AppSim-Bench: Bridging Real-world Apps and Reproducible Evaluation for Mobile GUI Agents

Jintian Feng[1,2]*, Long Chen[3], Xiao Yu[1,2], Jiayi Dai[1,2], Chenglong Liu[2], Haoru Wang[2], Zizhen Xue[3], Yuxuan Shi[1,2]†, Ziyang Wang[3], Yichen Gong[3]

[1]Laboratory for Artificial Intelligence and New Forms of Education, Central China Normal University
[2]Faculty of Artificial Intelligence in Education, Central China Normal University
[3]Agentic Labs, Acrab AI
fjt2018@mails.ccnu.edu.cn

Abstract

Mobile GUI agents can execute tasks from natural-language instructions, but their evaluation remains difficult to make both realistic and reproducible. Existing benchmarks typically trade off these goals: simplified apps lack real-world mobile complexity, whereas live commercial apps introduce uncontrolled variation from recommendations, advertisements, accounts, and changing content. We propose AppSim-Bench, which addresses this trade-off through controllable simulated apps that preserve task-relevant interaction logic while supporting deterministic evaluation. Built through a coding-agent-assisted and human-verified workflow, it contains 557 tasks across 17 high-frequency Chinese and English apps. Its controllable backend data and outcome-based verification remove major sources of environmental stochasticity, enabling reproducible cross-model comparison. Evaluating 19 GUI agents, spanning general-purpose and GUI-specialized systems, we find that autonomous mobile execution remains far from solved. The best model completes only 50.27% of tasks, and 28.55% of tasks are not solved by any agent. Further analysis shows that failures concentrate in longer workflows, numerical reasoning tasks, and inefficient trajectories marked by high action overhead and budget exhaustion. Our project is available at https://github.com/Acrab-Lab/AppSim.

1 Introduction

With the rapid development of Vision-Language Models (VLMs), Mobile GUI Agents are emerging as an important research direction in Human-Computer Interaction (HCI)(Liu et al., 2025b). These agents(Wang et al., 2025; Agashe et al., 2025; Ye et al., 2025; Qin et al., 2025; Lai et al., 2024; Zhang et al., 2023) can understand natural language instructions and autonomously complete multi-step tasks on mobile devices. However, how to systematically and reproducibly evaluate their true capabilities remains a critical, unsolved problem(Liu et al., 2025a).

Early evaluation methods often judge task completion by comparing the agent's execution path to a human-annotated "golden" path(Zhang et al., 2025; Rawles et al., 2023). However, such action-based approaches struggle with the common "multi-path problem" in mobile applications, making it difficult to measure the agent's true task-completion ability(Liu et al., 2025b,a; Nguyen et al., 2025). For example, in a shopping task, an agent may complete the purchase through a valid but unannotated path. A trace-based evaluator would still mark this attempt as a failure, exposing a basic limitation of static benchmarks.

To overcome these limitations, dynamic benchmarks are becoming the mainstream approach. Represented by AndroidWorld(Rawles et al., 2025), these benchmarks provide agents with a realistic and interactive environment to complete tasks autonomously. Success is judged by monitoring changes in the system state, such as UI state or database updates. This marks a shift in the evaluation paradigm from "trace verification" to "result verification"(Chen et al., 2025; He et al., 2025; Yang et al., 2025). This lays the foundation for a more realistic and accurate measurement of an agent's decision-making and execution abilities.

Despite these efforts, the development of Mobile GUI Agent benchmarks still faces several key challenges: (1) Some benchmarks evaluate agents mainly on system applications or open-source apps (Wang et al., 2024; Rawles et al., 2025), which differ substantially from everyday third-party mobile services. (2) Benchmarks involving commercial apps often require running live

*Work done as a research intern at Acrab AI.
†Yuxuan Shi is the corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 36823–36845
October 24-29, 2026 ©2026 Association for Computational Linguistics

