

# MobileWorld: Benchmarking Autonomous Mobile Agents in Agent-User Interactive and MCP-Augmented Environments

Quyuan Kong<sup>1\*</sup>, Xu Zhang<sup>1\*</sup>, Zhenyu Yang<sup>2</sup>, Nolan Gao<sup>3</sup>,  
Chen Liu<sup>1</sup>, Panrong Tong<sup>1</sup>, Chenglin Cai<sup>1</sup>, Hanzhang Zhou<sup>1</sup>,  
Jianan Zhang<sup>1</sup>, Liangyu Chen<sup>1</sup>, Zhidan Liu<sup>2</sup>, Steven HOI<sup>1</sup>, Yue Wang<sup>1†</sup>

<sup>1</sup>Tongyi Lab, Alibaba Group, <sup>2</sup>HKUST (GZ), <sup>3</sup>University of Florida

\*Equal contribution †Correspondence: yue.w@alibaba-inc.com

## Abstract

Among existing online mobile-use benchmarks, AndroidWorld has emerged as the dominant benchmark due to its reproducible environment and deterministic evaluation; however, recent agents achieving over 90% success rates indicate its saturation and motivate the need for a more challenging benchmark. To bridge these gaps, we introduce MobileWorld, a substantially more challenging benchmark designed to reflect real-world usage through 201 tasks across 20 applications. MobileWorld derives its difficulty from an emphasis on long-horizon, cross-application workflows, requiring nearly twice as many completion steps on average (27.8 vs. 14.3) and featuring a significantly higher proportion of multi-app tasks (62.2% vs. 9.5%) than AndroidWorld. MobileWorld balances production-grade utility and reproducible evaluation using open-source alternatives to industry standards (e.g., Mattermost for Slack), enabling full observability through source code modification and direct database access. Beyond standard GUI manipulation, MobileWorld introduces novel task categories including agent-user interaction and Model Context Protocol (MCP)-augmented tasks for evaluating agents in user-aware, hybrid-tool scenarios. We develop a planner-executor framework with extended action spaces supporting user interactions and MCP calls. Results show a sharp performance drop from AndroidWorld, with the best agentic framework and end-to-end model achieving 51.7% and 20.9% success rates, respectively, highlighting substantial room for future research.

## 1 Introduction

GUI agents on smartphones have emerged as a transformative technology, enabling users to automate complex mobile tasks through natural language instructions. With the rapid advancement of Vision-Language Models (VLMs), numerous mobile agents have been developed to navigate in-

terfaces automatically (Rawles et al., 2023; Deng et al., 2024; Chen et al., 2024; Yang et al., 2025; Zhou et al., 2025). Despite the existence of multiple benchmarks, AndroidWorld (Rawles et al., 2024) remains the go-to benchmark for mobile GUI agent evaluation, owing to its reproducible emulator environment and deterministic evaluation. However, it is approaching saturation—state-of-the-art (SOTA) frameworks now exceed 90% success rate—limiting our ability to distinguish incremental improvements from genuine breakthroughs.

Beyond saturation, current benchmarks exhibit several fundamental limitations: **(1) Limited task complexity.** Existing benchmarks omit critical application categories (e.g., e-commerce, enterprise communication) and typically feature short-horizon, single-application tasks, whereas real-world mobile assistance demands long-horizon planning and cross-application workflows. **(2) Fully-specified instructions.** Benchmarks neglect realistic scenarios where users provide vague or incomplete requests, requiring agents to engage in clarification dialogues. **(3) No external tool integration.** The Model Context Protocol (MCP) (Anthropic, 2024) has emerged as a standardized interface for external tools (Wang et al., 2025a). Next-generation agents must seamlessly combine GUI manipulation with tool invocations, yet no existing benchmark evaluates this hybrid paradigm.

To address these limitations, we introduce MobileWorld, a substantially more challenging benchmark designed to reflect real-world mobile usage. We compare MobileWorld with other online mobile-use benchmarks in Table 1. Detailed related work can be found in Section 2. In summary, our contributions are as follows:

**Increased Task Complexity.** As shown in Figure 1, MobileWorld tasks require on average 27.8 completion steps—nearly twice the 14.3 steps in AndroidWorld—with 62.2% involving cross-application workflows (vs. 9.5% in AndroidWorld).

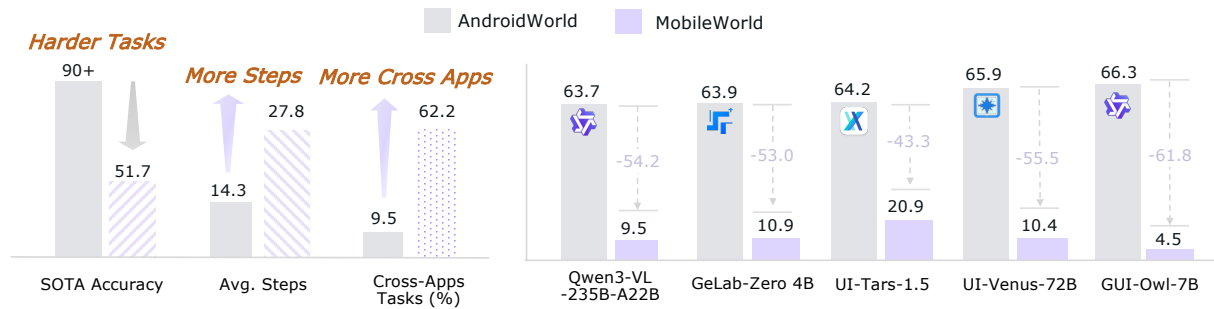


Figure 1: Compared to AndroidWorld, MobileWorld exhibits lower SOTA success rates, longer task horizons, more cross-application tasks, and sharp performance drops for recent models.

These tasks demand challenging capabilities including long-horizon multi-step planning, memory retention, and precise instruction following, etc. Consequently, the top-performing framework achieves only 51.7% success rate on MobileWorld, compared to over 90% on AndroidWorld.

**Agent-User Interaction Tasks.** Agent-user interaction tasks that evaluate an agent’s ability to handle ambiguous instructions through collaborative dialogue. As shown in the left side of Figure 2, consider the user request: “Send an email to Kevin with the message “Hello””. However, Kevin’s email address is not present in the device’s Email or Contacts app. In such cases, the agent must recognize the missing information and proactively trigger an ask\_user action to obtain the details, rather than hallucinating or failing silently. Inspired by Yao et al. (2024), we employ a user agent role-played by a LLM (GPT-4.1 in the example), which holds the key information in its context, and interacts with the GUI agent. This task type tests whether agents can identify the ambiguity in the instruction and ask for clarification. 22.4% of all tasks involve agent-user interaction in MobileWorld.

**MCP-Augmented Tasks.** MCP-augmented tasks that require hybrid-usage of the GUI navigation and external tool invocations. As illustrated in the right side of Figure 2, an agent is tasked with fetching a GitHub repository’s README and sending its summarization via a team collaboration software (Mattermost). Instead of navigating the repository through a web browser via a sequence of GUI actions, the agent can invoke an MCP tool to directly retrieve the content. After obtaining the relevant information from the GitHub MCP tool, the agent continues the task through GUI interactions to compose and send the message. With the rapid adoption of MCP as a standardized protocol for tool integration, evaluating an agent’s strate-

gic choice between GUI operations and API-based tools has become essential. MCP-augmented tasks constitute 19.9% of all tasks in MobileWorld.

### Reproducible and Deterministic Evaluation.

Prior benchmarks for commercial applications (shown in Table 1) suffer from authentication barriers and opaque internal states that necessitate noisy “MLLM-as-a-judge” evaluations. In contrast, we replace proprietary services with self-hosted open-source alternatives (e.g., Mattermost in place of Slack), enabling direct access to internal databases. Our framework employs a multi-faceted validation suite — comprising textual response verification, backend database inspection, local storage analysis, and application-specific callbacks — to ensure absolute determinism. Furthermore, Docker-based Android Virtual Devices (AVDs) coupled with snapshot management enable a standardized and fully reproducible environment.

### Comprehensive Evaluation on MobileWorld.

To address the unique requirements in our benchmark, we propose a planner-executor framework extending the action space with ask\_user and mcp\_call, enabling unified GUI manipulation, user interaction, and tool use. We evaluate 3 backbone models with our framework and 9 end-to-end GUI agents. As shown in Table 3, our framework achieves a SOTA (51.7%) success rate on MobileWorld. Crucially, end-to-end agents exhibit capability collapse on novel task types: most score below 10% on agent—user interaction tasks and near 0% on MCP tasks. These findings underscore fundamental deficiencies in current architectures regarding active user interaction and tool-use. Finally, we perform rigorous error analysis to reveal five research challenges for the community.

In summary, MobileWorld provides a rigorous and reproducible environment for advancing mobile agents in real-world scenarios that must excel

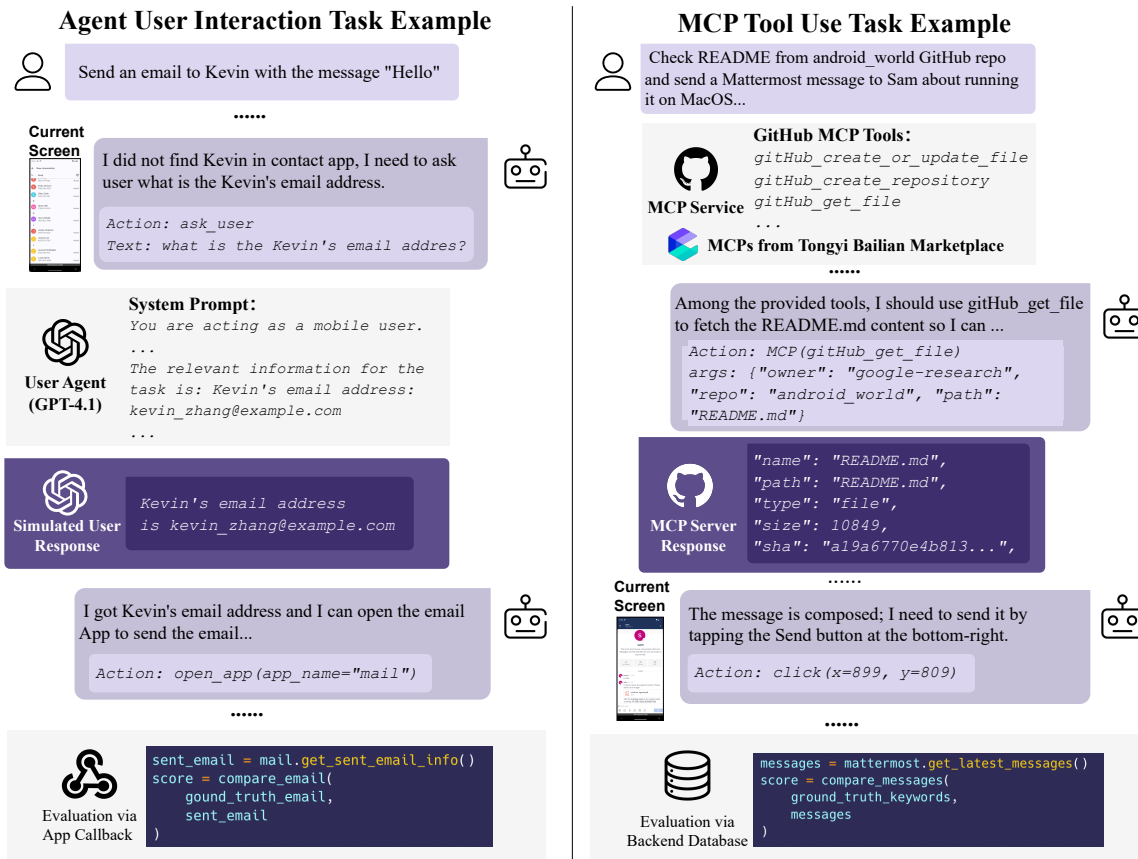


Figure 2: **Left:** An example of an agent-user interaction task, in which the agent must proactively request clarification from a LLM-simulated user when encountering incomplete information. Task completion is verified through the application’s callback cache. **Right:** An example of an MCP-augmented task, where the agent is initialized with a list of GitHub MCP tools and selects the appropriate tool to retrieve README content from a GitHub repository before completing the task via GUI operations. Task completion is verified through backend database inspection.

at **long-horizon reasoning**, **active user interaction**, and **MCP tool use**. By evaluating SOTA models, we identify core research challenges and offer the community a roadmap toward more realistic and autonomous mobile intelligence.

