MOBICLI: BENCHMARKING CLI AND HYBRID GUI–CLI INTERACTION FOR MOBILE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mobile agents have largely been developed and evaluated through GUI-only interaction, overlooking command-line interfaces (CLIs) that can improve execution efficiency and support operations beyond GUI actions. To bridge this gap, we introduce MobiCLI, a benchmark for evaluating CLI and hybrid GUI–CLI interaction on mobile devices. We identify five representative mobile scenarios where CLI capabilities are particularly useful: Batch Operations, File Management, Computational Tasks, Data Processing, and System Operations, and construct 122 tasks across 18 applications. We evaluate strong general-purpose models and representative phone-use agents on MobiCLI, with the best model achieving only 55.7% success. Our experiments reveal three key findings. First, CLI access substantially improves both success and efficiency for strong general-purpose models, exposing the limitations of GUI-only interaction. Second, existing phone-use agents generally fail to effectively exploit CLI capabilities. Third, hybrid tasks are substantially harder than CLI tasks, highlighting the challenge of GUI–CLI coordination. To improve hybrid execution, we further introduce MobiMux, a mobile agent framework that combines task verification with continual experience learning. MobiMux improves success rates by up to 16.4 percentage points on general-purpose models and achieves the best overall performance on MobiCLI.

1 INTRODUCTION

As smartphones have become a primary interface for daily digital services, mobile agents that can follow natural-language instructions and autonomously complete tasks are becoming increasingly important (Wen et al., 2024; Wang et al., 2024a). Recent advances in multimodal language models have greatly improved their ability to understand mobile interfaces, reason about application states, and execute long sequences of actions (Zhang et al., 2025b; Xu et al., 2026; Venus Team et al., 2026). At the same time, a growing number of benchmarks have been developed to evaluate mobile agents in increasingly diverse and interactive environments (Liu et al., 2025).

However, most existing mobile agent benchmarks remain centered on GUI interaction (Xu et al., 2025; Rawles et al., 2025; Liu et al., 2025), and current phone-use agents are also typically designed around GUI-based action spaces (Yan et al., 2025a; Zhou et al., 2025; Xu et al., 2026). GUI-only interaction can impose substantial limitations on many mobile tasks. Batch and file operations often require long sequences of GUI actions (Liu et al., 2026), system states may be difficult to inspect or manipulate through GUI interfaces, and complex computation or data transformation may rely on unreliable internal model reasoning (Gao et al., 2023). CLI access provides a complementary programmatic interface that can efficiently compose repeated operations, directly access system states and files, and invoke tools for reliable data processing. However, CLI cannot replace GUI interaction, since many mobile tasks still depend on visual information or GUI-specific states (Kong et al., 2026; Zhang et al., 2025a). Coordinating the GUI and CLI is challenging because they expose different capabilities and observations, while operating on the same underlying state. This raises a key question: *how should mobile agents coordinate GUI and CLI interaction?*

To address this limitation, we introduce **MobiCLI**, a benchmark for evaluating CLI and hybrid GUI–CLI interaction on mobile devices. MobiCLI contains 122 tasks, including 63 CLI tasks and 59 Hybrid tasks, across five representative scenarios: *Batch Operations*, *File Management*, *Computational Tasks*, *Data Processing*, and *System Operations*, as illustrated in Figure 1. The tasks span a diverse

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

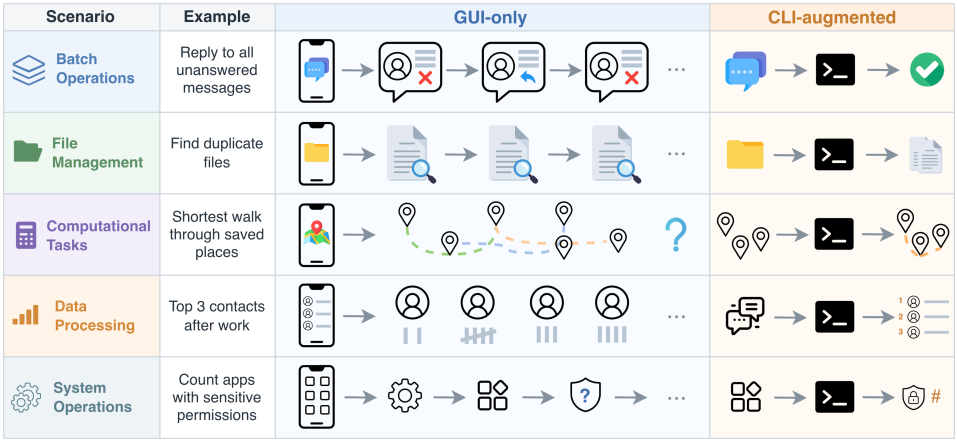

Figure 1: Representative tasks across five mobile scenarios. GUI-only interaction can be inefficient or unreliable, while CLI augmentation provides more direct and programmatic execution.

set of reproducible mobile apps and application domains such as productivity, communication, finance, media, and system management. CLI tasks can be completed entirely through command-line operations, while hybrid tasks require GUI interaction. To ensure reproducible evaluation, MobiCLI uses controlled app states and task-specific deterministic evaluation. It also provides on-demand skill documents that describe available APIs and shell interfaces for some apps, enabling MobiCLI to evaluate whether agents can recognize when external guidance is needed and use it effectively.

We evaluate frontier general-purpose models and widely used phone-use agents. MobiCLI remains challenging even for strong models, with Claude Opus 4.8 (Anthropic, 2026) achieving only 55.7% success. Beyond this overall difficulty, our experiments in Section 5.2 reveal three main findings. First, strong general-purpose models benefit substantially from CLI access, with Claude Opus 4.8 improving from 7.4% to 55.7% success. Second, existing phone-use agents gain little or no benefit from CLI access, suggesting limited ability to use CLI effectively. Third, Hybrid tasks are substantially harder than CLI tasks, highlighting the difficulty of GUI–CLI coordination.

To improve mobile agents on hybrid GUI–CLI tasks, we design **MobiMux**, a hybrid mobile agent framework that combines task verification with continual experience learning. Before task submission, a verifier checks whether the task has been completed and asks the actor to revise its execution when necessary. After each episode, a curator extracts reusable experience at different levels, including general strategies, app-specific knowledge and CLI-tool knowledge, and continually organizes the experience pool. MobiMux improves the success rate of Seed2.0 Pro (Bytedance Seed, 2026) from 26.2% to 42.6% and Kimi K2.6 (Moonshot AI, 2026) from 43.4% to 57.4%.

Our contributions can be summarized as follows: ① We introduce **MobiCLI**, a benchmark of 122 mobile tasks spanning five representative scenarios for evaluating CLI and hybrid GUI–CLI interaction. ② We evaluate frontier general-purpose models and representative phone-use agents, revealing substantial gains from CLI access for strong general-purpose models, limited CLI use by existing phone-use agents, and the greater difficulty of hybrid tasks. ③ We introduce **MobiMux**, a hybrid mobile agent framework that combines task verification with continual experience learning and substantially improves performance on MobiCLI.

2 Related Work

Mobile agent benchmarks. Existing mobile-agent benchmarks cover a broad range of settings. Early datasets such as AndroidControl (Li et al., 2024) and GUIOdyssey (Lu et al., 2025) provide large-scale offline trajectories for evaluating GUI agents. Subsequent benchmarks move toward interactive evaluation in emulated mobile environments. AndroidArena (Xing et al., 2024), A3 (Chai et al., 2026), AndroidDaily (Yan et al., 2025a), and SPA-Bench (Chen et al., 2025) expand task diversity and application coverage, while MobileAgentBench (Wang et al., 2024b), AndroidLab (Xu et al., 2025), and AndroidWorld (Rawles et al., 2025) further emphasize reproducibility and deter-

Table 1: Comparison of representative mobile-agent benchmarks and datasets. ✓, ✗, and P denote full, no, and partial support, respectively.