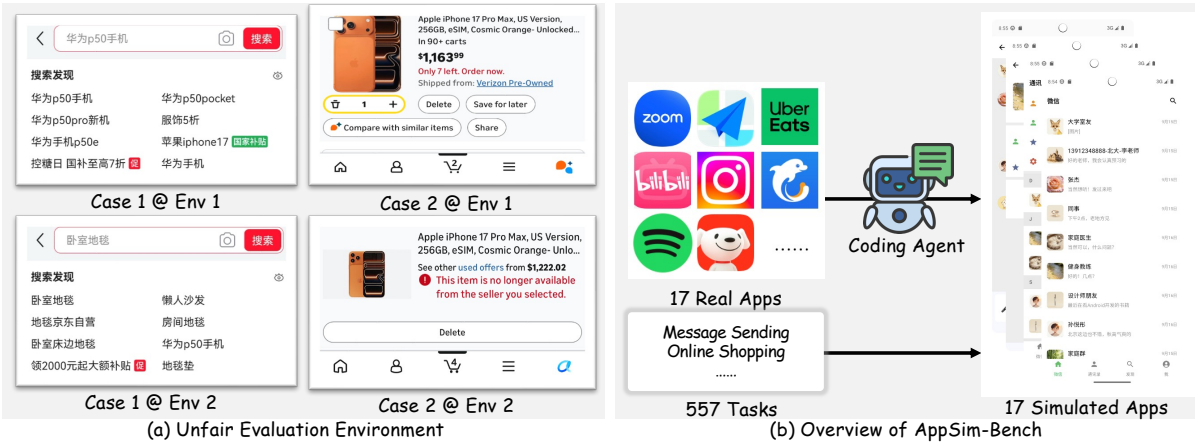


Figure 1: (a) Two cases showing inconsistent app states across time or devices. In Case 1, **search suggestions** differ across environments and may affect subsequent agent actions. In Case 2, **shopping-cart availability** changes across environments, making fair comparison difficult when the task depends on this state. (b) Overview of AppSim-Bench, which constructs 17 simulated apps covering 557 tasks.

apps on physical devices (Chen et al., 2025; Zhang et al., 2024b). These environments are difficult to reproduce because recommendation algorithms, advertisements, account states, and dynamic page loading can change across runs.

As shown in Fig.1 (a), even for the same task, different agent attempts may face entirely different environmental states. This difference directly violates the fundamental reproducibility principle of benchmark design, affecting the comparability and reliability of the results.

In summary, building an evaluation environment that is both realistic and interactive, yet also controllable and reproducible(Reuel et al., 2024; Kapoor et al., 2023), has become a core need for Mobile GUI Agent research. Although existing work(Zhang et al., 2024a) has noted the reproducibility issue, a general and efficient solution is still lacking. To this end, we propose AppSim-Bench: a new benchmark framework to systematically evaluate smartphone agents on everyday tasks.

AppSim-Bench addresses this need by constructing simulated apps with task-relevant UI flows and controllable backend data. Rather than interacting with live commercial services, agents operate in sandboxed environments whose initial states and content can be reset before each run. This design preserves realistic interaction patterns while removing major sources of evaluation noise from recommendation systems and dynamic content.

The benchmark is constructed through a coding-agent-assisted, human-verified workflow. Researchers specify the task requirements, data schema, page hierarchy, and representative screenshots, while a coding agent generates candidate implementations under these constraints. Human developers then inspect, execute, and iteratively correct the generated apps before they are used for evaluation. This workflow reduces repeated implementation effort, but the benchmark contribution lies in the resulting controlled evaluation environment and task suite.

Our main contributions are as follows:

- We introduce AppSim-Bench, a reproducible Mobile GUI Agent benchmark with 557 tasks across 17 simulated apps from Chinese and English mobile ecosystems.
- We design a deterministic evaluation protocol that separates benchmark specification, reference execution, and official scoring, covering state, answer, and hybrid task types.
- We evaluate 19 GUI agents and find that reliable mobile GUI execution remains far from solved: the strongest model completes only about half of the tasks, nearly one third of tasks are unsolved by any evaluated agent, and performance drops with task length, numerical constraints, and inefficient execution.

2 Related Work

2.1 Mobile GUI Agent

Benefiting from the rapid advances of vision-language models (VLMs) in visual understanding

Table 1: Comparison of benchmark characteristics. AppSim-Bench bridges the gap between fidelity and reproducibility, pioneering the reproducible evaluation of high-frequency tasks derived from real-world mobile usage. In the context, "Automated?" indicates whether the evaluation is fully automated without human intervention, whereas "Accurate?" indicates whether the evaluation is robust and noise-free.

Benchmark	3rd-party App	Reproducible	#Tasks	#Apps	Avg. Tasks	Automated?	Accurate?
AndroidArena(Xing et al., 2024)	×	×	221	13	17.0	×	×
AndroidWorld(Rawles et al., 2025)	✓	✓	116	20	5.8	×	✓
LlamaTouch(Zhang et al., 2024b)	✓	×	496	57	8.7	×	×
B-MoCA(Lee et al., 2025)	×	×	60	18	3.3	×	×
MobileAgentBench(Wang et al., 2024)	×	×	100	10	10.0	×	×
SPA-BENCH(Chen et al., 2025)	✓	×	340	68	5.0	✓	×
AppSim-Bench (ours)	✓	✓	557	17	32.8	✓	✓

and reasoning, research on Mobile GUI Agents has progressed rapidly in recent years (Guo et al., 2025; Qin et al., 2025; Wang et al., 2025). Leveraging these capabilities, GUI Agents can comprehend natural-language instructions, recognize the semantics and spatial layout of on-screen elements, and plan and execute multi-step actions accordingly.

Recent research has approached this direction from several perspectives: (1) employing prompt engineering to adapt general-purpose large models for GUI interaction tasks directly (Gao et al., 2024; Zheng et al., 2024); (2) performing task-specific fine-tuning on subtasks such as interface understanding and element grounding (Lai et al., 2024; Liu et al., 2025b; Wu et al., 2024; Zhang et al., 2025; Zhou et al., 2024); and (3) constructing agent frameworks where specialized modules cooperate to form complete GUI workflows (Agashe et al., 2025; Ye et al., 2025; Zhang et al., 2023).