## 2 Related Works

The development of autonomous agents capable of controlling computers has spurred the creation of various benchmarks across different platforms. For desktop environments, benchmarks such as WindowsAgentArena (Bonatti et al., 2024), OS-World (Xie et al., 2024), and WebArena (Zhou et al., 2023) evaluate agents on operating system tasks and web browsing scenarios. In the mobile domain, several benchmarks have emerged to evaluate Android agents. AndroidWorld (Rawles et al., 2024) introduces a fully functional Android environment with 116 programmatic tasks across 20 real-world apps, featuring dynamic task construction with parameterized natural language in-

structions. AndroidLab (Xu et al., 2025) provides a systematic framework supporting both LLMs and multimodal models. More recently, Android Agent Arena (A3) (Chai et al., 2025) addresses some limitations by incorporating 20 widely used third-party commercial apps and 201 tasks with real-time online information retrieval. Other notable efforts include SPA-Bench (Chen et al., 2024), ProBench (Yang et al., 2025), and AndroidDaily (Yan et al., 2025a), which further expand task coverage and evaluation methodologies.

Beyond standard GUI operations, recent work has recognized the importance of evaluating agents on their interaction capabilities and tool usage.  $\tau$ -bench (Yao et al., 2024) introduces dynamic conversations between simulated users and agents equipped with domain-specific API tools and policy guidelines. Building on this,  $\tau^2$ -bench (Barres et al., 2025) models a dual-control environment where both agent and user actively modify shared world states. Meanwhile, MCP has emerged as a

Table 1: Comparison of online Mobile GUI Agent Benchmarks. MobileWorld uniquely enables deterministic evaluation even for applications requiring backends (e.g., messaging) by utilizing self-hosted environments. We also introduce novel task paradigms: agent-user interaction and MCP augmentation. Backend-Req. Apps: Environment includes third-party apps requiring external backend authentication. (✓ = Supported, ✗ = Not Supported).

Benchmark	#Apps	#Tasks	Agent-User Int. Tasks	MCP-Aug. Tasks	Backend-Req. Apps	Deterministic Evaluation
AndroidArena	16	221	✗	✗	✓	✗
A3	20	201	✗	✗	✓	✗
Pro-Bench	34	200	✗	✗	✓	✗
AndroidDaily	48	235	✗	✗	✓	✗
SPA-Bench	66	340	✗	✗	✓	✗
MobileAgentBench	10	100	✗	✗	✗	✓
AndroidLab	9	138	✗	✗	✗	✓
AndroidWorld	20	116	✗	✗	✗	✓
<b>MobileWorld (Ours)</b>	<b>20</b>	<b>201</b>	✓	✓	✓	✓

mechanism for agents to invoke external tools (Luo et al., 2025; Wang et al., 2025b) alongside GUI operations. OSWorld-MCP (Jia et al., 2025) presents the first benchmark assessing tool invocation, GUI operation, and decision-making in real-world environments. Similarly, MCPWorld (Yan et al., 2025b) is a benchmark for API, GUI, and hybrid agents using "white-box apps" with source code availability.

### 3 MobileWorld

In this section, we formalize the task definition, describe the environment architecture and detail the benchmark construction methodology.

#### 3.1 Task Definition

**GUI-Only Tasks** We construct GUI-Only tasks following AndroidWorld (Rawles et al., 2024), which comprise Task Completion and Information Retrieval, requiring the agent to accomplish user objectives through standard mobile UI operations. Task Completion involves executing a sequence of GUI actions to drive the system into a specified target state (e.g., composing and sending an email), while Information Retrieval requires locating and gathering relevant information from internal mobile data or external sources (e.g., answering questions about today’s scheduled events).

**Agent-user Interaction Tasks** Previous mobile GUI agent benchmarks assume that user instructions are unambiguous. However, this assumption rarely holds in real-world scenarios. To reflect this challenge, we introduce a dedicated set of tasks in which critical information is deliberately omitted from the task goal, rendering successful completion impossible without actively seeking clarification.

**MCP-augmented Tasks** MCP provides a more straightforward way to access a wider range of resources and functionalities beyond standard GUI operations (Jia et al., 2025). However, in real-world scenarios, not all applications are equipped with well-designed MCP interfaces, and the GUI remains the predominant mode of interaction. Given this co-existence of GUI-based interaction and MCP-enabled tool access, we specifically design a set of tasks augmented with MCP tools. These tasks require agents to integrate MCP tool invocations with standard GUI operations to succeed.

#### 3.2 Environment Development

As illustrated in Figure 3, MobileWorld comprises two main components: the host machine running GUI agents and the containerized environment. We detail the key design elements below.

**Action Space** MobileWorld supports a hybrid action space combining GUI operations, agent-user interaction, and MCP tool invocation (see Section A for complete specification). *GUI Operations* include standard mobile manipulations: click, double\_tap, long\_press, drag, input\_text, scroll, navigation commands, answer for information retrieval, wait for screen updates, and status to signal completion. *Agent-User Interaction*: The ask\_user action routes clarification queries to an LLM-based user agent (e.g., GPT-4.1). *MCP Tool Integration*: The mcp\_call action invokes external tools from curated MCP servers<sup>1</sup>, spanning geospatial navigation, code analysis, document processing, etc.

<sup>1</sup><https://bailian.console.aliyun.com/?tab=app>

**Stable and Reproducible Environment** MobileWorld achieves stability and reproducibility through containerization and careful engineering.

*Containerized Architecture.* The entire evaluation environment is encapsulated in Docker-in-Docker containers (right side in Figure 3), including a rooted Android Virtual Device (AVD), self-hosted application backends, and an API server for orchestration. This design eliminates external dependencies and enables consistent deployment across different host systems.

*Open-Source Applications.* We construct stable, reproducible application environments based on popular open-source projects, including Mattermost (an open-source alternative to Slack), Mastodon (an open-source alternative to X/Twitter), and Mall4Uni (e-commerce platform). Self-hosting these applications provides full backend access, enabling precise control over task initialization states and deterministic verification of task outcomes. More details about the open-source applications can be referred in Section E.2.

*Snapshot-Based State Management.* To ensure consistent initial conditions across evaluation runs, we employ AVD snapshots that capture the complete device state. Each task execution begins from a predetermined snapshot, guaranteeing reproducible starting conditions.

**Task Evaluation** Deterministic evaluation is essential for reliable benchmarking. We implement multiple complementary verification methods to assess task success:

*Textual Answer Verification.* For information retrieval tasks where the expected output is a specific textual response, we employ pattern matching with regular expressions or exact string comparison against ground-truth answers.

*Backend Database Verification.* For tasks involving self-hosted applications such as Mattermost or Mastodon, we directly query the backend database to verify that expected state changes have occurred.

*Local Storage Inspection.* Leveraging rooted emulator access, we use Android Device Bridge (ADB) to inspect application-specific local storage and verify internal state.

*Application Callbacks.* For tasks involving customized applications, we implement callback APIs that capture intermediate states during execution. These states are persisted and subsequently queried by the evaluator to determine task success.

### 3.3 Task Construction

Based on this environment, we construct 201 tasks across diverse scenarios (Table 2). The categories span GUI-only (57.7%), agent-user interaction (22.4%), and MCP-augmented tasks (19.9%). Most tasks (62.2%) require cross-app coordination. For evaluation, we employ four verification modes: database verification (47.3%), storage inspection (36.8%), textual matching (10.9%), and app callbacks (5.0%). Additionally, we integrate 5 MCP servers with 64 tools.

**Task Instructions and Scenarios** To ensure our benchmark reflects diverse and realistic mobile usage scenarios, we curate tasks from a broad spectrum of real-world domains, including e-commerce, social platform, productivity collaboration, on-device system management, and information retrieval. All tasks are constructed based on the MobileWorld initial system state, which is preloaded by contacts, SMS messages, emails, etc. The preloaded materials are either synthesized by LLM or derived from the internet. The complete APP list can be found in Section E.1.

**Agent-User Interaction Task Construction** To construct agent-user interaction tasks, we first ask human annotators to write a clear and achievable task goal (e.g., “Send an email to kevin\_zhang@example.com with ‘Hello.’”), then remove critical information—such as replacing the email with just “Kevin”—making tasks ambiguous. To prevent unintended information leakage, annotators also verify that the device environment does not contain the omitted information (e.g., ensuring “Kevin” is not in Contacts). We deploy a LLM-simulated *user agent* responds to clarification requests, configured with the full task goal and the exact omitted information. In addition, the user agent is instructed not to assist beyond this scope. Full prompt for the user agent is provided in the following block.

#### User Agent System Prompt

```
# System Prompt: You are acting as a mobile phone user. An mobile GUI agent is executing a task on your phone. The task goal is: {self.goal}. You need to answer questions from the mobile GUI agent. The relevant information for the task is: {self.relevant_information}. If the question is not related to the task or no more task-related information is available, you need to refuse to answer in a polite manner. DO NOT make up any information. You can ONLY give the answer based on the relevant information and the task goal. Today is 2025-10-16, Thursday. If the question is about the date, you need to answer the correct date based on the current date.
```

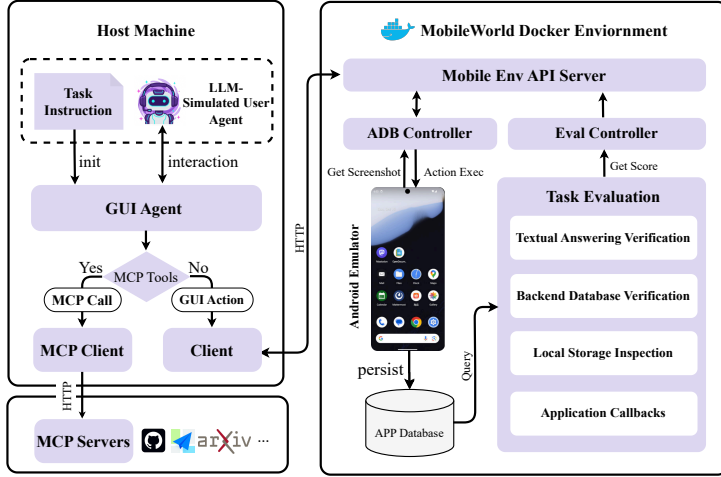


Figure 3: System architecture of MobileWorld. **Left:** GUI agents on the host receive instructions, interact with users, and execute GUI actions or MCP tool calls. **Right:** a dockerized Android environment with self-hosted app backends and an evaluator supporting 4 verification modes.

