OPENMOBILE-2: BUILDING VERSATILE MOBILE AGENTS WITH SCALABLE ENVIRONMENTS AND APP-NATIVE TOOLS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce OpenMobile-2, a near-frontier mobile agent with fully open training environments and recipes. We make three key advances: (1) *Diverse environments with simulated commercial apps*: We build MOBILEGYM++, featuring 35 realistic, functionally rich commercial-style apps with cross-app workflows, while preserving full controllability for reset and verification. Together with newly configured apps in Android emulators, this yields a diverse playground spanning over 110 apps. (2) *Open training data at scale*: Building on this foundation, we curate nearly 12K mobile interaction trajectories for supervised fine-tuning and the largest open collection of verifiable RL training data for mobile agents, comprising over 2K executable tasks with automatic rewards. (3) *Hybrid GUI and app-native tool use*: We explore an experimental mobile-use setting where apps expose selected functionalities as app-native tools alongside their GUIs, allowing agents to interleave GUI actions and tool calls within a task. We implement this setting in MOBILEGYM++ with over 300 carefully scoped tools across 50+ apps. Additionally, we introduce MOBILEGYM++ BENCH for complex, long-horizon commercial mobile scenarios, supporting both GUI-only and hybrid GUI–tool evaluation on a shared task suite. OpenMobile-2 performs competitively across established benchmarks, including AndroidWorld (79.9) and MobileWorld (50.4), while showing promising transfer to real-device mobile use, nearly doubling SPA-Bench performance from 31.9 to 59.6. All resources are available at this link.

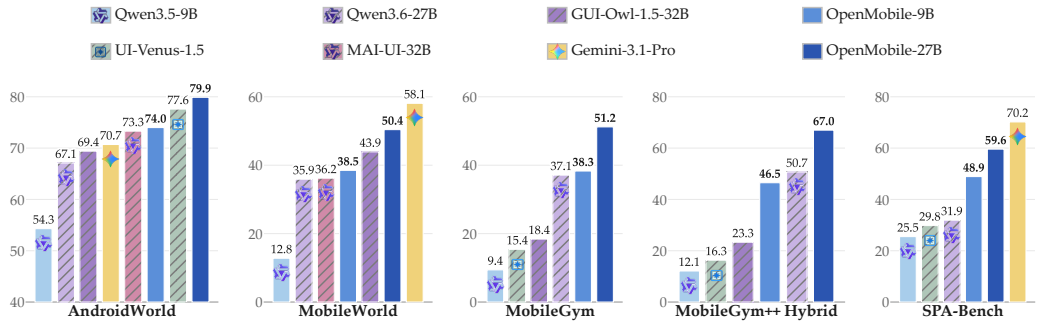
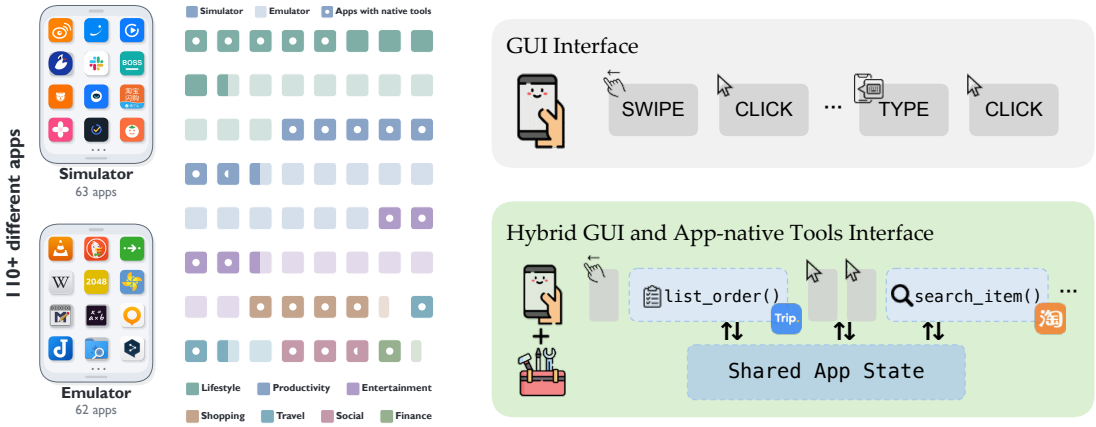


Figure 1: OpenMobile-2 performs strongly across established mobile agent benchmarks and achieves leading results among open-source models on real-device evaluation.

1 INTRODUCTION

Mobile agents built on vision-language models are increasingly capable of understanding smartphone interfaces and automating everyday tasks (Wang et al., 2025; Zhou et al., 2025; Xu et al., 2026; Gao et al., 2026; Zhou et al., 2026). Such progress depends heavily on large-scale interaction experience, typically in the form of trajectories collected across diverse tasks, applications, and environments (Cheng et al., 2026; Team et al., 2026b). Gathering such experience independently is

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.



(a) Application coverage

(b) GUI-only vs. Hybrid GUI and app-native tools

Figure 2: (a) Application coverage of OpenMobile-2. Each square represents two applications; dark and light tiles denote simulator and emulator apps, and dots mark apps with native tools. (b) Hybrid GUI and app-native tool use. The same task can be completed through GUI-only interaction (top) or by interleaving GUI actions with app-native tool calls (bottom) over a shared app state.

costly, while high-quality interaction data remains largely proprietary, leaving the open community with limited training resources and open models trailing well behind.

Scaling interaction experience without costly human annotation requires a diverse and controllable mobile playground. Especially for commercial applications, reproducing rich workflows and persistent user states at scale has remained a key obstacle. Existing open efforts (Sun et al., 2025; Cheng et al., 2026) largely rely on a small set of applications in Android emulators, mostly open-source apps and system utilities. Such environments provide realistic interaction, but third-party apps often lack reliable reset, while the full Android stack is costly to replicate for large-scale rollout and reinforcement learning (Rawles et al., 2025). Recent simulated environments offer an alternative path (Wu et al., 2026a; Liu et al., 2026b), providing efficient reset and parallelization, yet their applications remain few and shallow, with no scalable recipe for extending them.

We introduce OpenMobile-2, an open stack of environments, data, and training recipes that puts environment scaling into practice, as shown in Figure 2. We build MOBILEGYM++, a large-scale mobile environment featuring diverse, high-fidelity commercial-style applications, built upon the simulation framework of MobileGym (Wu et al., 2026a). In particular, we add 35 functional re-implementations of commercial apps with rich internal state and cross-app workflows, while preserving full controllability for clean reset and verification of task outcomes. Together with newly configured apps in Android emulator, this forms a diverse training playground spanning over 110 apps. Building on this playground, we curate nearly 12K trajectories for supervised fine-tuning and more than 2K executable tasks for reinforcement learning, each equipped with a reliable verifier.

We further explore an experimental mobile-use setting that combines GUI interaction with app-native tool use. This setting reflects an emerging shift in computer use, where universal but inefficient GUI interaction is increasingly complemented by efficient yet functionally bounded structured interfaces (Hu et al., 2026; Shi et al., 2026). In MOBILEGYM++, we expose selected app functionalities through over 300 app-native tools across 50+ apps, e.g., *search_course* in a fitness app. This enables us to train agents that interleave GUI actions and tool calls within a task.

To push mobile agent evaluation toward real-world commercial workflows and hybrid GUI–tool interaction, we introduce MOBILEGYM++ BENCH, a suite of 215 long-horizon cross-app tasks with persistent user states and programmatic verification. Every task can be solved with or without tools, so GUI-only and hybrid agents are compared on the same tasks. OpenMobile-2 achieves competitive results on established mobile agent benchmarks, reaching 79.9 on AndroidWorld and 50.4 on MobileWorld. It further transfers to real-device settings, improving SPA-Bench performance from 31.9 to 59.6 and outperforming strong open-source baselines. Further analysis highlights

the importance of scaling application diversity for improving agent generalization and reveals the complementary benefits of hybrid GUI and app-native tool use.

Our main contributions are:

- **Environment Scaling with Replicated Applications.** MobileGym++ faithfully replicates 35 commercial applications with accessible state in an extendable 110-application playground, overcoming the limited scope of existing open environments.
- **Open Training Data and Recipes at Scale.** We release a fresh collection of nearly 12K diverse trajectories and full recipes, together with over 2K executable RL tasks with reliable verifiers, which largely advance training mobile agents.
- **A Unified Framework for GUI and App-native Tool Use.** We integrate over 300 scoped app-native tools across 50+ apps and introduce MobileGym++ Bench, where GUI-only and hybrid GUI-tool agents can be trained and evaluated on the same long-horizon tasks.

