Agent-SAMA: State-Aware Mobile Assistant

Linqiang Guo[1], Wei Liu[1*], Yi Wen Heng[1], Tse-Hsun (Peter) Chen[1†], Yang Wang[2]

[1]Software PErformance, Analysis, and Reliability (SPEAR) lab, Concordia University, Montreal, Canada
[2]Concordia University, Canada
g_linqia@live.concordia.ca, w_liu201@encs.concordia.ca, he_yiwen@encs.concordia.ca, peterc@encs.concordia.ca, yang.wang@concordia.ca

Abstract

Mobile Graphical User Interface (GUI) agents aim to autonomously complete tasks within or across apps based on user instructions. While recent Multimodal Large Language Models (MLLMs) enable these agents to interpret UI screens and perform actions, existing agents remain fundamentally reactive. They reason over the current UI screen but lack a structured representation of the app navigation flow, limiting GUI agents' ability to understand execution context, detect unexpected execution results, and recover from errors. We introduce Agent-SAMA, a state-aware multi-agent framework that models app execution as a Finite State Machine (FSM), treating UI screens as states and user actions as transitions. Agent-SAMA implements four specialized agents that collaboratively construct and use FSMs in real time to guide task planning, execution verification, and recovery. We evaluate Agent-SAMA on two types of benchmarks: cross-app (Mobile-Eval-E, SPA-Bench) and mostly single-app (AndroidWorld). On Mobile-Eval-E, Agent-SAMA achieves an 84.0% success rate and a 71.9% recovery rate. On SPA-Bench, it reaches an 80.0% success rate with a 66.7% recovery rate. Compared to prior methods, Agent-SAMA improves task success by up to 12% and recovery success by 13.8%. On AndroidWorld, Agent-SAMA achieves a 63.7% success rate, outperforming the baselines. Our results demonstrate that structured state modeling enhances robustness and can serve as a lightweight, model-agnostic memory layer for future GUI agents.

Code — https://doi.org/10.5281/zenodo.15430187

Introduction

Mobile apps have become an important part of people's daily life, providing access to online shopping, communication, social media, and more. To automate and support these interactions, recent research has introduced graphical user interfaces (GUI) agents that are powered by Multimodal Large Language Models (MLLMs) (Lee et al. 2024; Wen et al. 2024a; Wang et al. 2024b; Zhang et al. 2023a, 2024; Li et al. 2024; Wang et al. 2024a, 2025b). Given a user instruction, such as "*using Chrome to search for the date of the next Winter Olympics opening ceremony, and then setting a reminder for that date in Calendar*", GUI agents can automatically complete these tasks without human intervention. The agents can analyze and reason app UI screens, identify actionable UI elements (e.g., buttons, input fields), and perform actions such as tapping, typing, or scrolling (Zhang et al. 2023a; Wang et al. 2024b,a, 2025b). By chaining these interactions together, GUI agents can complete complex multi-step tasks across diverse app environments.

However, existing GUI agents remain fundamentally reactive: they primarily reason the next action based on the current UI screen, without maintaining a structured representation of app behaviors—*like tourists who navigate a city one street at a time, knowing where they have been but lack a coherent view of the route or how different locations are connected*. This lack of structural understanding of an app's navigation logic limits GUI agents' ability to interpret execution context, detect unexpected execution outcomes, and recover from errors. To illustrate, Figure 1 shows how a user interacts with the Walmart app by performing a sequence of actions (e.g., tapping), where each action may lead to a transition to a new state (i.e., a new UI screen). The user types a query to search for products, selects a product from the results to view its details on *product page*, and taps the button to add the product to the shopping cart. Then, the user can tap "X" to return to the previously visited *product page*.

In app usages, every step depends and builds on the prior one to follow a task-oriented logical flow according to the app's requirements and design. Understanding such interaction flows is essential. As illustrated in the right side of Figure 1, app usage is not a series of isolated actions. Instead, the transitions between UI screens are triggered by specific user actions, and follow a pre-defined and structured changes in UI states. Having structured representations of the transition provides several benefits for GUI agents: it enables the agent to track its progress within a task, anticipate the outcomes of actions, verify whether the resulting state aligns with expectations, and assist in finding a recovery state among the action sequences.

In this paper, we propose **Agent-SAMA, a novel mobile GUI agent framework that leverages finite state machines (FSMs) to represent structured and state-aware task executions**. An FSM is a formal computational model that represents systems as a finite number of states with

*Corresponding author.
†Principal investigator.

state transitions triggered by specific actions. Given that mobile apps are inherently stateful systems, where each UI screen corresponds to a distinct state and user actions trigger transitions, FSMs provide a natural abstraction for app usages (Prajapati 2012; Wang 2004; Wagner et al. 2006; Espada et al. 2015; Shahbaz et al. 2022).

Specifically, Agent-SAMA consists of four phases: 1) **Planning**: given a task, the **Planner Agent** decomposes the task into a sequence of subtasks. To improve the planning process, Agent-SAMA generates several candidate plans and applies LLM-as-judges to select the best among all (Gu et al. 2025). 2) **Execution**: After planning, Agent-SAMA guides subtask executions through the collaboration of three agents. The **Screen Parser** extracts structured information from the current UI screen, generating a description of the screen and UI element locations (e.g., buttons). The output is passed to the **State Agent**, which incrementally constructs an FSM in real-time to model the navigation flow by mapping each screen's natural language description to a distinct state and associating user actions with transitions. State Agent adds pre- and post-conditions to every state and transition, allowing for better reasoning and verification. Finally, the **Actor Agent** selects and performs the appropriate action (e.g., tapping at a specific button) based on the output from the State Agent. 3) **Error Recovery and Verification**: If an unexpected error happens, the **Reflection Agent** compares the FSM-predicted transition (including post-conditions) against the current screen and uses the FSM to identify a previously verified stable state. It then generates a recovery plan to return to that point for retry. 4) **Knowledge Retention**: After completing a task, the **Mentor Agent** stores the constructed FSMs and executed actions in Agent-SAMA's long-term memory storage to create useful knowledge that can guide future task executions.