**MCP-Augmented Task Construction** Given the selected MCP servers and tools (see Section B), annotators are instructed to read each tool’s description and sample output. They then create standalone tasks that can be completed using only MCP tools (e.g., “List all commits in the main branch of the mastodon/mastodon repository from the past week” via the GitHub MCP). These tasks are non-trivial or error-prone via pure-GUI operations. Subsequently, annotators extend these into hybrid workflows by appending one or more GUI-based actions that consume the MCP output. For example, after retrieving a commit history via GitHub MCP tool calling, the agent is instructed to compose and send an email summarizing the changes using the native Email app.

### 3.4 Planner-Executor Agentic Framework

To support the novel task categories of MobileWorld, we propose a planner-executor agentic framework (China Mobile Jiutian Team, 2025). The *planner* interprets task instructions, analyzes the current screenshot and interaction history, and selects high-level actions (e.g., click, swipe, type, wait). Instead of outputting pixel coordinates, it describes the target UI element in natural language (e.g., “the ‘Send’ button at the bottom right corner”), which is then grounded to precise coordinates by a dedicated *grounding executor* using the screenshot. This decoupling allows any general-purpose VLM to be plugged in without requiring built-in pixel-level grounding capabilities. To handle MobileWorld’s unique requirements, we extend

the action space with `ask_user`, which queries an LLM-simulated user and `mcp_call`, which invokes external MCP tools by specifying their name and parameters. This unified loop seamlessly integrates GUI interaction, user clarification, and tool use. Further details are provided in Section D.

## 4 Experiments

In this section, we conduct comprehensive experiments to evaluate state-of-the-art models on MobileWorld and analyze their capabilities across different task categories.

### 4.1 Metrics

We define the following metrics to measure GUI agents on MobileWorld.

**Success Rate (SR)** measures the proportion of tasks successfully completed by the agent. For each task  $i$ , a binary score  $s_i \in \{0, 1\}$  is assigned, where  $s_i = 1$  if the task objective is fully achieved and  $s_i = 0$  otherwise. The overall success rate is computed as  $SR = \frac{1}{N} \sum_{i=1}^N s_i$ . We also report category-specific success rates: **GUI-Only SR** for GUI-only tasks, **Interaction SR** for agent–user interaction tasks (denoted by  $\mathcal{I}_{interact}$ ), and **MCP SR** for MCP-augmented tasks (denoted by  $\mathcal{I}_{MCP}$ ).

**Average Completion Steps (Ave. Steps)** denotes the average number of action steps taken across all execution trajectories. Let  $t_i$  be the number of steps in the trajectory for task  $i$  (including both successful and failed episodes). The average completion steps is defined as: Ave. Steps =

Table 2: Key statistics in MobileWorld.

Statistic	Number
Total tasks	201 (100%)
<i>Category Breakdown</i>	
– GUI-Only Tasks	116 (57.7%)
– Agent-User Int. Tasks	45 (22.4%)
– MCP-Aug. Tasks	40 (19.9%)
<i>App Complexity</i>	
– Single App Tasks	76 (37.8%)
– Two Apps Tasks	100 (49.8%)
– $\geq$ Three Apps Tasks	25 (12.4%)
<i>Evaluation Mode</i>	
– App Callbacks	10 (5.0%)
– Textual Matching	22 (10.9%)
– Storage Inspection	74 (36.8%)
– DB Verification	95 (47.3%)
Total MCP Servers	5
Total MCP Tools	64

$\frac{1}{N} \sum_{i=1}^N t_i$ . Under comparable success rates, a lower value reflects greater execution efficiency.

**Average User Queries (Ave. Queries)** measures the average number of ask\_user actions invoked by the agent across agent-user interaction tasks. Let  $c_i$  denote the number of ask\_user actions for task  $i \in \mathcal{I}_{\text{interact}}$ . This metric is defined as:

$$\text{Ave. Queries} = \frac{1}{|\mathcal{I}_{\text{interact}}|} \sum_{i \in \mathcal{I}_{\text{interact}}} c_i \quad (1)$$

This metric reflects how frequently agents seek user clarification, with optimal agents achieving task success while minimizing unnecessary queries.

**User Interaction Quality (UIQ)** evaluates both the effectiveness and efficiency of the agent’s ask\_user invocations. Let  $\mathcal{I}_{\text{interact}}$  denote the set of agent-user interaction task indices, and let  $\mathcal{I}_{\text{triggered}}$  denote the set of non-interaction task indices where the agent invoked at least one ask\_user action. For each task  $i$ , let  $c_i$  denote the number of ask\_user actions invoked and  $s_i \in \{0, 1\}$  the task success indicator. The quality score for task  $i \in \mathcal{I}_{\text{interact}}$  is computed as:

$$q_i = \begin{cases} \frac{s_i}{c_i} & \text{if } c_i > 0, \\ 0 & \text{if } c_i = 0. \end{cases} \quad (2)$$

The overall UIQ score is then defined as:

$$\text{UIQ} = \frac{\sum_{i \in \mathcal{I}_{\text{interact}}} q_i}{|\mathcal{I}_{\text{interact}}| + |\mathcal{I}_{\text{triggered}}|} \quad (3)$$

UIQ rewards agents that successfully complete interaction tasks with fewer queries, while penalizing agents that fail to seek necessary clarification or unnecessarily invoke ask\_user on non-interaction tasks. High UIQ indicates the agent appropriately recognizes when user clarification is needed and efficiently resolves ambiguities.

**Average MCP Tool Calls (Ave. MCP Calls)** measures the average number of MCP tool invocations by the agent in MCP-augmented tasks. It is defined as:

$$\text{Ave. MCP Calls} = \frac{1}{|\mathcal{I}_{\text{MCP}}|} \sum_{i \in \mathcal{I}_{\text{MCP}}} m_i \quad (4)$$

where  $m_i$  denote the number of MCP tool calls for task  $i \in \mathcal{I}_{\text{MCP}}$ . It indicates the agent’s propensity to leverage external tools, with effective agents appropriately integrating MCP tool access into their task execution workflows.

Table 3: Success rate (%) comparison of state-of-the-art mobile GUI agent models on MobileWorld under maximum 50 steps. We report overall SR and breakdown by task category: GUI-Only tasks, agent-user interaction tasks, and MCP-augmented tasks. **Bold** indicates the best result and underline indicates the second best.

Model	Overall	GUI-Only	User-Int.	MCP
Claude-4.5-Sonnet	<u>43.8</u>	47.8	<u>37.8</u>	<u>50.0</u>
Gemini-3-Pro	46.3	<b>55.6</b>	24.4	48.6
GPT-5	<b>51.7</b>	<u>54.0</u>	<b>62.2</b>	<b>51.6</b>
GUI-Owl-7B	4.5	7.7	-	-
GUI-Owl-32B	5.5	8.5	-	-
UI-Venus-7B	5.5	8.5	-	-
UI-Venus-72B	10.4	<u>16.4</u>	-	-
Qwen3-VL-8B	5.5	9.4	0.0	0.0
Qwen3-VL-32B	9.0	11.9	<u>6.7</u>	<u>2.7</u>
Qwen3-VL-235A22B	9.5	12.8	4.4	<b>5.4</b>
GELab-Zero-4B	<u>10.9</u>	16.1	<u>6.7</u>	-
Doubao-1.5-UI-TARS	<b>20.9</b>	<b>26.3</b>	<b>32.4</b>	-

## 4.2 Experimental Setup

We evaluate a suite of state-of-the-art Large Multimodal Models (LMMs), including Qwen3-VL-235B-A22B-Instruct (Qwen Team, 2025), Qwen3-VL-32B-Instruct (Qwen Team, 2025), GUI-Owl-32B (Ye et al., 2025), UI-Venus-72B (Gu et al., 2025), Doubao-1.5-UI-TARS (Seed, 2025), GELab-Zero (Team, 2025), OpenAI GPT-5 (OpenAI, 2025), Claude-4.5-Sonnet (Anthropic, 2025), and Gemini-3-Pro (Comanici et al., 2025). To enable consistent comparison of performance across models, we integrate general-purpose LMMs, such as GPT-5 and Claude-4.5-Sonnet, into the planner-executor agentic framework introduced in Section 3.4. We uniformly leverage UI-Ins-7B (Chen et al., 2025) as the grounding executor model. For end-to-end GUI agent models, including Doubao-1.5-UI-TARS, GUI-Owl, UI-Venus, GELab-Zero and Qwen3-VL, we employ their official implementations and adapt them to the MobileWorld environment. The details of the experiment setup can be found in Section C.

## 4.3 Main Results

Table 3 presents the success rate comparison across all evaluated models under different task categories. Several key observations emerge from our results.

First, MobileWorld presents a significant challenge even for state-of-the-art models. The best-performing model, GPT-5 + UI-Ins-7B, achieves only 51.7% overall success rate, substantially lower than the 90%+ success rates reported on AndroidWorld (Rawles et al., 2024). This confirms that

our benchmark effectively addresses the saturation problem in existing mobile agent evaluation.

Second, there exists a substantial performance gap between agentic frameworks and end-to-end GUI-specialized models. Agentic frameworks leveraging frontier LLMs (e.g., GPT-5, Claude-4.5-Sonnet and Gemini-3-Pro) consistently outperform end-to-end models by a large margin. The best agentic model achieves an overall success rate of 51.7%, compared to only 20.9% for the best end-to-end model (Doubao-1.5-UI-TARS). This gap stems from two key limitations of current end-to-end approaches: (1) insufficient capability in handling the complex reasoning and cross-app coordination required by MobileWorld, and (2) a lack of support for agent-user interaction and MCP tool invocation.

Third, agent-user interaction and MCP-augmented tasks prove particularly challenging and reveal critical capability gaps. On agent-user interaction tasks, Qwen3-VL and GELab-Zero achieve scores below 10%, while Doubao-1.5-UI-TARS achieves only 32.4%. This highlights the difficulties in recognizing when user clarification is needed. In contrast, GPT-5 demonstrates strong performance (62.2%) on these tasks, suggesting that advanced reasoning capabilities are essential for effective human-agent collaboration. Similarly, MCP-augmented tasks expose fundamental limitations in tool integration. Qwen3-VL is unable to properly utilize MCP tools when required to coordinate their invocation with GUI actions. On the other hand, among agentic frameworks, the best performance reaches only 51.6% (GPT-5). This reveals that current models struggle to effectively orchestrate between GUI actions and external tool invocations.

#### 4.4 In-depth Analysis

Table 4 presents a comprehensive analysis of agent behavior beyond task success rates. Several notable patterns emerge from examining execution efficiency and interaction quality.

**Execution Efficiency** Among all evaluated models, Gemini-3-Pro + UI-Ins-7B achieves the lowest average completion steps (24.2), followed by Claude-4.5-Sonnet + UI-Ins-7B (26.6) and GPT-5 + UI-Ins-7B (27.8). Interestingly, Doubao-1.5-UI-TARS and GUI-Owl-7B also exhibit relatively low step counts (20.9 and 20.6 respectively), but this is largely attributable to their lower success rates—these models often terminate early due to failure

Table 4: Detailed metrics comparison on MobileWorld under maximum 50 steps. We report Success Rate (SR), Average Completion Steps (Steps), Average User Queries (Queries), User Interaction Quality (UIQ), and Average MCP Calls (MCP). **Bold** indicates the best result and underline indicates the second best.