2 Related Work

Mobile Agents. Mobile agents follow natural language instructions to operate smartphones through actions such as taps, swipes, and typing. Early systems rely on proprietary multimodal models orchestrated by external pipelines for screen parsing, planning, and memory (Zhang et al., 2025; Wang et al., 2024), requiring no task specific training. Native agents instead pretrain vision language models for GUI grounding (Cheng et al., 2024; Hong et al., 2024; Wu et al., 2025) and fine-tune them into end-to-end action models (Qin et al., 2025; Liu et al., 2024; Ye et al., 2025). More recent systems further improve these models through multi-turn reinforcement learning on live devices and interactive sandboxes (Wang et al., 2025; Zhou et al., 2025; Team et al., 2026b; Xu et al., 2026; Gao et al., 2026). This progression makes interaction experience a central resource for scaling mobile agents, spanning high-quality trajectories for supervised training, efficiently resettable environments, and verifiable tasks for reinforcement learning.

Trajectory Data and Interactive Environments. Existing training trajectories are primarily obtained in two ways. Human-recorded trajectories (Rawles et al., 2023; Li et al., 2024; Zhang et al., 2024; Lu et al., 2025) are fixed once collected, expensive to annotate, and carry no executable notion of success. Interactive environments supply the missing loop and differ in how their apps are obtained. One family runs real apps on Android emulators or physical devices and grades tasks against device state (Rawles et al., 2025; Xu et al., 2025; Chai et al., 2026; Chen et al., 2025), which is realistic but resists clean reset and is costly to parallelize; benchmarks over real commercial apps (Wu et al., 2026b; Feng et al., 2026; Sui et al., 2026) inherit these constraints. The other rebuilds apps as controllable mock clients (Trivedi et al., 2024; Wu et al., 2026a; Liu et al., 2026b), trading fidelity for cheap parallelism, clean reset, and direct state checks. Both are now used as data factories, synthesizing trajectories and verifiable tasks from the environment itself (Sun et al., 2025; Cheng et al., 2026; Liu et al., 2026a; Cao et al., 2026; Team et al., 2026a), making application coverage a key bottleneck for scaling interaction experience. Yet applications in existing environments remain shallow, with no scalable way to capture interaction experience across commercial workflows.

Hybrid GUI and Tool Use. GUI actions reach any visible control but are slow and brittle; structured calls are efficient but cover only the functions an interface exposes. Structured interfaces such as CLIs and MCP are increasingly available alongside GUIs, and recent computer-using agents have begun to combine the two (Hu et al., 2026; Yan et al., 2025; Jia et al., 2026; Fan et al., 2026). On mobile the same split appears as API-only worlds with no phone GUI (Trivedi et al., 2024), as external MCP servers attached to a GUI agent (Kong et al., 2026), as shortcuts such as RPA over real apps (Zhao et al., 2026), or as actions an agent generates from its own GUI traces (Lee et al., 2026). Prior hybrid approaches typically attach external tools or system-level interfaces to a GUI agent, rather than functions exposed by the app itself. In contrast, MobileGym++ simulates commercial apps that selectively expose their own app-native tools alongside the GUI, enabling tool calls and GUI actions to interleave within a task over shared application state.

Table 1: Comparison of mobile agent environments across scale, interaction, and controllability. ✓ denotes supported, ✗ unsupported, and ∼ indicates limited system level cross-app interaction support in MobileGym. # Comm. Apps and # Tasks denote the number of covered commercial apps and benchmark tasks, respectively; Detailed application statistics are provided in Appendix C. MobileGym++ provides diverse, high-fidelity commercial-style applications with full controllability, while introducing an experimental app-native tool interface.

Environment	Platform	Scale		Interaction			Full Controllability
		# Comm. Apps	# Tasks	Cross-app	GUI	App-native Tools	
AppWorld (Trivedi et al., 2024)	API	9	750	✓	✗	✗	✓
AndroidLab (Xu et al., 2025)	Emulator	5	138	✗	✓	✗	✗
AndroidWorld (Rawles et al., 2025)	Emulator	0	116	✓	✓	✗	✗
SPA-Bench (Chen et al., 2025)	Real-device	58	340	✓	✓	✗	✗
MobileWorld (Kong et al., 2026)	Emulator	4	201	✓	✓	✗	✗
MobileGym (Wu et al., 2026a)	Simulator	12	256	∼	✓	✗	✓
MobileGym++ (ours)	Simulator	**35**	**215**	✓	✓	✓	✓

3 MobileGym++ Playground

3.1 A gap in controllable commercial applications

Mobile agents are increasingly moving toward real-world workflows across commercial applications, such as travel booking on Trip and online shopping on Taobao, where tasks are often long-horizon and grounded in real user needs. However, training and evaluation environments for such commercial applications remain largely missing. Real-device or Android-emulator setups introduce login friction, irreversible side effects, and costly reset procedures that make large-scale training impractical (Rawles et al., 2025; Kong et al., 2026). Prior simulated-app work, meanwhile, covers only a narrow range of commercial domains (Wu et al., 2026a; Trivedi et al., 2024). As a result, mobile agents have limited opportunities to practice and be evaluated on realistic mobile-use scenarios.

We built MobileGym++, a browser-hosted mobile environment that supports realistic commercial workflows in controllable simulated applications. The runtime extends MobileGym (Wu et al., 2026a), preserving its advantages in reset and programmatic evaluation while substantially expanding domain coverage, the number and richness of applications, and support for native cross-app interaction. Observations consist of raw screen pixels; there is no accessibility tree and the agent cannot read business state directly. Device state is fully serializable, enabling deterministic reset and lightweight parallel instantiation. We newly built 35 high-fidelity commercial applications in this round, bringing the total to 63 installable clients spanning travel, shopping, finance, content, social, office, and local services. Each application maintains persistent business state—orders, budgets, itineraries—that evolves through interaction and supports programmatic task verification. Table 1 compares MobileGym++ with existing mobile agent environments.

3.2 35 high-fidelity commercial applications

The 35 applications are structurally faithful replicas of their real-world counterparts. Construction starts from the source application's data schema and interaction graph, specified before any interface is rendered, so business-state richness and navigation depth are architectural properties of each replica. Human annotators then iteratively compare each replica against the live application and refine both functionality and visual fidelity until the state transitions, and interface details closely match the reference application (Appendix B). Every delivered app is checked in two ways before tasks are written on it: a coverage check maps each declared interaction to its implementation, and a replay of the core flows confirms that each transition produces the declared state change. Across these clients, the median application exposes 37 routes, 166 interactive actions, and 86 page transitions, with a median interaction depth of 3 from the home screen. Relative to the 12 prior MobileGym business apps retained in the same environment, the newly built clients reach $1.48\times$ the interactive actions and $1.91\times$ the page transitions, as shown in Figure 3.

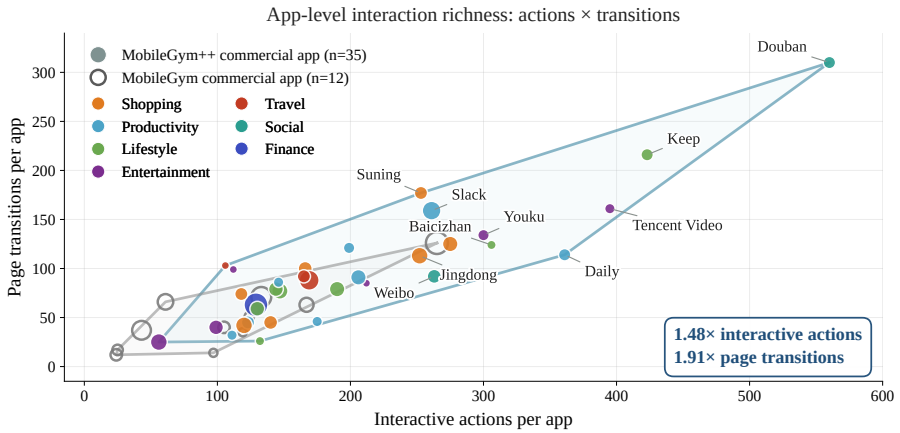


Figure 3: App-level interaction richness. Each marker denotes one app, positioned by its interactive actions and page transitions; marker size reflects the number of Bench tasks using that app. Our new apps reach 1.48× more actions and 1.91× more transitions.

3.3 Realistic cross-application workflows

Many real-world mobile workflows span multiple applications, requiring agents to retrieve information in one app, carry it across application boundaries, and complete an action in another. Single-application benchmarks cannot capture this behavior.

MobileGym++ extends simulated mobile use with explicit cross-application interaction mechanisms, including clipboard transfer, app-to-app sharing, payment redirection, and media selection. All applications run on a shared, resettable device whose full state can be serialized and restored across episodes. Each app maintains its own business objects—such as orders, expenses, and itineraries—which evolve through interaction. These mechanisms allow information and actions to flow across application boundaries, enabling realistic multi-app workflows rather than isolated single-app tasks.

3.4 Hybrid GUI and app-native tool use

Computer use is shifting beyond GUI-only interaction toward hybrid graphical and code interfaces. We bring this idea to mobile use through an experimental setting in which applications selectively expose app-native tools alongside their GUIs. Unlike prior mobile settings with external tools (Kong et al., 2026), our tools expose native app functions and operate on the same state as the GUI.

