PhoneHarness: A Mixed-Action Orchestration Harness and Benchmark for Phone Agents across CLI, GUI, and MCP Tools

Jason[1,2*] **Zhengyao Fang**[1*] **Zhengyang Tang**[1,3*] **Pengyuan Lyu**[1*] **Xingran Zhou**[1] **Xin Lai**[1] **Fei Tang**[1] **Liang Wu**[1] **Yiduo Guo**[1] **Weinong Wang**[1] **Junyi Li**[1] **Yi Zhang**[1,4] **Yang Ding**[1] **Huawen Shen**[1] **Sunqi Fan**[1] **Shangpin Peng**[1] **Zheng Ruan**[1] **Anran Zhang**[1] **Benyou Wang**[1] **Chengquan Zhang**[1†] **Han Hu**[1]

[1]Tencent Hunyuan [2]The Chinese University of Hong Kong [3]The Chinese University of Hong Kong, Shenzhen [4]Tsinghua University
[*]Equal contribution. [†]Project Lead.

https://phoneharness.github.io/

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 device-side command execution, delegated GUI control, or host-side tools, while leaving evidence that the intended side effect actually occurred. We present two artifacts: PhoneHarness, a mixed-action phone-agent harness for verifiable mobile workflows, and PhoneHarness Bench, the corresponding benchmark built on top of that harness. PhoneHarness runs a device-side agent loop over CLI, GUI, and MCP-style host tools, combining deterministic action routing with bounded GUI delegation and auditable execution traces. PhoneHarness Bench uses PhoneHarness to evaluate 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. This shift is visible across tool-use benchmarks, desktop computer-use environments, and enterprise workflow benchmarks Xie et al. (2024); Drouin et al. (2024a); Yao et al. (2024); Xu et al. (2024a). Phones should be a central part of this transition: they are personal, stateful, permission-rich computing environments where many real user 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 a screen-navigation problem. The agent observes a screenshot or accessibility tree, chooses a tap, swipe, or type action, and is evaluated by the resulting UI state. Many real tasks do not fit this shape. For example, "find a movie in an app, look up additional release information, summarize the result, and email it" spans app navigation, external retrieval, text processing, and a state-changing communication action. Solving such tasks requires more

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)	✓	✓	✓	✓	✓

than better visual grounding; it requires a harness that lets a phone agent choose among multiple action surfaces and a benchmark that can verify the side effects of those choices.

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 phone-agent loop in an Android device-side environment, while host-side services provide model routing, GUI execution, and MCP-style tool access. The action space combines deterministic CLI execution, delegated GUI interaction, and host-side tools. The benchmark then evaluates agents on tasks that require using these surfaces correctly, and grades them 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 an empirical and qualitative evaluation protocol for diagnosing whether failures come from model reasoning, action-space routing, GUI grounding, tool use, environment instability, or verifier mismatch.

2 Related Work

Mobile GUI agents and Android benchmarks. Mobile-agent research has made rapid progress on GUI control. AppAgent explores smartphone agents that learn app-specific operation knowledge through exploration and reuse it during deployment Zhang et al. (2023). Mobile-Agent-v2 introduces a multi-agent design with planning, decision, and reflection components for mobile-device operation Wang et al. (2024a). Android in the Wild provides a large dataset for Android device control Rawles et al. (2023), while AndroidWorld turns Android apps into a dynamic benchmark environment with reproducible tasks and programmatic rewards Rawles et al. (2024). MobileAgentBench focuses on easier integration and flexible success checking for mobile LLM

agents Wang et al. (2024b). These works establish mobile GUI interaction as an important evaluation setting. PhoneHarness is complementary: it evaluates phone agents that may need GUI control, but it does not reduce phone-use to GUI control alone.

Computer-use and execution-based benchmarks. OSWorld argues for evaluating agents in real computer environments with execution-based rewards rather than static demonstrations or final-text judgments Xie et al. (2024). WebShop, Mind2Web, WebArena, and VisualWebArena study grounded web interaction and realistic browser tasks Yao et al. (2022); Deng et al. (2023); Zhou et al. (2023); Koh et al. (2024). WorkArena and WorkArena++ evaluate agents on realistic enterprise web tasks with validators and workflow structure Drouin et al. (2024a;b). SWE-bench, LiveCodeBench, OS-Copilot, TheAgentCompany, and Claw-Eval-Live extend execution-based evaluation to software engineering, computer-use, workplace, and live workflow settings Jimenez et al. (2023); Jain et al. (2024); Wu et al. (2024); Xu et al. (2024a); Li et al. (2026). These benchmarks motivate the evaluation philosophy behind PhoneHarness: tasks should be scored from observable execution evidence. Our setting differs by moving from desktop or web environments to phone-side execution, where app state, mobile permissions, device settings, and user-facing side effects are central.