Experimental results show that Agent-SAMA achieves significant improvement over the baseline Mobile-Agent-E+Evo (Wang et al. 2025b) across two challenging cross-app benchmarks (Wang et al. 2025b; Chen et al. 2025) and also achieves a 63.7% success rate on AndroidWorld (Rawles et al. 2025). On Mobile-Eval-E (Wang et al. 2025b), Agent-SAMA improved the Success Rate by over 12% compared to the baseline, completing more tasks successfully. It also encounters less error during task execution (32 vs. 49) and yet achieves a 4.53% higher Recovery Success rate. The results highlight Agent-SAMA's ability to detect and recover from execution errors during runtime. We observe similar trends on SPA-Bench (Chen et al. 2025), where Agent-SAMA outperforms the baseline by 5% in Success Rate and a 13.81% improvement in Recovery Success rate with less errors (63 vs. 70). On AndroidWorld, Agent-SAMA outperforms several baselines such as Mobile-Eval-E+Evo (Wang et al. 2025b) and AgentS2 (Agashe et al. 2025), indicating its strong performance through stable planning and execution.

Our contributions are as follows:

- **Introducing FSM Modeling to Mobile GUI Agents.** We are the first to incorporate finite state machine (FSM) modeling into mobile GUI agents. By treating each app as a state machine—with UI screens as states and user actions as transitions—we enable structured, state-aware reasoning that addresses key limitations of reactive execution. Source code and evaluation data are available at Zenodo (Author s).
- **A State-Aware Agent Framework with Persistent Memory and Recovery.** We implement FSM modeling in Agent-SAMA, a mobile GUI agent that constructs per-app state graphs during execution, tracks visited states, infers state preconditions and postconditions, and uses this structured representation for proactive planning and robust error recovery.
- **An LLM-Based Judge for Plan Selection and Replanning.** To enhance planning and decision-making, we introduce a LLM–based judge that evaluates multiple candidate plans and selects the most reliable one based on context and execution history. This mechanism further improves the agent's ability to adapt and recover in complex mobile environments.
- **Improved Task Performance Across Benchmarks.** Agent-SAMA achieves significant performance gains over the baseline, Mobile-Agent-E + Evo, on both Mobile-Eval-E and SPA-Bench benchmarks, including up to 12% improvement in Success Rate, over 6.5% in Action Accuracy, over 7% in Satisfaction Score, and up to 13.81% in Error Recovery Rate.

Related Work

Mobile GUI Agents

Recent advances in MLLM have significantly improved GUI agents' ability in executing complex tasks. Some agents adopt a single-agent architecture (e.g., Mobile-agent (Wang et al. 2024b), AppAgent (Zhang et al. 2023a), AppAgent v2 (Li et al. 2024)), while others follow a multi-agent paradigm (e.g., MobileExperts (Zhang et al. 2024), Mobile-Agent-v2 (Wang et al. 2024a) and Mobile-Agent-E (Wang et al. 2025b)) that distributes perception, planning, and execution across specialized agents. To improve task success rates, most existing mobile agents (Lee et al. 2024; Wen et al. 2024a; Wang et al. 2024b; Zhang et al. 2023a, 2024; Li et al. 2024; Wang et al. 2024a, 2025b) rely on prompt engineering (Liu et al. 2025a), enriching LLM input with UI information (e.g., screenshots, UI trees, OCR), short action histories, and chain-of-thought (CoT) reasoning. A few studies also explore training-based approaches, including supervised fine-tuning (Ding 2024) and reinforcement learning (Liu et al. 2024b).

One similar study, GUI-Xplore (Sun et al. 2025), builds GUI transition graphs from offline videos to support static reasoning tasks such as screen recall. While both GUI-Xplore and Agent-SAMA use graph-based representations of app navigation, GUI-Xplore focuses on evaluating agents' ability to reason about app structure without real-time interaction or execution. In contrast, Agent-SAMA constructs FSMs online during task execution. Using the constructed FSM, Agent-SAMA integrates state tracking and reasoning, transition validation, and recovery planning into the agent's decision-making process. Our FSM-based architecture is model-agnostic and compatible with existing

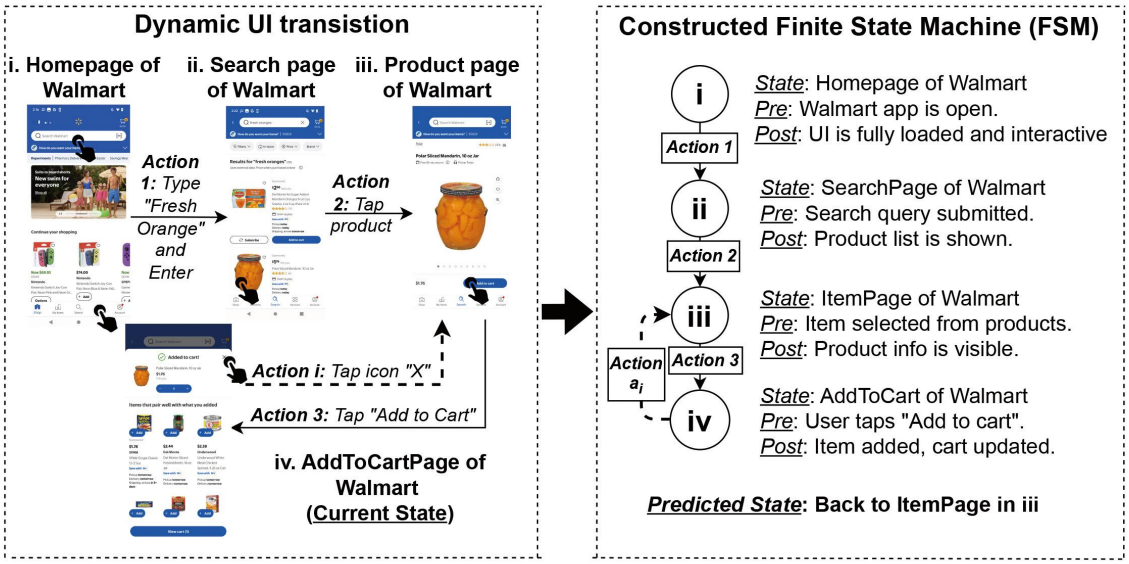

Figure 1: An example of how Agent-SAMA represents real-time UI interactions as a Finite State Machine (FSM). The left side shows the dynamic UI transitions of the Walmart App along with the user action (e.g., typing and tapping) that leads to a new UI screen. The right side shows the corresponding FSM, where each UI screen is represented as a state (a natural language description of the screen generated by MLLM) with its MLLM-generated pre- and post-condition. The user action defines the transition between the states. Given the current state and the entire FSM, Agent-SAMA also predicts the possible next state.