We follow a simple, principled design for tool construction: each tool exposes a structured lookup or write operation over an application's own business objects, while preserving semantic equivalence with the corresponding GUI interaction. A tool call and its GUI counterpart therefore modify the same persistent state and are evaluated by the same terminal condition. This enables controlled comparison of the two interaction channels without changing tasks, initialization, or success criteria. We implement over 300 such tools across more than 50 applications, including system clients such as Clock and Notes; tool depth varies by app, from a few search and lookup functions to more than ten read and write operations. All benchmark tasks remain solvable through GUI-only interaction, while hybrid agents can interleave GUI actions and app-native tool calls within the same task.

3.5 MobileGym++ Bench

MobileGym++ Bench enables reliable evaluation of mobile agents in commercial-style applications through resettable environments and programmatic verification over application-level business states. Unlike existing benchmarks that rely on device-level signals (Rawles et al., 2025; Kong et al., 2026), MobileGym++ Bench evaluates whether agents achieve the intended changes in application business objects. Furthermore, the same task suite supports both GUI-only and hybrid GUI–tool interaction, enabling controlled comparison between the two settings. See details in Appendix E.

Task Design. The primary source of difficulty is commercial navigation and information flow. A single-app task requires the agent to locate, filter, and fill fields across a deep page stack before a business object is written. A cross-application task further requires a live value from another application's store to be identified, carried across the application boundary, and written into the correct field. Instructions are phrased as user goals and do not name widgets, tools, or checker fields; the terminal is a write in the application's own business state.

Scale and Composition. The benchmark comprises 215 tasks across seven business domains, including 109 single-app, 74 two-app, 27 three-app, and 5 four-app tasks. Tasks are balanced across applications and business objects to avoid domination by a small subset of clients.

Evaluation Protocol. Each task specifies a user goal, a bound initial state, and a programmatic checker over the before/after diff of application business state. The checker does not inspect the trajectory or screenshots and does not rely on a model judge. GUI-only and Hybrid are evaluated on the same tasks, initial states, and checks. The primary metric is success rate.

4 OPENMOBILE-DATA

We release **OpenMobile-Data**: 11.6K demonstration trajectories and more than 2K executable RL tasks collected across our Android emulator environments and MOBILEGYM++. The 11.6K demonstrations are task–trace pairs for imitation, including 1.7K trajectories that interleave GUI actions with app-native tool calls. Each RL task specifies a goal, an initial state, and a programmatic reward, and can be rolled out in MobileGym++ in either GUI-only or hybrid mode.

4.1 CONSTRUCTION PIPELINE

Environment abstraction and task construction. We construct a state–action transition graph for each application to expose its available functionality and support the generation of executable user goals following OpenMobile (Cheng et al., 2026). Nodes represent representative application states, while edges capture actions that induce state transitions. On the Android emulator, we explore each application to discover screens, actions, and transitions, and consolidate perceptually similar screenshots into representative states. This exploration yields a connected view of the application's functionality: the global view captures the overall state–action structure and available functionality, while the local view captures the actions and states reachable from a particular state. For MOBILEGYM++, the same views are derived directly from the backend specification, which explicitly defines pages, actions, and business objects. Conditioned on these views, Gemini-3.1-Pro-Preview generates realistic and executable user instructions. We filter the generated instructions for quality and diversity, removing low-quality and highly similar tasks.

Trajectory collection and filtering. Conditioned on the generated user instructions, Gemini-3.1-Pro-Preview acts as the teacher agent and executes each task in its corresponding environment. Each execution produces a trajectory of screens, GUI actions, and, where available, native tool calls. We retain only rollouts that complete the specified task and pass the corresponding runtime-specific verifier: completion is assessed by a vision-language judge on the Android emulator (Sun et al., 2026), while MOBILEGYM++ uses its programmatic task verifier. After collecting successful trajectories, we perform step-level filtering to remove redundant interactions, including meaningless action loops and actions that leave the application state unchanged. The resulting data are released as two separate resources: demonstration trajectories retain the full observation–action sequence, while executable RL tasks retain only the instruction, initial state, and verifier.

4.2 DATA STATISTICS

OpenMobile-Data is diverse along three dimensions: domain distribution, task diversity, and workflow diversity (Figure 4). The data span seven domains with a relatively balanced distribution. We quantify domain balance using the Gini coefficient, where lower values indicate more uniform coverage. The full set achieves a Gini coefficient of 0.43, while the emulator subset is more concentrated (0.52) and simulated clients provide more balanced coverage (0.25). Task goals also show diversity, with a mean pairwise cosine similarity of 0.43, indicating varied requests without excessive repetition. Finally, the data cover both single-app and cross-app workflows, with 30% of trajectories spanning multiple

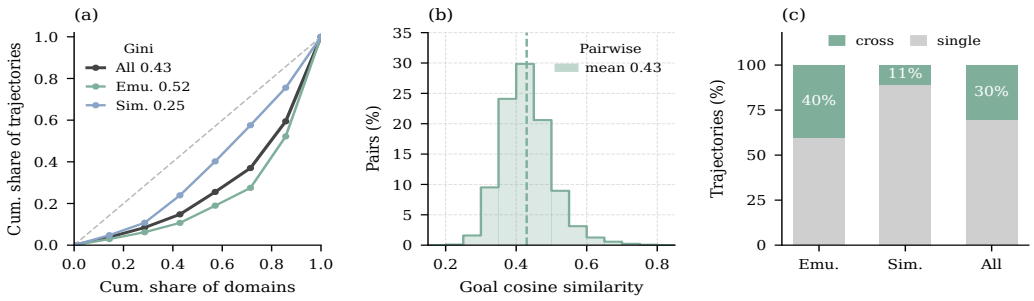


Figure 4: Diversity and coverage of OpenMobile-Data. **(a)** Domain distribution across seven categories for emulator and simulator data. Gini coefficients quantify domain balance. **(b)** Pairwise cosine similarity among task goals. **(c)** Single-app and cross-app workflow composition.

applications (40% on the emulator and 11% on simulated clients). Together, these statistics show that OpenMobile-Data provides diverse domains, task goals, and interaction workflows.

5 Experiments

In this section, we evaluate models trained with OpenMobile-Data on established mobile-agent benchmarks, real-device tasks, and MobileGym++.

5.1 Training and Evaluation Details

We train two supervised fine-tuning runs from each base model, both for 5 epochs. GUI-only SFT uses only GUI trajectories. Hybrid SFT uses the full mixture, combining those GUI trajectories with hybrid GUI–tool trajectories. RL is then initialized from the hybrid SFT checkpoint and trained on 512 tasks randomly sampled from the 2K+ executable tasks. We evaluate the resulting models on five benchmarks spanning three settings—emulator apps, physical devices, and simulated apps with both GUI-only and tool-augmented interaction. **AndroidWorld** (Rawles et al., 2025) contains 116 tasks across 20 real apps on the emulator, with success determined by the resulting device state; **MobileWorld** (Kong et al., 2026) contains 201 harder long-horizon and cross-app tasks evaluated in GUI-only and MCP-enabled settings; **MobileGym** (Wu et al., 2026a) uses resettable mock apps with state-based verification and reports success rate (SR) and progress rate (PR); **MobileGym++ Bench** (Section 3.5) contains 215 single- and cross-app tasks evaluated with identical tasks and checkers in GUI-only and hybrid settings; and **SpaBench** (Chen et al., 2025) evaluates tasks on physical Android devices, providing a real-device evaluation beyond the emulator-based training environments. Detailed training and evaluation configurations are provided in (Appendix A).

We analyze the overlap between our training data and benchmark tests to assess potential benchmark overfitting (Appendix A.4). Using `text-embedding-3-large`, only 1.11%, 0.39%, 0.46%, and 2.25% of SFT instructions exceed a cosine similarity of 0.7 with AndroidWorld, MobileWorld, MobileGym, and MobileGym++ Bench, respectively, and none exceeds 0.9. Among the 2,475 RL task goals, 6.1% exceed 0.7 with MobileGym++ Bench, while none exceeds 0.9.

5.2 Main Results

OpenMobile-Data improves performance across established mobile agent benchmarks. As shown in Table 2, models fine-tuned on OpenMobile-Data consistently outperform their corresponding base models across AndroidWorld, MobileWorld, and MobileGym++. The improvements are substantial at both model scales: the 9B model gains 19.7 points on AndroidWorld over its base (54.3 to 74.0), while the 27B model gains 12.8 points (67.1 to 79.9), with consistent improvements on MobileWorld and MobileGym++ as well. These results show that the interaction experience synthesized by OpenMobile-Data is useful across different mobile environments.

Table 2: **Main results** on established mobile agent benchmarks and real-device evaluation. $*$ denotes results reproduced by us, and $\pm$ denotes the standard deviation across evaluation seeds.

