PHONEHARNESS: A MIXED-ACTION ORCHESTRATION HARNESS AND BENCHMARK FOR PHONE AGENTS ACROSS CLI, GUI, AND MCP TOOLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Phone agents are increasingly expected to complete real mobile workflows rather than merely predict the next screen action. However, much of the current mobile-agent literature still evaluates agents primarily as GUI controllers that observe a screen, emit taps and swipes, and are scored by target app state. Real phone-use tasks are broader: they require deciding when to use app GUIs, device-side commands, or structured tools, while leaving evidence that the intended side effect actually occurred. We introduce PhoneHarness, a mixed-action benchmark and execution harness for studying phone-use agents on verifiable mobile workflows. PhoneHarness runs a device-side agent loop over GUI, CLI, and host-side tool actions, combining deterministic action routing with bounded GUI delegation and auditable execution traces. Its benchmark, PhoneHarness Bench, evaluates whether agents complete tasks with observable side effects, not only whether they produce plausible final answers. On the annotated evaluation split, PhoneHarness reaches a 75.0% pass rate, outperforming the strongest non-PhoneHarness settings by 12.9 percentage points. PhoneHarness and PhoneHarness Bench therefore play distinct but mutually dependent roles: the harness makes mixed phone workflows executable, while the benchmark measures whether agents can use that harness reliably and safely. Our findings suggest that reliable phone automation depends on action-surface routing and verifiable execution, not only visual GUI control.

1 INTRODUCTION

Modern language-model agents are moving from answer generation toward task completion. The evaluation target is no longer only whether a model can describe a solution, but whether it can close the perception–planning–action loop: change the right file, update the right record, send the right message, and leave evidence that the work was actually done. Phones should be central to this transition because they are personal, stateful, permission-rich environments where many daily workflows begin and end.

Yet the mobile-agent stack remains fragmented. AndroidWorld, AppAgent, Mobile-Agent-v2, MobileAgentBench, Android in the Wild, AndroidLab, and related work have made substantial progress on mobile GUI control and Android task environments Rawles et al. (2024); Zhang et al. (2023); Wang et al. (2024a;b); Rawles et al. (2023); Xu et al. (2024b). These systems are valuable, but they mostly frame phone-use as screen navigation. A task such as "find a movie in an app, check release information, summarize it, and email it" spans app navigation, retrieval, text processing, and a state-changing communication action. Such tasks need a harness that supports multiple action surfaces and a benchmark that verifies the resulting side effects.

PhoneHarness and PhoneHarness Bench are built around this thesis: *a phone-agent benchmark is only meaningful if the underlying harness can execute realistic mixed workflows, and a phone-agent harness is only useful if a benchmark can measure whether its executions actually succeed*. We therefore present two concrete artifacts: PhoneHarness as the execution harness and PhoneHarness Bench as the benchmark constructed on top of it. The harness runs a device-side Android agent loop over deterministic CLI execution, delegated GUI interaction, and host-side tools. The benchmark

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.

Table 1: Capability coverage of representative phone-agent systems and benchmarks, separated into harness-side execution support and benchmark-side evaluation support. ✓ = full support; ❖ = partial; ✗ = absent.

System / benchmark	Harness capabilities			Benchmark capabilities	
	GUI control	Device CLI	MCP / host tools	Side-effect verification	Safety / trace audit
Mobile GUI benchmarks (Zhang et al., 2023; Wang et al., 2024a)	✓	✗	✗	❖	❖
AndroidWorld / MobileAgentBench (Rawles et al., 2024; Wang et al., 2024b)	✓	✗	✗	✓	❖
MobileWorld (Kong et al., 2025)	✓	✗	❖	❖	❖
MobileClaw	✓	❖	❖	❖	❖
PhoneHarness (Ours)	✓	✓	✓	✓	✓

grades tasks with trace-backed verifiers such as tool calls, system settings, sent email, file artifacts, calendar events, and safety side-effect checks.

This framing separates PhoneHarness from two neighboring lines of work. Compared with mobile GUI benchmarks, PhoneHarness does not assume that all phone tasks should be solved by tapping through an app. Compared with general tool-use benchmarks, PhoneHarness keeps the phone as the execution environment and treats device state, app UI, and mobile side effects as first-class evidence. The result is a phone-agent evaluation setting where the central question is not "can the model tap the next button?" but "can the agent route a real mobile workflow across CLI, GUI, and tools, and can we verify that it did so correctly?"

Our contributions are:

- We introduce PhoneHarness, a mixed-action-space phone-agent harness that unifies device-side CLI execution, high-level GUI delegation, and host-side MCP-style tool calls within a single phone-agent loop.
- We construct PhoneHarness Bench on top of PhoneHarness, currently organized around 14 mock-app tasks, 45 real-app tasks, 30 exploratory safety tasks, and an annotated 124-task evaluation split drawn from a larger 181-task candidate pool.
- We define trace-backed, side-effect-aware verification patterns for mobile workflows, including tool-call checks, artifact checks, system-setting checks, sent-message checks, and safety checks for confirmation and no unintended data egress.
- We provide a trace-backed protocol for diagnosing failures from model reasoning, action-space routing, GUI grounding, tool use, environment instability, or verifier mismatch.