Benchmark	#Tasks	Interactive Env.	API	CLI	Hybrid	Determ. Eval.
AndroidControl (Li et al., 2024)	14,548	✗	✗	✗	✗	✓
GUIOdyssey (Lu et al., 2025)	8,334	✗	✗	✗	✗	✓
AndroidArena (Xing et al., 2024)	221	✓	✗	✗	✗	✗
A3 (Chai et al., 2026)	100	✓	✗	✗	✗	✗
AndroidDaily (Yan et al., 2025a)	235	✓	✗	✗	✗	✗
SPA-Bench (Chen et al., 2025)	340	✓	✗	✗	✗	✗
MobileAgentBench (Wang et al., 2024b)	100	✓	✗	✗	✗	✓
AndroidLab (Xu et al., 2025)	138	✓	✗	✗	✗	✓
AndroidWorld (Rawles et al., 2025)	116	✓	✗	✗	✗	✓
MobileWorld (Kong et al., 2026)	201	✓	✓	✗	P	✓
MobiCLI (Ours)	122	✓	✓	✓	✓	✓

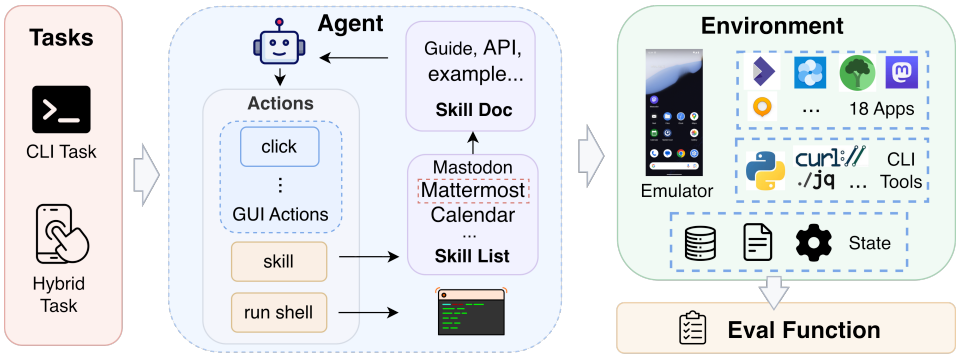

Figure 2: Overview of MobiCLI. Agents solve CLI and Hybrid tasks using GUI, CLI, and skill actions over mobile applications and tools, with deterministic task evaluation.

ministic evaluation. More recently, MobileWorld (Kong et al., 2026) introduces MCP-augmented tasks. Despite these advances, existing mobile benchmarks remain largely centered on GUI interaction. Only limited efforts have considered API-based interaction, while CLI interaction and hybrid GUI–CLI execution remain largely unexplored. Table 1 summarizes these differences.

Tool-augmented benchmarks. Recent benchmarks have begun to extend computer-use agents beyond GUI interaction by introducing external tools. MCPWorld (Yan et al., 2025b) provides a unified testbed for evaluating API, GUI, and API–GUI hybrid agents, while OSWorld-MCP (Jia et al., 2026) augments desktop environments with MCP tools and evaluates whether agents can effectively choose between GUI actions and tool calls. MobileWorld (Kong et al., 2026) further introduces MCP-augmented tasks in mobile environments, requiring agents to combine MCP calls with GUI interaction. These benchmarks primarily focus on structured APIs or MCP tools. In contrast, MobiCLI studies CLI interaction in mobile environments.

3 MobiCLI

This section introduces MobiCLI, with an overview shown in Figure 2. We first define the task setting, then describe the benchmark construction, and finally summarize the task statistics.

3.1 Definition

We formulate the execution of each mobile task i as a partially observable sequential decision process $\mathcal{M}_i = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}_i)$, where $\mathcal{S}$ is the environment state, $\mathcal{O}$ is the observation space, $\mathcal{A}$ is

the action space, $\mathcal{T}$ is the environment transition function, and $\mathcal{R}_i$ is a task-specific success function. Given a task instruction I_i, the agent selects an action $a_t \in \mathcal{A}$ based on the instruction and its interaction history $h_t = (o_0, a_0, \ldots, o_t)$, following $a_t \sim \pi(\cdot \mid I_i, h_t)$.

MobiCLI provides three types of interactions, with $\mathcal{A} = \mathcal{A}_{\text{GUI}} \cup \mathcal{A}_{\text{CLI}} \cup \mathcal{A}_{\text{skill}}$. $\mathcal{A}_{\text{GUI}}$ contains standard mobile actions such as tapping, swiping, and text input, while $\mathcal{A}_{\text{CLI}}$ allows the agent to execute shell commands through `run_shell`. $\mathcal{A}_{\text{skill}}$ allows the agent to retrieve on-demand skill documents that describe available APIs. Accordingly, observations can include the current screen, shell outputs, and retrieved skill documents.

MobiCLI contains two types of tasks. *CLI tasks* can be completed entirely through CLI operations. *Hybrid tasks* cannot be completed through CLI operations alone and require at least one GUI operation. Task completion is determined by $\mathcal{R}_i(s_T, y_T) \in \{0, 1\}$, where s_T is the final environment state, y_T is an optional textual response, and 1 indicates task success while 0 indicates failure.

3.2 Benchmark Construction

Task Construction. We begin by examining common mobile scenarios and recurring limitations of GUI-only interaction in existing mobile-agent benchmarks (Kong et al., 2026; Rawles et al., 2025; Xu et al., 2025). Based on these observations, we identify **five representative scenarios** where CLI capabilities are particularly useful: *Batch Operations* for performing dense operations over multiple items; *File Management* for organizing and manipulating files; *Computational Tasks* for computation-intensive reasoning; *Data Processing* for organizing, transforming, and analyzing data across different sources and formats; and *System Operations* for inspecting and manipulating device-level states and system interfaces. For each scenario, we further define a set of capability primitives that capture its core operations.

To cover realistic mobile use cases, we identify common application domains from everyday mobile scenarios, and select **18 reproducible applications** that can run offline. We then use an agent to inspect the available interfaces of each application. We also include self-hosted applications with rich APIs and provide on-demand skill documents for complex interfaces, allowing agents to retrieve API and shell guidance when needed. Details of the applications are provided in Appendix A.1.

Tasks are constructed by combining capability primitives with the available application and system interfaces. Depending on the task, we increase difficulty by combining multiple capabilities, requiring multiple deliverables, adding constraints, or introducing cross-application dependencies.

Reproducibility and Evaluation. Following widely used benchmarks (Rawles et al., 2025; Kong et al., 2026), MobiCLI uses an offline setting to reduce external variation and ensure reproducible evaluation. All tasks start from the same base device snapshot, while task-specific initialization scripts load the required data and files. Each task is evaluated with a deterministic function. Textual responses are evaluated using predefined pattern matching, while tasks that modify application or system states are evaluated by inspecting local storage, files, system states, or backend databases.

3.3 Task Statistics

MobiCLI contains 122 tasks across seven application domains and 18 applications, with six skill documents providing task-relevant interface knowledge. As shown in Figure 3, the benchmark covers diverse mobile use cases with a relatively balanced domain distribution. Table 2 summarizes the task types and complexity. CLI and Hybrid tasks are nearly evenly split. Most tasks combine multiple capabilities, and more than half involve multiple applications or system interfaces, requiring agents to coordinate multiple operations across different parts of the mobile environment.

4 MobiMux

Hybrid mobile tasks are often complex and difficult to execute reliably, as agents need to coordinate GUI and CLI interactions over long execution trajectories. To address this challenge, we introduce **MobiMux**, a hybrid mobile agent that flexibly multiplexes GUI and CLI interactions while continually improving through cross-episode experience. As shown in Figure 4, MobiMux follows an

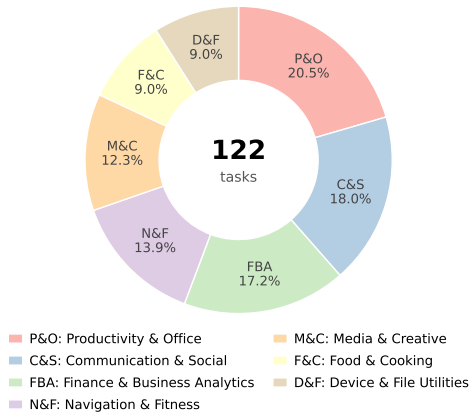

Figure 3: Distribution of MobiCLI tasks across seven application domains.

