backbones. Overall, these results demonstrate that the proposed memory design generalizes across architectural choices and supports flexible planner–actor deployment.

3.4 Generalization Discussion

Figure 4 evaluates generalization on template-shifted, app-shifted, and domain-shifted subsets, where agents encounter scenarios beyond source memory coverage. Despite operating under distribution shifts, memory-augmented agents outperform the baseline across most settings, indicating that external memory remains beneficial beyond training coverage. With Qwen2.5-VL-32B, stationary memory leads to modest but consistent SR improvements (+0.1% to +0.9%) across most shift types, while procedural memory exhibits higher variance, ranging from slight degradation under template- and app-shifted settings to moderate gains under domain shifts. In contrast, Gemini-2.5-Pro demonstrates more stable improvements from both memory types (+0.2% to +1.4% SR), indicating a stronger ability to integrate heterogeneous memory signals under distribution shifts. Overall, these trends suggest that different distribution shifts emphasize different forms of memory information: stationary memory being more effective when surface-level structures remain informative, and procedural memory becoming more critical when task objectives and execution patterns change. Moreover, the reduced sensitivity to shift type for Gemini-2.5-Pro suggests that stronger models can more flexibly integrate multiple memory signals. Together, these results provide empirical evidence that our memory design captures complementary aspects of past experience, enabling agents to adapt across diverse generalization scenarios.

Iteration	SR (%)	$\text{Proc}_{\text{Amex}}$ (%)	$\text{Stat}_{\text{Amex}}$ (%)
MAGNET	31.14	100	100
1 iteration	37.70	38	27
2 iterations	39.34	32	24
3 iterations	40.98	26	18

Table 5: **In-the-wild memory evolution on AndroidWorld.** $\text{Proc}_{\text{Amex}}$ and $\text{Stat}_{\text{Amex}}$ denote the percentage of retrieved procedural and stationary memories originating from the initial Amex memory bank.

3.5 Continual Adaption Discussion

To evaluate MAGNET's capacity for continual learning, we warm-start the agent with memories from Amex and then iteratively deploy it on AndroidWorld tasks. After each iteration, the newly acquired interaction trajectories and grounding experiences are incorporated into the memory base and used to initialize the agent for the subsequent iteration. Table 5 shows success rate rising from 31.14% to 40.98% over three iterations. The Proc_{Amex} and Stat_{Amex} columns measure the percentage of retrieved memory entries that still originate from the initial Amex bank (procedural and stationary, respectively). Both proportions decrease

sharply, from 100% to 26% for procedural memory and from 100% to 18% for stationary memory, indicating that Amex-derived knowledge is progressively replaced by AndroidWorld-specific experiences. This trend confirms that dynamic memory update mechanism effectively discards outdated or less relevant memories and adapts to new environments through continuous online learning.

4 Related Works

4.1 MLLM Agents

The emergent capabilities of Multimodal Large Language Models (MLLMs) have motivated their use as central controllers that orchestrate external components (Wu et al., 2023; Li et al., 2023; Yang et al., 2023; Wang et al., 2024e; Shen et al., 2024). These agents augment MLLMs with memory (Fan et al., 2024; Wang et al., 2024f), tool use (Schick et al., 2023; Wang et al., 2025a), complex reasoning (Yang et al., 2023; Wang et al., 2024e) and the ability of iterative learning in real environments (Qian et al., 2024; Xi et al., 2024). Currently, MLLM-driven agents are flourishing across a broad spectrum of applications, ranging from general-purpose tasks such as image generation and editing (Wang et al., 2024e) and video games (Wang et al., 2023; Li et al., 2025), to domain-specific areas including healthcare (Li et al., 2024) and e-commerce (Gong et al., 2025). Building on these advances, our work extends to perceive and operate GUI within dynamic mobile environments.

4.2 GUI Agents

As a specialized instantiation of MLLM agents, GUI agents focus on controlling software interfaces and operating systems. One line of work aims to construct specialist end-to-end agents (Hong et al., 2024; Xu et al., 2024; Zhang et al., 2024b) by fine-tuning small open-source MLLMs on task specific GUI datasets (Deng et al., 2023; Zhang et al., 2024a; Lu et al., 2024). Such methods achieve strong in-domain performance with efficient inference, but their heavy reliance on the high-quality labeled datasets (Chen et al., 2025) severely constrains their generalization ability on unseen applications. Another line of research, including AppAgent (Zhang et al., 2025; Li et al.; Jiang et al., 2025) and MobileAgent (Wang et al., 2024b,a, 2025c; Qin et al., 2025), adopts a planner–actor decomposition, where a planner leverages strong proprietary models (e.g., GPT-4o (OpenAI, 2024), Gemini (Gemini Team, Google, 2025)) to derive operation steps, and an actor executes actions on screens. These frameworks typically assume actors with strong grounding capabilities, such as SeeClick (Cheng et al., 2024), UGround (Gou et al., 2024), and OS-Atlas (Wu et al., 2024b). While adaptable and interpretable, they require careful system design. OS-Copilot (Wu et al., 2024a) further extends this paradigm to OS-level agents by unifying heterogeneous system control, but its text-centric self-directed learning offers limited support for visual interface modeling. In contrast, MAGNET specializes in mobile GUI agents with UI-centric memory and workflow-level planning, enabling robust adaptation to evolving interfaces. The most closely related work, Agent-S (Agashe et al., 2024), also augments a planner–actor framework with procedural memory, but records per-task summaries additively without addressing UI-level changes. MAGNET instead abstracts generalized workflow templates, applies decay-based memory evolution, and explicitly handles both workflow and appearance drift via its dual-memory design.

4.3 Memory-enhanced Agents

LLM agents increasingly incorporate memory to support long-term and complex reasoning. Early work (Park et al., 2023) introduced natural language memory streams for experience recording and reflection, followed by richer memory designs, including summarization and forgetting (Zhong et al., 2024), OS-inspired virtual memory management (Packer et al., 2023), and triplet-based or neurosymbolic representations for precise reasoning (Modarressi et al., 2023; Wang et al., 2024d). Hierarchical and dynamic memory further improve efficiency in long-horizon tasks, exemplified by subgoal summarization in (Hu et al., 2024a) and evolving note-like memory in (Xu et al., 2025). At a higher level, memory has been extended toward reusable reasoning patterns and skills: (Yang et al., 2024) studied buffered reasoning strategies for iterative problem solving, and (Wang et al., 2023) demonstrated expandable skill libraries that support continual learning in interactive environments. More recently, Chain-of-Memory (Gao et al., 2025) proposed modular memory organization for cross-application navigation and knowledge reuse. Inspired by these, we introduce a memory-driven framework MAGNET to evolve procedural and stationary knowledge about dynamic environments.