agent designs, offering a potential avenue for improving future mobile GUI agents.

Evaluation Benchmarks for GUI Agents

Numerous benchmarks have been proposed to evaluate mobile GUI agents, including MobileEnv (Zhang et al. 2023b), AutoDroid (Wen et al. 2024b), MobileAgentBench (Wang et al. 2024c), AndroidArena (Xing et al. 2024), and AndroidWorld (Rawles et al. 2025). However, many of these fall short in realism, long-horizon task modeling, or cross-app execution. For example, MobileEnv and AndroidWorld rely on emulator-based environments. MobileAgentBench mainly targets single-app interactions, while AndroidArena supports some cross-app tasks but still lacks long-horizon or highly complex workflows. In this work, We adopt Mobile-Eval-E (Wang et al. 2025b) and SPA-BENCH (Chen et al. 2025), which provide 364 actions and 262 actions (in English version apps), respectively. These benchmarks support realistic, multi-step, and cross-application tasks, enabling rigorous evaluation of our proposed Agent-SAMA. They also facilitate direct comparison with the baseline, Mobile-Agent-E (Wang et al. 2025b), particularly for complex behavior modeling.

Agent-SAMA

Figure 2 provides an overview of Agent-SAMA. Agent-SAMA employs various agents during the four phases of task execution: 1) ***Planning***, which generates a plan to fulfill a given task. 2) ***Execution***, which models app UI transition and user actions (e.g., tap or swipe) as a Finite State Machine (FSM), providing a structured representation of the task execution. 3) ***Verification and Error Recovery***, where the Reflection Agent leverages the constructed FSM to recover from any encountered error. 4) ***Knowledge Retention***, where the Mentor agent extracts reusable knowledge (i.e., guidance cues, action sequences, FSMs) from the execution history, FSM, and error history for guiding future tasks.

Planning Phase

The planning phase is handled by a **Planner agent**, which takes in the user's high-level task instruction, u, and generates a structured plan, π, that decomposes the task into sequential subtasks. The Planner first expands u to infer user intent. Then, the Planner generates $\pi = [(g_1, r_1), (g_2, r_2), \ldots, (g_k, r_k)]$ with additional reusable knowledge K, which generated by prior task execution. Each $g_i \in \mathcal{G}$ represents a subtask (e.g., "open profile settings"), and each $r_i \in \mathcal{R}$ is a rationale that explains how the subtask contributes to achieving the overall task u. As shown in recent work (Gu et al. 2025; Mahmud et al. 2025; Seo, Lee, and Bu 2025), relying on a single-path plan may lead to suboptimal results due to lack of consideration on other diverse strategies. Hence, the Planner agent employs a two-step planning pipeline. In the first step, the Planner generates multiple candidate plans, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ where $n = 5$. In the second step, we leverage LLM-as-judges and ask the LLM agent to evaluate the candidate plans against an evaluation rubric provided by the benchmark that considers factors such as goal relevance, execution efficiency, robustness, and clarity (Gu et al. 2025). The

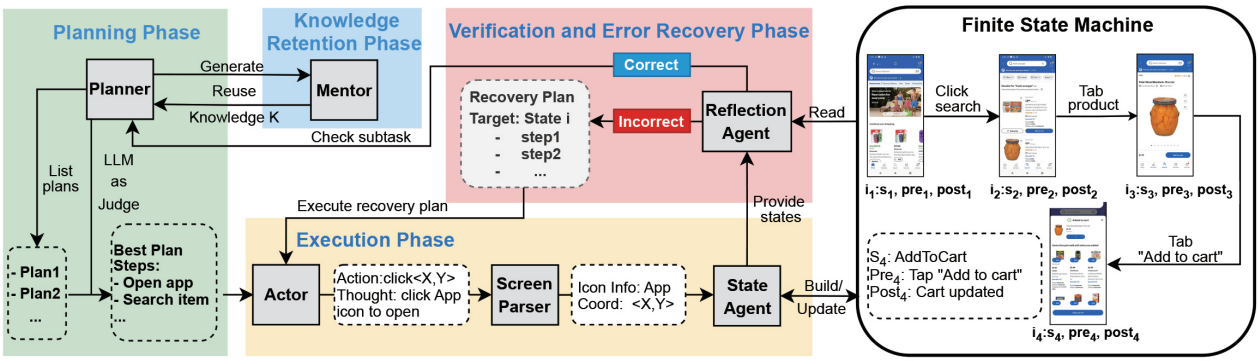

Figure 2: An overview of Agent-SAMA. The Planner, Actor, Screen Parser, StateAgent, and Reflection Agent are involved in the main agent loop for each task, while Mentor contributes to updating long-term reusable knowledge across tasks. Decision-making at each step is disentangled into high-level planning by the Planner and low-level actions by the Actor. The State Agent builds FSMs dynamically, and the Reflection Agent verifies the outcome of each action, tracks progress, and provides error recovery.

agent then selects the best candidate plan to guide the task execution.