Model	SR (%)	Steps	Queries	UIQ	MCP
Claude-4.5-Sonnet	<u>43.8</u>	<u>26.6</u>	<u>0.76</u>	<u>0.25</u>	1.91
Gemini-3-Pro	46.3	<b>24.2</b>	0.36	0.19	<b>2.63</b>
GPT-5	<b>51.7</b>	27.8	<b>1.11</b>	<b>0.40</b>	<u>2.23</u>
GUI-Owl-7B	4.5	<u>20.6</u>	-	-	-
GUI-Owl-32B	5.5	24.0	-	-	-
UI-Venus-7B	5.5	26.7	-	-	-
UI-Venus-72B	10.4	34.2	-	-	-
Qwen3-VL-8B	5.5	24.8	0.04	0.00	2.32
Qwen3-VL-32B	9.0	27.1	0.00	0.00	3.84
Qwen3-VL-235A22B	9.5	26.9	0.00	0.00	2.38
GELab-Zero-4B	10.9	29.9	0.37	0.02	-
Doubao-1.5-UI-TARS	20.9	20.9	1.22	0.13	-

rather than efficient task completion. In contrast, UI-Venus-72B requires the highest average steps (34.2) while achieving only 10.4% success rate, indicating inefficient behavior.

**User Interaction Quality** The Average Queries and UIQ metrics jointly reveal how models manage agent-user interaction tasks. GPT-5 demonstrates the most effective clarification behavior, with an average of 1.11 queries per interaction task — corresponding to its highest user-interaction success rate (62.2%) and UIQ (0.40). Claude-4.5-Sonnet follows, averaging 0.76 queries and achieving a UIQ of 0.25. In contrast, Gemini-3-Pro got a lower UIQ (0.19) and asks very few queries on average (0.36), even though it performs well on GUI-only tasks (55.6%). This suggests it underutilizes user clarification even when beneficial. Notably, UIQ scores are consistently lower than User-Interaction SR across all models, indicating the presence of unnecessary ask\_user operations that inflate query counts without contributing to task success. In particular for Doubao-1.5-UI-TARS, a User-Interaction SR of 32.4% was achieved with 1.22 average queries, but the model obtains a lower UIQ of only 0.13. This gap suggests that Doubao-1.5-UI-TARS frequently issues redundant or ineffective clarification requests.

**MCP Tool Integration** For MCP-augmented tasks, Gemini-3-Pro leads in average tool invocations (2.63 calls per task), followed by GPT-5 (2.23) and Claude-4.5-Sonnet (1.91). This higher tool utilization correlates with stronger MCP SR

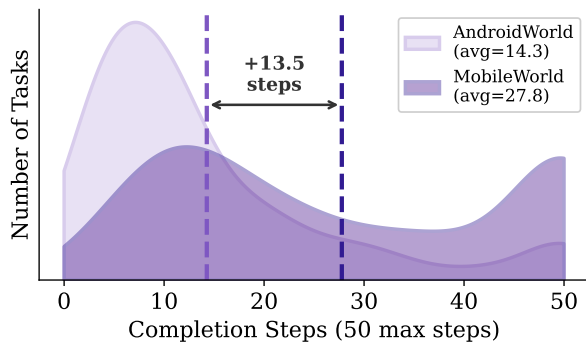


Figure 4: Comparison of completion steps between AndroidWorld and MobileWorld.

performance, indicating that successful MCP task completion requires agents to actively and appropriately leverage external tools. The average MCP Tool calls of Qwen3-VL (2.32 ~ 3.84) is comparable to that of the aforementioned models. However, due to inaccuracies in generated tool names and arguments, a large proportion of tool invocation failed, leading to a significantly lower MCP success rate (0.0 ~ 5.4). Other end-to-end models universally fail to invoke MCP tools, as they lack the architectural support for tool integration.

These findings highlight that effective mobile agents require not only strong task completion capabilities but also appropriate utilization of interactive clarification and external tool integration, where current end-to-end approaches show significant room for improvement.

**Completion Step Comparison with AndroidWorld** We compared the task completion steps with 50 max steps under the GPT-5 agentic framework on MobileWorld and AndroidWorld, respectively. As shown in Figure 4, MobileWorld exhibits a significantly higher complexity compared to AndroidWorld in terms of task length and step distribution. While AndroidWorld tasks are predominantly completed within 15 steps (with an average of 14.3 steps), MobileWorld tasks require substantially more actions, with 27.8 average steps. The distribution of MobileWorld tasks is notably right-skewed, with a large proportion requiring over 20 steps, reflecting its focus on long-horizon workflows involving GUI interactions, user clarification, and MCP tool calls.

**Case Study** To better understand the limitations of current GUI agents, we manually inspect failed tasks across evaluated models and categorize representative failure cases. Our case study in Sec-

tion K reveals five key challenges faced by current GUI agent models. (1) *Ambiguity detection*. Without the ability to query the user, agents frequently hallucinate missing arguments rather than requesting clarification—e.g., silently assuming an arbitrary hometown when asked to compute a driving distance, which produces a plausible-looking but incorrect result. (2) *Long-context management for MCP tool invocation*. MCP tool responses can easily overflow the agent’s context window; a PDF-extraction tool, for instance, may return a 20k-token document that buries the few target values and causes the agent to extract the wrong numbers. (3) *Long-term memory tracking*. In repetitive multi-step workflows, agents lose track of which sub-tasks have already been completed. In a bulk file-renaming task, the agent repeatedly re-processes previously renamed files, entering a destructive loop that corrupts the final file state. (4) *Complex logic reasoning*. Tasks requiring multi-step symbolic or numerical reasoning—such as identifying the top k most expensive items in a shopping cart and summing their prices—routinely fail due to mis-ranking, mis-reading prices, or arithmetic slips during aggregation. (5) *Temporal-spatial context awareness*. Agents rarely ground their actions in the device’s real-world time or location; when asked to “schedule a lunch tomorrow,” most models hallucinate a date instead of consulting the system clock, resulting in mis-scheduled events. More details can be found in Section L.

## 5 Conclusion

We introduce MobileWorld, a challenging benchmark for evaluating mobile GUI agents that addresses the saturation of existing benchmarks. MobileWorld advances mobile agent evaluation through complex workflows, agent-user interaction tasks that assess ambiguity recognition, and MCP-augmented tasks involving GUI operations and external tools. Built on production-grade, open-source apps in a containerized environment, it supports deterministic evaluation via multiple verification methods. Experiments show SOTA agents achieve only 51.7% (agentic framework) and 20.9% (end-to-end) SR, with further drops on interaction and MCP tasks, exposing critical gaps in ambiguity handling, dialogue, and hybrid planning. We hope MobileWorld will serve as a rigorous testbed to drive progress toward mobile agents capable of handling real-world automation complexity.

## Limitations

We identify the following limitations of our work, which may not be exhaustive but represent our best effort to acknowledge potential constraints.

**Simulated User Agent** Agent-user interaction relies on LLM-simulated users (GPT-4.1) rather than real humans, which may not capture the full variability of human communication (e.g., impatience, ambiguous responses, unexpected follow-ups). However, this trade-off is necessary to ensure reproducible and scalable evaluation.

**Open-Source Application Substitutes** We use open-source alternatives (Mattermost for Slack, Mastodon for X/Twitter) to enable deterministic evaluation. These may differ from commercial apps in UI complexity and edge-case behaviors, but this approach is essential for full backend access and reproducibility.

**Benchmark Scope** Our benchmark comprises 201 tasks spanning 20 mobile applications and therefore cannot exhaustively cover the full diversity of real-world mobile scenarios. Certain application domains may be underrepresented, and all tasks are currently limited to English and Chinese. Expanding domain diversity and multilingual coverage remains an important direction for future work, though it is constrained by the high cost of expert annotation.

**Emulator-Based Evaluation** Experiments use Android Virtual Devices rather than physical devices, which may differ in rendering latency, touch response, and hardware-specific features. This trade-off enables containerized, reproducible deployment across diverse host systems.

**Evaluation Design** Binary success metrics do not capture partial completion, and temperature 0.0 may differ from real deployments. These choices prioritize determinism and fair model comparison over fine-grained assessment.

## References

- Anthropic. 2024. What is the model context protocol (mcp)? <https://modelcontextprotocol.io/docs/getting-started/intro>.
- Anthropic. 2025. Claude Sonnet 4. <https://www.anthropic.com/claude/sonnet>. [Accessed 31-08-2025].

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025.  $\tau^2$ -bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Rogério Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, and 1 others. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149*.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, and 1 others. 2024. Spabench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*.

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, and 1 others. 2025. Ui-ins: Enhancing gui grounding with multi-perspective instruction-as-reasoning. *arXiv preprint arXiv:2510.20286*.

China Mobile Jiutian Team. 2025. *JT-GUIAgent: An advanced gui agent framework with planner-grounder architecture*. GitHub repository.

Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Liu Jianfeng Liu Jianfeng, Ang Li, Jian Luan, Bin Wang, Rui Yan, and 1 others. 2024. Mobilebench: An evaluation benchmark for llm-based mobile agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8813–8831.

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*.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. 2025. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. *arXiv preprint arXiv:2510.24563*.

- Adam Jones and Conor Kelly. 2024. [Code execution with MCP: Building more efficient agents](#). Anthropic Engineering Blog. Accessed: 2024-12-XX.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. 2025. [Mcp-universe: Benchmarking large language models with real-world model context protocol servers](#). *Preprint*, arXiv:2508.14704.
- OpenAI. 2025. [Introducing gpt-5](#). Technical report, OpenAI.
- Qwen Team. 2025. [Qwen3-vl: The multimodal large language model series](#). GitHub repository.
- Christopher Rawles, Sarah Clinckemahill, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyi Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.
- Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728.
- ByteDance Seed. 2025. [Ui-tars-1.5](#). <https://seed-tars.com/1.5>.
- GELab Team. 2025. [Gelab-zero: An advanced mobile agent inference system](#).
- Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. [Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning](#). *arXiv preprint arXiv:2509.02544*.
- Wenhao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, and Siheng Chen. 2025b. [Mcp-flow: Facilitating llm agents to master real-world, diverse and scaling mcp tools](#). *Preprint*, arXiv:2510.24284.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. [Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments](#). *Advances in Neural Information Processing Systems*, 37:52040–52094.
- Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025. [Androidlab: Training and systematic benchmarking of android autonomous agents](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166.
- Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, Shiliang Yang, Zhirui Wang, Brian Li, Kang An, Chenyang Li, Lei Lei, Mengmeng Duan, Danxun Liang, Guodong Liu, and 78 others. 2025a. [Step-gui technical report](#). *Preprint*, arXiv:2512.15431.
- Yunhe Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xinge Wang, Xin Yuan, Xu Han, and 1 others. 2025b. [Mcp-world: A unified benchmarking testbed for api, gui, and hybrid computer use agents](#). *arXiv preprint arXiv:2506.07672*.
- Leyang Yang, Ziwei Wang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. 2025. [Probench: Benchmarking gui agents with accurate process information](#). *arXiv preprint arXiv:2511.09157*.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024.  [\$\tau\$ -bench: A benchmark for tool-agent-user interaction in real-world domains](#). *arXiv preprint arXiv:2406.12045*.
- Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025. [Mobile-agent-v3: Fundamental agents for gui automation](#). *arXiv preprint arXiv:2508.15144*.
- Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and 1 others. 2025. [Mai-ui technical report: Real-world centric foundation gui agents](#). *arXiv preprint arXiv:2512.22047*.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. [Webarena: A realistic web environment for building autonomous agents](#). *arXiv preprint arXiv:2307.13854*.