5 Conclusions

We introduce MAGNET, addressing the appearance drift and workflow drift challenges in evolving applications through dual-level memory: stationary memory enables robust grounding despite interface changes, while procedural memory adapts to workflow evolution. A dynamic update mechanism refines memories by prioritizing frequently accessed knowledge. Comprehensive evaluations on AndroidWorld and offline benchmarks demonstrate effectiveness across diverse scenarios, with ablation studies confirming complementary benefits of both components. This work establishes that exploiting semantic and intent stability beneath surface changes enables practical deployment of adaptive agents in evolving software ecosystems.

Limitations

While MAGNET demonstrates robust adaptation, it has limitations in both memory construction and retrieval. During construction, procedural memory requires successful trajectories, limiting cold-start effectiveness in novel domains, and clustering-based workflow abstraction may struggle with highly diverse task structures; practical remedies include transferring knowledge from known apps, open-ended exploration, human demonstrations, and subtask-level clustering. During retrieval, stationary memory assumes functional stability—when both appearance and function change simultaneously, stale entries may mislead the actor despite partial mitigation from the retention score and self-correction; moreover, task-level retrieval keys may miss step-level workflow coverage, and using step-level context as keys with outcome-based retention signals are promising directions for future work.

Acknowledgments

The research is supported by the AI for Science Program, Shanghai Municipal Commission of Economy and Informatization (Grant No.2025-GZL-RGZN-BTBX-02028). The project's computational resources are partially supported by CFFF platform of Fudan University.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

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

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, and 1 others. 2025. Guicourse: From general vision language model to versatile gui agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21936–21959.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.

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

Hermann Ebbinghaus. 1885. *Über das gedächtnis: untersuchungen zur experimentellen psychologie*. Duncker & Humblot.

Yue Fan, Xiaojian Ma, Rujie Wu, Yuntao Du, Jiaqi Li, Zhi Gao, and Qing Li. 2024. Videoagent: A memory-augmented multimodal agent for video understanding. In *European Conference on Computer Vision*, pages 75–92. Springer.

Xinzge Gao, Chuanrui Hu, Bin Chen, and Teng Li. 2025. Chain-of-memory: Enhancing gui agents for cross-application navigation. *arXiv preprint arXiv:2506.18158*.

Gemini Team, Google. 2025. Gemini-2.5-pro. https://deepmind.google/models/gemini/pro/.

Ming Gong, Xucheng Huang, Chenghan Yang, Xianhan Peng, Haoxin Wang, Yang Liu, and Ling Jiang. 2025. Mindflow: Revolutionizing e-commerce customer support with multimodal llm agents. *arXiv preprint arXiv:2507.05330*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, and 1 others. 2025. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024a. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, and 1 others. 2024b. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, and 1 others. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*.

Edwin L Hutchins, James D Hollan, and Donald A Norman. 1986. Direct manipulation interfaces. In *User centered system design*, pages 87–124. CRC Press.

Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. 2025. Appagentx: Evolving gui agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*.

Binxu Li, Tiankai Yan, Yuanting Pan, Jie Luo, Ruiyang Ji, Jiayuan Ding, Zhe Xu, Shilong Liu, Haoyu Dong, Zihao Lin, and 1 others. 2024. Mmedagent: Learning to use medical tools with multi-modal agent. *arXiv preprint arXiv:2407.02483*.

Chenliang Li, Hehong Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, and 1 others. 2023. Modelscope-agent: Building your customizable agent system with open-source large language models. *arXiv preprint arXiv:2309.00986*.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions, 2024a. *URL https://arxiv. org/abs/2408.11824*, 2.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Weili Guan, Dongmei Jiang, and Liqiang Nie. 2025. Optimus-3: Towards generalist multimodal minecraft agents with scalable task experts. *arXiv preprint arXiv:2506.10357*.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, and 1 others. 2024. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*.

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

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*.

Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2023. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*.

OpenAI. 2024. Gpt-4o. https://openai.com/index/hello-gpt-4o/.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Chen Qian, Jiahao Li, Yufan Dang, Wei Liu, YiFei Wang, Zihao Xie, Weize Chen, Cheng Yang, Yingli Zhang, Zhiyuan Liu, and 1 others. 2024. Iterative experience refinement of software-developing agents. *arXiv preprint arXiv:2405.04219*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Pascal J Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F Grewe, and Thilo Stadelmann. 2025. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions. *arXiv preprint arXiv:2501.16150*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei

Huang. 2024. Small llms are weak tool learners: A multi-llm agent. *arXiv preprint arXiv:2401.07324*.

Thomas B Sheridan and Raja Parasuraman. 2005. Human-automation interaction. *Reviews of human factors and ergonomics*, 1(1):89–129.

Ben Shneiderman. 2010. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India.

Yuan Sun, Xiaodong Li, and Andreas Ernst. 2019. Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems. *IEEE transactions on pattern analysis and machine intelligence*, 43(5):1746–1760.

Chenyu Wang, Weixin Luo, Sixun Dong, Xiaohua Xuan, Zhengxin Li, Lin Ma, and Shenghua Gao. 2025a. Mllm-tool: A multimodal large language model for tool agent learning. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6678–6687. IEEE.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025b. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024b. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024c. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383.

Siyuan Wang, Zhongyu Wei, Yejin Choi, and Xiang Ren. 2024d. Symbolic working memory enhances language models for complex rule application. *arXiv preprint arXiv:2408.13654*.

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

Zhenyu Wang, Aoxue Li, Zhenguo Li, and Xihui Liu. 2024e. Genartist: Multimodal llm as an agent for unified image generation and editing. *Advances in Neural Information Processing Systems*, 37:128374–128395.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, and 1 others. 2024f. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024a. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024b. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, and 1 others. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

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

Jingqi Yang, Zhilong Song, Jiawei Chen, Mingli Song, Sheng Zhou, Xiaogang Ouyang, Chun Chen, Can Wang, and 1 others. 2025. Gui-robust: A comprehensive dataset for testing gui agent robustness in real-world anomalies. *arXiv preprint arXiv:2506.14477*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Wenwen Yu, Zhibo Yang, Jianqiang Wan, Sibo Song, Jun Tang, Wenqing Cheng, Yuliang Liu, and Xiang Bai. 2025. Omniparser v2: Structured-points-of-thought for unified visual text parsing and its generality to

multimodal large language models. *arXiv preprint arXiv:2502.16161*.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024a. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Jiwen Zhang, Yaqi Yu, Minghui Liao, Wentao Li, Jihao Wu, and Zhongyu Wei. 2024b. Ui-hawk: Unleashing the screen stream understanding for gui agents.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.