Method	AndroidWorld		MobileWorld		MobileGym		MobileGym++		SPA-Bench
	Pass@1↑	Pass@3↑	GUI	MCP	SR	PR	GUI	Hybrid	L3
Commercial & Closed-Weight Models									
GPT-5.6-Sol	–	–	70.1	–	–	–	–	–	68.1*
Gemini-3.1-pro-preview	70.7	–	58.1	–	–	–	–	–	70.2*
MAI-UI-32B	73.3	–	36.2	30.0	–	–	–	–	–
MAI-UI-235B-A22B	76.7	–	39.7	37.5	–	–	–	–	–
Open-Weight Models									
UI-Venus-1.5-8B	73.7	–	–	–	15.4	28.3	19.5*	16.3*	29.8*
GUI-Owl-1.5-8B	69.0	–	38.2	37.5	17.2*	31.3*	13.0*	12.6*	–
GUI-Owl-1.5-32B	69.4	–	43.9	42.5	18.4*	30.9*	18.6*	23.3*	–
MAI-UI-8B	70.7	–	27.5	20.0	16.8*	28.3*	19.1*	18.6*	–
Open-Data Models									
OpenMobile-1-8B	64.7 $\pm$ 3.2	78.0	17.7	–	13.7*	25.5*	13.5*	7.0*	–
OpenMobile-2-9B									
Base (Qwen3.5-9B)	54.3 $\pm$ 3.7 *	69.8*	12.8*	12.5*	9.4*	20.0*	9.3*	12.1*	25.5*
GUI-only SFT	72.8 $\pm$ 2.2	81.9	38.5	12.5	34.8	51.6	40.9	41.9	–
Hybrid SFT	73.3 $\pm$ 1.5	83.6	39.3	27.5	35.9	52.2	39.1	44.7	48.9
+ RL	74.0 $\pm$ 2.9	83.6	38.5	27.5	38.3	52.0	40.5	46.5	48.9
OpenMobile-2-27B									
Base (Qwen3.6-27B)	67.1 $\pm$ 4.1 *	81.9*	35.9*	32.5*	37.1*	50.1*	46.5*	50.7*	31.9*
GUI-only SFT	80.8 $\pm$ 2.1	89.7	50.4	35.0	46.9	63.1	54.9	57.7	–
Hybrid SFT	80.8 $\pm$ 3.5	90.5	48.7	45.0	48.4	64.5	55.3	65.6	57.4
+ RL	79.9 $\pm$ 3.0	90.5	50.4	42.5	51.2	66.6	57.2	67.0	59.6

Starting from hybrid SFT, we further explore agentic RL on executable mobile tasks. RL improves training-aligned environments, raising MobileGym success rate from 35.9 to 38.3 (9B) and from 48.4 to 51.2 (27B), while also improving 27B SPA-Bench performance from 57.4 to 59.6, suggesting improved transfer to real-device mobile use. Gains on other benchmarks remain limited, suggesting that RL benefits depend on the alignment between training environments and evaluation scenarios.

Generalization beyond simulated environments to real devices. Beyond established online benchmarks, we evaluate the resulting agents on real Android devices using SPA-Bench L3. Hybrid SFT achieves 48.9 and 57.4 for the 9B and 27B models, respectively, substantially improving over their corresponding base models (25.5 and 31.9) and outperforming strong open-source counterparts. These results show that the benefits of OpenMobile-Data transfer beyond the controlled environments used for training and evaluation to physical devices.

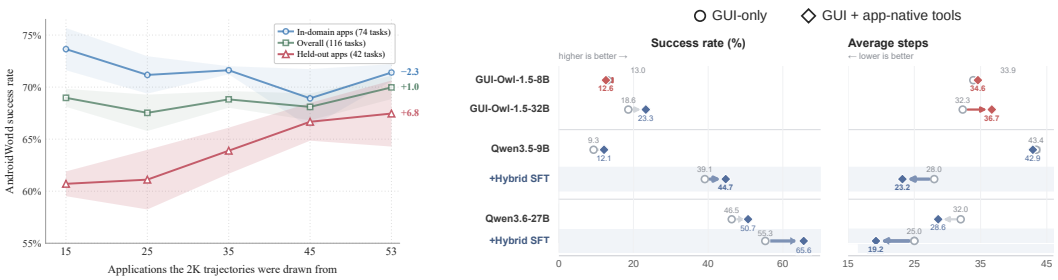
App-native tools provide a complementary action channel. MobileGym++ enables agents to interleave GUI actions with application-specific tool calls within a task. On MobileGym++ Bench, hybrid SFT improves the hybrid evaluation over GUI-only SFT at both scales (9B: 41.9 to 44.7; 27B: 57.7 to 65.6), while GUI-only success stays comparable (Table 2). This result demonstrates that app-native tools provide a useful complementary action channel for mobile agent interaction.

Hybrid training enables generalizable tool-use behavior. We compare hybrid and GUI-only SFT to evaluate whether app-native tool training transfers beyond the observed tools. On MobileWorld MCP, whose tools are unseen during training, hybrid SFT achieves 27.5 and 45.0 versus 12.5 and 35.0 for GUI-only SFT. This transfer suggests that hybrid training learns general tool-use strategies rather than memorizing specific tool APIs.

5.3 Analysis

We focus our analysis on two aspects of mobile agent interaction: how environment coverage relates to generalization, and how app-native tools complement GUI interaction.

Broader environment coverage improves transfer under a fixed data budget. With the backbone, optimizer, and total training data fixed at 2K emulator trajectories, we vary application coverage from 15 to 53 while keeping the same 15 InD applications. Broader coverage improves transfer to



(a) Environment coverage

(b) App-native tool use

Figure 5: **Environment coverage and app-native tool use.** (a) Broader coverage improves held-out AndroidWorld performance at a fixed 2K-trajectory budget. (b) App-native tools improve success and efficiency, with larger gains after Hybrid SFT.

unseen applications, with Pass@1 increasing from 60.7 to 67.5 (Figure 5a). Meanwhile, the InD data share drops from 100% to 39.1%, but InD Pass@1 remains largely stable (73.7→71.4; Table 3). These results show that broader application coverage improves generalization without significantly compromising performance on familiar applications.

Table 3: Environment-coverage settings under a fixed budget of 2K trajectories. InD share denotes the fraction of training trajectories from the 15 in-distribution applications.

# Train apps	InD data share
15	100%
25	64.0%
35	48.9%
45	41.1%
53	39.1%

Table 4: App-native tool use on MobileGym++. TR (tool-call rate) is the average number of app-native tool calls per task. #Tools is the number of distinct tools invoked across the benchmark, and #Tasks is the number of tasks (out of 215) in which at least one app-native tool is called.

Model	Stage	TR	#Tools	#Tasks
GUI-Owl-1.5-8B	-	0.58	15	20
GUI-Owl-1.5-32B	-	0.51	51	45
Qwen3.5-9B	Base	0.64	28	21
	+ Hybrid SFT	1.41	110	113
Qwen3.6-27B	Base	1.64	97	102
	+ Hybrid SFT	2.23	115	128

Hybrid training unlocks the benefits of app-native tools. Providing app-native tools alone does not guarantee effective tool use: without hybrid training, models rarely learn when to invoke them. Before hybrid SFT, tool calls appear in only 20–102 of 215 tasks (Table 4), bringing limited gains: Qwen models improve by only 2.8–4.2 points, while GUI-Owl-1.5 shows inconsistent results and can even degrade (Figure 5b). After hybrid SFT, tool usage becomes substantially more effective: tool calls appear in 113 and 128 tasks, covering 110 and 115 distinct tools for the 9B and 27B models, respectively. More importantly, tool-enabled gains increase to 5.6 and 10.3 points, while trajectories become 4.8 and 5.8 steps shorter. These results show that hybrid training teaches models to effectively leverage app-native tools, improving both task success and interaction efficiency.

6 Conclusion

We presented OpenMobile-2, an open training stack for mobile agents spanning diverse environments, open training data, and trained agent models. On the environment side, MobileGym++ and newly configured emulator apps establish a diverse playground spanning over 110 applications, including high-fidelity commercial-style apps with clean reset and verification. On the data side, we release nearly 12K demonstration trajectories and over 2K executable RL tasks with automatic rewards. Models trained on this stack demonstrate strong performance across established mobile agent benchmarks and transfer to real-device evaluation. We further explore app-native tools as a complementary action interface for mobile agents, finding that tool access alone brings limited gains while hybrid training substantially amplifies their benefits. We open-source environments, data, benchmarks, and trained models to support further open research on mobile agents.

AI USE STATEMENT

In this work, we used generative AI tools for writing assistance, including grammar, spelling, and clarity improvements. We also used AI coding assistants for implementation support, including the development of MOBILEGYM++ simulated applications. Additionally, we used VLMs for task instruction synthesis, trajectory evaluation, and automated verification as described in the methodology. The authors reviewed all AI-assisted outputs and take responsibility for the research design, experiments, analysis, claims, artifacts, and final presentation.