Execution Phase

The execution phase consists of i) **Screen Parser**, ii) **State Agent**, and iii) **Actor Agent**. The **Screen Parser** serves as the "*eyes*" of Agent-SAMA. For every screen that reflects the current state of the device, the Screen Parser performs OCR-based text detection, icon localization, and screen segmentation to identify clickable elements and invokes the MLLM to generate descriptive captions for visual elements, helping agents interpret ambiguous or icon-based UI components. The Screen Parser extracts screen perception information on current screenshot s_i before executing any actions and gets $p_i = [(e_1, c_1), (e_2, c_2), \ldots, (e_k, c_k)]$, where each (e_j, c_j) represents a pair of a GUI element (e.g., screen texts or icon descriptions) and its coordinate. The perception result p_i and the current screenshot s_i are provided to the **State Agent** as input.

The **State Agent constructs and incrementally expands app interactions in real-time as a finite state machine (FSM) (Wagner et al. 2006)**—an abstract computational model that represents the system as a set of discrete states and transitions triggered by UI actions. This structured representation of the app's navigation flow enables the agent to maintain execution context, detect abnormal transitions, and reason about both past and future states. As illustrated in Figure 1, the FSM represents the UI transitions during task execution, where each state corresponds to a distinct app screen (i.e., **a state is represented as the natural language description of the screen generated by the State Agent**), and edges represent transitions triggered by user actions such as taps and swipes, maintaining a structured representation of the app's navigation flow.

The FSM is defined as $\mathcal{M} = (S, A, T, s_0, G)$, where S is the set of discovered UI states, A is the action space, T is the transition function, s_0 is the initial state, and G denotes the goal state for current subgoals in each iteration during the task execution. For every perception result p_i, screenshot i_i and subtask g_i, **each generated state node s_i contains**:

1. a natural language *state description* d_{i}, summarizing the current screen;
2. a *description of next-state prediction* $d_{\text{i+1}}$, estimating the next expected screen based on the subtask; and
3. a pre- and post-condition: pre^{i+1} and post^{i}.

To reduce **state explosion and avoid redundant states**, Agent-SAMA assigns each state a state beacon—a concise semantic label derived from its state description d_{i}. Before creating a beacon for the new FSM node, **State Agent** will check the state description and decide whether this node can be matched against previously seen beacons. Existing nodes are reused if a match is found, which reduces state explosion caused by redundant states. Each transition is defined as $T(s_i, a_i) = (s_{i+1}, \text{pre}^{i+1}, \text{post}^{i})$, where s_i is the UI state at step i, a_i is the corresponding action, s_{i+1} is the resulting state, pre^{i+1} is the precondition of s_{i+1}, indicating what must hold before the next state begins, and post^{i} is the post-condition of s_i, representing what should hold after achieving the state in step i. The **Actor Agent** selects the corresponding tool (e.g., the mobile API to perform tapping on a specific coordinate) to perform the action a_i based on the current subtask g_i and the perception information p_i to advance to state s_{i+1}. Note that for cross-app tasks, we create and maintain separate FSMs for each app.

Verification and Error Recovery Phase

One major benefit of using an FSM to provide a structured representation of app navigation and task execution is its effectiveness in detecting and recovering from execution errors. By explicitly modeling discrete UI states, transitions triggered by user actions, and expected pre- and post-conditions, the FSM allows the system to detect deviations from the intended task flow. Hence, in this phase, the **Reflection Agent** determines whether an action, a_i, was successful by examining the result from the FSM (i.e., the prior

state s_i, the predicted next-state description d_i, and pre_{i-1} and post_i), the perception result on the new screen p_{i+1}, the screenshots (i and $i+1$), the current subtask g_j, and the knowledge base K . The agent then outputs one of the three outcomes: Success, NoChange, or Fail (unexpected transition or violation of the predicted state).

In the cases of NoChange or Fail, the Reflection agent also generates the reason for the failures. Then, the agent activates the recovery mode and uses the FSM to identify a previously verified and stable state, s_j to resume the current subtask g_i. Then, the reflection agent constructs a recovery plan, $\pi_r = [a_{i-1}, a_{i-2} \ldots]$, to revert to a recovery point. When executing the recovery plan, the Reflection Agent verifies each step using the same process as in the normal execution flow. If Agent-SAMA accumulates repeated failures while executing the recovery plan (n=2), the recovery process is terminated and Agent-SAMA re-invokes the **Planner Agent** to reassess the current state and generate a revised plan. This fallback mechanism prevents the agent from getting stuck in repetitive recovery loops and ensures that high-level reasoning can reorient execution when low-level strategies fail (Wang et al. 2025b; Chang and Geng 2025).

Knowledge Retention Phase

Agent-SAMA generates various types of process data during task execution, including: 1) the action history, 2) error details, and 3) transition history. As shown in prior studies (Wang et al. 2025b; Liu et al. 2025b), data from prior task execution can be useful to guide future tasks. Hence, at the end of every task, **Mentor Agent** analyzes this process data to extract reusable knowledge K, which includes: (i) *action sequences*, which are sequential actions annotated with preconditions; and (ii) *guidance cues*, which are natural language tips distilled from prior executions to help with future task reasoning. In addition, Mentor also stores the FSM in K to better support further tasks if they share similar functionality or design (e.g., shopping events can be similar between Amazon and Walmart). When a new task begins, Agent-SAMA checks long-term memory for K, and selectively loads it as external context (i.e., part of the prompt) to improve planning, execution efficiency, and robustness.

Evaluation

Benchmarks and Evaluation Settings

We evaluate Agent-SAMA on two sets of benchmarks: (1) fully cross-app tasks: Mobile-Eval-E (Wang et al. 2025b) and SPA-Bench (Chen et al. 2025), and (2) mostly single-app tasks: AndroidWorld (Rawles et al. 2025).

Running cross-app benchmarks on a physical device with human evaluation. Cross-app tasks, such as *e.g., using Chrome to search for the date of the next Winter Olympics opening ceremony, and then setting a reminder for that date in Calendar*, are more realistic and inherently more complex to plan, execute, and recover from (Liu et al. 2025b). Thus, Mobile-Eval-E and SPA-Bench provide a more rigorous test of GUI agents' capabilities (Table 1). To run these two benchmarks, we deploy Agent-SAMA on a physical Google Pixel 7 Pro, controlled via Android Debug Bridge (ADB).