A Validation of UI-40K Annotation Quality

To ensure the reliability of UI-40K annotations, we employed a hybrid validation approach combining automated SOTA MLLM prediction and manual verification.

Automatic Validation. We sampled 1,000 instances and queried two state-of-the-art MLLMs (GPT-4o and Gemini-2.5-Pro) to independently generate bounding boxes based on the tuple ⟨Current Screen, Click Description⟩. We calculated the IoU between the outputs of these two models to filter for consensus. Instances where the inter-model IoU exceeded 0.75 were treated as high-confidence pseudo-ground truth (787 instances), while the remaining 213 instances were assigned to a manual verification set. For the 787 high-confidence instances, we computed the IoU against the OmniParserV2 annotations:

Metric	Value
Mean IoU	0.55
Median IoU	0.63
IoU ≥ 0.5	68.5%
IoU ≥ 0.7	42.3%

Table 6: Automatic validation metrics for UI-40K annotations.

Analysis of Discrepancies. Although the mean IoU is 0.55, manual inspection reveals that discrepancies primarily arise from valid variations in annotation granularity (e.g., the parser marks a larger clickable container while the model marks the specific icon inside), rather than functional errors.

Manual Verification. For the remaining 213 cases where models disagreed, manual review confirmed that the annotated regions were semantically correct regarding the clickable element. This validation confirms the high reliability of UI-40K for training and evaluation purposes.

B Datasets

B.1 Dataset Splitting

AITZ The AITZ (Zhang et al., 2024a) dataset comes pre-split. We directly use the training set as the **source** data for memory construction and the test set as the **in-distribution (ID)** validation set.

GUI-Odyssey The GUI-Odyssey dataset constructs instructions by first generating tasks from templates and then rephrasing them into natural language. To create ID and TS splits, we proceed in two steps:

- **Template-level split.** We randomly sample 20% of all templates, and treat the corresponding data as **template-shifted (TS)**.
- **Instruction-level split.** For the rest portion, we further divide instructions: 75% are used as the **source** data for memory construction, while the remaining 25% form the **in-distribution (ID)** test set.

This design ensures that all TS instructions originate from unseen templates, while ID instructions come from seen templates. Given the large scale of GUI-Odyssey (over 110k images), we downsample the final evaluation sets to 309 ID samples and 311 TS samples, yielding a representative size comparable to other datasets.

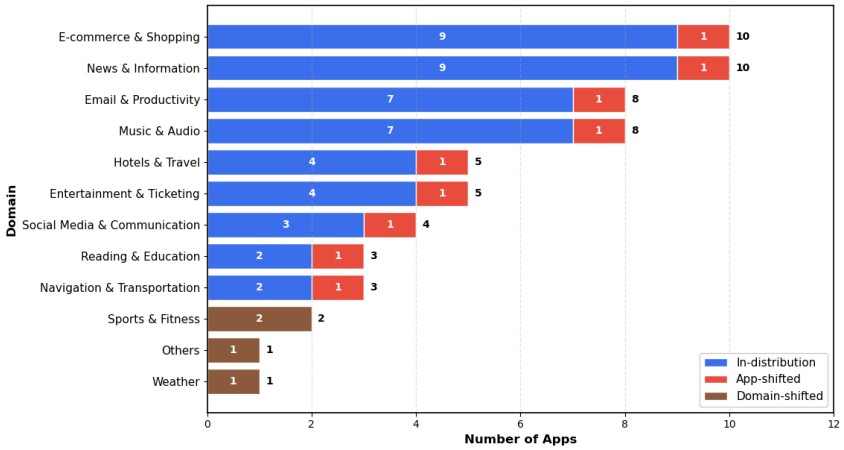


Figure 5: Domain/App distribution of the Amex dataset.

Amex The Amex dataset instructions also follow certain templates, similar to GUI-Odyssey, although this is not explicitly stated in the original paper or dataset. To address this, we manually classified the instructions, identifying 58 apps and grouping them into 12 domains.

The Amex splitting strategy is defined as follows:

1. **Domain-level split:** We select three domains (Sports & Fitness, Others, and Weather) as unseen domains, and the episodes from these domains form the **domain-shifted (DS)** subset.
2. **App-level split:** From each of the remaining domains, we randomly choose one app to form the unseen-apps set, while the rest constitute the seen-apps set. Although unseen-apps are disjoint from seen-apps, they still belong to domains already represented in seen-apps; therefore, their data is referred to as the **app-shifted (AS)** subset.
3. **Instruction-level split:** For each app in the seen-apps set, we randomly partition instructions into **source** and **in-distribution (ID)** subsets using a 4:1 ratio.

This yields four subsets in total: source, ID, AS, and DS. The distribution is summarized in Figure 5.

B.2 Statistics

Table 7 summarizes the statistics of all datasets and their subsets.

Dataset	Subset	Episodes	Screens
AITZ	source	1,998	13,919
	in-domain	506	4,724
Odyssey	source	4,635	71,803
	in-domain	309	4,920
	template-shifted	311	4,386
Amex	source	1,794	22,639
	in-domain	448	5,563
	app-shifted	487	8,035
	domain-shifted	259	2,472

Table 7: Dataset Statistics

C Experiment Details

C.1 Hyperparameter Settings

Implementation Details. Table 8 lists the key hyperparameters used in MAGNET. For memory retrieval, stationary memory uses a cosine similarity threshold of 0.7 with top-20 candidates, while procedural memory uses a threshold of 0.4 with top-5 candidates. Task clustering for procedural memory construction uses a cosine similarity threshold τ=0.5 (MiniCPM-Embedding). For the specialized grounding model adaptation, template matching uses a threshold of 0.3. OmniParser is configured with a box detection threshold of 0.05, IoU threshold of 0.1, and OCR threshold of 0.9.