## A Action Space

The action space supported by MobileWorld encompasses standard mobile GUI operations: *click*, *double\_tap*, *long\_press*, *drag*, *input\_text*, *scroll*, and navigation commands (*navigate\_home*, *navigate\_back*), alongside auxiliary actions: *answer* for responding to queries, *wait* for screen updates, and *status* to signal task completion or infeasibility. It is further augmented with *ask\_user* to request clarifications from a user when task instructions lack necessary context. Additionally, agents can leverage external functionalities through *mcp\_call*. The complete actions are listed in [Table 5](#).

## B MCP Tools

Agents can invoke external tools via the *mcp\_call* action. We curate a collection of popular MCP servers from the Bailian platform, spanning diverse domains including geospatial navigation (Amap), code repository analysis (GitHub), document processing (Jina), financial data retrieval (StockStar), and scholarly literature fetching (arXiv). These 61 tools enable functionalities that significantly exceed the capabilities of standard GUI operations, such as retrieving commit histories or querying real-time stock data. The complete tool catalog is provided in [Table 6](#).

## C Experimental Setup Details

To enable consistent comparison, we develop a planner-executor agentic framework ([China Mobile Jiutian Team, 2025](#)) where general-purpose LMMs (e.g., GPT-5, Claude-4.5-Sonnet) serve as the planner, and UI-Ins-7B ([Chen et al., 2025](#)) acts as the grounding executor. We extend the action space with *ask\_user* for user clarification and *mcp\_call* for external tool invocation (details in [Section D](#)). For end-to-end GUI agent models, including Doubao-1.5-UI-TARS, GUI-Owl, UI-Venus, GELab-Zero and Qwen3-VL, we adapt their official implementations to MobileWorld.

All models are evaluated with a maximum of 50 steps using metrics defined in [Section 4.1](#). Agentic frameworks and Qwen3-VL models are evaluated on all task categories. MCP tools are integrated into the system prompt following the tool call format for agentic frameworks, while Qwen3-VL models adopt the OSWorld-MCP format ([Jia et al., 2025](#)). Additionally, we augment the Qwen3-VL action space with *ask\_user* action to enable evaluation on

agent-user interaction tasks. Other end-to-end models are evaluated only on compatible tasks: MCP-augmented tasks are excluded due to lack of tool invocation support, and agent-user interaction tasks are excluded for models without user query actions (e.g., GUI-Owl, UI-Venus).

Open-source models are deployed using vLLM ([Kwon et al., 2023](#)) on a server with 8×NVIDIA H20 GPUs. Proprietary models including GPT-5, Claude-4.5-Sonnet, Gemini-3-Pro, and Doubao-1.5-UI-TARS are accessed through their official API. The temperature is set to 0.0. The user agent for agent-user interaction tasks is implemented using GPT-4.1.

## D Planner-Executor Agentic Framework Details

To accommodate the unique requirements of MobileWorld, we develop a planner-executor agentic framework consisting of two core components: a *planner* and a *grounding executor* ([China Mobile Jiutian Team, 2025](#)).

The planner is responsible for interpreting the task instruction, analyzing the current screenshot along with the interaction history, and deciding the next action (e.g., click, swipe, wait, or type). When the planned action involves clicking (e.g., click or long\_press), the planner generates a natural language description of the target UI element instead of outputting coordinates directly. This description is then passed to the grounding executor, which takes the current screenshot as input and predicts the precise pixel coordinates to perform the click or long press. This design allows any general-purpose VLM to be integrated into our framework without requiring pixel-level grounding capabilities.

To support agent-user interaction and MCP-augmented tasks, we extend the action space with two special operations:

**Ask User Action** When the planner issues an *ask\_user* action, the system forwards the query to the LLM-simulated user agent. The user agent’s textual response—along with the current screenshot—is appended to the interaction history, enabling the agent to acquire missing information through clarification dialogues.

**MCP Call Action** For MCP-augmented tasks, we preload the specifications of the involved MCP tools into the system prompt. When the planner

Table 5: Complete action space supported by MobileWorld.

Action	Parameters	Description
<i>GUI Operations</i>		
click	x, y	Tap at the specified coordinates
double_tap	x, y	Double-tap at the specified coordinates
long_press	x, y	Long-press at the specified coordinates
drag	start_x, start_y, end_x, end_y	Drag from start to end coordinates
input_text	text	Type text into the focused field
scroll	direction	Scroll in the specified direction (up/down/left/right)
<i>Navigation</i>		
navigate_home	—	Return to the home screen
navigate_back	—	Navigate to the previous screen
keyboard_enter	—	Press the enter key
wait	—	Wait for screen content to update
<i>Task Control</i>		
answer	text	Provide a textual response to the user (for IR tasks)
status	goal_status	Mark task as complete or infeasible
<i>Extended Actions</i>		
ask_user	text	Request clarification from the user
mcp_call	tool_name, params	Invoke an MCP tool with specified parameters

triggers `mcp_call`, the invocation request is sent to the MCP server, which returns a structured output. This result, along with the current screenshot, is injected into the interaction history as the system’s observation for that step.

This architecture enables the seamless unification of GUI manipulation, multi-turn user interaction, and external tool use within a single closed-loop decision process.

## D.1 Planner

The planner prompt is shown in the *Planner System Prompt* code block. We define an action space that aligns with the one implemented in MobileWorld. In addition, we integrate the `ask_user` action to enable clarification of ambiguous instructions. Furthermore, when a task is tagged as MCP-augmented, we dynamically inject the specifications of the relevant MCP tools into the prompt.

## D.2 Grounding Executor

For the grounding executor in our agentic framework, we design it to take click-related instructions generated by the upstream planner and output precise screen coordinates for clicking. The prompt used for this module is shown in the *Grounding Executor System Prompt* code block.

## E APP Information

### E.1 APP List

Table 7 lists the GUI applications included in MobileWorld, along with the number of tasks associated with each. These apps cover a broad range of everyday mobile usage scenarios, including communication (e.g., Mail, Messages), productivity (e.g., Calendar, DocsReader), navigation (Google Map), social interaction (Mastodon, Mattermost), and e-commerce (Taodian). To ensure reproducibility and enable deterministic evaluation, we prioritize open-source or publicly available implementations that closely mimic the functionality of popular commercial applications, such as Taobao (via Taodian), Gmail (via Mail), and Slack (via Mattermost). This design choice allows us to maintain full control over the environment while still providing realistic, real-world-aligned tasks. By using these open-source equivalents, we strike a balance between ecological validity and rigorous, verifiable assessment, enabling fair and repeatable benchmarking of mobile agents.

### E.2 Open-source APP Environment Construction

To ensure reproducibility and enable deterministic evaluation, we construct self-hosted backend environments for key applications. This section details

Table 6: MCP tools integrated in MobileWorld. These tools provide advanced capabilities beyond standard GUI operations, enabling agents to complete complex cross-domain tasks.

MCP Server	# Tools	Description
Amap Maps	15	Provides comprehensive geospatial services including geocoding, reverse geocoding, IP location, weather queries, and multi-modal route planning (cycling, walking, driving, public transit). Supports distance measurement and location-based search (keyword, nearby, and detail queries).
GitHub	26	Official GitHub integration offering advanced automation and interaction capabilities for developers. Enables repository querying, commit history retrieval, issue tracking, and code analysis.
Jina AI	3	Document processing and search capabilities powered by Jina AI, supporting multi-modal content analysis and retrieval tasks.
Stockstar	16	Financial intelligence service providing comprehensive data for A-share and Hong Kong stocks. Includes fundamental data, derived metrics, financial analysis, and business model evaluation.
arXiv	4	Academic paper search and retrieval from the arXiv repository, enabling agents to access and process scholarly literature.

Table 7: List of MobileWorld GUI apps and number of tasks for each one.

App	Description	Comparable Commercial App	#Tasks
Calendar	Manage events and schedules	Google Calendar	30
Camera	Take photos and videos	-	3
Chrome	Web browser for internet browsing	-	15
Clock	Alarms, timers, and world clock	-	7
Contacts	Manage contact information	-	11
Doreader	View and read documents	Adobe Reader	10
Files	File manager for storage	-	32
Gallery	View and manage photos	-	11
Mail	Email client for messaging	Gmail	61
Google Map	Navigation and location services	-	1
Mastodon	Decentralized social network	Twitter/X	41
Mattermost	Team collaboration and messaging	Slack	18
Messages	SMS and chat messaging	-	41
Settings	System configuration	-	7
Taodian	E-commerce shopping platform	Taobao	16

the implementation strategies for our four primary open-source applications.

**Mattermost** We build the Mattermost environment based on the official Docker deployment repository<sup>2</sup>, which provides a Docker Compose-based setup for the Mattermost service. Initial chat histories are manually generated following the official import guide using the Mattermost CLI tool. All backend data, including PostgreSQL database contents and file storage, are consolidated into a single directory. To ensure consistent initialization across task executions, we snapshot this directory and restore it at the beginning of each evaluation by copying the contents to the designated location before launching the Docker Compose stack. This approach guarantees that each task starts from an identical initial state. Additionally, we develop aux-

<sup>2</sup><https://github.com/mattermost/docker>

iliary tools that leverage the CLI to dynamically generate new chat messages during initialization when required by specific tasks. For evaluation, we directly query the PostgreSQL database to verify task outcomes, such as the presence of target messages or the creation of chat groups.

**Mastodon** The Mastodon environment is constructed using the official Docker setup from the Mastodon repository<sup>3</sup>. We manually create initial posts and user accounts within the platform, then capture a snapshot of the complete backend state, including the PostgreSQL database and media storage. Similar to Mattermost, we employ a snapshot-and-restore strategy: the backend data directory is preserved and restored before each task execution to ensure reproducibility. Since the original Mastodon Android client enforces HTTPS con-

<sup>3</sup><https://github.com/mastodon/mastodon>

nections, we apply minimal modifications to the client application to enable communication with the locally hosted HTTP backend service. Task verification is performed by directly querying the PostgreSQL database to validate outcomes such as new post creation or user interactions.

**Mail App** We develop the Mail application based on a pure-frontend React Native Gmail clone<sup>4</sup>. We resolve multiple compilation errors that arose with recent Android SDK versions and extend the application with critical functionalities, including attachment selection, email search capabilities, and email composition via the Android Share interface. To enable programmatic evaluation, we further augment the app with a callback mechanism that persists email sending events to a local file. When an email is sent, relevant metadata (recipient, subject, body, attachments) is recorded and subsequently retrieved by the evaluation function to verify task completion.

**Taodian App** The Taodian e-commerce application is adapted from the Mall4Uni full-stack platform<sup>5</sup>. To simplify deployment and eliminate external dependencies, we replace the original backend service with a lightweight file-based mock server. Product catalogs are manually curated from publicly available online sources. We enhance the user interface with improved theming and implement additional features such as SMS-based login and a refined shopping cart management interface. Similar to the Mail app, we instrument the application with callback hooks at critical interaction points (e.g., checkout button clicks) to capture shopping cart contents and shipping details. This information is transmitted to a callback server and subsequently used for automated task validation.

This comprehensive construction approach ensures that all applications are fully self-contained, reproducible, and amenable to automated evaluation while closely approximating real-world mobile application functionality.