Metric	Mobile-Eval-E	SPA-Bench
#Tasks	25	20
#Multi-App Tasks	19	20
#Apps	15	25
Avg # Actions	14.56	13.10
Total # Actions	364	262

Table 1: An overview of the cross-app benchmarks, showing the number of tasks, multi-apps, apps, and actions.

Following the setup in prior work (Wang et al. 2025b), we record execution traces and perform human evaluation based on the rubrics defined by Mobile-Eval-E. Mobile-Eval-E is composed of 25 manually designed tasks across 15 apps. It has the highest complexity (i.e., requires significantly more actions per task) compared to other existing benchmarks (Wang et al. 2025b), and requires multi-app interactions in 76% of the tasks. SPA-Bench provides 40 cross-app tasks, 20 in English and 20 in Chinese, covering both system and third-party apps. For our evaluation, we focused on the 20 English cross-app tasks.

Running single-app benchmark in an emulator with automated evaluation. To complement the real-device experiments, and enable large-scale automated evaluation, we also evaluate Agent-SAMA using AndroidWorld. It provides 116 task templates across 20 apps in an emulator environment and supports automated evaluation. Unlike the benchmarks mentioned above, around 90% of AndroidWorld's tasks are limited to single-app interactions.

Evaluation Metrics

Cross-app benchmarks. App tasks are inherently open-ended and do not always allow a strict binary success criterion (Wang et al. 2025b). Therefore, following prior work (Wang et al. 2025b; Liu et al. 2025b; Wang et al. 2024a, 2025a), we rely on human evaluation using the task-specific rubrics provided by Mobile-Eval-E. We then manually adapt the rubrics for SPA-Bench. The first two authors assess the task outcomes separately. In the event of disagreement, the third author serves as the deciding vote to resolve any conflicts. We evaluate Agent-SAMA using five metrics.

- **Success Rate (SR)** measures the percentage of tasks that are completed successfully (Chen et al. 2025).
- **Satisfaction Score (SS)** quantifies the proportion of rubrics fulfilled for each task, providing a finer-grained view of partial task completion.
- **Action Accuracy (AA)** captures how closely the agent's executed actions align with the human reference trajectory, computed as the ratio of correctly matched steps.
- **Termination Rate (TR)** reflects the percentage of tasks that terminate unsuccessfully. Following the definition in Mobile-Agent-E (Wang et al. 2025b), this includes exits to the home screen, closing the app, or entering unrecoverable states before task completion.
- **Recovery Success (RS)** evaluates the performance of GUI agent's recovery ability, and is defined as the percentage of failed subtasks that are successfully recovered.

MLLM Agent	SS%	AA%	TR%	SR%	RS%
	Mobile-Eval-E				
AppAgent*	25.20	60.70	96.00	–	–
Mobile-Agent*	45.50	69.80	68.00	–	–
Mobile-Agent-v2*	53.00	73.30	52.00	–	–
Mobile-Agent-E+Evo	78.97	76.65	24.00	72.00	67.34
Agent-SAMA	**86.15**	**83.24**	**16.00**	**84.00**	**71.88**
	SPA-Bench				
Mobile-Agent-E+Evo	80.30	77.86	25.00	75.00	52.86
Agent-SAMA	**88.64**	**84.35**	**20.00**	**80.00**	**66.67**

Table 2: Comparison of evaluation metrics on Mobile-Eval-E and SPA-Bench. Higher is better except for Termination Rate (TR). We adopted the results from prior work (Wang et al. 2025b) for the agents marked with *.

MLLM Agent	Success Rate (%)
GPT-4 Turbo*	30.6
GPT-4o*	34.5
GPT-4o+UGround*	44.0
GPT-4o+Aria-UI*	44.8
Mobile-Agent-E	45.7
UI-TARS*	46.6
Mobile-Agent-E+Evo	53.4
AgentS2*	54.3
V-Droid*	59.5
Agent-SAMA	63.7

Table 3: Task Success Rate (%) on AndroidWorld benchmark. We adopted the results from V-Droid (Dai et al. 2025) for the agents marked with *.

Single-app benchmark. AndroidWorld includes an automated evaluation script that verifies task correctness. We also use **Success Rate (RS)** as the primary metric, where a task is considered successful only if the final UI state precisely matches the expected outcome.

Experiment Settings

Backbone MLLMs. We use GPT-4o-2024-11-20 (OpenAI 2024) as the backbone MLLM in our experiments with a temperature of 0 to reduce variations. GPT-4o is a state-of-the-art MLLM capable of processing both text and images, making it suitable for complex mobile interaction tasks. GPT-4o offers strong performance with particularly low latency and efficient image-text alignment. **Screen Parser Implementation.** Our Screen Parser is implemented in alignment with Mobile-Agent-E (Wang et al. 2025b). For OCR detection and recognition, we adopt DBNet (Liao et al. 2020) and ConvNextViT-document from ModelScope, respectively. Icon grounding is performed using GroundingDINO (Liu et al. 2024a), while Qwen-VL-Plus (Bai et al. 2023) is used to generate textual captions for each detected icon crop.

Experiment and Results

Performance of Agent-SAMA. Table 2 compares Agent-SAMA with prior baselines, including AppAgent (Zhang et al. 2023a) and Mobile-Agent series (Wang et al. 2024b,a, 2025b) on the Mobile-Eval-E benchmarks across five evaluation metrics. We re-ran Mobile-Agent-E+Evo as our main baseline because it is the only prior agent with long-term memory and knowledge reuse, which are key features for fair comparison with Agent-SAMA. Other baselines lack memory or structured agent frameworks and have been consistently outperformed. We then compare Agent-SAMA with Mobile-Agent-E+Evo on the SPA-Bench. To ensure consistency, we re-ran Mobile-Agent-E+Evo five times using the same MLLM as Agent-SAMA, with two authors evaluating the results independently. Finally, we report the average.