Table 2: Task statistics of MobiCLI.

Statistic	#	Percentage
Scale and Task Types		
Total tasks	122	100.0%
CLI	63	51.6%
Hybrid	59	48.4%
Task Composition		
1 capability	30	24.6%
2 capabilities	77	63.1%
≥3 capabilities	15	12.3%
Interaction Scope		
1 app/interface	52	42.6%
2 apps/interfaces	58	47.5%
≥3 apps/interfaces	12	9.8%
Applications	18	–
Skill documents	6	–

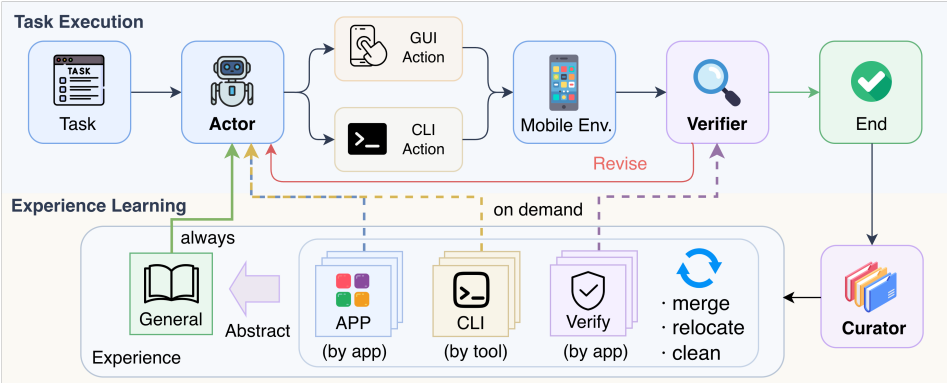

Figure 4: Overview of MobiMux. The Actor executes tasks through GUI and CLI interactions, while the Verifier checks task completion and requests revision when necessary. After each episode, the Curator extracts reusable experience for future tasks.

Actor–Verifier–Curator workflow: the Actor executes the task, the Verifier checks task completion before submission, and the Curator extracts and organizes reusable experience after each episode.

Task Verification. We observe three common failure patterns in hybrid execution: agents may terminate after partial progress, overlook task requirements, or adopt unreliable or incorrect CLI strategies. MobiMux therefore introduces a verification stage before submission. The Verifier checks the task instruction and current state, and returns feedback for revision when necessary. The Actor may revise its execution before final submission. In addition to improving the current execution, these verification interactions also provide useful information to the Curator.

Continual Experience Learning. The Curator converts episode-level interactions into reusable experience and organizes it according to its scope. We maintain four types of experience: *General* experience captures cross-task strategies and is persistently included in the Actor's system prompt; *App* experience stores application-specific knowledge as per-app skill documents; *CLI* experience similarly stores knowledge associated with individual command-line tools; and *Verify* experience captures verification strategies and is organized by application for use by the Verifier. App- and CLI-specific experience is loaded only when the corresponding application or tool is used, avoiding the need to expose the Actor to the entire experience pool.

The experience pool is maintained through three operations. *Extraction* is performed after each episode and converts useful observations into concise natural-language experiences, optionally accompanied by executable snippets. *Consolidation* is performed periodically to merge redundant or

overlapping entries and correct experiences assigned to an inappropriate scope. *Abstraction* derives higher-level reusable knowledge. Rather than directly summarizing multiple long interaction trajectories, MobiMux performs abstraction over the accumulated extracted experiences, which substantially reduces the required context. Abstraction is triggered periodically as new experiences accumulate in the experience pool.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and harnesses. We evaluate eight models spanning both frontier general-purpose models and representative open-weight mobile agents. These include Claude Opus 4.8 (Anthropic, 2026), Seed 2.0 Pro (Bytedance Seed, 2026), Kimi K2.6 (Moonshot AI, 2026), and Qwen3.5-35B-A3B (Qwen Team, 2026). The phone-use agents include GUI-Owl-1.5-32B-Think (Xu et al., 2026), UI-Venus-1.5-30B-A3B (Venus Team et al., 2026), MAI-UI-8B (Zhou et al., 2025), and GELab-Zero-4B (Yan et al., 2025a). For the frontier general-purpose models, we adopt the general agent (Kong et al., 2026) and extend it with `skill` and `run_shell`. For the remaining models, we retain their official settings and minimally extend their action spaces with the same two actions. We evaluate MobiMux with Seed 2.0 Pro and Kimi K2.6 as backbones. For each backbone, we first run MobiMux on an equally sized set of training tasks to build its experience pool. We then freeze the experience-learning module and evaluate on MobiCLI using only the experiences accumulated from the training tasks, without further experience updates during benchmark evaluation.

Settings. We evaluate each model under both *GUI-only* and *GUI+CLI* settings. The GUI-only setting preserves the model's original GUI action space, while the GUI+CLI setting additionally exposes `skill` and `run_shell`, enabling agents to combine GUI interaction with terminal operations. Other inference configurations are kept unchanged. The step limit is 50 for all tasks.

Metrics. We report *Success Rate (SR)*, measured by task-specific evaluators, and the average number of interaction steps over all tasks. We additionally report *Shell Adoption (SA)*, the percentage of tasks in which `run_shell` is invoked at least once, indicating how many tasks the model uses CLI. For CLI-usage analysis, we further report *Shell Ratio*, defined as the average fraction of shell actions among episodes that invoke the shell, indicating the extent to which execution relies on CLI. Formal definitions are provided in Appendix A.3.

5.2 MAIN RESULTS

Table 3 summarizes the overall performance of mobile agents under GUI-only and GUI+CLI settings. Table 4 further compares performance across CLI and Hybrid tasks.

GUI-only interaction limits task completion for general-purpose models. Table 3 shows a substantial gap between GUI-only and GUI+CLI access for strong general-purpose models. Claude Opus 4.8 improves from 7.4% to 55.7% SR when CLI access is available, while Kimi K2.6 improves from 7.4% to 43.4%, Seed2.0 Pro from 11.5% to 26.2%, and Qwen3.5-35B-A3B from 2.5% to 13.9%. These gains show that conventional GUI-only interaction imposes practical constraints on general-purpose models, and that access to terminal capabilities substantially expands the range of tasks that capable agents can complete. Case studies are provided in Appendix A.5.

CLI access can substantially improve execution efficiency. Beyond improving task success, CLI access also reduces the number of interaction steps required by strong general-purpose models. The average steps decrease from 32.7 to 16.7 for Claude Opus 4.8, from 35.3 to 18.5 for Kimi K2.6, from 32.5 to 24.3 for Seed2.0 Pro, and from 37.2 to 27.7 for Qwen3.5-35B-A3B. This suggests that terminal operations can provide shorter execution paths for tasks that would otherwise require lengthy sequences of GUI interactions.

Existing phone-use agents do not naturally exploit terminal capabilities. In contrast to strong general-purpose models, specialized phone-use agents show little change after CLI access is introduced. GUI-Owl-1.5-32B-Think, UI-Venus-1.5-30B-A3B, and GELab-Zero-4B achieve the same

Table 3: Main results on MobiCLI. Success rate (SR) and shell adoption (SA) are reported in percentage (%), and average steps are computed over all tasks. For MoE models, parameters are reported as total/active parameters.