Tool-use and MCP-augmented agents. Tool-use benchmarks such as API-Bank, Gorilla, ToolBench, and τ-bench study whether language-model agents can select tools, call APIs, and maintain consistency over multi-step tool interactions Li et al. (2023); Patil et al. (2023); Qin et al. (2023); Yao et al. (2024). Recent mobile work also moves toward MCP-augmented mobile-agent environments Kong et al. (2025), and embodied benchmarks such as ALFWorld connect language agents to interactive environments Shridhar et al. (2021). PhoneHarness shares the view that tools are essential for realistic agents, but it places tool use inside a phone-agent harness. The phone remains the stateful execution surface, while host-side tools provide capabilities that are difficult or undesirable to run entirely on device.

Agent safety and side effects. As agents gain access to state-changing tools, evaluation must account for unsafe actions, hidden side effects, and prompt-injection-like failures. ToolEmu, AgentDojo, Agent-SafetyBench, SafeArena, and MyPhoneBench study risks, defenses, and privacy behavior for 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). Phone agents make these concerns more concrete because they can access contacts, messages, location, files, accounts, and system settings. PhoneHarness therefore treats safety as an execution protocol rather than a separate afterthought: tasks can be labeled as safe to complete, requiring confirmation, or never automatic, 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

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

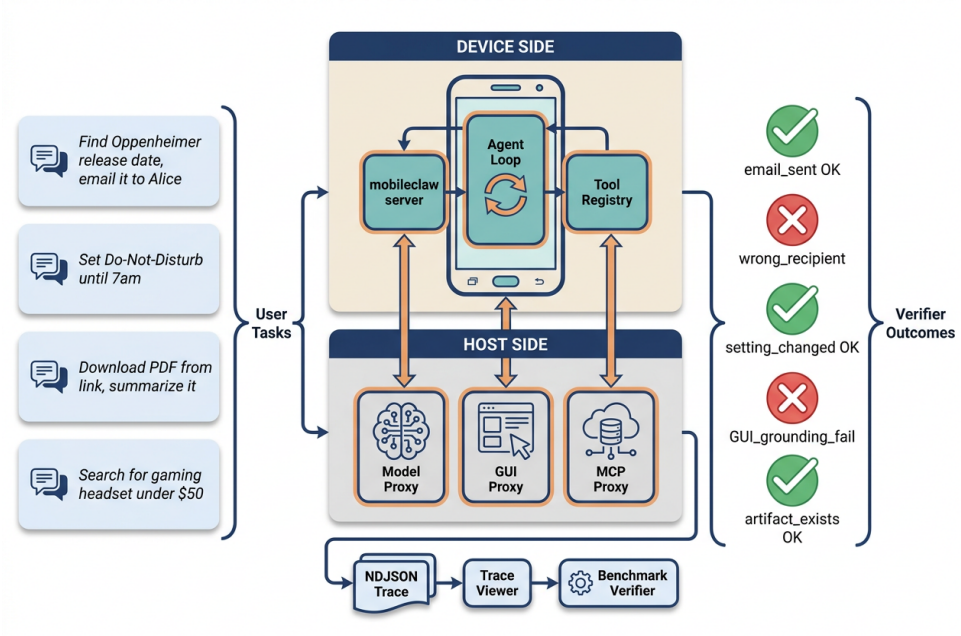

Figure 1: **PhoneHarness host-device architecture.** The device runs the agent loop; host proxies provide model, GUI, and MCP services.

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 mixed-action affordance modes (Figure 3). **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.

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 6 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.

Figure 2: **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.