Hyperparameter	Value
Stationary retrieval threshold	0.7
Procedural retrieval threshold	0.4
Stationary top-N/top-K	20
Procedural top-N/top-K	5
Task clustering threshold τ	0.5
Template matching threshold	0.3
OmniParser box threshold	0.05
OmniParser IoU threshold	0.1

Table 8: Key hyperparameters of MAGNET.

C.2 Online Agent Configurations

Choice of Easy Split. AndroidWorld (Rawles et al., 2024) provides three difficulty levels: Easy (61 tasks), Medium (36 tasks), and Hard (19 tasks). We evaluate on the Easy split for the following reasons:

1. **Iterative Learning Requirements**: Our continual learning approach requires multiple training iterations (three memory-free passes) to evolve memories from online experiences. The Easy split provides sufficient task diversity while maintaining experimental tractability for this iterative process.
2. **Research Focus**: Our primary contribution is the memory-driven adaptation framework demonstrated through comprehensive offline benchmarks (AITZ, GUI-Odyssey, Amex with multiple distribution shifts). The online evaluation serves to validate real-world applicability rather than benchmark performance optimization.

C.3 Baselines Details

C.3.1 Offline Baselines

For offline evaluation, we compare MAGNET with both specialized models and agentic frameworks.

Specialized models including UI-Venus-Navi-7B (Gu et al., 2025), Atlas-Pro-7B (Wu et al.,

2024b), GUI-R1-7B (Luo et al., 2025), and InfiGUI-R1-3B (Liu et al., 2025) are implemented following their respective papers, including model architectures, input formats, and inference settings. For fair comparison across datasets, we unify the action space while preserving each model's original design assumptions.

COAT (Zhang et al., 2024a) is evaluated using the Gemini-2.5-Pro backbone, consistent with its original implementation. COAT does not employ explicit memory mechanisms and serves as a representative memory-free agentic baseline.

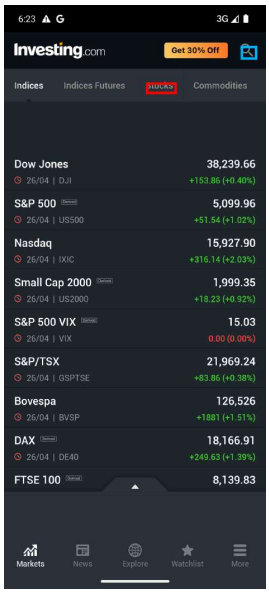
Agent-S (Agashe et al., 2024) is adapted to offline datasets following its original framework design. Memory construction is performed using demonstrations from the corresponding offline dataset, consistent with the experience-augmented retrieval mechanism described in the Agent-S paper. Since offline benchmarks do not support interactive self-exploration, the Episodic Memory component—which relies on online trajectory collection—is omitted. Additionally, the web knowledge module is removed to ensure a fair comparison in a closed-world offline setting. All other components, including hierarchical planning and narrative memory retrieval, are retained without modification.

Unless otherwise specified, all agentic frameworks are evaluated under the same action space.

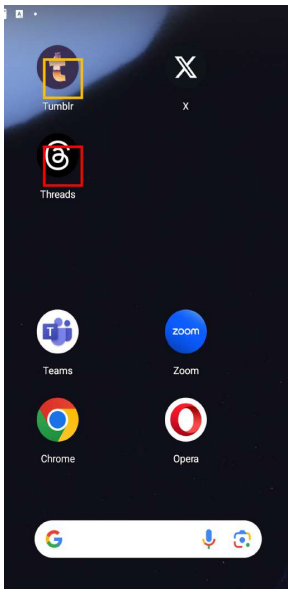
C.3.2 Online Baselines

For completeness, we document how each online baseline prepares its memory before evaluation on the AndroidWorld Easy split:

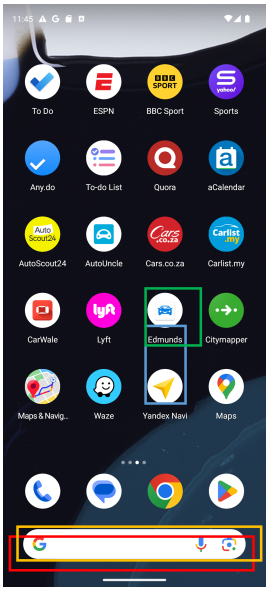
- **AppAgent** (Zhang et al., 2025): For each of the 24 applications contained in the Easy set, AppAgent is paired with a manual exploration task. It runs the task once to produce structured documentation for that application and then reuses the resulting documentation during inference on the downstream tasks.
- **Agent-S** (Agashe et al., 2024): The agent first executes one round of generic tasks together with the official exploration tasks to build its internal memory. This single-run memory is then reused for evaluation without additional updates.
- **MAGNET**: We run three memory-free passes across the 61 tasks to harvest trajectories, filter them through the metabolism rule, and then conduct the reported evaluation. For the continual-learning study, the initial memories are constructed from the Amex dataset and updated after each full pass over the Easy tasks.



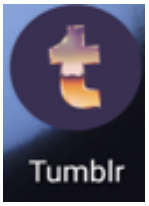
(a) Search for the stock price trends of Google



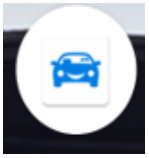
(b) Use Tumblr to share the meeting link



(c) Verify the price of the leading product for Subaru with Edmunds



(d) Icon retrieved from stationary memory in 6b



(e) Icon retrieved by MAGNET in 6c

Figure 6: Case studies on memory components. Red boxes mark the baseline, cyan denote procedural memory, orange denote stationary memory, and green denote MAGNET. Procedural (a) and stationary (b) memory both improve over the baseline, while MAGNET (c) outperforms them by combining both. (d) and (e) show the icon retrieval process for (b) and (c), respectively.

C.4 Evaluation Metrics

Success Rate (SR) Following (Wu et al., 2024b), we count a step as successful if the predicted action type and all parameters exactly match the ground truth. The success rate is the average accuracy under this criterion. Matching rules differ depending on the output format (see below).

Grounding (Grd.) We adopt the criterion from (Wu et al., 2024b), counting a prediction as correct if the relative distance between the predicted and ground-truth positions is less than 14%.

C.5 Qualitative Case Studies

Figure 6 presents three case studies showing how different components improve task execution.

(a) **Searching for Google stock.** The baseline clicks a misleading "stack" tab that only shows a stock leaderboard. With procedural memory, the agent recalls the correct workflow `Search for Stock Price Trends`, which includes the step `Tap the search icon and enter the company name`, thereby completing the task effectively.