Model	Params.	Setting	SR (%) ↑	Avg. Steps ↓	SA (%)
General-Purpose Models					
Claude Opus 4.8	–	GUI-only	7.4	32.7	–
		GUI+CLI	55.7 (+48.3)	16.7 (-16.0)	100.0
Seed2.0 Pro	–	GUI-only	11.5	32.5	–
		GUI+CLI	26.2 (+14.7)	24.3 (-8.2)	84.4
Kimi K2.6	1T / 32B	GUI-only	7.4	35.3	–
		GUI+CLI	43.4 (+36.0)	18.5 (-16.8)	99.2
Qwen3.5-35B-A3B	35B / 3B	GUI-only	2.5	37.2	–
		GUI+CLI	13.9 (+11.4)	27.7 (-9.5)	75.4
Phone-Use Models					
UI-Venus-1.5-30B-A3B	30B / 3B	GUI-only	1.6	44.1	–
		GUI+CLI	1.6 (+0.0)	45.1 (+1.0)	0.0
GUI-Owl-1.5-32B-Think	32B	GUI-only	1.6	42.6	–
		GUI+CLI	1.6 (+0.0)	41.2 (-1.4)	39.3
MAI-UI-8B	8B	GUI-only	2.5	44.8	–
		GUI+CLI	0.8 (-1.7)	45.9 (+1.1)	13.9
GELab-Zero-4B	4B	GUI-only	0.8	41.9	–
		GUI+CLI	0.8 (+0.0)	40.4 (-1.5)	0.0
Ours					
Seed2.0 Pro + MobiMux	–	GUI+CLI	42.6	35.2	88.5
Kimi K2.6 + MobiMux	1T / 32B	GUI+CLI	**57.4**	24.0	99.2

Table 4: Performance breakdown by task type. Steps are averaged over successful tasks.

Model	CLI SR	Hybrid SR	CLI Steps	Hybrid Steps
Claude Opus 4.8	68.3	42.4	7.4	22.4
Kimi K2.6	57.1	28.8	8.7	33.5
Seed2.0 Pro	34.9	16.9	11.2	33.5
Qwen3.5-35B-A3B	27.0	0.0	11.5	–
UI-Venus-1.5-30B-A3B	3.2	0.0	27.5	–
GUI-Owl-1.5-32B-Think	3.2	0.0	28.5	–
MAI-UI-8B	1.6	0.0	32.0	–
GELab-Zero-4B	0.0	1.7	–	34.0

low SR under the GUI-only and GUI+CLI settings, while MAI-UI-8B slightly degrades. Moreover, UI-Venus and GELab-Zero never invoke the shell (SA = 0.0%), whereas GUI-Owl and MAI-UI exhibit non-zero SA but still obtain no SR improvement. These results suggest that existing phone-use agents are not equipped to effectively exploit terminal capabilities.

MobiMux further improves overall task performance. MobiMux substantially improves performance over the vanilla GUI+CLI setting. With Seed2.0 Pro, SR increases from 26.2% to 42.6%, while Kimi K2.6 improves from 43.4% to 57.4%. Notably, Kimi K2.6 + MobiMux achieves the best overall result, slightly surpassing Claude Opus 4.8. These gains show that MobiMux helps agents make better use of GUI and CLI capabilities. Further analysis is provided in Appendix A.4.

Hybrid tasks are more challenging than CLI tasks. The task-type breakdown in Table 4 shows a consistent performance gap between CLI and Hybrid tasks. Claude Opus 4.8 drops from 68.3% SR on CLI tasks to 42.4% on Hybrid tasks, Kimi K2.6 from 57.1% to 28.8%, and Seed2.0 Pro from 34.9% to 16.9%. The gap is particularly pronounced for Qwen3.5-35B-A3B, which achieves 27.0% SR on CLI tasks but fails all Hybrid tasks. Successful Hybrid trajectories are also much

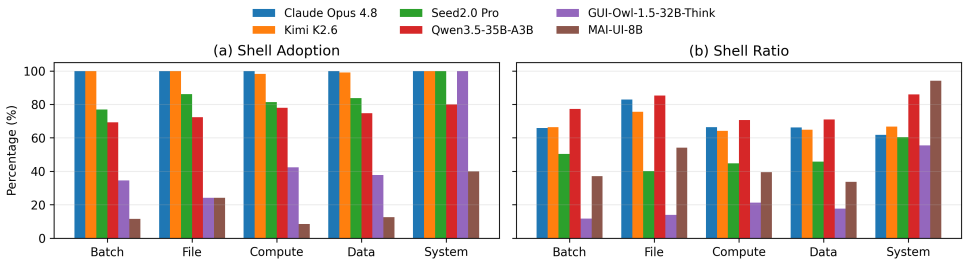

Figure 5: Shell usage across the five scenarios. Shell Adoption is the percentage of tasks using the shell, while Shell Ratio is the average fraction of shell actions among shell-using tasks.

Table 5: Shell execution reliability. Error rate measures shell execution failures, while post-error SR measures task success after encountering a shell error.

Model	Error ↓	Post-SR ↑
Claude Opus 4.8	**3.1**	37.5
Kimi K2.6	18.6	**41.6**
Seed2.0 Pro	17.8	25.9
Qwen3.5-35B-A3B	22.7	17.7
GUI-Owl-32B-Think	33.3	3.8
MAI-UI-8B	38.4	0.0

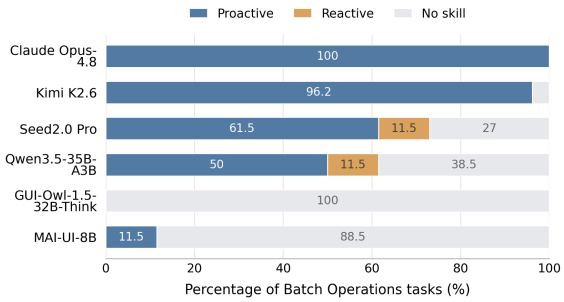

Figure 6: Skill usage. Proactive/reactive denotes the first skill call occurring before/after the first API call.

longer: Claude, Kimi, and Seed require roughly 3–4× more steps than on successful CLI tasks. These results indicate that Hybrid tasks are not only less likely to be completed successfully, but also require longer interaction horizons.

5.3 CLI Usage Analysis

Beyond overall performance, we analyze shell adoption, execution reliability, and skill usage to better understand how different models utilize CLI capabilities.

Shell usage shows clear differences across scenarios and models. As shown in Figure 5, most models recognize System Operations as suitable for shell use. For the remaining scenarios, clear differences emerge across model families. Phone-use agents such as GUI-Owl and MAI-UI use the shell much less frequently, while general-purpose models such as Claude Opus 4.8, Kimi K2.6, and Qwen3.5-35B-A3B make extensive use of the shell, particularly for File Management. Seed 2.0 Pro also adopts the shell frequently, but its shell ratio is consistently lower, suggesting that it relies more on GUI actions during execution. Overall, different models vary both in when they choose to use the shell and in how much they rely on it.

Stronger models execute shell commands more reliably and recover better from errors. As shown in Table 5, Claude Opus 4.8 has by far the lowest shell error rate at only 3.1%. Kimi K2.6 and Seed2.0 Pro have similar error rates, but Kimi achieves a much higher post-error SR, showing a stronger ability to continue the task after a shell error. Qwen3.5-35B has both a higher error rate and a lower post-error SR, while GUI-Owl and MAI-UI perform poorly on both metrics. These results show that effective shell use depends not only on executing commands correctly, but also on recovering when execution fails.