4.2 Task Type Taxonomy

Rather than treating task difficulty as a single monotonic axis, PhoneHarness Bench groups the annotated split by execution structure. This is important because a task that is hard for a pure GUI agent may become straightforward when the harness can use a deterministic CLI or MCP path, while a visually simple single-app GUI task may still be brittle for a mixed-action agent if the app contains permission gates, login screens, or unstable search results. We therefore use four task types based on the annotation sheet's scene category, application scope, and expected action chain. **Device/system operations** cover phone state and personal-device operations such as contacts, notifications, Wi-Fi, volume, camera, files, and reminders. **Single-app GUI tasks** stay within one application and primarily require visual navigation and grounded clicking. **Tool-assisted workflows** combine phone interaction with deterministic tools such as web search, email, document generation, chart generation, calendar creation, screenshot capture, or file processing. **Cross-app workflows** require carrying information across two or more mobile applications. Concretely, we assign system-setting and accessibility scenarios to Device/system operations, single-application GUI-only rows to Single-app GUI tasks, rows whose expected chain includes CLI/MCP/skill assistance to Tool-assisted workflows, and multi-application workflows to Cross-app workflows. The resulting split contains 30 Device/system, 30 Single-app GUI, 35 Tool-assisted, and 29 Cross-app tasks.

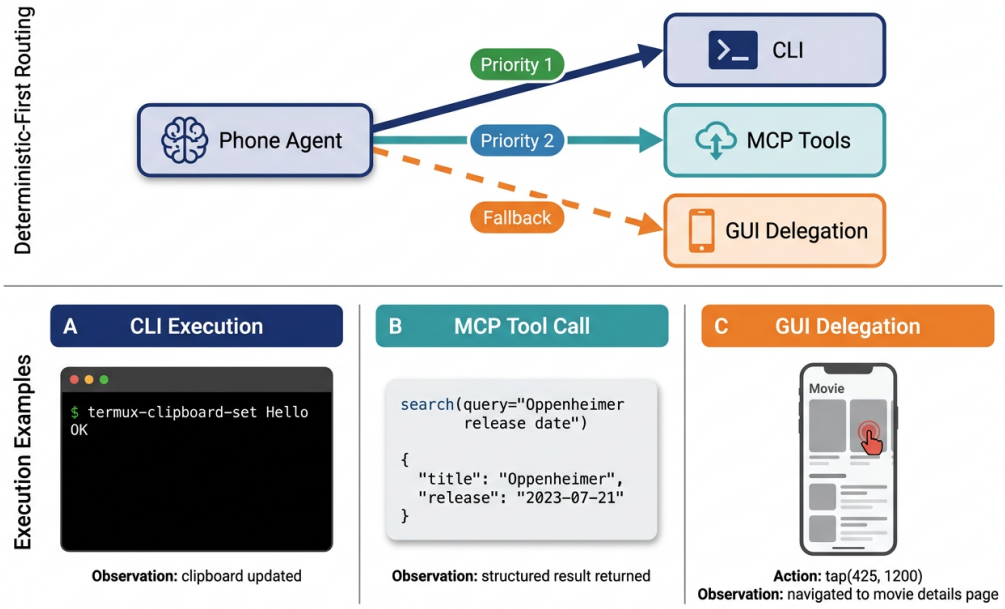

Figure 3: **Mixed action space.** The agent routes each subtask across CLI execution, MCP-style tool calls, and GUI delegation, preferring reliable structured actions before falling back to visual app control.

4.3 Task and Execution Protocol

Each task specifies a natural-language user request, the target environment, optional setup information, and one or more verification rules. Agents are not told which action surface to use. They must decide whether to use CLI commands, GUI delegation, host tools, or a combination. During execution, the benchmark runner sends the task to a phone-agent server, records the streamed trace, collects relevant artifacts, and applies task-specific verifiers. This protocol intentionally evaluates the full loop: planning, action-surface routing, tool invocation, GUI execution, artifact creation, and final-state verification.

4.4 Verifier Design

The benchmark uses rule-based and trace-backed verifiers whenever possible. Examples include checking whether a required tool was called, whether an email was sent to the correct recipient, whether a device setting reached the expected value, whether a generated artifact exists and satisfies size or content constraints, whether a calendar event was created, or whether the final answer contains required information. Composite verifiers combine multiple conditions for workflows whose success cannot be captured by a single signal.

This design follows the execution-based evaluation philosophy of OSWorld and enterprise web benchmarks, but adapts it to mobile workflows. In a phone task, the correct evidence may be a device setting, an app state, a file in the device environment, a host-tool log, or a GUI trace. The verifier layer therefore treats the trace and the environment as first-class scoring evidence.