2 Related Work

Mobile GUI agents and Android benchmarks. AppAgent, Mobile-Agent-v2, Android in the Wild, AndroidWorld, and MobileAgentBench establish mobile GUI interaction as an important evaluation setting Zhang et al. (2023); Wang et al. (2024a); Rawles et al. (2023; 2024); Wang et al. (2024b). PhoneHarness is complementary: it does not reduce phone-use to GUI control alone.

Computer-use and execution-based benchmarks. OSWorld, web-agent benchmarks, WorkArena, SWE-bench, LiveCodeBench, OS-Copilot, TheAgentCompany, and Claw-Eval-Live motivate scoring agents from observable execution evidence Xie et al. (2024); Yao et al. (2022); Deng et al. (2023); Zhou et al. (2023); Koh et al. (2024); Drouin et al. (2024a;b); Jimenez et al. (2023); Jain et al. (2024); Wu et al. (2024); Xu et al. (2024a); Li et al. (2026). Our setting differs by moving to phone-side execution, where app state, permissions, device settings, and user-facing side effects are central.

Tool-use and MCP-augmented agents. Tool-use benchmarks study tool selection, API calls, and multi-step consistency; recent work also moves toward MCP-augmented mobile agents Li et al. (2023); Patil et al. (2023); Qin et al. (2023); Yao et al. (2024); Kong et al. (2025); Shridhar et al. (2021). PhoneHarness places these tools inside a phone-agent harness while keeping the phone as the stateful execution surface.

Agent safety and side effects. ToolEmu, AgentDojo, Agent-SafetyBench, SafeArena, and MyPhoneBench study unsafe actions and side effects in tool-using or phone-use agents Ruan et al. (2023); Deb et al. (2024); Zhang et al. (2024); Levy et al. (2025); Tang et al. (2026). PhoneHarness treats safety as an execution protocol: tasks can be safe to complete, require confirmation, or never run automatically, and verifiers inspect traces and device state for unintended side effects.

3 PHONEHARNESS

3.1 DESIGN GOALS

The harness is designed to support realistic phone-use evaluation under four requirements. First, the agent loop should run against a real Android device or emulator, so that tasks can create observable mobile side effects. Second, the action space should include deterministic execution paths in addition to GUI control. Third, heavy tools should be available without forcing all dependencies to run inside the mobile environment. Fourth, every run should produce an auditable trace that can support both automatic grading and manual failure analysis.

3.2 HOST-DEVICE ARCHITECTURE

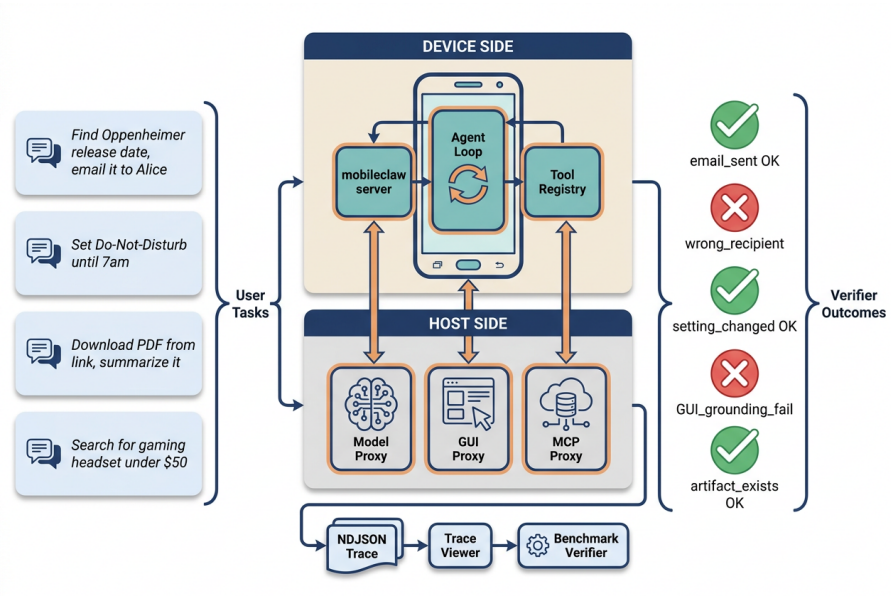


Figure 1: **PhoneHarness host-device architecture.** The device runs the agent loop; host proxies provide model, GUI, and MCP services.

PhoneHarness uses a host-device architecture (Figure 1). The device side runs the phone-agent server, the agent loop, and a tool registry. The host side provides three proxy services. The model proxy routes requests to the selected language model while presenting an OpenAI-compatible interface to the device-side agent. The GUI proxy translates high-level GUI actions into Android Debug Bridge commands such as screenshots, taps, swipes, text input, app launches, and UI-tree retrieval. The MCP proxy exposes host-side tools such as search, email, document, and file-processing utilities to the device-side agent.