Benchmark	Metric	GPT-4o	Claude 3.5	Gemini 1.5 Pro
M-Eval-E	SS	168 / 195 (86.15%)	158 / 195 (81.03%)	137 / 195 (70.25%)
	AA	303 / 364 (83.24%)	289 / 364 (79.40%)	241/ 364 (66.20%)
	TR	4 / 25 (16.00%)	5 / 25 (20.00%)	8 / 25 (32.00%)
	SR	21 / 25 (84.00%)	20 / 25 (75.00%)	17/ 25 (68.00%)
SPA-Bench	SS	117 / 132 (88.64%)	108 / 132 (81.82%)	98 / 132 (74.24%)
	AA	221 / 262 (84.35%)	215 / 262 (82.06%)	181 / 262 (69.08%)
	TR	4 / 20 (20.00%)	4 / 20 (20.00%)	7 / 20 (35.00%)
	SR	16 / 20 (80.00%)	16 / 20 (80.00%)	12 / 20 (60.00%)

Table 4: Comparison of Agent-SAMA's performance across Mobile-Eval-E and SPA-Bench using different backbones.

Across both benchmarks, Agent-SAMA improves all evaluation metrics compared to the baseline by 4.53% to over 13.00%, indicating Agent-SAMA is able to finish more cross-app tasks with higher accuracy and quality. On Mobile-Eval-E, Agent-SAMA achieves a 7.18% improvement in Satisfaction Score (SS), 6.59% in Action Accuracy (AA), 8% in Termination Rate (TR), 4.53% in Recovery Success(RS), and a notable 12% in Success Rate (SR) in comparison with the strongest baseline Mobile-Agent-E+Evo. We see similar improvements on SPA-Bench, where Agent-SAMA achieves an 8.33% improvement in SS, 6.49% in AA, 5% in TR, a significant 13.81% in RS, and a 5% in SR. These results indicate that Agent-SAMA's FSM-based architecture contributes significantly to robust, accurate, and resilient task execution in GUI agents.

Table 3 presents the result of Agent-SAMA on the AndroidWorld benchmark. The table contains popular baselines such as AgentS2 (Agashe et al. 2025), V-Droid (Dai et al. 2025), and Mobile-Agent-E (Wang et al. 2025b) and its Evo version. Agent-SAMA demonstrates competitive performance, achieving a success rate of 63.7%, outperforming most existing mobile agent baselines. While AndroidWorld tasks are generally shorter than those in Mobile-Eval-E and SPA-Bench, the benchmark enforces strict step limits and low fault tolerance (Rawles et al. 2025), making recovery more costly. Despite this, Agent-SAMA maintains strong performance through stable planning and execution.

Sensitivity of the results when changing the backbone MLLM. Table 4 shows Agent-SAMA's overall performance across three widely used MLLMs: GPT-4o (OpenAI 2024), Claude-3.5-Sonnet (Anthropic 2024), and Gemini 1.5 Pro (Gemini Team et al. 2024) on both Mobile-Eval-E and SPA-Bench benchmarks. The backbone MLLM has a large impact on the performance of Agent-SAMA, with GPT4o achieves the highest scores, followed by Claude 3.5, with Gemini 1.5 Pro performing the lowest. This trend is consistent across all three major evaluation metrics, suggesting that stronger MLLMs improve GUI agent's performance. Never-

Ablation Setting				Mobile-Eval-E			SPA-Bench		
Planning	Multi-plans	Pre/Post conditions	Mentor	SS (%)	AA (%)	SR (%)	SS (%)	AA (%)	SR (%)
		✓	✓	61.54	62.91	52.00	56.06	54.96	45.00
✓		✓	✓	72.31	67.22	68.00	71.97	69.47	60.00
✓	✓		✓	82.68	78.25	72.00	81.77	79.44	70.00
✓	✓	✓		73.85	73.08	68.00	78.79	74.05	70.00
✓	✓	✓	✓	**86.15**	**83.24**	**84.00**	**88.64**	**84.35**	**80.00**

Table 5: The results of the ablation study when removing 1) Planner agent (i.e., no planning at all), 2) Selecting the best plan across multiple plans, 3) Pre and Post conditions in State Agent, and 4) Mentor agent. The last row shows results from the full version of Agent-SAMA.

theless, when using a weaker MLLM like Claude 3.5, Agent-SAMA still outperforms Mobile-Agent-E Evo (GPT-4o version), demonstrating the effectiveness of Agent-SAMA.

A notable comparison is on error occurrence and recovery, as Agent-SAMA uses FSM to assist in reasoning and error recovery. We find that Agent-SAMA encounters notably fewer less errors during execution compared to Mobile-Agent-E + Evo (32 vs. 49 on Mobile-Eval-E and 63 vs. 70 on SPA-Bench) with a higher recovery rate (4.53% and 13.81% higher). The findings show that Agent-SAMA's structured FSM representation improves the agent's ability to make correct decisions, while also enhancing its capacity to recover from errors.

Ablation Studies. Table 5 presents the results of our ablation study on the contributions of four key components in Agent-SAMA: 1) the entire planning (removing the Planner Agent completely), 2) LLM-as-judges (Gu et al. 2025) to select the best plan among five generated plans, 3) pre- and post-conditions in the State Agent, and 4) the entire knowledge retention (removing the Mentor agent completely). We observe that each component contributes significantly to Agent-SAMA's performance, ***and their combination leads to more substantial improvements***. Removing each component reduces the performance metrics significantly, with planning having the largest impact (e.g., SR decreased from 84% to 52% in Mobile-Eval-E and from 80% to 45% on SPA-Bench), followed by multi-plan selection, Mentor agent, and then pre- and post-conditions. Importantly, when all components are combined, Agent-SAMA achieves the highest performance across all metrics. This demonstrates that these components complement and reinforce one another: effective planning helps the system select better paths, selecting the best plan from multiple ones may help reduce the effects of having excessive states on subtask planning or re-planning during error recovery, pre- and post-conditions enable better validation and recovery, and the Mentor allows learning across tasks. Overall, the full integration leads to an effective state-aware GUI agent framework.