## F Task Evaluation Mode

Table 8 lists some examples of tasks and demonstrate how conduct deterministic evaluation on them. This verification system ensures that task outcomes are evaluated deterministically, eliminating

<sup>4</sup>[https://github.com/PrincewillIroka/gmail\\_clone](https://github.com/PrincewillIroka/gmail_clone)

<sup>5</sup><https://gitee.com/gz-yami/mall4uni>

the evaluation noise inherent in MLLM-as-a-judge approaches used by prior work.

### F.1 Detailed Evaluation Pipeline Examples

To provide a clearer understanding of each evaluation method, we present concrete implementation examples below.

**Backend Database Verification** For tasks involving self-hosted applications, we directly query the backend PostgreSQL database to verify expected state changes. For example, in the task “*Create a channel on Mattermost called ‘reading’ for paper reading. Add everyone to the channel and greet everyone with a welcome message,*” the evaluator connects to the Mattermost PostgreSQL database and executes SQL queries to check: (1) whether a channel named reading exists (`SELECT * FROM channels WHERE LOWER(name) = LOWER('reading')`), (2) whether all 11 expected members are present in the channel, and (3) whether the latest message in the channel matches the expected welcome greeting (`SELECT * FROM posts ORDER BY createat DESC`). The same pattern applies to Mastodon tasks, where we verify post creation, replies, and user interactions through the Mastodon PostgreSQL backend.

**Local Storage Inspection** For tasks modifying device-level settings or system app state, we leverage ADB (Android Debug Bridge) shell commands on the rooted emulator to inspect local storage. For example, in the task “*Turn on device flight mode,*” the evaluator executes `adb shell settings get global airplane_mode_on` and checks whether the returned value is 1. Similarly, for SMS verification tasks, we query the SMS content provider (`content query -uri content://sms/sent`) to inspect sent messages; for alarm tasks, we query the alarm database to verify alarm time, repeat settings, and ringtone configuration.

**Application Callbacks** For customized applications (Mail and Taodian), we instrument the app code with callback hooks that persist interaction events to local files. For example, in the Taodian e-commerce task “*Help me order a temporary tat-too set on Taodian,*” when the user reaches the checkout page, the app writes a JSON callback file containing the order details (product name, quantity, price). The evaluator reads this callback file and verifies that the product name matches the expected item. For the Mail app, when an email is

Table 8: The example tasks with their evaluation modes and corresponding verification logics.

Task	Eval Mode	Eval Logic
I want to drive to Tianjin. Please check the driving distance in kilometers. Response only one integer number. No other text.	Textual Answering Verification	Compare the agent answer text with the ground-truth distance.
Reply to the toot of gourmet user about Greek food Moussaka, and the reply content should be 'Nice sharing, i love it'.	Backend Database Verification	Fetch the reply content from the Mastodon backend database and compare with the ground-truth content.
Set a weekend alarm for 8:25 a.m. with the ringtone "beebeep" and vibration off.	Local Storage Inspection	Leverage adb command to query the local storage of Alarm app.
I want to remove some electronic products in the shopping cart of the TaoDian app.	Application Callbacks	Implement callback APIs that capture the cart item changes in Taodiao app and persist in a local file for evaluation.

sent, the app persists metadata (recipient, subject, body, attachments) to a JSON file on the device’s local storage, which the evaluator retrieves via ADB and validates against the expected content.

**Textual Answer Verification** For information retrieval tasks, we compare the agent’s textual response against ground-truth answers using pattern matching. For example, in the task “*How many days of conference meetings did I schedule in October?*” (expected answer: 12), the evaluator first applies noise-cleaning regular expressions to remove potential false positives from date mentions (e.g., “October 12”), time values (e.g., “12:00”), and ordinal patterns (e.g., “12th”). It then checks whether the expected answer 12 appears as a standalone token in the cleaned response using word-boundary matching.

## G Per-Task Difficulty Analysis

To provide a more granular understanding of benchmark difficulty, we analyze the per-task success rates across three frontier agentic models: GPT-5, Claude-4.5-Sonnet, and Gemini-3-Pro (all using UI-Ins-7B as the grounding executor). For each of the 198 evaluated tasks, we compute how many of the three models successfully complete it, yielding a four-level difficulty categorization.

**Difficulty Distribution** As shown in Table 9, the benchmark exhibits a roughly bimodal difficulty distribution. Approximately 31.3% of tasks (62 tasks) are unsolved by all three models, representing genuinely hard challenges that exceed the capabilities of current frontier models. Conversely, 29.8% of tasks (59 tasks) are solved by all three models, serving as accessible baseline tasks. The remaining 38.9% of tasks (77 tasks) show differen-

tial difficulty across models, providing discriminative power for model comparison.

Table 9: Difficulty distribution of MobileWorld tasks based on how many of the three agentic models (GPT-5, Claude-4.5-Sonnet, Gemini-3-Pro) successfully complete each task.

Solved by	# Tasks	Percentage
0 models (Hardest)	62	31.3%
1 model	41	20.7%
2 models	36	18.2%
3 models (Easiest)	59	29.8%

**Impact of Agent-User Interaction** We also analyze difficulty differences between tasks requiring agent-user interaction and standard tasks. Among the 45 interaction tasks, GPT-5 achieves 64.4%, while Claude-4.5-Sonnet and Gemini-3-Pro achieve 40.0% and 24.4%, respectively. Notably, Gemini-3-Pro performs comparably to the other models on non-interaction tasks (53.6%) but drops significantly on interaction tasks, suggesting that the ability to recognize ambiguity and initiate clarification is a distinct capability that varies substantially across models. Furthermore, we conduct an ablation study in which agents are evaluated on interaction tasks *without* access to the ask\_user action: all models achieve a 0% success rate, confirming that these tasks genuinely require agent-initiated clarification and cannot be completed through GUI manipulation alone.

## H Deployment and Reproducibility Details

**Simplified Deployment** While MobileWorld employs a Docker-in-Docker architecture to encapsulate the Android Virtual Device and self-hosted

backends, we have invested substantial effort in minimizing deployment friction. The entire evaluation environment is consolidated into a single Docker image. We provide a command-line tool that automatically verifies whether the host hardware meets the system requirements (e.g., KVM support, available memory) and launches the full environment with a single command. This reduces the setup process to pulling the image and running one command, regardless of the host system configuration.

**On-Demand Service Startup** To minimize resource consumption, backend services for individual applications (e.g., Mattermost PostgreSQL, Mastodon services) are started on demand — only when evaluating tasks targeting those specific applications — rather than launching all services simultaneously. This design significantly reduces the memory and CPU footprint during evaluation, making it feasible to run the benchmark on machines with moderate resources.

## I Simulated User Agent Design

The LLM-simulated user agent is a critical component for evaluating agent-user interaction tasks. We address key design considerations below.

**Run-to-Run Stability** The simulated user is configured with a fixed system prompt and deterministic hidden context for each task, using a low temperature setting (0.0) to minimize response variability. The hidden context contains exactly the information intentionally omitted from the task instruction, ensuring consistent and predictable responses across runs.

**Guardrails Against Prompt Injection** The simulated user’s system prompt explicitly restricts it to answering only task-relevant clarification questions and prohibits disclosure of information beyond the specified hidden context. If the GUI agent asks irrelevant or off-task questions, the user agent is instructed to decline assistance, preventing information leakage that could artificially inflate task success rates.

## J MCP Determinism and Reproducibility

MCP-augmented tasks that query external services (e.g., GitHub, arXiv, StockStar) introduce potential reproducibility concerns due to the dynamic nature of these resources. We address this through a task-design-level strategy.

**Stable Content Selection** We deliberately construct MCP-augmented tasks that target stable, well-established content to minimize temporal variability. For example, arXiv queries reference classic, widely-cited papers whose metadata is unlikely to change. GitHub queries target mature, well-maintained repositories with stable commit histories and README content. This design choice ensures that task outcomes remain consistent across evaluation runs conducted at different times.

**Limitations and Future Directions** We currently rely on external MCP services without response caching or proxying. While these are popular, well-maintained services with low risk of disruption, this introduces a potential reproducibility risk if service endpoints change or become temporarily unavailable. We recognize that stronger determinism could be achieved by pinning specific content versions (e.g., GitHub commit SHAs), caching API responses within the container, or proxying MCP servers inside the Docker environment to serve pre-recorded responses. We plan to investigate these enhancements in future iterations of the benchmark.

## K Research Challenge Analysis

To better understand the limitations of current GUI agents, we manually inspect failed task trajectories across evaluated models and categorize representative cases. This analysis reveals several challenges that current agents must overcome to achieve reliable real-world performance.

### Challenge 1: Ambiguity Detection and User Engagement

When the model operates without the ability to query the user for clarification, it often generates plausible-sounding but factually incorrect or unsupported responses. This hallucination arises because the model attempts to infer missing or ambiguous information on its own, rather than seeking confirmation. In the example shown in [Figure 9](#), the task instruction is: *“I want to drive to Tianjin, China from my hometown. Please check the driving distance.”* The GUI agent opens Google Maps and correctly inputs the destination (Tianjin), but it hallucinates “Shanghai” as the hometown without asking the user to specify the departure city. This incorrect assumption leads to a wrong distance calculation and ultimately causes task failure. Allowing interactive clarification significantly reduces these risks by grounding the model’s ac-

tions in verified user intent.

**Challenge 2: MCP Tool Descriptions and Output Management** External tools invoked via MCP may return excessively long outputs that overwhelm the agent’s context window. As illustrated in [Figure 10](#), when extracting specific benchmark scores from an academic PDF to email a comparison summary, the MCP tool returns the entire 20k-token document as raw text. This floods the context with irrelevant content, preventing the agent from locating target information and causing incorrect extractions or downstream failures. Effective MCP integration requires content-aware retrieval strategies and context management mechanisms ([Jones and Kelly, 2024](#)).

**Challenge 3: Long-Term Memory and State Tracking** The GUI agent struggles to maintain awareness of actions it has already performed without long-term memory tracking mechanism. As illustrated in [Figure 11](#), the agent is instructed to rename all files in the Downloads folder with the prefix `bid_`, ordering them by creation date and renaming them uniformly as `bid_index.extension`. However, after successfully renaming several files, the agent forgot the processed files. In subsequent steps, it attempts to rename these renamed files again. This leads to repeated, conflicting modifications and ultimately results in an incorrect final state. The absence of a reliable memory mechanism to track completed subtasks causes the agent to fall into a destructive loop, highlighting a critical gap in its ability to handle multi-step operations in real-world environments.

**Challenge 4: Complex Logic Reasoning** The GUI agent exhibits limited capability in tasks requiring multi-step logical reasoning and accurate numerical calculation. For instance, in the task “Find the three most expensive items in the TaoDian app shopping cart and calculate their total price,” the agent must first scan all items in the cart, maintain a running record of the top three highest-priced products, and then sum their prices precisely. While the model attempts this process and outputs a final number, the result is incorrect, either because it fails to correctly identify the most expensive items (e.g., due to misreading prices or overlooking items) or because it makes arithmetic errors during summation. This reflects a broader weakness in handling structured reasoning and exact calculation, which are essential for reliable per-

formance in real-world e-commerce or financial tasks.

**Challenge 5: Temporal-Spatial Context Awareness** The model generally lacks awareness of real-world time and location unless these are explicitly stated in the prompt. In the task “I’ve received a lunch invitation via text message; please reply ‘OK’ and schedule a lunch event tomorrow” the agent is expected to determine today’s date — by observing system UI elements such as the desktop clock — and create a calendar entry for an appropriate upcoming lunchtime. Most tested agents either hallucinate an arbitrary date or fail to consult time. This inability to ground actions in the actual temporal context leads to incorrectly scheduled events and highlights a critical gap in the agent’s situational awareness. Reliable access to the real-time information on the device is essential for performing scheduling tasks accurately.