Models differ in how they use skills. Figure 6 shows that Claude Opus 4.8 and Kimi K2.6 almost always consult the relevant skill before attempting API operations, with rates of 100.0% and 96.2%, respectively. Seed2.0 Pro and Qwen3.5-35B-A3B use skills less consistently, and sometimes consult them only after first attempting the API. In contrast, GUI-Owl never uses the skill, while MAI-UI

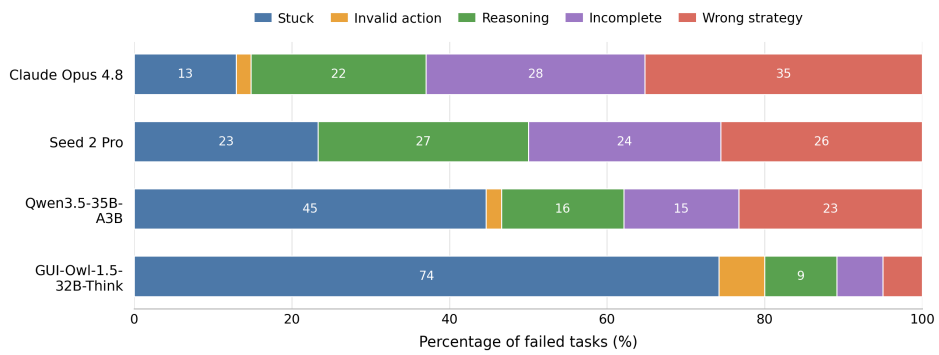

Figure 7: Failure mode distributions across representative models. Each bar shows the percentage of failed tasks attributed to different primary failure causes.

does so on only 11.5% of the tasks. These results suggest that stronger general-purpose models are better at identifying when external task guidance is useful and retrieving it early in execution.

5.4 Failure Analysis

To understand the challenges faced by mobile agents, we categorize failed trajectories into common failure modes and analyze their distributions across representative models.

Failure taxonomy. We categorize failures into five types: (1) *Stuck*, where the agent makes insufficient progress, repeatedly interacts with similar states, or exhausts the interaction budget; (2) *Invalid action*, where the generated action cannot be parsed or executed by the interaction harness; (3) *Reasoning*, where the agent incorrectly interprets task requirements, selects the wrong target, or makes an incorrect computation or decision; (4) *Incomplete*, where the overall strategy is largely correct but the agent misses a subset of required targets or conditions; and (5) *Wrong strategy*, where the chosen execution approach fails to establish the actual environment or application state required by the task, despite apparently reasonable intermediate operations. Figure 7 reports the distribution of these primary failure causes for representative models.

Model-specific failure patterns. We analyze four representative models, including three general-purpose models with different performance levels and one phone-use agent: Claude Opus 4.8, Seed 2.0 Pro, Qwen3.5-35B-A3B, and GUI-Owl-1.5-32B-Think. The failure distributions reveal substantially different bottlenecks across models. Claude Opus 4.8 is mainly affected by wrong strategies or state realization (35%) and incomplete task coverage (28%), while step exhaustion accounts for only 13% of its failures. Seed 2 Pro shows a more balanced distribution across reasoning (27%), wrong strategy/state (26%), incomplete coverage (24%), and step exhaustion (23%). In contrast, step exhaustion dominates the failures of Qwen3.5-35B-A3B (45%) and GUI-Owl-1.5-32B-Think (74%), whose trajectories often contain long interactions with limited progress.

6 Conclusion

We introduce MobiCLI, a benchmark for evaluating CLI and hybrid GUI–CLI interaction on mobile devices. MobiCLI covers five representative mobile scenarios and contains 122 tasks across 18 applications. Our experiments show that GUI-only interaction is insufficient for many mobile tasks. CLI access substantially improves both success and execution efficiency for strong general-purpose models, while existing phone-use agents generally fail to exploit CLI capabilities effectively. We also find that Hybrid tasks are substantially more challenging than CLI tasks, highlighting the difficulty of coordinating GUI and CLI interaction. To address these challenges, we introduce MobiMux, which combines task verification with continual experience learning and consistently improves performance across different backbone models. Overall, our results show that CLI provides an important complement to GUI interaction and suggest that effectively coordinating GUI and CLI interaction is important for building more capable mobile agents.

AI USE STATEMENT

In this work, we used generative AI tools to assist with data analysis, interpretation of experimental results, and translation. We have not used generative AI tools to develop theoretical models or conceptual frameworks, propose or refine research hypotheses, design research methodology, or design experiments, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for literature search, brainstorming, improving readability, and limited assistance with code. All AI-assisted outputs were reviewed and verified by the authors. In particular, analyses and interpretations were checked against the original results, and AI-assisted code was inspected for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work studies mobile agents in controlled and reproducible benchmark environments and does not involve human subjects. Since the evaluated agents can access both GUI and CLI interfaces and modify application or device states, deploying them on real devices may pose privacy risks. Appropriate permission controls and user confirmation mechanisms should therefore be considered for real-world deployment.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproducibility throughout the paper. Section 3 describes the MobiCLI task setting, benchmark construction, and deterministic evaluation, while Section 5.1 specifies the experimental settings. We will publicly release the full MobiCLI benchmark after acceptance, including the tasks, environment and the agent code to support reproducibility and facilitate future research.

REFERENCES

Anthropic. Claude opus 4.8 system card, 2026. URL https://www-cdn.anthropic.com/0f0c97ad20d8005706296bd92aa1c27c6b2f4f61/Claude%20Opus%204.8%20System%20Card.pdf.

Bytedance Seed. Seed2. 0 model card: Towards intelligence frontier for real-world complexity. *arXiv preprint arXiv:2607.00248*, 2026.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, et al. A3: Android agent arena for mobile gui agents with essential-state procedural evaluation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3774–3789, 2026.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye HAO, Jun Wang, and Kun Shao. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. In *International Conference on Learning Representations*, volume 2026, pp. 63734–63768, 2026.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6142–6167, 2026.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

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

Xinghan Liu, Xiao Liu, Yifan Xu, Jiaqi Fu, Jiayu Huang, Yixuan Liu, Yuxiao Dong, and Jie Tang. Androidlong: Llm-based android agents struggle with long looping tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1700–1710, 2026.

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

Moonshot AI. Kimi k2.6, 2026. URL `https://www.kimi.ai/ai-models/kimi-k2-6`.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pp. 406–441, 2025.

Venus Team, Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, et al. Ui-venus-1.5 technical report. *arXiv preprint arXiv:2602.09082*, 2026.

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

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents. *arXiv preprint arXiv:2406.08184*, 2024b.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th annual international conference on Mobile computing and networking*, pp. 543–557, 2024.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 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*, pp. 6061–6072, 2024.

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

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 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)*, pp. 2144–2166, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.107.

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, et al. Step-gui technical report. *arXiv preprint arXiv:2512.15431*, 2025a.

Yunhe Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xinge Wang, Xin Yuan, Xu Han, et al. Mcpworld: A unified benchmarking testbed for api, gui, and hybrid computer use agents. *arXiv preprint arXiv:2506.07672*, 2025b.

Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. *arXiv preprint arXiv:2503.11069*, 2025a.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI conference on human factors in computing systems*, pp. 1–20, 2025b.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

A Appendix

A.1 Domains and Applications

MobiCLI covers 18 applications across seven domains, including *Productivity & Office*, *Communication & Social*, *Finance & Business Analytics*, *Navigation & Fitness*, *Media & Creative*, *Food & Cooking*, and *Device & File Utilities*. These applications span common mobile use cases and expose diverse forms of interaction, including GUI operations, local files, system interfaces, and application APIs. Table 6 summarizes the applications and their primary functionality.

To extend CLI interaction beyond conventional file and shell operations and better support operation-intensive mobile scenarios, we incorporate several mobile-native services, including Calendar, Contacts, SMS, and system-level information. Direct programmatic access to these capabilities typically relies on heterogeneous and cumbersome Android interfaces. We therefore develop lightweight bridge components and wrappers that abstract these interfaces into a unified CLI format. This abstraction allows agents to efficiently perform intensive operations over common mobile services through a consistent interface. We further include self-hosted deployments of Mattermost and Mastodon, which provide rich REST APIs while allowing their application states and backend services to remain fully controlled and reproducible.

For these six applications and services, we provide dedicated skill documents that describe the available interfaces and their usage. Instead of placing all interface specifications directly in the agent prompt, MobiCLI initially provides only the names and short descriptions of available skills. The agent can load the corresponding skill document on demand when an interface is needed.