ETHICS STATEMENT

The MOBILEGYM++ applications reproduce the interface structure and business logic of real commercial clients for only research purposes. They run entirely in a browser sandbox, do not connect to any real back end, and do not bypass login, anti-bot, or other access controls of the original services. All users, orders, balances, and other records in these applications are synthetic, so the released trajectories and tasks contain no real personal information. Application names and logos are shown only to identify the workflows being replicated. Human annotators verified the replicated applications against real clients; they were team members or were compensated at standard rates. No personal data were collected from them, and the study involves no other human subjects.

REPRODUCIBILITY STATEMENT

Anonymous source code is available at https://anonymous.4open.science/r/openmobile2. Implementation details are in Appendix A.

REFERENCES

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, et al. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training. arXiv preprint arXiv:2602.14093, 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, et al. SPA-Bench: A comprehensive benchmark for smartPhone agent evaluation. In The Thirteenth International Conference on Learning Representations, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 9313–9332, 2024.

Kanzhi Cheng, Zehao Li, Zheng Ma, Nuo Chen, Jialin Cao, Qiushi Sun, Zichen Ding, Fangzhi Xu, Hang Yan, Jiajun Chen, et al. Openmobile: Building open mobile agents with task and trajectory synthesis. arXiv preprint arXiv:2604.15093, 2026.

Siqi Fan, Minghao Li, Xiaoqian Ma, Wenhui Tan, Xiusheng Huang, Juntong Wu, Liujie Zhang, Shuo Shang, and Weihang Chen. Screenshots or tools? eliciting tool use and managing multimodal context in hybrid gui-mcp computer-use agents. arXiv preprint arXiv:2608.03327, 2026.

Yunfei Feng, Xi Zhao, Cheng Zhang, Dahu Feng, Daolin Cheng, Jianqi Yu, Yubin Xia, and Erhu Feng. Mobiflow: Real-world mobile agent benchmarking through trajectory fusion. arXiv preprint arXiv:2604.09587, 2026.

Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, Beitong Zhou, et al. Ui-venus-1.5 technical report. arXiv e-prints, pp. arXiv–2602, 2026.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14281–14290. IEEE, 2024.

Xuhao Hu, Xi Zhang, Haiyang Xu, Kyle Qiao, Jingyi Yang, Xuanjing Huang, Jing Shao, Ming Yan, and Jieping Ye. Toolcua: Towards optimal gui-tool path orchestration for computer use agents. arXiv preprint arXiv:2605.12481, 2026.

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.

Juyong Lee, Woogyeol Jin, and Kimin Lee. Improving proficiency and efficiency of android gui agents via self-generating tool actions. arXiv preprint arXiv:2609.06792, 2026.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. Advances in Neural Information Processing Systems, 37:92130–92154, 2024.

Guohong Liu, Jialei Ye, Pengzhi Gao, Wei Liu, Jian Luan, Yunxin Liu, and Yuanchun Li. Scalewob: Guiding gui agents with coding agents via large-scale environmental synthesis. arXiv preprint arXiv:2605.25160, 2026a.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. Autoglm: Autonomous foundation agents for guis. arXiv preprint arXiv:2411.00820, 2024.

Yuxuan Liu, Xin Lai, Junyi Li, Pengyuan Lyu, Yiduo Guo, Zhengyao Fang, Yang Ding, Yi Zhang, Weinong Wang, Huawen Shen, et al. Phoneworld: Scaling phone-use agent environments. arXiv preprint arXiv:2605.29486, 2026b.

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 2025 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 22404–22414. IEEE, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025.

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.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. Advances in Neural Information Processing Systems, 36:59708–59728, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In Proceedings of the Twentieth European Conference on Computer Systems, pp. 1279–1297, 2025.

Haoting Shi, Wenhao Wang, Weicheng Fang, Yaozhong Liang, Tian Jin, Pengxiang Zhao, Guangyi Liu, Siheng Chen, and Yanfeng Wang. Cua-universe: A scalable and dynamic environment for hybrid gui+ cli agents. arXiv preprint arXiv:2609.05374, 2026.

Yifan Sui, Xin Huang, Hongbing Li, Fang Xu, Jiahe Lv, Haolong Yan, Yeqing Shen, Litao Liu, Zhimin Fan, Ziyang Meng, et al. Androiddaily: A verifiable benchmark for mobile gui agents on real-world closed-source applications. arXiv preprint arXiv:2605.27761, 2026.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 5555–5579, 2025.

Qiushi Sun, Kanzhi Cheng, Yian Wang, Bowen Yang, Hang Yan, Liheng Chen, Fangzhi Xu, Zichen Ding, Nuo Chen, Jialin Cao, et al. Osreward: Instituting standardized evaluation for cross-platform computer-use reward models. arXiv preprint arXiv:2607.28609, 2026.

Hy Vision Team, Huawen Shen, Zhengyang Tang, Shangpin Peng, Liang Wu, Anran Zhang, Weinong Wang, Yiduo Guo, Chenxin Li, Zhengyao Fang, et al. Hymobileagent: Data-environment co-scaling for efficient gui agents. arXiv preprint arXiv:2607.14548, 2026a.

Venus Team, Zhuohan Cai, Haoxing Chen, Jiaxuan Chen, Weizhi Chen, Changlong Gao, Zhangxuan Gu, Yuan Guo, Yusong Hu, Jianrong Jiang, et al. Ui-venus-2 technical report. arXiv preprint arXiv:2609.00028, 2026b.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 16022–16076, 2024.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. arXiv preprint arXiv:2509.02544, 2025.

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, 2024.

Dingbang Wu, Rui Hao, Haiyang Wang, Shuzhe Wu, Han Xiao, Zhenghong Li, Bojiang Zhou, Zheng Ju, Zichen Liu, Lue Fan, et al. Mobilegym: A verifiable and highly parallel simulation platform for mobile gui agent research. arXiv preprint arXiv:2605.26114, 2026a.

Qinzhuo Wu, Zhizhuo Yang, Hanhao Li, Pengzhi Gao, Wei Liu, and Jian Luan. Mobilebench-ol: A comprehensive chinese benchmark for evaluating mobile gui agents in real-world environment. In Findings of the Association for Computational Linguistics: ACL 2026, pp. 13659–13693, 2026b.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: Foundation action model for generalist gui agents. In International Conference on Learning Representations, volume 2025, pp. 5090–5108, 2025.

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, 2025.

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, 2025.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. arXiv preprint arXiv:2508.15144, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. Advances in Neural Information Processing Systems, 38:113222–113244, 2026.

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, 2025.

Jiwen Zhang, Jihao Wu, Teng Yihua, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents. In Findings of the Association for Computational Linguistics: EMNLP 2024, pp. 12016–12031, 2024.

Pengxiang Zhao, Guangyi Liu, Yaozhen Liang, Weiqing He, Zhengxi Lu, WenHao Wang, Yuehao Huang, Yuxiang Chai, Zhaolu Kang, Yaxuan Guo, et al. Mas-bench: A unified benchmark for shortcut-augmented hybrid mobile gui agents. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 6960–6985, 2026.

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.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, Jianan Zhang, Long Li, Long Chen, et al. Qwen-ui-agent technical report: Toward next-generation real-world centric foundation gui agents. arXiv preprint arXiv:2607.28227, 2026.

A Training and Evaluation Details

A.1 Supervised fine-tuning

Two SFT runs. Supervised fine-tuning consists of two separate 5-epoch runs from the base model. They share the hyperparameters in Table 5 and differ in training data (Table 6). The GUI-only run uses emulator trajectories together with GUI simulator trajectories, and does not use app-native tools. The hybrid run uses the full emulator and simulator mixture, including 1,774 simulator trajectories that interleave app-native tool calls. GUI trajectories use the GUI-only system prompt, whereas hybrid trajectories use a tool-aware system prompt. RL is initialized from the hybrid-run checkpoint.

Training samples. Each trajectory yields one training sample per step. A sample contains the task instruction, the interaction history up to that step, and a target turn consisting of a `Thought:` paragraph followed by one tool call. The three most recent screenshots are retained as images; older screenshots are replaced with the text "This screenshot has been collapsed." The full mixture contains 118,582 step-level samples. SFT screenshots are fed at their native 1080×2400 resolution, and the model predicts coordinates in a normalized 999×999 space.

Action space. The GUI-only run uses the `mobile_use` tool with actions `click`, `long_press`, `swipe`, `type`, `answer`, `system_button` (Back, Home, Menu, or Enter), `wait`, and `terminate` (success or failure). The hybrid run adds app-native tools, including actions such as `clock__create_alarm` and `notes__create_note`. An application's tool catalog is available only while that application is in the foreground, so the policy chooses between a GUI action and an available app-native action at each step.

Table 5: Training hyperparameters. SFT values are shared by both runs; – indicates that a setting does not apply.