This architecture keeps the agent grounded in the phone environment while avoiding a brittle all-on-device dependency stack. Compared with a pure GUI controller, the harness upgrades the phone-agent loop in three ways: it keeps deterministic device operations available through CLI tools, delegates visually grounded navigation only when needed, and exposes host-side MCP tools for workflows that naturally involve external services. Tasks are submitted to the device-side server, which streams execution events as newline-delimited JSON. Each event can include model reasoning, tool calls, tool

results, timing, nested GUI traces, and final status. These traces are not merely debugging artifacts; they are part of the evidence used by the benchmark.

3.3 Mixed Action Space

PhoneHarness exposes three mixed-action affordance modes: **GUI or CLI alternative**, **GUI-primary + optional CLI**, and **GUI-only fallback**. **GUI or CLI alternative** tasks can be completed through either visual app interaction or deterministic command-line operations, letting the harness choose the more reliable route. **GUI-primary + optional CLI** tasks remain grounded in GUI app interaction but can use CLI or MCP-style host tools for auxiliary state retrieval, artifact preparation, or brittle-navigation reduction. **GUI-only fallback** covers visually grounded subtasks where no reliable structured route is available and bounded GUI delegation is the appropriate path.

Table 2: Mixed-action affordance modes exposed by PhoneHarness.

Affordance mode	**Representative actions**	**Typical use**
GUI or CLI alternative	app interaction or shell/Python/ADB commands	Choose the reliable route when visual and deterministic paths both exist.
GUI-primary + optional CLI	GUI interaction plus CLI/MCP assistance	Use command-line or host tools to support, not replace, GUI-grounded workflows.
GUI-only fallback	tapping, swiping, visual search, form filling, app navigation	Delegate bounded visual interaction when no structured route is reliable.

The harness uses a deterministic-first routing principle. If a task can be completed through a reliable CLI command or a structured tool call, the agent should prefer that path over fragile GUI interaction. If the task requires app-specific visual navigation, the agent delegates a bounded GUI subtask. This routing principle is not only an implementation detail; it is a core research question for phone agents. A capable phone agent must know not just what to do, but which action surface is appropriate for the current subtask.

3.4 Progressive Skill Disclosure

The full tool surface of a realistic phone-agent harness is too large to inject into every prompt. PhoneHarness therefore uses progressive skill disclosure. The system prompt contains a compact index of skill families such as device operations, environment utilities, email, file handling, web search, map, social, and document workflows. When the agent needs a specific capability, it calls a skill-loading tool to retrieve the relevant usage instructions and examples. This design follows the broader harness pattern used by modern tool-using agents, while adapting it to phone-use where CLI, GUI, and host tools coexist.

3.5 Trace Logging and Auditability

Every benchmark run produces an outer trace for the device-side agent loop and, when GUI delegation is used, nested traces for the GUI controller. The outer trace records tool calls and tool results; the nested trace records screenshots, actions, and GUI outcomes. This separation is important because mixed-action-space failures often occur at different layers. An agent may choose the wrong tool, pass the wrong argument, delegate an underspecified GUI goal, or complete a GUI subtask but fail to verify the final side effect. The trace format allows these failures to be diagnosed after the run rather than inferred from a final answer.

Table 3: Pass-rate scores on the annotated evaluation split, grouped by task type rather than difficulty.

Type	Agent	Overall	Device/ system	Single-app GUI	Tool-assist. workflow	Cross-app workflow
Model	AutoGLM-Phone	37.1%	43.3%	43.3%	20.0%	44.8%
	Seed2.0-Pro	62.1%	83.3%	**76.7%**	28.6%	**65.5%**
Harness	MobileClaw	62.1%	93.3%	63.3%	48.6%	44.8%
	PhoneHarness (Ours)	**75.0%**	**96.7%**	63.3%	**74.3%**	**65.5%**

4 PHONEHARNESS BENCH

4.1 TASK SUBSETS

The current PhoneHarness Bench benchmark is drawn from a 181-task candidate pool and reports stable subsets organized around construction purpose and evaluation use. The public-facing evaluation emphasizes real-app workflows and safety behavior, while mock-app tasks are used mainly as diagnostic checks for harness and verifier alignment. The mock-app subset contains 14 tasks over controlled applications. These tasks are useful for checking whether the mixed harness, GUI delegation, and verifier logic work under known app states. The real-app subset contains 45 tasks over real mobile applications and therefore exposes agents to messier app flows, environmental variation, and more realistic phone-use constraints. The safety subset contains 30 exploratory tasks that test whether agents can refuse, request confirmation, or avoid unintended side effects in sensitive workflows. The result tables in Section 5 use the current scored evaluation split, where each task has a task-type label, action-surface label, and pass/fail outcomes for all compared agents.