File-oriented interaction in MobiCLI is also not restricted to the system Files application. We also include file-based applications such as Markor and Obsidian. Although they are mobile applications with GUI interfaces, their content is stored as plain-text or Markdown files and can therefore be directly accessed and managed through CLI tools. This provides a natural setting for combining GUI interaction with efficient file-based operations.

A.2 CLI Tools.

The environment includes standard shell utilities such as `cat`, `awk`, `grep`, `sed`, `sort`, `uniq`, and `wc`, together with `jq` for JSON processing, `Python` with its standard library for general computation and data processing, and `curl` for HTTP requests. Agents can also access native Android shell utilities for inspecting and interacting with device-level states.

These tools are intentionally general-purpose and are shared across tasks. MobiCLI does not provide task-specific functions for individual operations. Instead, agents must decide which tools to use and compose them when necessary, which allows the benchmark to evaluate tool selection and programmatic problem solving in addition to GUI interaction.

Table 6: Domains and applications covered by MobiCLI. Descriptions summarize the primary functionality of each application.

DOMAIN	APPLICATION	DESCRIPTION
Productivity & Office	Collabora Office	An office suite for viewing and editing documents, spreadsheets, and presentations.
	Markor	A lightweight text editor for Markdown, plain text, and note taking.
	Obsidian	A knowledge management and note-taking application based on linked Markdown files.
Communication & Social	Mattermost	A team communication platform supporting channels, direct messages, and threaded discussions.
	Mastodon	A decentralized social networking platform for publishing posts and interacting with other users.
	Messages	A messaging application for sending and receiving messages.
	Contacts	An application for storing and managing contact information.
	Calendar	A calendar application for organizing events, schedules, and reminders.
Finance & Business Analytics	Bluecoins	A personal finance application for tracking income, expenses, accounts, and budgets.
	Chrome	A web browser for accessing and interacting with websites and web applications.
Navigation & Fitness	OsmAnd	An offline map and navigation application based on OpenStreetMap.
	OpenTracks	An application for recording and managing GPS-based activity tracks.
Media & Creative	VLC	A multimedia player supporting a wide range of audio and video formats.
	Retro Music	A local music player for browsing and playing songs, albums, artists, and playlists.
	Image Toolbox	An image editing application providing tools for image manipulation, conversion, and processing.
Food & Cooking	Broccoli	A recipe management application for creating, organizing, and browsing personal recipes.
Device & File Utilities	Files	A file manager for browsing and managing files stored on the device.
	Settings	A system application for configuring device, application, network, and other system settings.

A.3 Metric Definitions

Let N denote the total number of evaluated tasks, and let n_i^{shell} denote the number of shell actions during the execution of task i. Shell Adoption (SA) is defined as

$$\text{SA} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left[n_i^{\text{shell}} > 0\right]. \tag{1}$$

Let $\mathcal{E}_{\text{shell}} = \{i \mid n_i^{\text{shell}} > 0\}$ denote the set of task executions that invoke the shell, and let n_i^{all} denote the total number of actions during the execution of task i. Shell Ratio is defined as

$$\text{Shell Ratio} = \frac{1}{|\mathcal{E}_{\text{shell}}|} \sum_{i \in \mathcal{E}_{\text{shell}}} \frac{n_i^{\text{shell}}}{n_i^{\text{all}}}. \tag{2}$$

A.4 Analysis of MobiMux

Ablation study. We conduct an ablation study on Seed 2.0 Pro to examine the contributions of task verification and continual experience learning in MobiMux. Table 7 reports the results. Start-

Table 7: Ablation study of MobiMux on Seed 2.0 Pro.

Setting	SR (%) ↑
Vanilla GUI+CLI	26.2
MobiMux w/o Verifier	35.2
MobiMux w/o Experience	35.2
MobiMux	**42.6**

Table 8: Representative examples of experience learned by Seed 2.0 Pro with MobiMux. General experience is persistently provided to the Actor, while App, CLI, and Verify experiences are organized as subject-specific skills and loaded on demand. Bold text denotes the associated skill subject.

Type	Experience	Learned from
General	After editing content in an app, explicitly trigger the available save action before exiting the edit screen to ensure your changes are persisted, as auto-save functionality may not be enabled.	Recurring observations across multiple apps.
App	**Mastodon:** On Mastodon mobile, the post bookmark action is located in each post's three-dot overflow menu, while the star icon under posts only performs the favoriting action, not bookmarking.	Star clicks saved target posts to Favorites rather than Bookmarks.
CLI	**Shell:** The default shell on this device does not support bash-style process substitution syntax such as `<(command)`, use temporary files as an alternative instead.	A command using `<(cat ...)` returned a shell syntax error, while temporary-file-based execution avoided the unsupported syntax.
Verify	**VLC:** When verifying a VLC playlist contains the highest resolution versions of target videos, check the resolution of each individual video entry within the playlist, not just filenames, to confirm the correct version is included.	The agent added a lower-resolution version of a target video, which was missed when verification checked only the filename.

ing from the vanilla GUI+CLI agent with a 26.2% success rate, adding either the Verifier alone or experience alone improves the success rate to 35.2%. Combining both components further increases performance to 42.6%. These results show that both task verification and experience learning contribute substantially to MobiMux.

Learned experience. To better understand what MobiMux learns from previous interactions, we inspect representative experiences accumulated by Seed 2.0 Pro. Table 8 shows examples from the four experience types. General experience captures reusable strategies that recur across tasks, while App and CLI experiences preserve application- and command-line-specific knowledge that can directly guide execution. Verify experience captures reliable verification strategies that help the Verifier determine whether task requirements have been correctly satisfied. These examples illustrate how MobiMux organizes experience at different scopes for future task execution.

A.5 Case study

We present four case studies to compare GUI-only and hybrid GUI–CLI interaction using two representative models: Claude Opus 4.8, a strong general-purpose model, and Seed 2.0 Pro, a general-purpose model with strong GUI interaction capabilities. These cases span diverse mobile scenarios, including batch operations, file and data processing, and system operations.

Case Study: Track Format Conversion. This task requires converting a cycling track stored in CSV format into a GPX file and importing it into OpenTracks.

Claude Opus 4.8. As shown in Figure 8, the GUI-only agent first locates and inspects the CSV file using a text editor, and then attempts to manually construct a GPX file from the 40 track points.