Hyperparameter	SFT	RL
Framework	Megatron-SWIFT	verl
Objective	Next-token prediction	DAPO (group-relative)
Trainable parameters	LLM (vision frozen)	LoRA
Optimizer	Adam (β_1=0.9, β_2=0.95)	AdamW
Weight decay / gradient clip	0.1 / 1.0	0.1 / 1.0
Learning rate	5×10^{-6}	1×10^{-6}
Schedule	Cosine; 3% warmup; minimum 10^{-7}	40 warmup steps
Batch size	32 (microbatch 1)	64 tasks $\times$ 8 rollouts
Training length	5 epochs	150 steps (budget 300); 2 updates each
Group size	–	8
Clipping ϵ_{low} / ϵ_{high}	–	0.2 / 0.28
KL coefficient	–	0.01 (low-variance estimator)
Token budget	32,768 (packed)	32,768 (4,096 prompt + 28,672 response; 1,024 per turn)
Screenshot input	1080×2400 (native)	Short side 448 px
Screenshot history	Last 3	Full episode (at most 32 images)
Coordinate space	999×999	999×999
Step cap	None (maximum observed: 50)	30
Precision	bf16	bf16
Hardware	8 GPUs; tensor parallelism 4	32 H800 GPUs; sequence parallelism 4

A.2 Reinforcement learning

Algorithm. We initialize reinforcement learning from the hybrid SFT checkpoint and optimize the policy with GRPO-style group-relative advantages (Shao et al., 2024) and the DAPO objective (Yu et al., 2026), implemented in verl (Sheng et al., 2025). For each task, we sample a group of 8

Table 6: Trajectories in each SFT run. The hybrid run adds 1,774 simulator trajectories that interleave app-native tool calls, so the simulator count rises from 2,144 to 3,918.

	GUI-only run		Hybrid run	
Source	Trajectories	Steps	Trajectories	Steps
Emulator	7,725	83,631	7,725	83,631
Simulator	2,144	22,857	3,918	34,951
Total	9,869	106,488	11,643	118,582

rollouts in MOBILEGYM++ and use asymmetric clipping with ϵ_{low}=0.2 and ϵ_{high}=0.28. We add a KL loss toward the SFT reference policy, with coefficient 0.01 and a low-variance estimator. The hyperparameters are listed in Table 5.

Tasks. Training uses all 512 tasks. Each task has a bound initial state and a programmatic verifier in MOBILEGYM++. Validation uses the official MobileGym split of 256 tasks, run once before training and every 10 steps, with one rollout per task.

Rollouts. Each rollout is a multi-turn episode in the hybrid setting. Every assistant turn contains a `Thought:` paragraph and one tool call, with at most 1,024 generated tokens per turn. Unlike SFT, the screenshot history of the episode remains in context, resized so that the short side is 448 pixels and capped at 32 images. The action space contains nine `mobile_use` actions (including `open_app`) and app-native tools. An application's tool catalog is provided when it comes to the foreground; each tool call returns a structured receipt containing its output or an error code. An episode ends when the agent calls `terminate` or reaches the smaller of the task-specific and global step limits (Table 5).

Reward. The reward of a rollout is

$$r = \underbrace{v}_{\text{verifier}} \cdot \underbrace{m_{\text{tool}}}_{\text{tool bonus}} \cdot \underbrace{\ell}_{\text{length factor}} - 0.2\, n_{\text{fmt}} - 0.2\, f_{\text{succ}}.$$

Here $v \in \{0, 1\}$ is a relaxed state verifier: the goal is met, and any extra state changes are benign. Reported accuracy uses the strict verifier, which also requires a clean state. In the hybrid setting, $m_{\text{tool}} = 2$ when the trajectory contains at least one app-native tool call, and $m_{\text{tool}} = 1$ otherwise. The length factor ℓ is 1 for responses up to 12,000 tokens and decreases linearly to 0.5 at 28,672 tokens. n_{fmt} is the number of malformed turns, and $f_{\text{succ}} = 1$ when the agent claims completion but the verifier fails.

Infrastructure. Training runs on 32 H800 GPUs across 4 nodes. The actor is a rank-64 LoRA (α=64) on the attention and MLP projections, with the vision tower frozen, trained in bf16 with FSDP2 and Ulysses sequence parallelism of degree 4. No parameters, optimizer states, or activations are offloaded. Rollouts are generated by a colocated vLLM engine with tensor parallelism 4. Each node hosts a MOBILEGYM++ simulator, and rollouts drive 64 browser pages concurrently (32 workers, two pages each). The learning rate is 1×10^{-6} after 40 warmup steps, with an entropy coefficient of 0.001.

A.3 EVALUATION BENCHMARKS

We evaluate on five benchmarks that differ in where the apps run, which actions the agent may take, and how completion is judged, so a gain that holds on all of them is unlikely to come from fitting one environment.

AndroidWorld (Rawles et al., 2025) is a widely adopted benchmark for evaluating mobile agents in reproducible Android environments. It comprises 116 programmatic tasks across 20 real-world apps, with deterministic evaluation based on the resulting device state. We report task success rate.

MobileWorld (Kong et al., 2026) is a recent benchmark designed to evaluate more challenging mobile-agent capabilities. It contains 201 tasks across 20 apps, with an emphasis on long-horizon

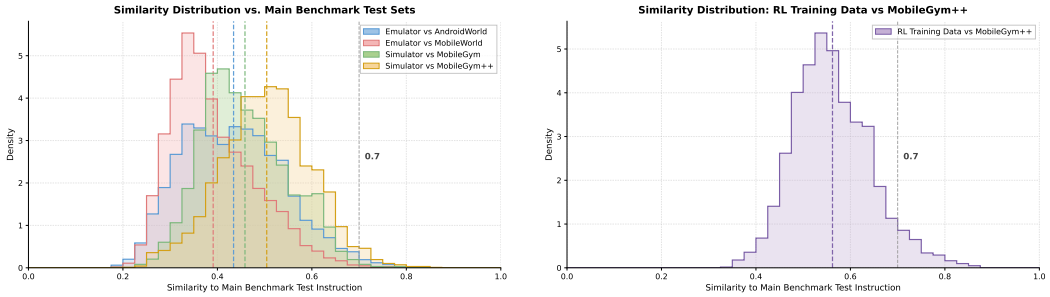
and cross-application workflows. We evaluate both its GUI-only tasks and MCP-enabled tasks using the official task checker.

MobileGym (Wu et al., 2026a) is a benchmark built around resettable mock applications with programmatic state-based evaluation. It evaluates agents on tasks requiring application interactions and state transitions, and reports both success rate (SR) and progress rate (PR), capturing complete and partial task completion, respectively.

MOBILEGYM++ Bench (Section 3.5) is designed to evaluate mobile agents on tasks involving application business state and native tool interactions. It contains 215 tasks spanning single-app and cross-app workflows, each initialized from a fixed application state and evaluated by a programmatic checker over the application business state. We evaluate both GUI-only and hybrid GUI and app-native tool interaction on the same tasks, initial states, and checker.

SpaBench (Chen et al., 2025) evaluates mobile agents on real Android devices, providing an evaluation setting that goes beyond the controlled emulator and simulator environments used in training and the other benchmarks. We use SpaBench to assess whether the capabilities learned from OpenMobile-2 transfer to real-device interaction.

A.4 OVERLAP WITH EVALUATION BENCHMARKS



(a) SFT goals (b) RL goals

Figure 6: Per-goal maximum cosine similarity. (a) SFT goals against the matched benchmark. (b) RL task goals against MOBILEGYM++ Bench. Vertical lines mark each mean and the 0.7 threshold.

We compare training goals with the evaluation instructions of the four benchmarks that share a runtime with training. Emulator goals (7,718 unique) are compared with AndroidWorld and MobileWorld; simulator goals (3,916 unique) are compared with MobileGym and MOBILEGYM++ Bench. Each goal is embedded with `text-embedding-3-large` (3,072 dimensions, L2-normalized), and we record its maximum cosine similarity to that benchmark. Table 7 and Figure 6a summarize these scores. No training goal is an exact string match of an evaluation instruction, and none exceeds similarity 0.9. SpaBench is not included, because its tasks run on physical devices outside these two pools.

Table 7: Maximum cosine similarity of each SFT training goal to the matched benchmark. Above 0.7 counts goals whose maximum exceeds 0.7.

Benchmark	Train goals	Mean	Max	Above 0.7
AndroidWorld	7,718	0.434	0.892	1.11% (86)
MobileWorld	7,718	0.391	0.851	0.39% (30)
MobileGym	3,916	0.458	0.826	0.46% (18)
MOBILEGYM++ Bench	3,916	0.504	0.861	2.25% (88)

The same embedding is applied to the 2,475 RL task goals, compared only with MOBILEGYM++ Bench (Figure 6b). Exact string overlap is again zero, and none exceeds 0.9. The mean maximum similarity is 0.562, the maximum is 0.862, and 6.1% (151) exceed 0.7.

B MOBILEGYM++ APP CONSTRUCTION

Behavior specification. We represent core application workflows as a structured graph of page nodes and meaningful UI states. Each state-changing interaction names its UI element and action, describes the expected state effect, and includes a check for the resulting state. This makes navigation and business-state writes explicit before implementation. The specification is then used as the contract for implementing and checking routes, actions, and state changes.