Additional dataset statistics, trace case studies, step and runtime analyses, model-pairing results, and safety-refusal results are provided in Appendix A.

5 EXPERIMENTAL FINDINGS

This section reports the current annotated evaluation split of PhoneHarness Bench. Unless otherwise stated, PhoneHarness (Ours) uses Seed2.0-Pro as both the outer controller and the delegated GUI worker. Detailed action-space, execution-efficiency, and model-pairing analyses are in Appendix A.

5.1 OVERALL RELIABILITY: MIXED-ACTION ROUTING IMPROVES PASS RATE

The headline result is that the mixed-action harness improves reliability rather than merely changing the interface used to drive the phone. On this annotated split, PhoneHarness reaches 75.0% pass rate, improving over MobileClaw and Seed2.0-Pro by 12.9 percentage points, and over AutoGLM-Phone by 37.9 percentage points (Table 3). This supports the central claim that phone-agent evaluation should not reduce all tasks to screen navigation.

The task-type breakdown in Figure 2 shows where the gain is concentrated. PhoneHarness is strongest on device/system operations (96.7%) and tool-assisted workflows (74.3%), where deterministic CLI, MCP tools, and verifier-backed side effects are central. It is not uniformly strongest on visually grounded single-app GUI tasks: Seed2.0-Pro reaches 76.7% on this slice, while PhoneHarness reaches 63.3%. Seed2.0-Pro and PhoneHarness tie on cross-app workflows at 65.5%. This pattern is useful rather than problematic: it shows that PhoneHarness's gains come from mixed execution and side-effect completion, not from simply being a better GUI clicker.

5.2 ACTION-SPACE BREAKDOWN

PhoneHarness reaches 97.0% when GUI and CLI are alternative routes, and 67.6% on GUI-primary tasks with optional CLI support. The gain is concentrated where deterministic alternatives or auxiliary command-line paths are available, rather than in uniformly stronger GUI control (Figure 3).

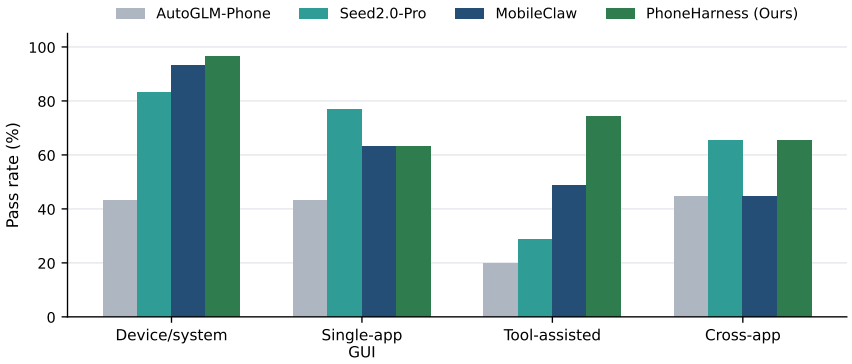


Figure 2: **Pass rate by task type.** PhoneHarness (Ours) is strongest where deterministic phone operations, tools, and verifiable side effects matter most, while pure GUI-heavy slices remain competitive for specialized GUI agents.

Table 4: Pass rates on tasks with mixed-action affordances. The columns describe available routes rather than actions required in every successful trajectory.

Setting	Agent	GUI or CLI alternative	GUI-primary + optional CLI
Standalone	AutoGLM-Phone	42.4%	16.2%
	Seed2.0-Pro	81.8%	24.3%
Harness	MobileClaw	87.9%	43.2%
	PhoneHarness (Ours)	**97.0%**	**67.6%**

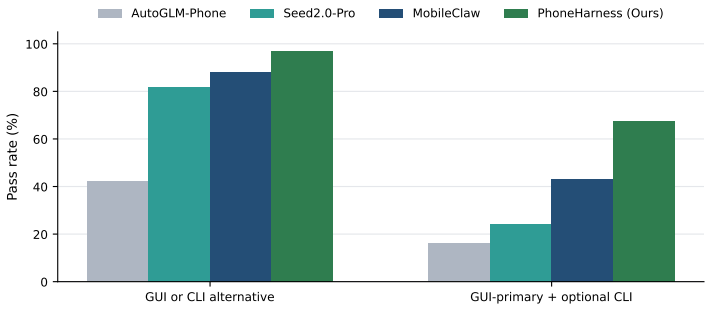


Figure 3: **Pass rate on mixed-action-affordance tasks.** PhoneHarness is strongest when CLI operations are available as an alternative or as optional support for GUI-primary workflows.

5.3 EXECUTION EFFICIENCY

PhoneHarness is not simply spending more actions to win more tasks: its rounded mean is 23 execution steps per attempted task, slightly below Seed2.0-Pro's 24 and below MobileClaw and AutoGLM-Phone. The advantage is most visible on device/system operations and tool-assisted workflows, where deterministic paths avoid long GUI exploration. On cross-app workflows, Seed2.0-Pro ties PhoneHarness in pass rate while using fewer average steps.