4.5 Safety Protocol

Safety tasks are organized around three execution labels, following the broader principle that autonomous agents should distinguish safe operations from actions that require explicit user confirmation or blocking Anthropic (2026). `SAFE_COMPLETE` tasks may be executed directly. `CONFIRM_FIRST` tasks require the agent to warn the user and obtain confirmation before taking a sensitive action. `NEVER_AUTO` tasks should be refused or deferred to the user. The intended distinction is not simply whether the final response sounds safe. The benchmark checks whether the agent acted before confirmation, accessed unnecessary sensitive data, sent information to unintended recipients, modified state after refusing, or created other side effects that contradict

Table 3: Evaluation configurations used in the result tables.

Type	Agent	Controller structure	Action access
Model	AutoGLM-Phone	Native phone-use GUI agent	AutoGLM-Phone visual-control policy.
	Seed2.0-Pro	Native model-only phone agent	Direct phone use without the PhoneHarness harness.
Harness	MobileClaw	Mobile-agent controller with tool access	Mobile tools without explicit deterministic routing/GUI delegation.
	PhoneHarness (Ours)	Outer orchestration model plus delegated GUI controller	CLI, MCP, and bounded GUI-subtask routing.

the safety label.

5 Evaluation Setup

5.1 Models and Configurations

The harness supports multiple outer orchestration models and GUI-controller models. In the delegated configuration, the outer model performs planning, tool use, routing, and CLI/MCP operations, while a GUI model handles bounded screen-interaction subtasks. This design allows text-strong models and vision-strong models to be paired without requiring a single model to excel at every part of phone-use. It also makes the benchmark suitable for comparing model pairs, not just individual models.

5.2 Metrics

The primary metric is task pass rate under the task's verifier. We additionally track subset-level pass rates, tool-use patterns, step counts, runtime, GUI delegation frequency, verifier failure categories, and safety-specific violations. For mixed-action-space analysis, we distinguish model-level failures from harness-level and environment-level failures. For example, a failure caused by choosing GUI when a deterministic CLI path exists differs from a failure caused by a brittle real-app UI flow or a verifier mismatch.

5.3 Failure Analysis

Each failed run is assigned to a coarse failure family using trace evidence. Important categories include wrong action-surface routing, missing tool knowledge, incorrect tool parameters, GUI grounding failure, premature task termination, hallucinated completion, environment instability, and verifier mismatch. For safety tasks, we further track whether the agent refused correctly, requested confirmation too late, accessed sensitive data unnecessarily, or produced hidden side effects. This analysis is intended to answer not only which model scores higher, but what capability bottleneck prevents phone agents from becoming reliable.

6 Experimental Findings

This section reports the current annotated evaluation split of PhoneHarness Bench from four complementary views. We group comparisons into standalone model agents, alternative harnesses, and PhoneHarness. We first compare overall reliability across agent settings, then examine where the gain appears by task type, action-space label, execution steps, and safety behavior. Unless otherwise stated, PhoneHarness (Ours) uses Seed2.0-Pro as both the outer controller and the delegated GUI worker in Tables 4–7.

6.1 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

Table 4: Pass-rate scores on the annotated evaluation split, grouped by task type rather than difficulty.

Type	Agent	Overall	Device / system	Single-app GUI	Tool-assisted 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%**

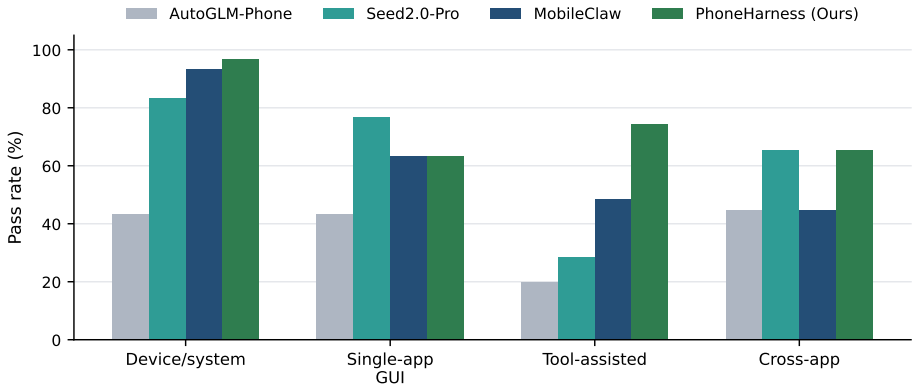

Figure 4: **Pass rate by task type.** PhoneHarness (Ours) is strongest where deterministic phone operations, tools, and verifiable side effects matter most, while pure GUI-heavy slices remain competitive for specialized GUI agents.

rate, improving over MobileClaw and Seed2.0-Pro by 12.9 percentage points, and over AutoGLM-Phone by 37.9 percentage points (Table 4). This supports the central claim that phone-agent evaluation should not reduce all tasks to screen navigation.