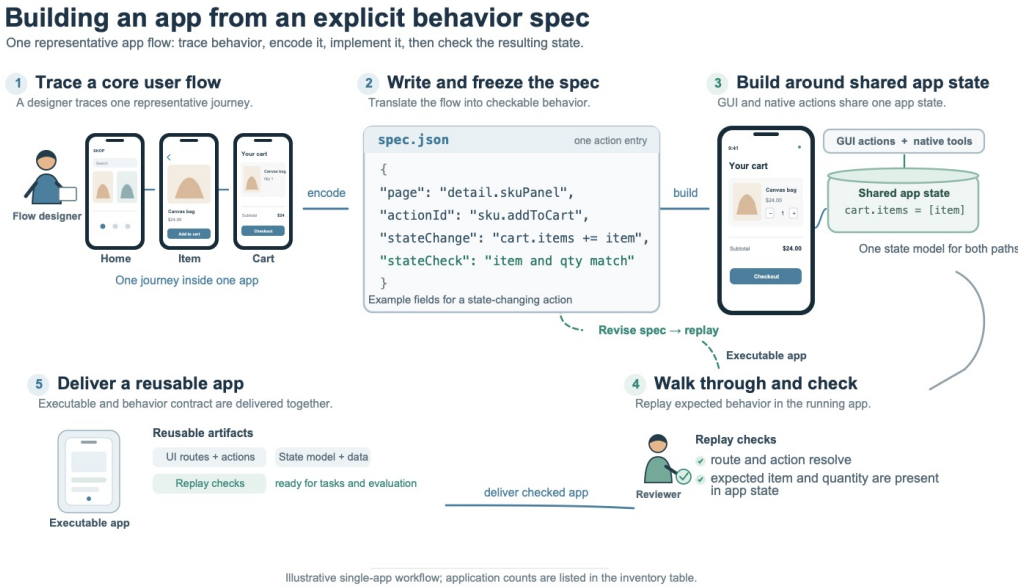


Figure 7: Illustrative single-app workflow. The phone screens show one journey; the numbered circles mark construction stages, not application counts.

Specification entry. The excerpt below shows the form of a state-changing interaction. A page identifier groups its controls; the action identifier names the operation; and the state change and check make the outcome observable.

```
{
  "id": "detail.skuPanel",
  "interactions": [{
    "element": "Add to cart",
    "effect": "state",
    "actionId": "sku.addToCart.confirm",
    "stateChange": "cart.items += item",
    "stateCheck": "item and quantity match"
  }]
}
```

Implementation and checks. Routes and actions operate on app-owned state and seeded data. GUI actions and app-native tools use the same app state. A coverage check maps declared interaction identifiers to implementation bindings; runtime replay checks representative transitions and their expected state effects. When a mismatch appears, the specification is corrected and the affected checks are replayed. Each delivered app includes its executable interface, state model, seeded data, and reusable checks for core flows. Benchmark instructions and terminal verifiers are authored separately on top of the completed apps.

C PLAYGROUND INVENTORY

MOBILEGYM++ contains 63 installable simulated clients: 35 applications built in this work and 28 inherited from MobileGym. Of these, 47 are commercial-style applications and 16 are system clients. The Android emulator inventory contains 62 applications. The tables below list applications by domain; the emulator inventory is not divided by source batch.

Table 8: Application counts by domain in the two runtimes.

Domain	MOBILEGYM++	Emulator
Lifestyle	18	19
Productivity	15	23
Entertainment	9	15
Shopping	8	1
Travel	5	3
Social	6	0
Finance	2	1
Total	63	62

C.1 MOBILEGYM++ APPLICATIONS

Table 9 gives the complete 63-client inventory; * marks the 35 applications built in this work.

Table 9: The 63 MOBILEGYM++ clients. * marks the 35 applications built in this work and † marks the 16 system clients.

Domain	Applications
Lifestyle (18)	Baicizhan*, Cainiao*, Dianping*, Keep*, Luckin Coffee*, Maoyan*, Meituan*, Meiyou*, Weather†, Answer Sheet†, Browser†, Calculator†, Calculator 2†, Clock†, Compass†, Gallery†, Settings†, Themes†
Productivity (15)	BOSS Zhipin*, Days Matter*, TickTick*, Fanqie ToDo*, flomo*, Google Drive*, Mail*, Slack*, Daily*, Tencent Meeting, Calendar†, Phone†, Files†, Notes†, Messages†
Entertainment (9)	Podcasts*, Fanqie Novel*, Tencent Video*, NetEase Cloud Music†, Ximalaya*, Youku*, Bilibili, Spotify, WeRead
Shopping (8)	JD*, Pinduoduo*, Suning*, Taobao*, Taobao Instant Commerce*, Vipshop*, Xianyu*, eBay
Travel (5)	Ctrip*, DiDi*, Qunar*, Maps, Railway 12306
Social (6)	Douban*, Weibo*, RedNote, Reddit, WeChat, X
Finance (2)	Alipay, Kapi*

C.2 EMULATOR APPLICATIONS

Table 10: The 62 Android emulator applications, grouped by domain. AndroidWorld and MiniWoB denote the task-support apps installed with the AndroidWorld suite.

Domain	Applications
Lifestyle (19)	Camera, Chrome, Clock, Settings, Simple Gallery Pro, AndroidWorld, Broccoli, Clipper, MiniWoB, OpenTracks, APKPure, Breezy Weather, Coursera, DuckDuckGo, F-Droid, NerdCalci, Snapseed, Quark, Xiachufang
Productivity (23)	Files, Contacts, Dialer, Simple SMS Messenger, Simple Calendar Pro, Joplin, Markor, Tasks, Simple Draw Pro, Audio Recorder, Amaze, Material Files, MiXplorer, MT Manager, Etar, Omni Notes FOSS, Code Editor, DeepL, GeminiAssist, Doubao, Element, Mattermost, Session
Entertainment (15)	VLC, Retro Music, iQIYI, Youku, Bilibili, NetEase Cloud Music, NewPipe, AntennaPod, Toutiao, Yahoo News, Yahoo Sports, Wikipedia, Librera, 2048, Angry Birds Friends
Shopping (1)	eBay
Travel (3)	OsmAnd, Booking.com, Citymapper
Finance (1)	Pro Expense

D DATA COLLECTION

D.1 CONSTRUCTION DETAILS

Environment representation and task synthesis. Task synthesis is grounded in the functions an application supports and the states reachable from a given screen. For Android emulator applications, exploration is used to map screens and transitions. For MOBILEGYM++, the page, action, and business-object declarations are available from each simulated client. A global view summarizes an application's features and where they are located; a local view lists states directly reachable from the current screen. Instructions are phrased as user goals rather than naming the controls, tools, or verifier fields to use.

Trajectory rollout and filtering. The same model executes each instruction step by step, producing a `Thought:` paragraph followed by one action per turn. When the foreground application exposes app-native tools, its tool catalog is available and the model may choose a tool call or GUI action. Both channels act on the same application state. A rollout is retained when the task succeeds under its runtime-specific check: the programmatic verifier for MOBILEGYM++ or a vision-language judge for Android emulator tasks. Degenerate transitions that leave application state unchanged are removed.

D.2 EXECUTABLE RL TASKS

Each executable RL task is a triple of instruction, initial state, and verifier. The initial state is a snapshot of the MOBILEGYM++ application stores, so repeated rollouts of a task start from the same state. The verifier compares application state before and after an episode and checks whether the requested change was written. Since it reads application state rather than the screen, it evaluates GUI and app-native tool actions alike.

D.3 DATA COMPOSITION

Source coverage. The SFT mixture contains 11,643 trajectories. Of these, 7,725 come from emulator(about two third), while 3,918 come from simulator (about one third; Table 6). The mixture therefore combines trajectories from Android emulator applications with controllable simulated-client workflows.

Trajectory length and action channels. Mean trajectory lengths range from 6.8 steps for hybrid trajectories to 11.9 for AndroidWorld, with MobileGym at 10.7 and MobileWorld at 8.8. The 1,774 hybrid trajectories use app-native tools alongside GUI actions.