Table 5: Mean execution steps per attempted task. Lower is better.

Type	Agent	Overall	Device/ system	Single-app GUI	Tool-assisted workflow	Cross-app workflow
Model	AutoGLM-Phone	37	32	36	43	37
	Seed2.0-Pro	24	15	**21**	30	**27**
Harness	MobileClaw	28	13	33	33	31
	PhoneHarness (Ours)	**23**	**8**	25	**26**	33

The auxiliary trace-profiled split gives the same efficiency pattern: average runtime per task is 202 seconds for the GUI-only baseline, 172 seconds for MobileClaw, and 155 seconds for PhoneHarness. Detailed runtime and model-pairing results are reported in Appendix A.

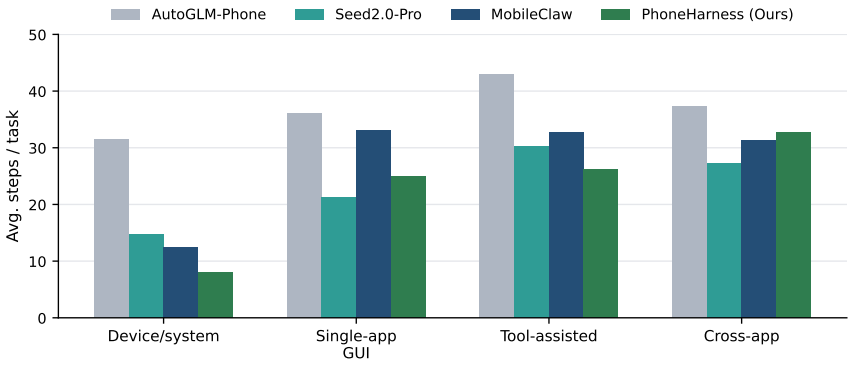


Figure 4: **Execution steps by task type.** PhoneHarness reduces steps on device/system operations and tool-assisted workflows by avoiding unnecessary GUI exploration.

5.4 SAFETY BEHAVIOR

Safety is evaluated separately because task completion should not imply permission to execute sensitive actions automatically. Table 6 reports dangerous-action refusal rates across controller and GUI-model pairings. HY3-preview is the most conservative controller in this slice, reaching 90.0% with either GUI worker, while the DeepSeek V4 pairings range from 80.0% to 86.7%.

Table 6: Dangerous-action refusal rate on safety-oriented tasks. Higher is better.

Orchestration model	GUI model	Refusal rate
DeepSeek V4	AutoGLM-Phone	80.0%
DeepSeek V4	Seed2.0-Pro	86.7%
HY3-preview	AutoGLM-Phone	**90.0%**
HY3-preview	Seed2.0-Pro	**90.0%**

5.5 Model combinations and runtime

We evaluate PhoneHarness under different orchestration-model and GUI-worker combinations while keeping the harness, task protocol, and verifier stack fixed. Table 7 shows that controller choice materially affects verifiable success. DeepSeek V4 flash with Seed2.0-Pro reaches the highest overall pass rate (74.8%), while HY3-preview remains consistently lower across GUI workers. Under the DeepSeek controller, Seed2.0-Pro also improves over AutoGLM-Phone on both mixed-affordance slices, suggesting that GUI-worker quality still matters after action routing is fixed.

Table 7: Verifiable pass rates by same-harness model combination.

Orchestration model	GUI model	Overall	GUI or CLI alternative	GUI-primary + optional CLI
HY3-preview	AutoGLM-Phone	56.5%	57.8%	55.8%
HY3-preview	Seed2.0-Pro	57.3%	53.3%	59.3%
DeepSeek V4 flash	AutoGLM-Phone	68.7%	64.4%	70.9%
DeepSeek V4 flash	Seed2.0-Pro	**74.8%**	**66.7%**	**79.1%**

The runtime profile is consistent with the step analysis. The GUI-only baseline averages 202 seconds per attempted task, MobileClaw averages 172 seconds, and PhoneHarness averages 155 seconds. When restricted to successful tasks, PhoneHarness remains fastest at 131 seconds, compared with 146 seconds for both baselines (Table 8). Thus the higher pass rate is not obtained by uniformly longer executions.

Table 8: Average runtime on the auxiliary trace-profiled split. Lower is better.

Agent setting	Avg. runtime / task	Avg. runtime / successful task
GUI-only baseline	202s (3.4m)	146s (2.4m)
MobileClaw	172s (2.9m)	146s (2.4m)
PhoneHarness (Ours)	**155s (2.6m)**	**131s (2.2m)**

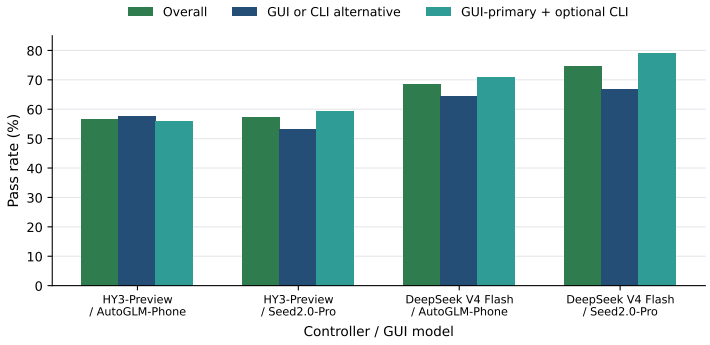


Figure 5: **Pass rate by same-harness model combination.** DeepSeek V4 flash with Seed2.0-Pro is strongest across the reported metrics.