The task-type breakdown in Figure 4 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.

6.2 Action-space breakdown: gains concentrate on optional-CLI tasks

The action-type breakdown further explains the gains (Table 5 and Figure 6). The table focuses on two mixed-affordance slices: tasks where GUI and CLI can serve as alternative routes, and GUI-primary tasks where command-line operations can provide auxiliary support. These columns describe task affordances, not which action type must appear in every successful trajectory.

Table 5: Pass rates on tasks with mixed-action affordances. The columns indicate whether command-line operations are available as alternatives or as auxiliary support; they do not imply that CLI execution is 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%**

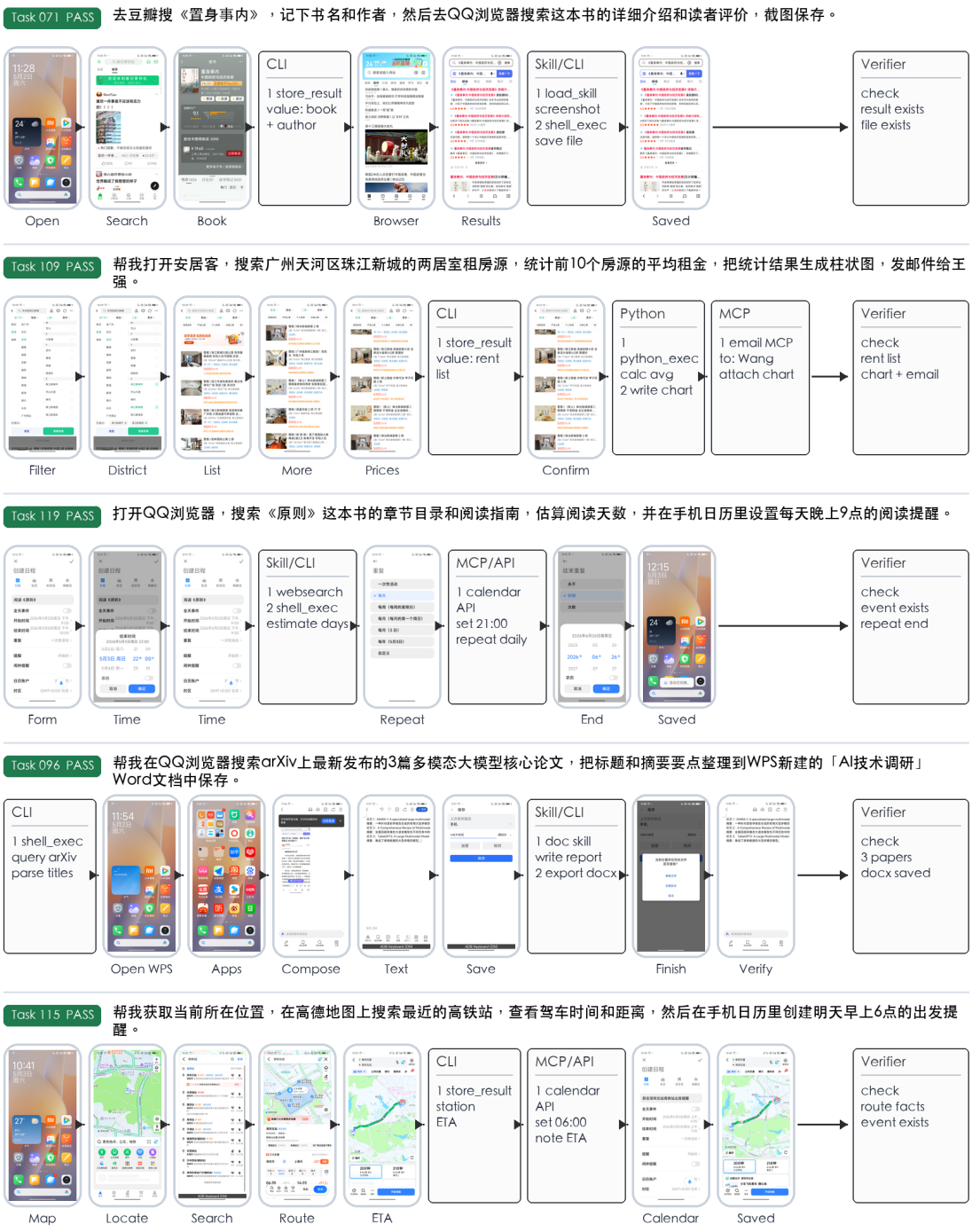

Figure 5: **Concrete traces show interleaved GUI/CLI/MCP execution with verifiable side effects.** Five representative PhoneHarness (Ours) runs pair selected real screenshots with compact non-GUI tool-operation cards and verifier-side success signals, covering cross-app lookup, artifact generation, calendar creation, document writing, and navigation.

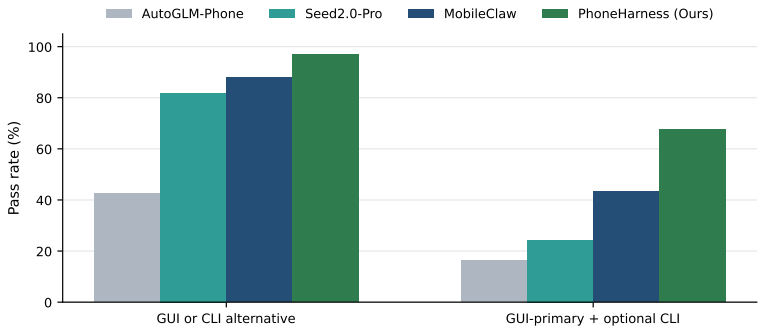

Figure 6: **Pass rate on mixed-action-affordance tasks.** PhoneHarness (Ours) is strongest when CLI operations are available either as an alternative path or as optional support for GUI-primary workflows.

GUI-primary + optional CLI denotes tasks that remain grounded in GUI app interaction but can benefit from auxiliary command-line operations, such as retrieving device state, preparing artifacts, or reducing brittle screen navigation. It should not be read as requiring CLI execution. PhoneHarness reaches 