## L Case Study

In this section, we demonstrate various types of examples to illustrate the tasks in MobileWorld.

### L.1 GUI-Only Task Example: Task Completion

[Figure 5](#) demonstrates editing a Mastodon poll by removing the “USA” option and replacing “Brazil” with “Canada”, guided by precise click and input actions. The process involves navigating to the post, accessing the edit menu, deleting the unwanted option, and typing the new one.

### L.2 GUI-Only Task Example: Information Retrieval

[Figure 8](#) presents a real-time mobile task in which the user retrieves Beijing’s highest temperature for *today* using the Chrome browser. Starting from the home screen, the agent should launch Chrome, inputs the query “Beijing highest temperature today”, and immediately extracts the current forecast from Google’s dynamic weather widget. Both the execution and validation of the task are inherently real-time, ensuring that the final output (11°C) reflects the actual temperature condition on the day of execution.

### L.3 Agent-User Interaction Task Example

[Figure 6](#) highlights a task involving calendar event management with an ambiguous user instruction: “My schedule on 10/20 is a bit full, please remove a

few events.” The directive lacks specificity regarding which events to delete, making it inherently ambiguous. To resolve this uncertainty, the agent must engage in interactive clarification. In this scenario, the system identifies multiple events on October 20 and explicitly asks whether to delete only the occurrence of “Meet with Sam” or the entire recurring series. This interaction underscores the sensitivity and importance of user confirmation in critical operations such as event deletion, where incorrect actions could disrupt scheduling.

#### **L.4 MCP-Augmented Task Example**

The example shown in [Figure 7](#) demonstrates how a mobile GUI agent completes a complex task with the help of MCP tool invocation: the user requests the three most recent commits from the google-research/android\_world repository, formatted as “author: commit message,” and sent via email to a specified address. The agent first invokes an MCP tool to fetch the commit data, then opens the Mail app, taps “Compose”, and fills in the recipient, subject, and body with the retrieved information before successfully sending the email. The example highlights the agent’s ability to seamlessly integrate external API calls with precise UI interactions.

**# User Instruction**

**Edit my Mastodon poll about which country has the largest area by removing the option “USA” and changing the option “Brazil” to “Canada”.**

The sequence of screenshots illustrates the following steps:

- Click [162, 960]**: Home screen of the mobile app.
- Click [1019, 349]**: Tap on the poll post titled "Which country has the largest land area?".
- Click [648, 1073]**: Tap on the three-dot menu icon to open the edit options.
- Click [124, 748]**: Tap on the "Edit post" option.
- Click [411, 635]**: Tap on the "USA" option to remove it from the poll.
- Click [136, 1254]**: Tap on the "Brazil" option to edit it.
- LongPress [136, 1254]**: Long-press on the text "Brazil" to bring up the text editing menu.
- Type(Canada)**: Type the word "Canada" to replace "Brazil".
- Click [1029, 206]**: Tap on the checkmark icon to save the changes.
- Finish**: The final state of the poll with "Canada" as an option.

Figure 5: Task Completion Task Example

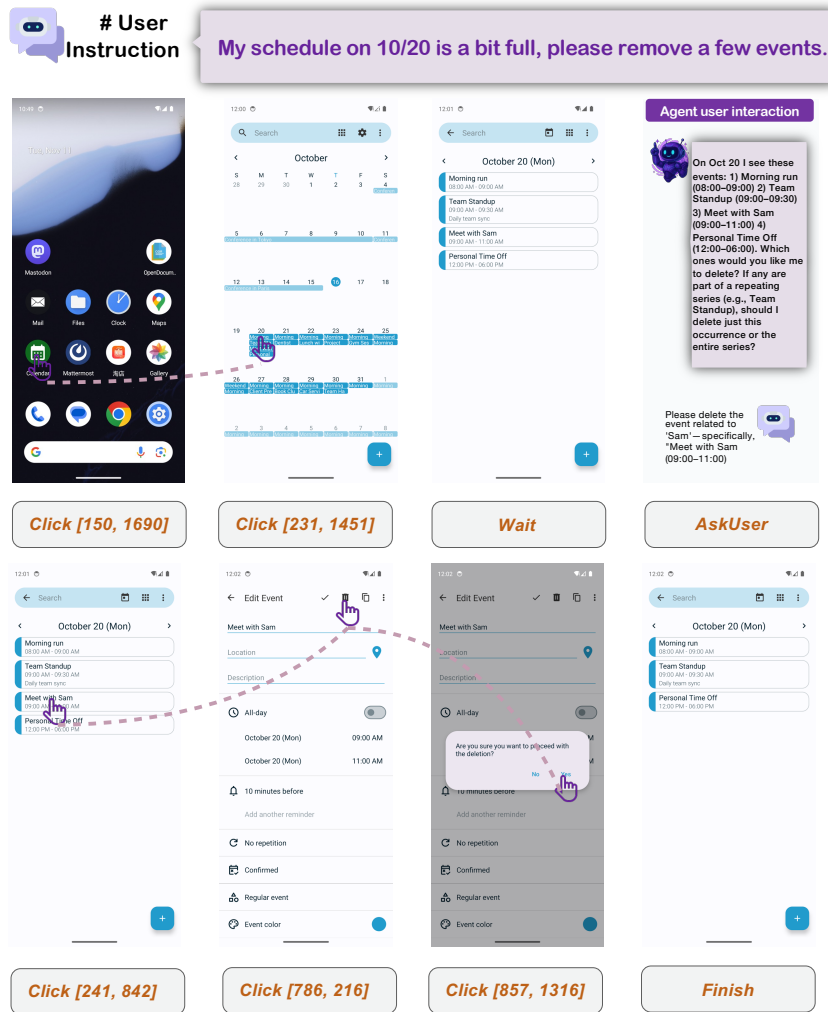


Figure 6: Agent-User Interaction Task Example

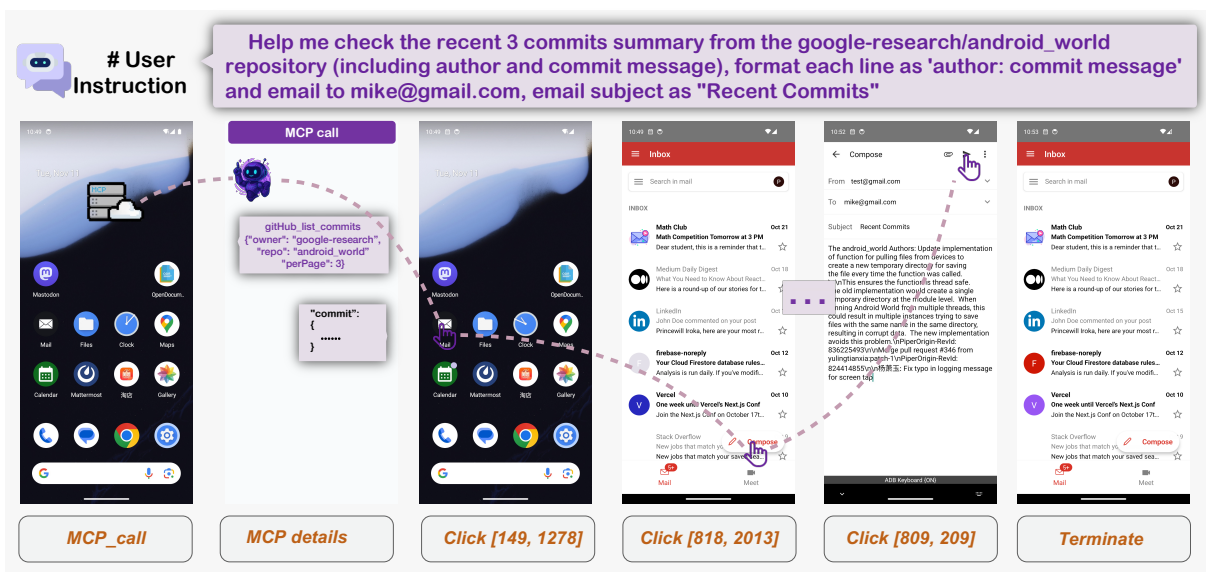


Figure 7: MCP-Augmented Task Example

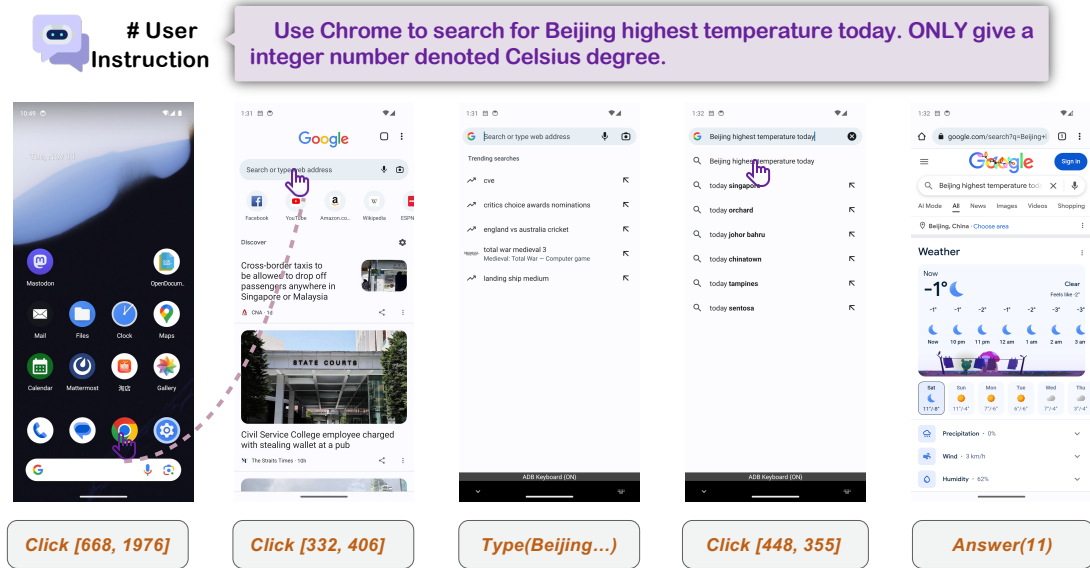


Figure 8: Information Retrieval Task Example

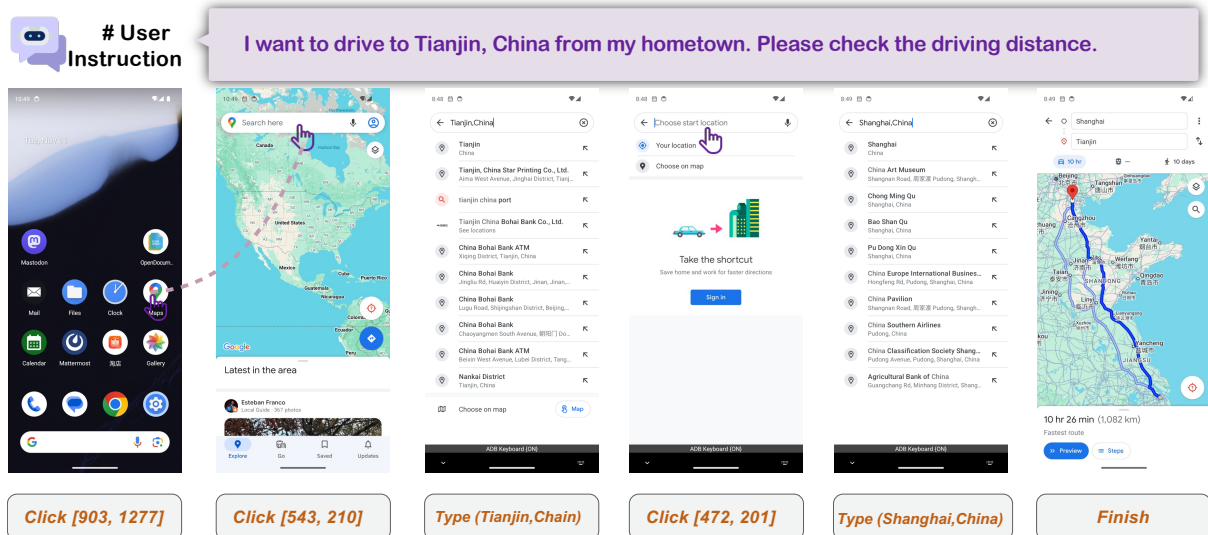


Figure 9: Hallucination without user clarification: A representative failure case showing how the mobile GUI agent hallucinates actions when faced with ambiguous scenarios that require user clarification.