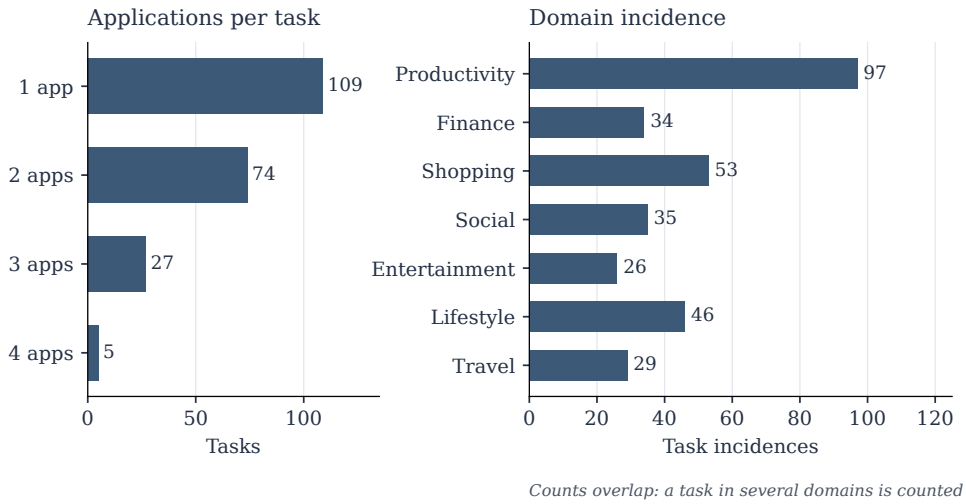


Figure 8: MobileGym++ Bench composition. Left: tasks by the number of applications they touch. Right: task incidences by domain. Domain counts are not mutually exclusive.

Table 11: MobileGym++ Bench task incidences by domain. A task that touches more than one domain is counted in each, so the column does not sum to 215.

Domain	Task incidences
Productivity	97
Finance	34
Shopping	53
Social	35
Entertainment	26
Lifestyle	46
Travel	29

E MobileGym++ Bench

The following subsections document the task split and domain distribution for MobileGym++ Bench, and give two worked examples that illustrate the evaluation protocol concretely.

E.1 Task composition

The 215 tasks span one to four applications. Of these, 109 use one app, 74 use two, 27 use three, and 5 use four; 106 tasks therefore require cross-app interaction. Figure 8 and Table 11 report the same split and the task incidences by domain. Because a cross-domain task is counted once in each domain it touches, these incidences overlap and do not sum to 215.

E.2 Example tasks

GUI-only: transfer search results into Notes. Task 202 (`HTV6Case27`) asks the agent to search RedNote (Xiaohongshu) using the Chinese term for travel, identify the two most-liked posts among the first ten results, and write their full titles to the system Notes app, one per line (not to flomo). In the GUI-only setting, the agent can use the `mobile_use` interface actions but receives no app-native tools. The programmatic checker independently retrieves the top two titles from the initial search results and verifies that both appear in the latest Notes entry.

Hybrid: complete a Taobao purchase. Task 111 (`HTV6Case69`) asks the agent to buy a Jianmu large-capacity insulated flask made of 316 stainless steel, choose the saved address marked School, and

pay using Huabei installments. The hybrid action set combines GUI actions with Taobao app-native tools; relevant calls include `taobao__search_items`, `taobao__get_item_summary`, and `taobao__add_to_cart`. `taobao__prepare_checkout` opens the confirmation page as a non-persisted draft, so the agent must still complete the payment flow in the GUI. The checker requires a new paid order for the target item, with the school-tagged address and Huabei installment payment recorded in the order.

F APP-NATIVE TOOL EXAMPLES

App-native tools occupy the same action slot as `mobile_use` GUI interactions: at each step, the agent chooses one action from a single list that includes both. When an application is in the foreground, its tool catalog is appended to that list; when the application leaves the foreground, its tools are removed. The following Taobao schemas illustrate read, state-changing, and navigation-to-GUI operations. The tool handlers and the visible application use the same `useTaobaoStore`: for example, `add_to_cart` updates the cart state that the GUI renders. Tool schemas reject undeclared input keys.

```
taobao__search_items(query: string, limit?: integer)
taobao__get_item_summary(item_id: string)
taobao__add_to_cart(item_id: string, sku_description: string,
                    quantity: integer [1..99])
taobao__prepare_checkout(cart_line_ids: string[1..30])
```

The first two calls read the catalog; `add_to_cart` validates the item, SKU, and quantity before writing the shared cart. `prepare_checkout` accepts one to 30 valid cart-line IDs, opens the confirmation page, and creates neither an order nor a payment. The GUI then continues from the same app state.

G SYSTEM PROMPT

The prompt below is the evaluation harness for Qwen3.5-9B, Qwen3.6-27B, and the OpenMobile-2 checkpoints trained from them. It is used in both the GUI-only and hybrid settings. The GUI tool `mobile_use` is always available; when an application exposes app-native tools, the environment lists them in the tool message for as long as that application stays in the foreground. In the GUI-only setting no such catalog is attached. Results marked $*$ for other models in Table 2 are reproduced by us, but not with this harness. Each of those models keeps its own action format and system prompt.

System prompt for Qwen and OpenMobile-2 evaluation

```
# Tools

You have access to the following functions:

<tools>
{"type": "function", "function": {
  "name": "mobile_use",
  "description": "Use a touchscreen to interact with a mobile device, and take
  ↪  screenshots.\n* This is an interface to a mobile device with touchscreen. You can
  ↪  perform actions like clicking, typing, swiping, etc.\n* Some applications may take
  ↪  time to start or process actions, so you may need to wait and take successive
  ↪  screenshots to see the results of your actions.\n* The screen's resolution is
  ↪  999x999.\n* Make sure to click any buttons, links, icons, etc with the cursor tip in
  ↪  the center of the element. Don't click boxes on their edges unless asked.",
  "parameters": {
    "properties": {
      "action": {
        "description": "The action to perform. The available actions are:\n* `click`: Click
        ↪  the point on the screen with coordinate (x, y).\n* `long_press`: Press the
        ↪  point on the screen with coordinate (x, y) for specified seconds.\n* `swipe`:
        ↪  Swipe from the starting point with coordinate (x, y) to the end point with
        ↪  coordinates2 (x2, y2).\n* `type`: Input the specified text into the activated
        ↪  input box.\n* `answer`: Output the answer.\n* `system_button`: Press the system
        ↪  button.\n* `wait`: Wait specified seconds for the change to happen.\n*
        ↪  `terminate`: Terminate the current task and report its completion status.",
        "enum": ["click", "long_press", "swipe", "type", "answer", "system_button",
        ↪  "wait", "terminate"],
```

```
        "type": "string"},
      "coordinate": {
        "description": "(x, y): The x and y coordinates. Required only by `action=click`,
        ↪  `action=long_press`, and `action=swipe`.",
        "type": "array"},
      "coordinate2": {
        "description": "(x, y): The end coordinates. Required only by `action=swipe`.",
        "type": "array"},
      "text": {
        "description": "Required only by `action=type` and `action=answer`.",
        "type": "string"},
      "time": {
        "description": "The seconds to wait. Required only by `action=long_press` and
        ↪  `action=wait`.",
        "type": "number"},
      "button": {
        "description": "Required only by `action=system_button`.",
        "enum": ["Back", "Home", "Menu", "Enter"],
        "type": "string"},
      "status": {
        "description": "Required only by `action=terminate`.",
        "enum": ["success", "failure"],
        "type": "string"}
    },
    "required": ["action"],
    "type": "object"}
}}
</tools>

If you choose to call a function ONLY reply in the following format with NO suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format: an inner <function=...></function>
↪  block must be nested within <tool_call></tool_call> XML tags
- Required parameters MUST be specified
- Put reasoning after `Thought:` BEFORE the function call, and do not write anything after
↪  </tool_call>
- Every assistant turn MUST include exactly one function call. Do not answer the user in
↪  plain text.
- The always-available GUI tool is `mobile_use`. When a tool message lists extra app tools
↪  inside `<tools>...</tools>`, you may call one of those tools with the same XML format
↪  while that app stays in the foreground. Do not call tools from a previous app, and do
↪  not invent tool names.
</IMPORTANT>

You are a mobile GUI agent. Prefer `mobile_use` for touchscreen interaction; use extra app
↪  tools only when they are currently listed and clearly help the task.

# Session

- The task query is provided only once in the first user message, together with the
↪  initial screenshot.
- After each tool call, the environment returns a tool message containing the resulting
↪  screenshot, and may also attach a foreground-app tool catalog.
- Previous assistant reasoning and tool calls remain in the conversation history.
- Each turn: look at the screenshot(s) still provided as images, plus the full history
↪  (including any currently listed extra tools), then make exactly one tool call.
- When the task is complete or cannot be completed, call `mobile_use` with action=terminate
↪  and status=success or status=failure.

# Tool call

- Every assistant turn must be exactly: a `Thought:` paragraph, then one
↪  <tool_call>...</tool_call> block.
- Emit Qwen XML:
↪  <tool_call><function=NAME><parameter=...>...</parameter></function></tool_call>
```

```
- Do not emit a JSON object such as {"action":"click","coordinate":[x,y]} or
↪  {"name":"...","arguments":{...}}.
- One <parameter> block per argument. Arrays/objects must be valid JSON inside the block
↪  (e.g. [163, 718]).
- Do not output anything after </tool_call>.

# Visual context

- Only the most recent screenshot slots remain as images. Older slots are replaced with
↪  the text "This screenshot has been collapsed."
- Act only on images that are actually present. Do not recover pixels from collapsed slots.

# Robustness

- If the history shows repeated similar actions without progress, try a different action.
- Never assume an earlier action succeeded without checking the latest screenshot.
```