Limitations

Benchmark Coverage. Our evaluation is limited to Mobile-Eval-E and SPA-Bench. Although these are the state-of-the-art benchmarks with the highest difficulties, cross-app tasks, and represent realistic usage scenarios, they cannot fully reflect all interactions. Some interactions (e.g., dynamic content, external interruptions like third-party ads) may even trigger unpredictable states. Future studies are needed to evaluate Agent-SAMA on more diverse scenarios and tasks.

Baseline Comparison. We included the Mobile Agent series as our primary baseline, given its strong performance on long-horizon mobile tasks. To provide border comparison, we also included additional agent frameworks such as AppAgent (Zhang et al. 2023a) and AgentS2 (Agashe et al. 2025). Notably, the results we obtained from re-running Mobile-Agent-E+Evo differ from those reported in its original paper. To migrate variation across runs, we re-ran five times with the same settings, and two authors evaluated the results independently. Finally, we reported the average. We also observe similar results across runs.

Conclusion

In this paper, we introduced Agent-SAMA, a state-aware mobile GUI agent that models app navigation and task execution as a Finite State Machine (FSM). By design, specialized agents for various phases of task execution—planning, execution, verification, and recovery, and knowledge retention—Agent-SAMA provides a structured representation and effective framework for mobile task automation. Our evaluation on two real-world cross-app benchmarks(Mobile-Eval-E and SPA-Bench) and one mostly single-app benchmark(AndroidWorld) shows that Agent-SAMA achieves up to 12% improvement in Success Rate, and 13.8% in Recovery Success compared to the baseline Mobile-Agent-E+Evo (Wang et al. 2025b) on cross-app benchmarks, and it also achieves a high success rate of 63.7% on AndroidWorld. These results highlight the value of FSM-based modeling for improving agent planning and error recovery in complex mobile environments.

References

Agashe, S.; Wong, K.; Tu, V.; Yang, J.; Li, A.; and Wang, X. E. 2025. Agent S2: A Compositional Generalist-Specialist Framework for Computer Use Agents. arXiv:2504.00906.

Anthropic. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/3-5-models-and-computer-use. Accessed: 2025-04-22.

Author(s), A. 2025. Agent-SAMA: State-Aware Mobile Assistant.

Bai, J.; Bai, S.; Yang, S.; Wang, S.; Tan, S.; Wang, P.; Lin, J.; Zhou, C.; and Zhou, J. 2023. Qwen-VL: A Versatile

Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. arXiv:2308.12966.

Chang, E. Y.; and Geng, L. 2025. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. arXiv:2503.11951.

Chen, J.; Yuen, D.; Xie, B.; Yang, Y.; Chen, G.; Wu, Z.; Yixing, L.; Zhou, X.; Liu, W.; Wang, S.; Zhou, K.; Shao, R.; Nie, L.; Wang, Y.; HAO, J.; Wang, J.; and Shao, K. 2025. SPA-Bench: A Comprehensive Benchmark for SmartPhone Agent Evaluation. In *The Thirteenth International Conference on Learning Representations*.

Dai, G.; Jiang, S.; Cao, T.; Li, Y.; Yang, Y.; Tan, R.; Li, M.; and Qiu, L. 2025. Advancing Mobile GUI Agents: A Verifier-Driven Approach to Practical Deployment. arXiv:2503.15937.

Ding, T. 2024. MobileAgent: enhancing mobile control via human-machine interaction and SOP integration. arXiv:2401.04124.

Espada, A. R.; Gallardo, M. d. M.; Salmerón, A.; and Merino, P. 2015. Using Model Checking to Generate Test Cases for Android Applications. *Electronic Proceedings in Theoretical Computer Science*, 180: 7–21.

Gemini Team; et al. 2024. Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context. https://arxiv.org/abs/2403.05530. ArXiv preprint arXiv:2403.05530, Accessed: 2025-04-22.

Gu, J.; Jiang, X.; Shi, Z.; Tan, H.; Zhai, X.; Xu, C.; Li, W.; Shen, Y.; Ma, S.; Liu, H.; Wang, S.; Zhang, K.; Wang, Y.; Gao, W.; Ni, L.; and Guo, J. 2025. A Survey on LLM-as-a-Judge. arXiv:2411.15594.

Lee, S.; Choi, J.; Lee, J.; Wasi, M. H.; Choi, H.; Ko, S. Y.; Oh, S.; and Shin, I. 2024. Explore, Select, Derive, and Recall: Augmenting LLM with Human-like Memory for Mobile Task Automation. arXiv:2312.03003.

Li, Y.; Zhang, C.; Yang, W.; Fu, B.; Cheng, P.; Chen, X.; Chen, L.; and Wei, Y. 2024. AppAgent v2: Advanced Agent for Flexible Mobile Interactions. arXiv:2408.11824.

Liao, M.; Wan, Z.; Yao, C.; Chen, K.; and Bai, X. 2020. Real-time Scene Text Detection with Differentiable Binarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 11474–11481. AAAI.

Liu, G.; Zhao, P.; Liu, L.; Guo, Y.; Xiao, H.; Lin, W.; Chai, Y.; Han, Y.; Ren, S.; Wang, H.; Liang, X.; Wang, W.; Wu, T.; Li, L.; Wang, H.; Xiong, G.; Liu, Y.; and Li, H. 2025a. LLM-Powered GUI Agents in Phone Automation: Surveying Progress and Prospects. arXiv:2504.19838.

Liu, S.; Zeng, Z.; Ren, T.; Li, F.; Zhang, H.; Yang, J.; Jiang, Q.; Li, C.; Yang, J.; Su, H.; Zhu, J.; and Zhang, L. 2024a. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. arXiv:2303.05499.