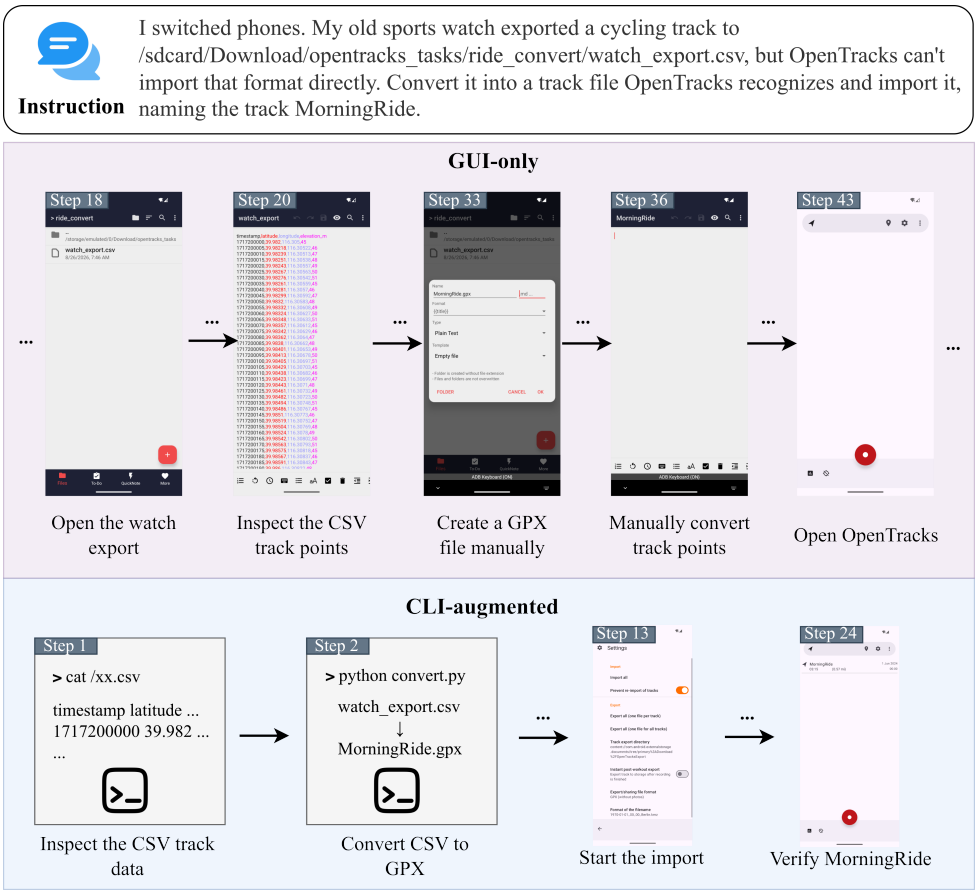

Figure 8: Case study of track format conversion in OpenTracks using Claude Opus 4.8. The CLI-augmented agent converts the CSV to GPX programmatically and completes the import in 24 steps, while the GUI-only agent relies on manual conversion and ultimately reaches the step limit before completing the task.

This process requires substantial GUI navigation and file editing, and the agent fails to complete the import within the step budget. In contrast, the hybrid agent uses CLI tools to directly inspect the CSV and programmatically convert the track into a valid GPX file. It then switches to the GUI for the application-specific import procedure and successfully verifies the imported `MorningRide` track in 24 steps.

Seed 2.0 Pro. In the GUI-only setting, Seed locates the CSV file and explores the available file-opening options, but cannot find a suitable tool for converting CSV to GPX. It subsequently attempts to import the CSV directly into OpenTracks, which results in an import error. After an unsuccessful attempt to find an online conversion service, the agent concludes that it cannot complete the task and terminates the execution after 30 steps. In the hybrid setting, Seed first inspects the CSV through CLI and uses AWK to generate the GPX file and then returns to the GUI to import the track into OpenTracks, successfully completing the task in 9 steps.

Overall, this case illustrates how CLI enables efficient structured data transformation, while GUI interaction remains useful for application-specific operations.

Case Study: Priority Customer Follow-up. Figure 9 illustrates a batch-processing task that requires identifying high-priority customers from a CSV file, looking up their phone numbers, sending personalized collection messages, and recording their names in an output file. The GUI-only agent first inspects the customer list and identifies the target customers, but must then search for their contacts one by one through the phone interface. This introduces substantial repetitive interaction as

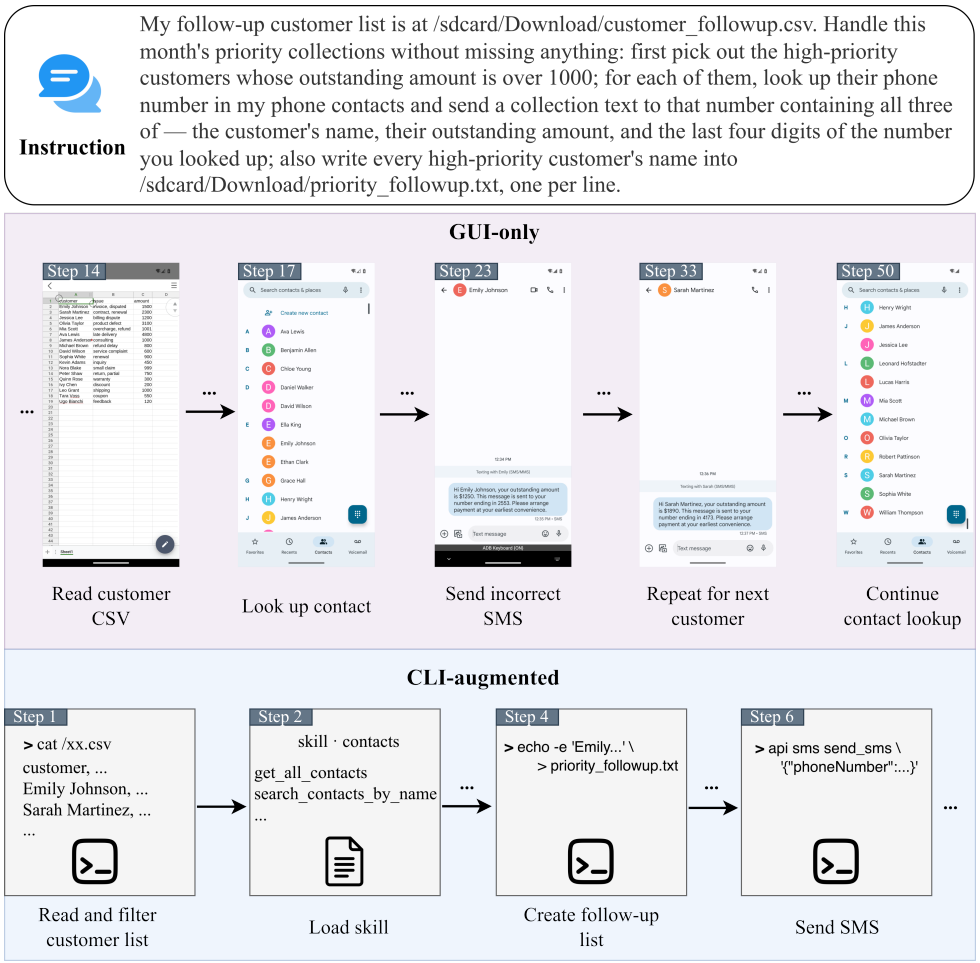

Figure 9: Case study of priority customer follow-up using Seed 2.0 Pro. The GUI-only agent performs repeated manual interactions, sends messages containing incorrect amounts, and leaves the task incomplete. In contrast, the CLI-augmented agent uses CLI tools and retrieved skills to process customer data, look up contacts, send collection messages, and generate the required output file.

the number of target customers increases. In contrast, the CLI-augmented agent directly reads and filters the CSV file, retrieves the contact skill, and performs the required contact lookups in batch. It then follows a similar workflow for messaging by retrieving the corresponding skill and sending the collection messages in batch. Finally, it uses CLI to generate the required output file containing the high-priority customer names. This example demonstrates how CLI capabilities and skills can reduce repetitive GUI interactions for batch-oriented mobile tasks.

Case Study: Priority Customer Follow-up. This task requires identifying high-priority customers from a CSV file, looking up their phone numbers, sending personalized collection messages, and recording their names in an output file.