(b) **Launching Tumblr.** The baseline incorrectly identifies Tumblr as the Threads app. Stationary memory retrieves the exact Tumblr icon (6d), allowing the agent to select the correct app.

(c) **Opening Edmunds for Subaru's leading product price verification.** Although the Subaru's leading model was already identified, the baseline ignores this and reverts to searching in the browser. Even with stationary memory, it still clicks the search bar. With procedural memory, the agent infers that the next step is to open Edmunds, but incorrectly selects Yandex Navi due to the absence of visual grounding. By combining both memories, MAGNET plans the correct action and grounds it visually (6e), successfully opening Edmunds.

C.6 Detailed Architecture Ablation Results

Table 9 presents the complete per-dataset results for the architecture ablation study summarized in Table 4 of the main paper.

D Efficiency Analysis

We analyze the per-step computational overhead introduced by MAGNET on the Amex benchmark using Qwen2.5-VL-32B on 2×A6000 GPUs.

Backbones		MAG.	AITZ		GUI-Odyssey		Amex	
Planner	Actor		SR (%)	Grd. (%)	SR (%)	Grd. (%)	SR (%)	Grd. (%)
Standard End-to-End Baselines								
Qwen2.5-VL-32B		-	40.62	38.71	47.54	48.86	59.02	66.93
Gemini-2.5-Pro		-	50.14	53.21	48.27	55.18	59.76	72.68
Paired Backbone Ablation: Base vs. + MAGNET (Ours)								
GPT-4o	Atlas-Base-7B	×	36.92	36.54	33.84	35.30	42.17	50.16
		✓	**38.38**	**38.55**	**36.20**	**37.64**	**42.80**	**50.32**
GPT-4o	Gemini-2.5-Pro	×	42.48	41.94	38.22	39.15	46.21	54.08
		✓	**43.71**	**42.87**	**40.54**	**41.28**	**46.71**	**54.82**
GPT-4o	Qwen2.5-VL-32B	×	34.37	34.29	32.04	33.10	39.48	47.26
		✓	**36.15**	**35.82**	**34.29**	**34.64**	**41.72**	**49.28**
Qwen2.5-VL-32B	Atlas-Base-7B	×	43.64	41.53	49.93	51.58	53.92	70.59
		✓	**45.73**	**46.51**	**52.07**	**54.91**	**62.37**	**72.57**
Qwen2.5-VL-32B	Qwen2.5-VL-32B	×	41.09	39.28	48.13	49.38	59.68	67.69
		✓	**43.50**	**43.78**	**50.16**	**51.91**	**62.84**	**71.53**
Gemini-2.5-Pro	Atlas-Base-7B	×	45.31	48.48	44.54	52.10	56.31	69.42
		✓	**47.44**	**53.03**	**45.40**	**53.67**	**58.32**	**71.04**
Gemini-2.5-Pro	Gemini-2.5-Pro	×	50.87	53.88	48.92	55.95	60.35	73.34
		✓	**52.77**	**57.35**	**49.74**	**57.31**	**62.23**	**75.54**

Table 9: Backbone-agnostic effectiveness of MAGNET. By grouping backbones in pairs, this layout highlights the consistent performance gain across different LLM/VLM combinations. **Bold** indicates the better performance within each backbone pair.

Retrieval Latency. The RAG retrieval is implemented as dense vector search over pre-built indices and introduces only 0.31 s/step in total (procedural: 0.19 s/step; stationary: 0.12 s/step), which is negligible relative to MLLM inference time.

Token Overhead. Retrieved memory entries add an average of 499.83 tokens/step to the prompt (procedural: 422.56 tokens/step; stationary: 77.27 tokens/step). This additional context is the primary driver of inference overhead.

Pipeline Overhead. MAGNET's per-step pipeline comprises three stages: Observe (screen parsing), Think (procedural memory retrieval and planning), and Predict (stationary memory retrieval and grounding). Memory retrieval affects only the latter two stages. Table 10 reports the full breakdown.

Stage	w/o MAGNET	w/ MAGNET	Increase
Think	13.19 s	15.46 s	+2.27 s
Predict	6.95 s	9.39 s	+2.44 s
Full Pipeline	29.92 s	34.28 s	+4.35 s

Table 10: Per-step wall-clock time breakdown (seconds/step).

The overall per-step wall-clock increase is only +14.55%, as the Observe stage is unaffected by memory retrieval and dilutes the inference overhead. We consider this a modest and acceptable trade-off given the consistent performance gains.

E Retrieval Robustness Analysis

To empirically characterize how the agent behaves under mismatched procedural memory retrieval, we sample 200 steps from the offline evaluation—each with a retrieved workflow—and conduct a structured analysis using Qwen2.5-VL-32B as the judge.

Definitions. A retrieved workflow is judged a *match* if it covers the current step, i.e., the workflow actions explicitly contain or clearly imply the ground-truth action. Among mismatched workflows, we further distinguish *relevant* (task-related but not step-covering) from *irrelevant* (unrelated to the task entirely). We additionally assess *reasoning correctness*: whether the planner's generated action description refers to the same action as the ground truth.

Results. Among the 200 sampled steps, 123 have matched retrievals and 77 have mismatched retrievals. For matched retrievals, the planner reasons correctly in 78.0% of cases (96/123). Table 11 breaks down the mismatched cases by relevance and reasoning correctness.

	R-Correct	R-Wrong
W-Relevant	41	30
W-Irrelevant	4	2

Table 11: Breakdown of mismatched retrieval cases (77/200 steps). R-Correct/R-Wrong: reasoning correct/wrong; W-Relevant/W-Irrelevant: retrieved workflow is task-relevant/irrelevant.

The vast majority of mismatched cases remain task-relevant (71/77), and 57.7% of these (41/71) still produce correct reasoning. Truly irrelevant retrievals account for only 6 out of 200 steps, confirming that MAGNET's two-stage retrieval mechanism—similarity thresholding followed by retention-based ranking—effectively ensures task-level relevance even when exact workflow coverage is unavailable.

F OmniParser Annotation Quality Validation

A key question for stationary memory reliability is whether OmniParser-predicted patches introduce meaningful noise compared to GT patches during memory construction. To assess this, we compare the downstream Grounding accuracy (Grd.) on GUI-Odyssey using Qwen2.5-VL-32B when stationary memory is built from OmniParser-predicted patches versus GT-annotated patches. The difference is only 0.62%, confirming that OmniParser-predicted patches are of sufficient quality for practical memory construction, and that any residual gap reflects the backbone model's visual reasoning capacity rather than annotation error.