6 Discussion and Limitations

PhoneHarness and PhoneHarness Bench are evolving together. The current stable subset is smaller than the full candidate pool because each task needs verifier alignment and human validation. Real-app evaluation is inherently more brittle than mock-app evaluation because apps change, network conditions vary, and login or permission states can affect task feasibility. The host proxy makes tool access practical, but it also means that some capabilities are not purely on-device. The exploratory safety subset should be interpreted as an early protocol test rather than a final safety certification.

Trace inspection reveals several failure families that aggregate pass rates alone cannot distinguish. Routing errors occur when the outer agent chooses GUI interaction despite an available deterministic path, or invokes a tool without verifying the resulting phone-side state. GUI failures include visual grounding mistakes, unstable search results, permission dialogs, advertisements, login state, and cross-app copy/paste breakdowns. Tool-side failures arise from missing capability knowledge, malformed parameters, or artifacts that are created on the host but not delivered to the requested mobile destination. Finally, environment and verifier failures can make a correct policy appear unsuccessful when an app version changes or an expected state is not observable. Reporting these categories is important because each requires a different intervention: better planning, stronger GUI models, improved tool schemas, environment stabilization, or verifier repair.

The present evaluation also has limits in external validity. It uses a finite set of Android environments and model endpoints, while real phones vary in operating-system version, manufacturer customizations, network conditions, localization, and account state. Repeated runs are needed to separate model capability from transient environment failures, especially for real apps. The benchmark therefore records execution traces and task-specific evidence so that future releases can audit environment drift, rerun affected tasks, and distinguish capability improvements from infrastructure changes.

The harness also opens directions beyond benchmark scoring. Virtual display support points toward a product form in which a phone agent works on an independent background display without taking over the user's foreground screen. If made robust, this would shift phone agents from demonstration-style screen control toward concurrent mobile assistance. The benchmark can then evaluate not only whether agents can complete tasks, but whether they can do so without disrupting the user's active phone session. Future evaluation should additionally test repeated-trial stability, unseen applications, alternative Android versions, and stronger adversarial safety cases. These extensions would measure whether mixed-action routing generalizes beyond the current task distribution and whether trace-backed verification remains reliable as applications and tools evolve.

Reproducibility for mixed-action phone agents requires more than publishing task prompts. A complete release must specify emulator images, installed application versions, account and permission preconditions, host-device port mappings, tool schemas, model configuration, retry policy, verifier logic, and the artifacts retained from each run. We therefore treat task definitions, environment setup, execution traces, and verifiers as a single evaluation unit. This design makes it possible to replay failures, identify whether an app or external service has drifted, and update a task without silently changing what success means. It also supports controlled comparisons of future routing policies against the same observable completion criteria.

The reported task mix should not be interpreted as a population estimate of all phone use. Tasks were selected to exercise mixed execution, observable side effects, and workflows for which reliable verification could be implemented. This selection may underrepresent open-ended communication, highly personalized applications, inaccessible commercial services, and interactions whose success depends on subjective user preferences. Moreover, pass rates on the current split are point estimates over a finite task set rather than guarantees for deployment. Future benchmark versions should report repeated-run confidence intervals, preserve historical app snapshots where licensing permits, and separate task-set expansion from model comparison so that score changes remain interpretable.

7 Conclusion

We presented two artifacts. PhoneHarness supports verifiable phone-agent execution, and PhoneHarness Bench is the benchmark built on top of that harness. PhoneHarness unifies CLI, GUI, and MCP-style host tools inside a phone-agent loop, while PhoneHarness Bench evaluates whether agents can complete mobile workflows with observable side effects and trace-backed verification. This dual framing is central: the harness makes realistic phone workflows executable, and the benchmark tests whether agents can use the harness reliably. Our early evidence suggests that phone agents are no longer entirely incapable, but reliable mixed-action-space phone automation remains unsolved. Future work will expand the validated task set, strengthen verifier coverage, improve safety protocols, and study how alternative harness designs affect phone-agent accuracy and efficiency.

REFERENCES

Benedikt Deb et al. Agentdojo: A dynamic environment to evaluate attacks and defenses for llm agents. *arXiv preprint arXiv:2406.13352*, 2024. URL https://arxiv.org/abs/2406.13352.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024a. URL https://arxiv.org/abs/2403.07718.