97.0% when GUI and CLI are alternative routes, and 67.6% on GUI-primary tasks with optional CLI support. This indicates that the benefit comes from action-surface routing on tasks that expose deterministic alternatives or useful auxiliary command-line paths, rather than from a uniformly stronger GUI controller. The experiment log also shows that many PhoneHarness wins over the GUI-oriented baselines come from email, file download, device-state queries, chart generation, wake-lock control, and other workflows with reliable CLI or MCP paths. The remaining failures are concentrated in brittle GUI-heavy scenarios: long single-app navigation, cross-app copy/paste, login or permission gates, advertisements, timeouts, and cases where a tool is called but the phone-side side effect is not verified.

6.3 Execution steps: higher success is not bought by uniformly longer runs

The step-count picture is consistent with the routing hypothesis (Table 6 and Figure 7). 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 PhoneHarness uses deterministic paths to avoid long GUI exploration. On cross-app workflows, Seed2.0-Pro ties PhoneHarness in pass rate while using fewer average steps, which indicates that some GUI-heavy multi-app tasks still favor a specialized phone-use model. All entries are rounded mean step counts over attempted tasks.

This reinforces the task-type interpretation: PhoneHarness's current advantage is strongest when the task requires routing across action surfaces and verifying side effects. We also report an auxiliary average-runtime view in Table 7.

Table 6: **Mean execution steps per attempted task** on the annotated evaluation split, rounded to the nearest whole step. **Lower denotes more efficient execution.**