Alternative Annotation Strategies. OmniParser is not the only annotation strategy available; its use is a practical choice for offline datasets lacking XML metadata. The framework supports multiple strategies depending on the deployment context:

- **Online deployment (XML metadata):** XML accessibility metadata is directly available alongside screenshots, enabling GT-quality bounding boxes without any external parser.
- **Offline datasets (OmniParser):** For datasets lacking XML metadata, OmniParserV2 serves as a practical fallback, as validated by the 0.62% Grd. difference above.

- **High-precision scenarios (human annotation):** For domain-specific deployments where annotation quality is critical, human-annotated bounding boxes can initialize stationary memory with maximum precision.

These strategies can be combined depending on data availability and quality requirements.

G Stationary Memory Correction Rate Analysis

To directly validate the role of stationary memory, we identify click steps where the planner's action description was correct but the actor failed to ground the target element. We then inject ground-truth (GT) visual knowledge directly into the actor and re-run the Predict stage, measuring how many previously failed steps are corrected. We test two injection forms: **Crop** (a cropped UI element patch, MAGNET's default stationary memory format, providing visual knowledge without a localization hint) and **Redbox** (the GT element region annotated on the full screenshot, providing both visual knowledge and a localization hint).

Model	Setting	Failed	Corrected	Rate
7B	Crop	70	38	54.3%
7B	Redbox	70	47	67.1%
32B	Crop	33	20	60.6%
32B	Redbox	33	20	60.6%

Table 12: Correction rates when GT visual knowledge is injected into the actor.

On 32B, both injection forms achieve the same correction rate (60.6%), demonstrating that injecting visual knowledge alone corrects over 60% of previously failed grounding steps. This confirms that *knowledge sufficiency*—not localization accuracy—is the operative bottleneck, and that MAGNET's crop-based stationary memory directly addresses this gap. On 7B, Redbox substantially outperforms Crop (67.1% vs. 54.3%), indicating that the smaller model benefits more from explicit localization hints alongside visual knowledge.

H Generalization with Smaller Backbones

We evaluate MAGNET with Qwen2.5-VL-7B as the backbone to assess generalization beyond the 32B model used in the main experiments.

Offline Results. Table 13 reports results on the Amex in-distribution subset.

	7B		32B	
Setting	SR (%)	Grd. (%)	SR (%)	Grd. (%)
w/o MAGNET	23.49	21.92	59.68	67.69
w/ MAGNET	27.14	23.97	62.84	71.53

Table 13: Offline results on Amex (in-distribution) for 7B and 32B backbones.

MAGNET yields consistent gains on both model sizes, confirming that the framework generalizes beyond the 32B backbone.

Online Memory Evolution. Table 14 reports iterative memory evolution results on AndroidWorld. Both models show consistent evolution behavior—old memory ratios drop substantially after each iteration—but 7B stops improving after one iteration while 32B continues to improve over three iterations. We attribute this to 7B's low base success rate (22.95%): fewer completed tasks per iteration means fewer valid trajectories, limiting new memory accumulation. MAGNET's iterative gains are thus partially gated by the backbone's execution quality.

	7B			32B		
Iteration	SR	Proc	Stat	SR	Proc	Stat
0 iter	22.95	100	100	31.14	100	100
1 iter	26.22	45	42	37.70	38	27
2 iter	26.22	39	38	39.34	32	24
3 iter	—	—	—	40.98	26	18

Table 14: Online memory evolution on AndroidWorld for 7B and 32B. SR (%), Proc and Stat denote the percentage of retrieved procedural and stationary memories from the initial memory bank.

I Effect of Trajectory Filtering

We ablate the effect of trajectory quality filtering on online memory construction. Table 15 compares three settings on AndroidWorld: the M3A baseline (no memory), MAGNET with unfiltered trajectories (all collected trajectories used for memory construction regardless of task success), and MAGNET with filtered trajectories (only successful trajectories used for procedural memory construction).

The gap between filtered and unfiltered memory (+4.91%) confirms that trajectory-level quality screening is essential for constructing reliable procedural memory. Erroneous trajectories introduce noisy or repetitive action sequences that degrade abstracted workflow quality, whereas filtering en-

Method	SR (%)
M3A (baseline)	32.78
MAGNET (unfiltered memory)	34.43
MAGNET (filtered memory)	39.34

Table 15: Effect of trajectory filtering on AndroidWorld task success rate.

sures that only correct execution patterns are consolidated into reusable workflows.

J The Use of Large Language Models (LLMs)

Large Language Models (LLMs) were used to aid in the writing and polishing of the manuscript. Specifically, we used an LLM to assist in refining the language, improving readability, and ensuring clarity in various sections of the paper. The model helped with tasks such as sentence rephrasing, grammar checking, and enhancing the overall flow of the text.

It is important to note that the LLM was not involved in the ideation, research methodology, or experimental design. All research concepts, ideas, and analyses were developed and conducted by the authors. The contributions of the LLM were solely focused on improving the linguistic quality of the paper, with no involvement in the scientific content or data analysis.

The authors take full responsibility for the content of the manuscript, including any text generated or polished by the LLM. We have ensured that the LLM-generated text adheres to ethical guidelines and does not contribute to plagiarism or scientific misconduct.