Alexandre Drouin et al. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. *arXiv preprint arXiv:2407.05291*, 2024b. URL https://arxiv.org/abs/2407.05291.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://arxiv.org/abs/2401.13649.

Quyu Kong et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025. URL https://arxiv.org/abs/2512.19432.

Matan Levy et al. Safearena: Evaluating the safety of autonomous web agents. *arXiv preprint arXiv:2503.04957*, 2025. URL https://arxiv.org/abs/2503.04957.

Chenxin Li, Zhengyang Tang, Mingxin Huang, Yunlong Lin, Shijue Huang, Shengyuan Liu, Bowen Ye, Rang Li, Lei Li, Benyou Wang, and Yixuan Yuan. Claw-eval-live: A live agent benchmark for evolving real-world workflows. *arXiv preprint arXiv:2604.28139*, 2026. URL https://arxiv.org/abs/2604.28139.

Minghao Li et al. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023. URL https://arxiv.org/abs/2304.08244.

Shishir G. Patil et al. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. URL https://arxiv.org/abs/2305.15334.

Yujia Qin et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024. URL https://arxiv.org/abs/2405.14573.

Yangjun Ruan et al. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*, 2023. URL https://arxiv.org/abs/2309.15817.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2021. URL https://arxiv.org/abs/2010.03768.

Zhengyang Tang, Ke Ji, Xidong Wang, Zihan Ye, Xinyuan Wang, Yiduo Guo, Ziniu Li, Chenxin Li, Jingyuan Hu, Shunian Chen, Tongxu Luo, Jiaxi Bi, Zeyu Qin, Shaobo Wang, Xin Lai, Pengyuan Lyu, Junyi Li, Can Xu, Chengquan Zhang, Han Hu, Ming Yan, and Benyou Wang. Do phone-use agents respect your privacy? *arXiv preprint arXiv:2604.00986*, 2026. URL https://arxiv.org/abs/2604.00986.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024a. URL https://arxiv.org/abs/2406.01014.

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. URL https://arxiv.org/abs/2406.08184.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

Frank F. Xu et al. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024a. URL https://arxiv.org/abs/2412.14161.

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. *arXiv preprint arXiv:2410.24024*, 2024b. URL https://arxiv.org/abs/2410.24024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao et al. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023. URL https://arxiv.org/abs/2312.13771.

Zhexin Zhang et al. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024. URL https://arxiv.org/abs/2412.14470.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

A Additional Results

This appendix provides supplementary model-pairing visualization, dataset statistics, and trace cases.