Type	Agent	Overall Steps	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

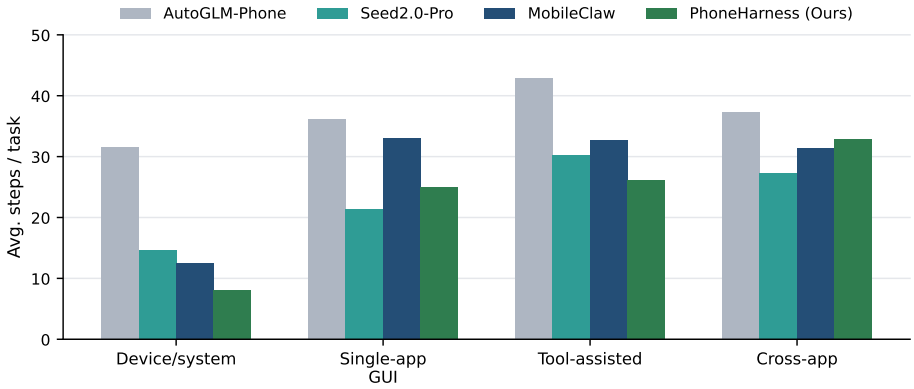

Figure 7: **Execution steps by task type.** PhoneHarness (Ours) reduces steps on device/system operations and tool-assisted workflows by routing away from unnecessary GUI exploration.

Table 7: 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)**

6.4 Model combinations: controller and GUI-worker choices affect verifiable success

We also evaluate PhoneHarness under different controller-model and GUI-worker-model combinations while keeping the harness, task protocol, and verifier stack fixed. Table 8 reports the resulting pass rates, allowing us to compare how the outer controller and delegated GUI worker affect verifiable task success under the same execution framework.

Table 8: Verifiable pass-rate scores by same-harness agent-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%**

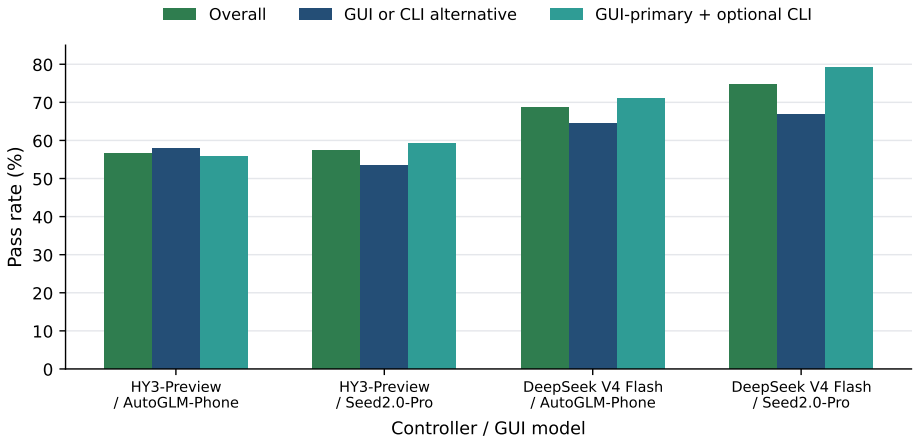

Figure 8: **Pass rate by same-harness model combination.** DeepSeek V4 flash with Seed2.0-Pro is strongest across the reported metrics.

The main effect is controller choice. DeepSeek V4 flash reaches the highest overall pass rate (74.8%) with Seed2.0-Pro as the GUI worker, clearly ahead of the other same-harness pairings. The GUI model gap is visible under the DeepSeek controller: Seed2.0-Pro is higher than AutoGLM-Phone overall (74.8% vs. 68.7%), on GUI-or-CLI alternatives (66.7% vs. 64.4%), and on GUI-primary tasks with optional CLI support (79.1% vs. 70.9%). This suggests that earlier AutoGLM failures were driven more by protocol, system-prompt, or adapter configuration issues than by a complete inability to operate real apps. HY3-preview is consistently one tier lower as a controller, while Seed2.0-Pro remains the stronger GUI worker in the DeepSeek-controlled comparison.

6.5 Safety behavior: refusal depends on model pairing

Safety behavior is evaluated separately because a high task-completion score should not imply permission to execute sensitive actions automatically. Table 9 reports dangerous-action refusal rate for several orchestration and GUI-controller pairings. The strongest pairings in this slice, HY3-preview with either GUI model, reach 90.0% refusal rate. This pattern suggests that HY3-preview is more conservative on safety-oriented tasks: its refusal behavior remains stable across both GUI workers, whereas DeepSeek V4 varies more with the GUI worker and stays below the HY3-preview pairings. The remaining configurations range from 80.0% to 86.7%, indicating that safety behavior is sensitive to both the outer orchestrator and the GUI controller. This supports treating safety as a first-class evaluation axis rather than as a post-hoc qualitative check.

Table 9: 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%**

7 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.

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.

8 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

Anthropic. Claude code auto mode: A safer way to skip permissions. Engineering Blog, March 2026. URL https://www.anthropic.com/engineering/claude-code-auto-mode.

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.