K Algorithm Details

Algorithm 1 Instruction Clustering via Maximal Cliques

Require: Instructions $\{I_1, I_2, \ldots, I_n\}$, similarity threshold τ
Ensure: Instruction clusters $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$
Compute embeddings $e_i = \text{ENCODE}(I_i)$ for all instructions
Initialize similarity graph $\mathcal{G} = (V, \emptyset)$ where $V = \{I_1, I_2, \ldots, I_n\}$
for each pair (I_i, I_j) where $i \neq j$ **do**
  if $\text{COSINE}(e_i, e_j) > \tau$ **then**
    Add edge (I_i, I_j) to $\mathcal{G}$
  end if
end for
$\mathcal{C} = \text{FIND\_MAXIMAL\_CLIQUES}(\mathcal{G})$
return $\mathcal{C}$

Algorithm 2 Memory Retrieval and Update Mechanism

Require: Query x_q (natural language description of next action)
Require: Memory Bank $\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}$ where each entry $m_i = \langle k_i, v_i, \tau_i, t_i, n_i \rangle$ contains:
k_i: retrieval key (workflow description or UI function)
v_i: stored value (workflow template or UI element patch)
τ_i: creation timestamp (when this entry was added to memory)
t_i: last access counter (value of C_{global} when last retrieved)
n_i: total retrieval count (number of times this entry has been retrieved)
Require: Global retrieval counter C_{global} (tracks total number of retrieval events)
Require: Parameters: N (number of candidates to filter), K (number of memories to retrieve)
Ensure: Retrieved memories $\mathcal{M}_{\text{ret}}$, Updated $\mathcal{M}$ and C_{global}
// Stage 1: Semantic Filtering
Compute semantic similarity $s_i = \text{COSINE}(E(x_q), E(k_i))$ for all $m_i \in \mathcal{M}$
    where $E(\cdot)$ encodes text into embedding vectors
$\mathcal{C}_N = \text{TOP\_N}(\mathcal{M}, \text{key} = s_i)$ {Select top-N most semantically similar entries}
// Stage 2: Dual-Key Ranking
for each $m_i \in \mathcal{C}_N$ **do**
    $g_i \leftarrow C_{\text{global}} - t_i$ {Retrieval gap: events since last access}
    $R_i \leftarrow \exp(-g_i/n_i)$ {retention score: usage-adaptive decay}
end for
Sort $\mathcal{C}_N$ by (R_i, τ_i) in descending order
    {Primary key: retention score R_i; Secondary key: creation time τ_i}
$\mathcal{M}_{\text{ret}} = \text{TOP\_K}(\mathcal{C}_N)$ {Select top-K entries after sorting}
// Stage 3: Update Retrieved Memories
$C_{\text{global}} \leftarrow C_{\text{global}} + 1$ {Increment global counter}
for each $m_i \in \mathcal{M}_{\text{ret}}$ **do**
    $t_i \leftarrow C_{\text{global}}$ {Record current retrieval event}
    $n_i \leftarrow n_i + 1$ {Increment retrieval count}
end for
if New workflow or UI element $\langle k^*, v^* \rangle$ extracted from successful task **then**
    $\tau^* \leftarrow \text{CURRENT\_TIMESTAMP}()$ {Record creation time}
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{\langle k^*, v^*, \tau^*, C_{\text{global}}, 1 \rangle\}$
        {Initialize: creation time τ^*, last access C_{global}, count 1}
end if
return $\mathcal{M}_{\text{ret}}$, $\mathcal{M}$, C_{global}

L Prompts

L.1 Memory Construction

Extract workflow for procedural memory.

You are an expert in analyzing and abstracting user behavior on mobile devices. Given a list of mobile tasks, each described by a natural language instruction and its corresponding action sequence, extract commonly used workflows shared across multiple tasks.

Instructions:

1. Identify repetitive subsequences of actions that appear in two or more tasks.
2. Each extracted workflow should be a useful and reusable subroutine, not too specific to any one task.
3. Do NOT output overlapping or highly similar workflows; each should be distinct and meaningful.
4. Each workflow must contain at least 3 steps.
5. Represent variable elements (such as user input, contact names, app names) using descriptive placeholders (e.g., `[SearchQuery]`, `[AppName]`, `[ContactName]`).
6. Focus on semantic repetition, not just literal string match.

Workflow Examples:

```
[
  {
    "title": "Install: Search and Install an App",
    "plan": [
      "Open the App Store or App Market or Play Store.",
      "Tap the search bar.",
      "Type [AppName] into the input field.",
      "Locate the correct app in the result list.",
      "Tap the [InstallButton] to download and install."
    ]
  },
  ...
]
```

Input Mobile Tasks: {tasks}

Output: Extract all valid reusable workflows from the input above, following the rules. Output strictly in JSON format as shown in the examples, without extra explanation or commentary.

Restate click action description for stationary memory.

Format the given context into: `click [ui element] to [purpose]`

Requirements:

1. `ui element`: concise name or description of the clicked UI element.
2. `purpose`: clear explanation of the click's goal.

Examples:

1. click the Chrome icon to search for information online
2. click the More Info button to view quiz-related details
3. click the second search result to read about hiking trails

Input Context:

Action Description: {action description}
Action Result: {action result}

Output the formatted action description directly without any additional text.

Action Inference Prompt

You are an assistant for inferring user actions based on mobile UI screenshots.
You will be given two screenshots:

1. **Before Action**: A screenshot of the UI before the user performed the action.
2. **After Action**: A screenshot showing the result of the action.

Your task is to:

General Action

- Provide a concise **action description** of what the user did. (e.g., "type 'blender recommendation' into the search bar.", "stop and set the task as completed/impossible."). Start with verb.
- Provide a concise **action result** describing the effect of the action.

Output format (strictly JSON, no extra explanation or formatting):

```
{
  "action_desc": "<description of the user action>",
  "action_result": "<description of the outcome of the action>"
}
```

Swipe Action

- Provide a concise **action description** of what the user did (e.g., "scroll up on the home feed").
- Provide a concise **action result** describing the effect of the action (e.g., "By doing this...").

Output format (strictly JSON, no extra explanation or formatting):

```
{
  "action_desc": "<a concise description of the user action. For swipe
      actions, only use direction terms: 'up', 'down', 'left', or 'right
      '>",
  "action_result": "<description of the outcome>"
}
```

Click Action

- Provide a concise **action description** of what the user clicked. Start with a verb. (e.g., "click on the settings icon in the top right")
- Provide a concise **action result** describing what happened after the click (e.g., "By doing this...").

Output format (strictly JSON, no extra explanation or formatting):

```
{
  "action_desc": "<description of the user action. (e.g., \"click on the
      settings icon in the top right\")>",
  "action_result": "<description of the outcome>"
}
```

Action Type {action_type}: {action_meaning}

L.2 GUI Agents

Observe: Generate screen description.

You are a smart and helpful visual assistant that is well-trained to describe smartphone screenshots.
1. You are provided with a screenshot of the current mobile phone.
2. You are required to describe this screen's main content and its functionality. The output must be less than five sentences.
3. You are required to keep the description as concise and brief as possible.

Input:
CURRENT SCREENSHOT: {screenshot}
YOUR RESPONSE:

Plan: Generate next action description.