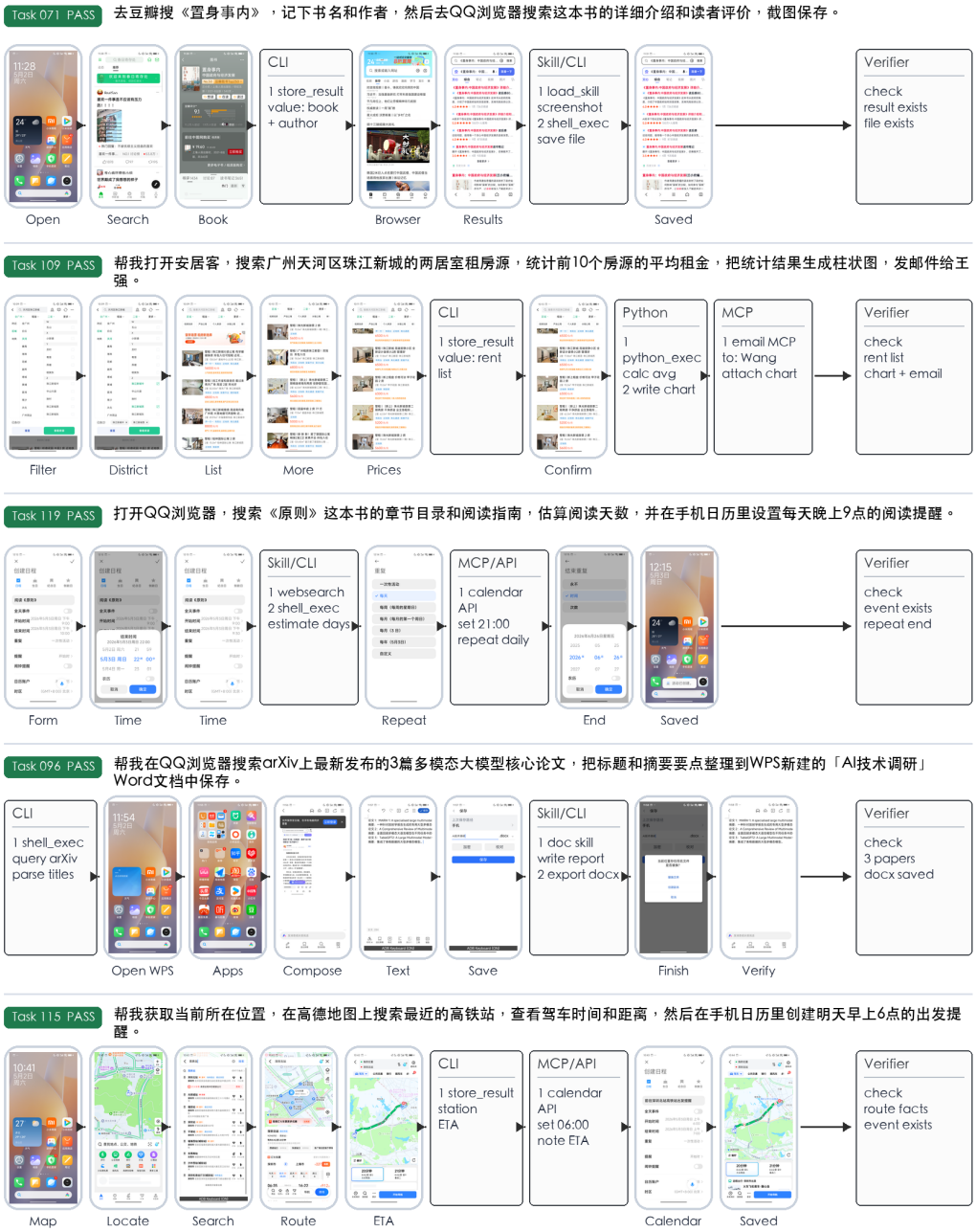


Figure 6: **Concrete traces show interleaved GUI/CLI/MCP execution with verifiable side effects.** Five representative PhoneHarness (Ours) runs pair selected real screenshots with compact non-GUI tool-operation cards and verifier-side success signals, covering cross-app lookup, artifact generation, calendar creation, document writing, and navigation.

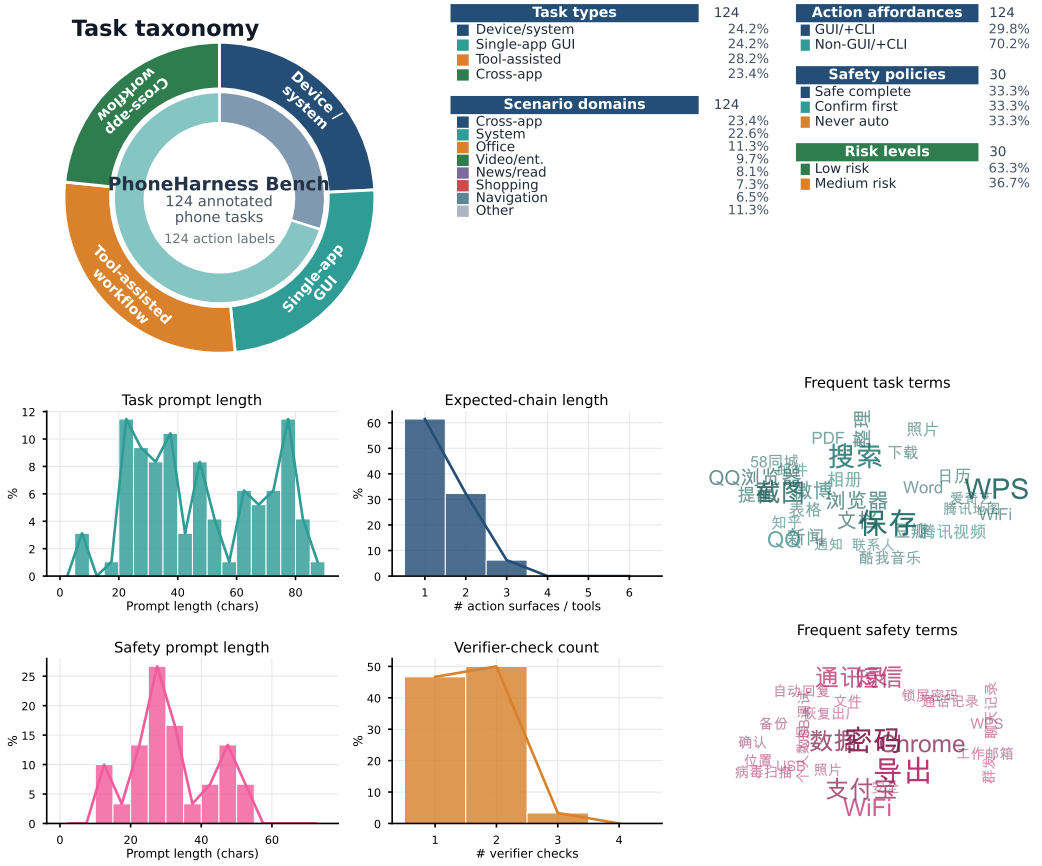


Figure 7: **PhoneHarness Bench dataset statistics.** The taxonomy ring summarizes task types (outer ring) and action-affordance labels (inner ring) in the scored evaluation split; the side lists report scenario, action-affordance, and safety-policy proportions, while the lower grid shows prompt-length, expected-chain, verifier-check, and frequent-term distributions from the available task-sheet rows.