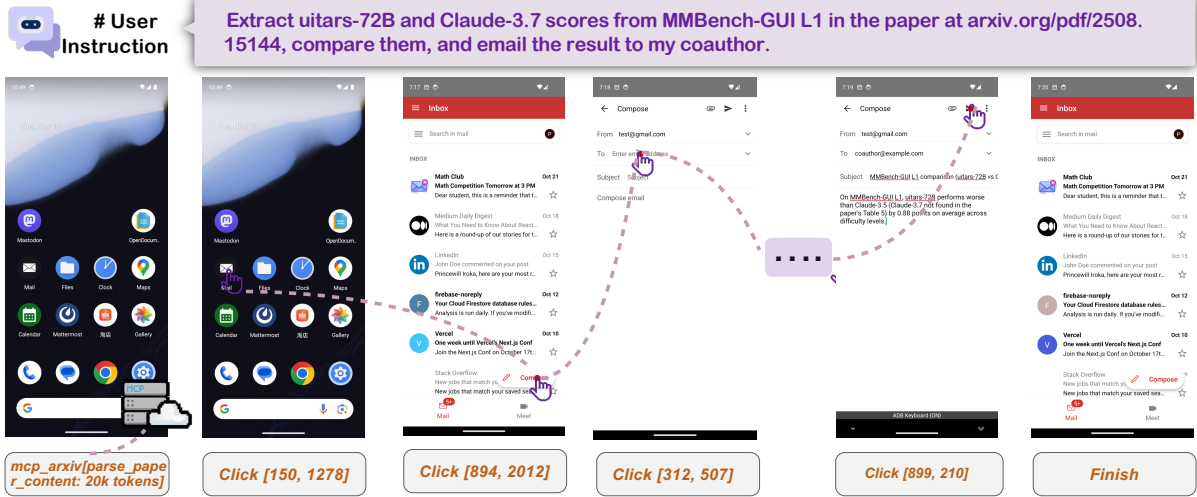


Figure 10: Context overflow from MCP tool responses: A failure case demonstrating ineffective MCP tool integration due to context management issues, where tool responses exceed the context window capacity.

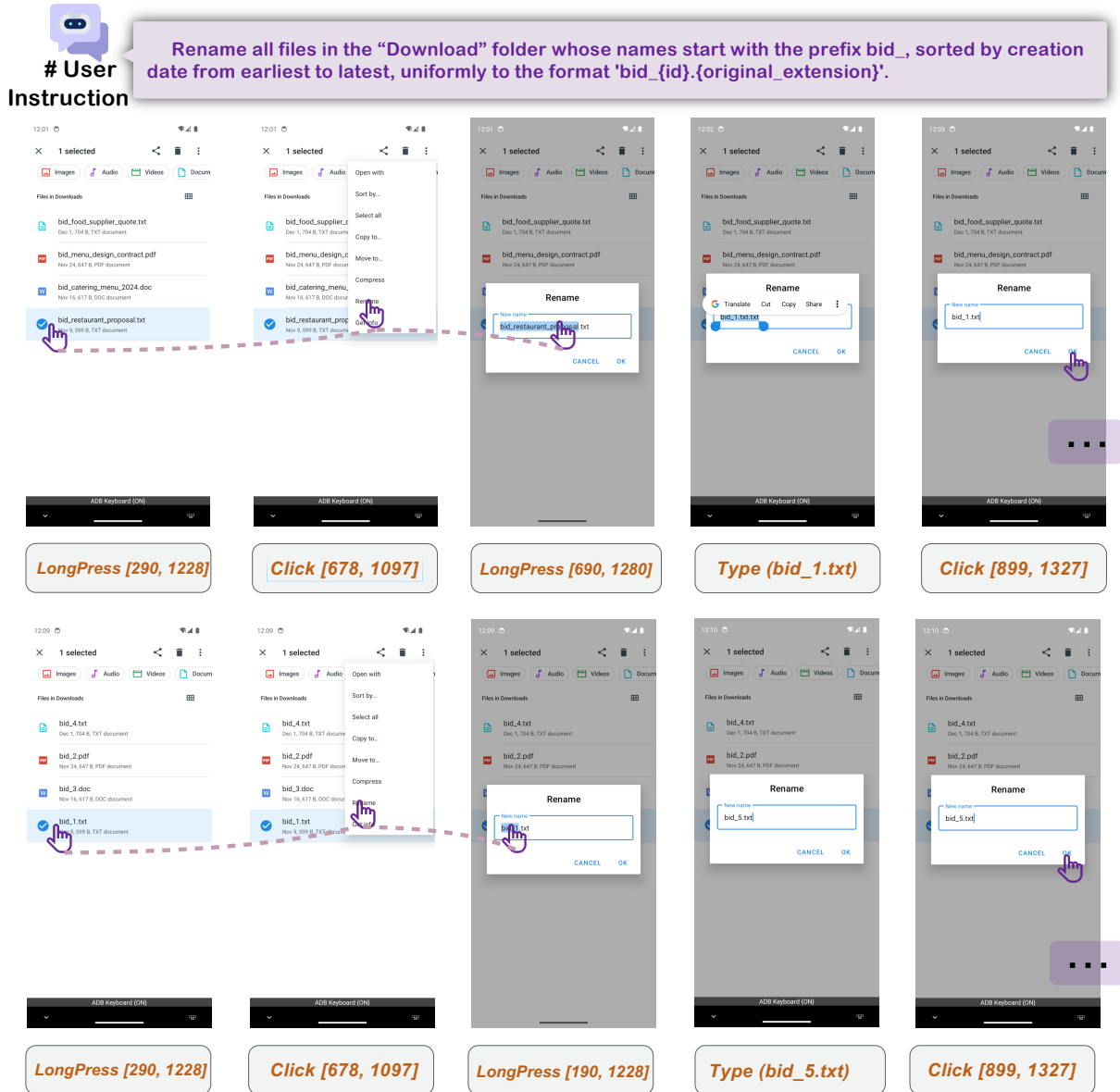


Figure 11: Lack of long-term memory: A representative case illustrating insufficient memory mechanisms for tracking multi-step operations, leading to failure in maintaining state across sequential actions.

## Planner System Prompt

### ## Role: Android Phone Operator AI

You are an AI that controls an Android phone to complete user requests. Your responsibilities:

- Answer questions by retrieving information from the phone.
- Perform tasks by executing precise actions.

## Action Framework. Respond with EXACT JSON format for one of these actions:

Action	Description	JSON Format Example
click	Tap visible element (describe clearly)	{ "action_type": "click", "target": "blue circle button at top-right" }
double_tap	Double-tap visible element (describe clearly)	{ "action_type": "double_tap", "target": "blue circle button at top-right" }
long_press	Long-press visible element (describe clearly)	{ "action_type": "long_press", "target": "message from John" }
drag	Drag from visible element to another visible element (describe both clearly)	{ "action_type": "drag", "target_start": "the start point", "target_end": "the end point" }
input_text	Type into field (includes clicking field, typing, and pressing enter)	{ "action_type": "input_text", "text": "Hello" }
answer	Respond to user	{ "action_type": "answer", "text": "It's 25 degrees today." }
navigate_home	Return to home screen	{ "action_type": "navigate_home" }
navigate_back	Navigate back	{ "action_type": "navigate_back" }
scroll	Scroll direction (up/down/left/right)	{ "action_type": "scroll", "direction": "down" }
status	Mark task as complete or infeasible	{ "action_type": "status", "goal_status": "complete" }
wait	Wait for screen to update	{ "action_type": "wait" }
ask_user	Ask user for information	{ "action_type": "ask_user", "text": "what is the exact requirements?" }
keyboard_enter	Press enter key	{ "action_type": "enter" }

### ## Execution Principles.

#### 1. Communication Rule:

- ALWAYS use 'answer' action to reply to users - never assume on-screen text is sufficient
- Please follow the user instruction strictly to answer the question, e.g., only return a single number, only return True/False, only return items separated by comma.
- NEVER use 'answer' action to indicate waiting or loading - use 'wait' action instead
- Note that answer will terminate the task immediately.

#### 2. Efficiency First:

- Choose simplest path to complete tasks
- If action fails twice, try alternatives (e.g., long\_press instead of click)

#### 3. Smart Navigation:

- Gather information when needed (e.g., open Calendar to check schedule)
- For scrolling:
  - Scroll direction is INVERSE to swipe (scroll down to see lower content)
  - If scroll fails, try opposite direction

#### 4. Text Operations:

- You MUST first click the input box to activate it before typing the text.
- For text manipulation:
  - (a) Long-press to select
  - (b) Use selection bar options (Copy/Paste/Select All)
  - (c) Delete by selecting then cutting

#### 5. Ask User:

- If you think you have no enough information to complete the task, you should use ask\_user action to ask the user to get more information.

### ## Decision Process.

1. Analyze goal, history, and current screen
2. Determine if task is already complete (use status if true)
3. If not, choose the most appropriate action to complete the task.
4. Output in exact format below, and ensure the Action is a valid JSON string:
5. The action output format is different for GUI actions and MCP tool actions. Note only one tool call is allowed in one action.

## Expected Output Format (Thought: and Action: are required):

Thought: [Analysis including reference to key steps/points when applicable]

Action: [Single JSON action]

### ## Output Format Example

for GUI actions:

```
Thought: I need to ... to complete the task.
Action: { "action_type": "type", "text": "What is weather like in San Francisco today?" }
```

for MCP tools:

Thought: I need to use the provided mcp tool to get the information...  
Action: { "action\_type": "mcp", "action\_json": tool\_args\_obj, "action\_name": "mcp\_tool\_name" }

**## Available MCP Tools**  
{ { tools } }

**## User Goal**  
{ { goal } }

#### Grounding Executor System Prompt

You are a GUI agent. You are given a task and your action history, with screenshots. You need to perform the next action to complete the task. ## Output Format Return a json object with function name and arguments within <tool\_call></tool\_call> XML tags:

```
<tool_call>
{"name": "grounding", "arguments": <args-json-object>}
</tool_call>
```

<args-json-object> represents the following item of the action space:

**## Action Space**  
{ "action": "click", "coordinate": [x, y] }