You are a smart and helpful visual assistant that is well-trained to manipulate mobile phones. Your task is to navigate and take action on the current screen step-by-step to complete the user request.
1. You are provided with a screenshot of the current mobile phone, together with the textual screen description.
2. You are provided with your history actions to decide on your next action. You can backtrack to revise the previous actions when necessary.
3. You are provided with some relevant workflows for reference.
4. You are required to analyze the task status and detail a reasonable future action plan to accomplish the user request.

Analysis Guidelines:
1. You should check whether the historical actions have accomplished the user request.
2. You should check the apps, icons, and buttons that are visible on the current screen and might pertain to the user request.
3. You should combine the above information and describe your future action plan. If the given workflows are relevant and helpful, you may refer to them for guidance.
4. The "Future Action Plan" must consist of a sequence of concrete, low-level action steps.

Output Format You are required to respond in a JSON format, consisting of 3 distinct parts with the following keys and corresponding content:

```
[
  {
    "Thought": "<Analyze the logic behind your next single-step action and
       your future action plan to fulfill the user request.>",
    "Future Action Plan": [
      {"type": "<ACTION_TYPE>", "description": "<Natural language description
         of the action>"},
      ...
    ]
  }
]
```

CURRENT SCREENSHOT: {screenshot}
SCREEN DESCRIPTION: {screen description}
HISTORY ACTIONS: {history actions}
USER REQUEST: {user request}
RELEVANT WORKFLOWS: {relevant workflows}
YOUR RESPONSE:

Predict: Generate detailed action arguments.

You are a smart and helpful visual assistant that is well-trained to manipulate mobile phones. Your task is to navigate on the current screen to complete the user request.

1. You are provided with a screenshot of the current mobile phone.
2. You are provided with a brief summarization of the screen content.
3. You are provided with history actions trying to accomplish the user request, together with the previous action result that indicates how the current screenshot is obtained.
4. You are provided with a **Relevant UI Element** that visually represents the most relevant UI component for the next action.
5. You are required to decide on the next single-step valid action to be conducted on the current screen so as to fulfill the user request.

Valid Actions on the Screen: {action_space}

Output Format: You must choose one of the valid APIs provided above and respond in the corresponding API call format. Your response should be strictly structured in JSON format, consisting of the following keys and corresponding content:

```
{
  "THINK": "<Analyze the logic behind your next single-step action.>",
  "NEXT": "<Describe the next single-step action in words, e.g. 'click on the
      .... located at ...'>",
  "ACTION": "<Specify the precise API function name without arguments, e.g.,
      click_element. Leave it empty if none applies or task is complete.>",
  "ARGS": "<Specify arguments in dictionary format, e.g., {'bbox': [0,1,2,3]}.
       Leave empty if not needed or task complete.>",
  "REASON": "<Explain your reasoning for choosing this action.>"
}
```

Output Examples:

```
{
  "THINK": "...",
  "NEXT": "...",
  "ACTION": "click_element",
  "ARGS": {"bbox": [100, 345, 219, 826]}
}
```

```
{
  "THINK": "...",
  "NEXT": "...",
  "ACTION": "scroll",
  "ARGS": {"direction": "down"}
}
```

```
{
  "THINK": "...",
  "NEXT": "...",
  "ACTION": "press_home",
  "ARGS": {}
}
```

Inputs:

- CURRENT SCREENSHOT: {screenshot}
- SCREEN CONTENT: {screen_desc}
- HISTORY ACTIONS: {history_actions}
- PREV ACTION RESULT: {prev_action_result}
- USER REQUEST: {user_request}
- RELEVANT UI ELEMENT: {relevant_ui}
- YOUR RESPONSE:

Prompt for OS-Atlas.

You are a foundational action model capable of automating tasks across various digital environments, including desktop systems (Windows, macOS, Linux), mobile platforms (Android, iOS), and web browsers. You interact with digital devices in a human-like manner: by reading screenshots, analyzing them, and taking appropriate actions.

Expertise:

1. **Grounding:** Given a screenshot and description, assist users in locating elements. Infer best-fit elements when implicit.
2. **Executable Language Grounding:** Given a screenshot and task instruction, determine executable actions to complete the task.

You are now operating in **Executable Language Grounding mode**. Your goal is to suggest executable actions that best fit user needs.

1. Basic Actions (available across all platforms):

```
[
    {
      "name": "CLICK",
      "purpose": "Click at the specified position.",
      "format": "CLICK <point>[[x-axis, y-axis]]</point>",
      "example": "CLICK <point>[[101, 872]]</point>"
    },
    {
      "name": "TYPE",
      "purpose": "Enter specified text at the designated location.",
      "format": "TYPE [input text]",
      "example": "TYPE [Shanghai shopping mall]"
    },
    {
      "name": "SCROLL",
      "purpose": "Scroll in the specified direction.",
      "format": "SCROLL [direction (UP/DOWN/LEFT/RIGHT)]",
      "example": "SCROLL [UP]"
    }
]
```

2. Custom Actions (environment-specific): Custom actions extend functionality to handle unseen or user-defined tasks.

{action_space}

Instruction: In most cases, task instructions are high-level and abstract. Carefully read the instruction and action history, then reason to determine the most appropriate next action.

Output Requirements: You must strictly generate two sections:

- `Thoughts`: concise reasoning (max 20 words).
- `Actions`: the actual next one-step action.

Inputs:

- Screenshot: {screenshot}
- Task: {user_request}
- History: {history_actions}

Output Format:

```
thoughts: "<a concise description of reasoning>"
actions: "<only next one step action usage>"
```

Table 16: Prompt template for OS-Atlas model.

Action space for OS-Atlas.

```
[
    {
      "name": "PRESS_ENTER",
      "purpose": "Press an enter button to confirm the input, or submit the
          input, or start a new line of text.",
      "format": "PRESS_ENTER",
      "example": "PRESS_ENTER"
    },
    {
      "name": "PRESS_HOME",
      "purpose": "Press a home button to navigate to the home page.",
      "format": "PRESS_HOME",
      "example": "PRESS_HOME"
    },
    {
      "name": "PRESS_BACK",
      "purpose": "Press a back button to navigate to the previous screen.",
      "format": "PRESS_BACK",
      "example": "PRESS_BACK"
    },
    {
      "name": "STOP",
      "purpose": "Stop and set the state of the task.",
      "format": "STOP [task_status (SUCCESS/FAILURE)]",
      "example": "STOP [SUCCESS]"
    }
]
```