Liu, X.; Qin, B.; Liang, D.; Dong, G.; Lai, H.; Zhang, H.; Zhao, H.; Iong, I. L.; Sun, J.; Wang, J.; Gao, J.; Shan, J.; Liu, K.; Zhang, S.; Yao, S.; Cheng, S.; Yao, W.; Zhao, W.; Liu, X.; Liu, X.; Chen, X.; Yang, X.; Yang, Y.; Xu, Y.; Yang, Y.; Wang, Y.; Xu, Y.; Qi, Z.; Dong, Y.; and Tang, J. 2024b. AutoGLM: Autonomous Foundation Agents for GUIs. arXiv:2411.00820.

Liu, Y.; Sun, H.; Liu, W.; Luan, J.; Du, B.; and Yan, R. 2025b. MobileSteward: Integrating Multiple App-Oriented Agents with Self-Evolution to Automate Cross-App Instructions. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, 883–893. ACM.

Mahmud, T.; Duan, B.; Pasareanu, C.; and Yang, G. 2025. Enhancing LLM Code Generation with Ensembles: A Similarity-Based Selection Approach. arXiv:2503.15838.

OpenAI. 2024. GPT-4o System Card. https://cdn.openai.com/gpt-4o-system-card.pdf. Accessed: 2025-04-22.

Prajapati, D. 2012. Hierarchical state machines for native mobile apps. In *2012 Annual IEEE India Conference (INDICON)*, 640–642.

Rawles, C.; Clinckemaillie, S.; Chang, Y.; Waltz, J.; Lau, G.; Fair, M.; Li, A.; Bishop, W.; Li, W.; Campbell-Ajala, F.; Toyama, D.; Berry, R.; Tyamagundlu, D.; Lillicrap, T.; and Riva, O. 2025. AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents. arXiv:2405.14573.

Seo, W.; Lee, J.; and Bu, Y. 2025. SPIO: Ensemble and Selective Strategies via LLM-Based Multi-Agent Planning in Automated Data Science. arXiv:2503.23314.

Shahbaz, M.; Aichernig, B. K.; Rafi, M.; and Ali, S. A. A. 2022. FSMApp: Model-based testing of mobile applications. *Advances in Computers*, 125: 167–203.

Sun, Y.; Zhao, S.; Yu, T.; Wen, H.; Va, S.; Xu, M.; Li, Y.; and Zhang, C. 2025. GUI-Xplore: Empowering Generalizable GUI Agents with One Exploration. *arXiv preprint arXiv:2503.17709*.

Wagner, F.; Schmuki, R.; Wagner, T.; and Wolstenholme, P. 2006. *Modeling Software with Finite State Machines: A Practical Approach*. Auerbach Publications, 1st edition.

Wang, J.; Xu, H.; Jia, H.; Zhang, X.; Yan, M.; Shen, W.; Zhang, J.; Huang, F.; and Sang, J. 2024a. Mobile-Agent-v2: Mobile Device Operation Assistant with Effective Navigation via Multi-Agent Collaboration. arXiv:2406.01014.

Wang, J.; Xu, H.; Ye, J.; Yan, M.; Shen, W.; Zhang, J.; Huang, F.; and Sang, J. 2024b. Mobile-Agent: Autonomous Multi-Modal Mobile Device Agent with Visual Perception. arXiv:2401.16158.

Wang, J.; Xu, H.; Zhang, X.; Yan, M.; Zhang, J.; Huang, F.; and Sang, J. 2025a. Mobile-Agent-V: Learning Mobile Device Operation Through Video-Guided Multi-Agent Collaboration. arXiv:2502.17110.

Wang, L.; Deng, Y.; Zha, Y.; Mao, G.; Wang, Q.; Min, T.; Chen, W.; and Chen, S. 2024c. MobileAgentBench: An Efficient and User-Friendly Benchmark for Mobile LLM Agents. arXiv:2406.08184.

Wang, Y. 2004. Using Statecharts to Model User Interface Behavior for Mobile Applications. In *Proceedings of the 42nd Annual ACM Southeast Regional Conference (ACMSE '04)*, 52–57. Association for Computing Machinery.

Wang, Z.; Xu, H.; Wang, J.; Zhang, X.; Yan, M.; Zhang, J.; Huang, F.; and Ji, H. 2025b. Mobile-Agent-E: Self-Evolving Mobile Assistant for Complex Tasks. *arXiv preprint arXiv:2501.11733*.

Wen, H.; Li, Y.; Liu, G.; Zhao, S.; Yu, T.; Li, T. J.-J.; Jiang, S.; Liu, Y.; Zhang, Y.; and Liu, Y. 2024a. AutoDroid: LLM-powered Task Automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, 543–557. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704895.

Wen, H.; Li, Y.; Liu, G.; Zhao, S.; Yu, T.; Li, T. J.-J.; Jiang, S.; Liu, Y.; Zhang, Y.; and Liu, Y. 2024b. AutoDroid: LLM-powered Task Automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, 543–557. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704895.

Xing, M.; Zhang, R.; Xue, H.; Chen, Q.; Yang, F.; and Xiao, Z. 2024. Understanding the Weakness of Large Language Model Agents within a Complex Android Environment. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, 6061–6072. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704901.

Zhang, C.; Yang, Z.; Liu, J.; Han, Y.; Chen, X.; Huang, Z.; Fu, B.; and Yu, G. 2023a. AppAgent: Multimodal Agents as Smartphone Users. arXiv:2312.13771.

Zhang, D.; Shen, Z.; Xie, R.; Zhang, S.; Xie, T.; Zhao, Z.; Chen, S.; Chen, L.; Xu, H.; Cao, R.; and Yu, K. 2023b. Mobile-Env: Building Qualified Evaluation Benchmarks for LLM-GUI Interaction. *CoRR*, abs/2305.08144.

Zhang, J.; Zhao, C.; Zhao, Y.; Yu, Z.; He, M.; and Fan, J. 2024. MobileExperts: A Dynamic Tool-Enabled Agent Team in Mobile Devices. arXiv:2407.03913.