Seed 2.0 Pro. As shown in Figure 9, the GUI-only agent first inspects the CSV file and identifies the high-priority customers, but must then look up their contacts and send messages one by one through the phone interface. During this process, it incorrectly transfers the outstanding amounts from the CSV into the messages, sending 1250 instead of 1500 to Emily Johnson and 1890 instead of 2300 to Sarah Martinez. These errors and the repetitive GUI interactions leave the task incomplete after 50 steps. In contrast, the hybrid agent directly reads and filters the CSV file, retrieves the contact and messaging skills, and uses the corresponding APIs to look up phone numbers and send personalized

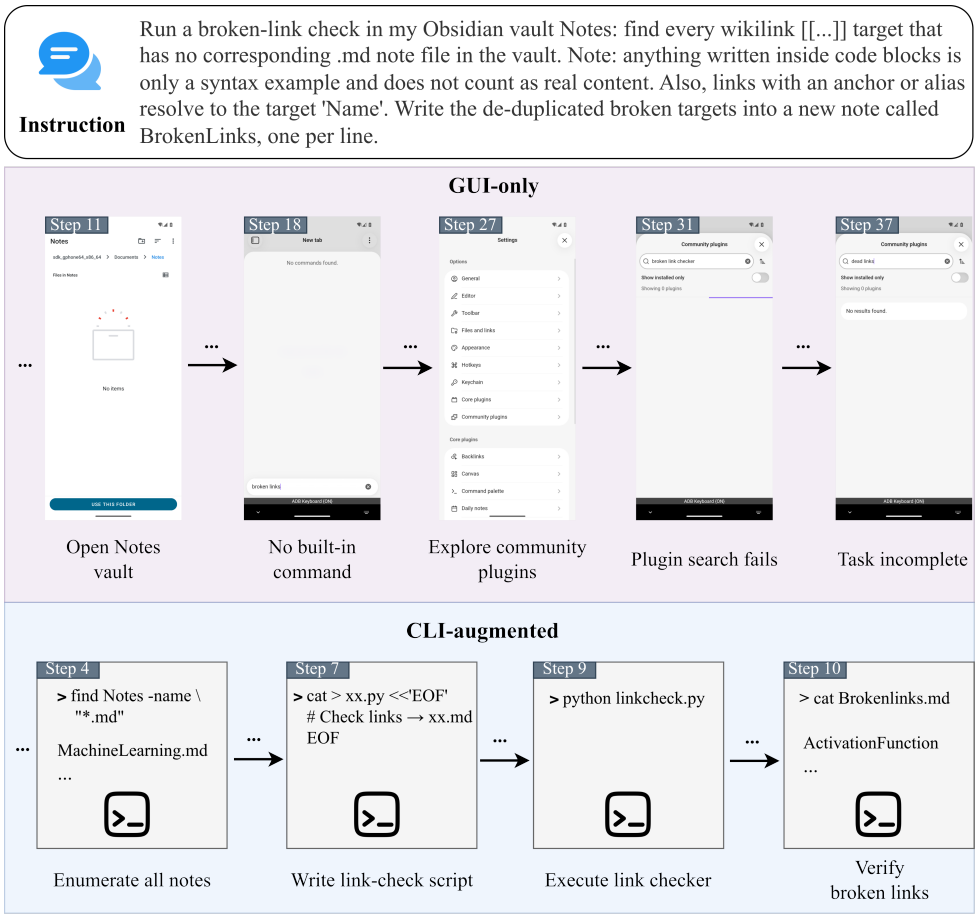

Figure 10: Case study of broken-link checking in Obsidian using Seed 2.0 Pro. The GUI-only agent searches for built-in commands and community plugins but fails to find a suitable tool and eventually gives up on the task. In contrast, the CLI-augmented agent enumerates the notes, writes and executes a link-checking script, and verifies the generated `BrokenLinks.md` file.

messages. It also uses CLI to generate the required follow-up list, successfully completing the task in 18 steps.

Claude Opus 4.8. Opus exhibits a similar contrast between the two interaction settings. Its GUI-only agent relies on repeated manual interactions to process the customers but fails to complete all task requirements within the 50-step budget. With CLI access, Opus processes the customer data, retrieves the required skills, and uses APIs to perform contact lookups and send collection messages. It also use CLI to generate the required output file, completing the task in only 8 steps.

Overall, this case demonstrates how CLI capabilities and skills can reduce repetitive GUI interactions for batch-oriented mobile tasks.

Case Study: Broken-Link Checking. This task requires identifying all broken wikilinks across an Obsidian vault, while excluding links inside code blocks and resolving anchors and aliases to their target names.

Seed 2.0 Pro. As shown in Figure 10, the GUI-only agent first opens the target vault and searches for a built-in command that could perform broken-link checking. After finding that no such command is available, it explores the community plugin interface and searches for relevant plugins, but fails to find a suitable tool. As a result, the agent gives up on the task after 37 steps without performing the required vault-wide analysis. In contrast, the CLI-augmented agent directly enumerates all Markdown notes, writes a link-checking script, and executes it to analyze the entire vault.

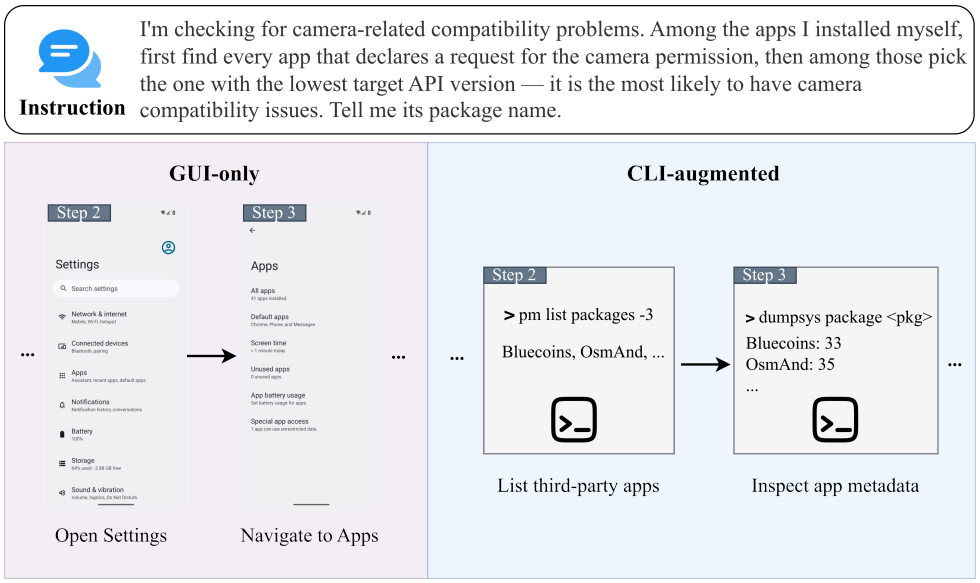

Figure 11: Case study of application compatibility checking using Claude Opus 4.8. The GUI-only agent navigates Android Settings but cannot access the target SDK information required to compare candidate applications. In contrast, the CLI-augmented agent enumerates third-party apps, inspects their metadata through commands, and identifies the correct package.

It then verifies the generated `BrokenLinks.md` file containing the de-duplicated broken targets, successfully completing the task in 10 steps.

Claude Opus 4.8. In the GUI-only setting, it first browses the vault to enumerate the available notes and then opens individual notes one by one to inspect their wikilinks. Because the analysis must cover the entire vault, this requires substantial repetitive navigation and manual comparison, and the agent reaches the limit while still inspecting individual notes. In contrast, the CLI-augmented agent directly enumerates all Markdown files and uses a Python script to parse the entire vault, exclude code-block examples, resolve link targets, and identify the de-duplicated broken wikilinks. It then uses CLI to create the required `BrokenLinks` note, completing the task in 8 steps.

Overall, this case illustrates how CLI enables efficient global analysis over many files, while GUI-only interaction is poorly suited to repository-wide reasoning.

Case Study: Application Compatibility Checking. This task requires identifying the installed third-party application that requests the CAMERA permission and has the lowest target API version.

Claude Opus 4.8. As shown in Figure 11, the GUI-only agent opens Android Settings and navigates to the installed application list, but cannot access the target SDK information required for comparison through the standard GUI. As a result, it gives up on the task after only 4 steps. In contrast, the CLI-augmented agent first enumerates the installed third-party applications and then inspects their metadata through commands. By comparing the target SDK versions of applications requesting CAMERA permission, it identifies the correct package and successfully completes the task in 4 steps.

Seed 2.0 Pro. Seed also relies on navigating the Settings interface, but cannot obtain the target SDK information needed to solve the task and eventually gives up after 34 steps. With CLI access, Seed is able to retrieve application information, but it returns the wrong package. This contrast shows that while CLI makes the required metadata accessible, successful task completion still depends on correctly interpreting and comparing the retrieved information.

Overall, this case illustrates that CLI can provide access to system-level information that is unavailable through the standard GUI, thereby expanding the range of mobile tasks that agents can solve.