Despite these advances, deploying current GUI Agents in real-world or production environments remains challenging due to issues such as semantic ambiguity of user intents and the complex functionality of mobile applications (Koh et al., 2024; Liu et al., 2025a; Nguyen et al., 2025). These challenges highlight the urgent need for a benchmark that can reflect realistic usage scenarios while ensuring reproducibility and fairness.

2.2 GUI Agent Benchmarks

To systematically evaluate GUI Agents, early benchmarks often relied on static screenshots, recorded demonstrations, or human-annotated action traces (Zhang et al., 2025; Rawles et al., 2023). These settings are scalable and easy to standardize, but they often judge task completion by comparing the agent's action sequence with a human-annotated "golden" trace. Such trace-based evaluation struggles with the common multi-path problem in real Apps, where the same goal can be completed through different valid trajectories.

Subsequent studies began exploring dynamic evaluation frameworks, which execute interactive applications on virtual machines or physical devices to provide agents with testing environments that more closely resemble real-world usage scenarios (Bonatti et al., 2024; Xie et al., 2024; Mialon et al., 2023; Deng et al., 2023). For example, AndroidWorld(Rawles et al., 2025) offers agents a full execution interface and determines task completion by monitoring changes in system states. However, such approaches typically rely on system apps or open-source applications (Rawles et al., 2025; Zhang et al., 2024a), as accessing backend states of commercial apps is often infeasible.

To evaluate on more everyday tasks, recent works have adopted physical-device-based evaluation, where agents interact directly with real mobile apps, and task success is assessed manually or via VLM-based evaluators (He et al., 2025; Chen et al., 2025; Liu et al., 2025b). While these attempts significantly increase the realism of evaluation, they still suffer from poor reproducibility and a lack of environmental control. Recommendation algorithms, advertisement placements, and dynamic content updates can cause the same task to present different UI states at different times, thereby undermining the fairness and reliability of model comparisons. Table 1 summarizes these benchmark differences in terms of app coverage, reproducibility, scale, and evaluation protocol. In summary, existing benchmarks face a clear trade-off between realism and reproducibility. This has become a major bottleneck for further research in GUI Agents.

A complementary line of work pursues the same goals of controllability and reproducibility through simulated app environments. AppWorld (Trivedi et al., 2024) pioneered this direction with a control-

lable execution environment of 9 apps and state-based unit tests, and shares with us the principles of interactivity, reproducibility, and outcome-based verification. It targets a different agent paradigm, however: agents act by generating code against 457 APIs, whereas AppSim-Bench targets interface-based GUI agents that must ground every action in on-screen pixels through primitives such as tap, swipe, and type. The two settings therefore probe distinct capabilities, namely program synthesis and API composition as opposed to visual grounding and UI navigation. More recent platforms share our simulation-based premise but optimize for scalable training rather than evaluation, and accordingly adopt browser-hosted or synthesized implementations (Wu et al., 2026; Liu et al., 2026). AppSim-Bench instead runs on a native Android runtime and covers both Chinese and English ecosystems.

3 AppSim-Bench

AppSim-Bench serves as a robust framework for evaluating mobile GUI agents within realistic, controllable, and reproducible environments. The benchmark comprises three core components: (1) **Tasks**, which describe user intentions in natural language; (2) **Simulated Apps**, a suite of task-relevant mobile application environments equipped with controllable backend data; and (3) **Automated Evaluation** that determines task completion via deterministic outcome verification across state-based, answer-based, and hybrid task types. The overall benchmark configuration is illustrated in Figure 1(b), while a comprehensive summary of the task composition is provided in Figure 2.

Task Construction AppSim-Bench comprises 557 tasks across 17 simulated mobile applications from both Chinese and English application ecosystems, including WeChat, JD, RedNote, Tencent Meeting, Amap, Ctrip, Bilibili, Ele.me, NetEase Cloud Music, Amazon, Booking.com, Instagram, Spotify, Uber Eats, WhatsApp, YouTube, and Zoom. These applications encompass a broad spectrum of high-frequency, ubiquitous mobile scenarios, spanning social communication, e-commerce, travel, food delivery, navigation, video sharing, music streaming, and video conferencing.

To ensure realism and relevance, we conducted a user study with 15 volunteers to collect representative, real-world interactions, such as "*sending message to friend*" and "searching and purchasing a product online". These raw interaction samples formed the basis of our initial task templates, which were subsequently refined and standardized through human annotation to yield natural, unambiguous, and machine-interpretable instructions.

Furthermore, we curate a specialized subset of **numerical-reasoning tasks**. These tasks demand that the agent extract numerical information from the graphical interface and execute mathematical or logical operations during interaction. We categorize these tasks into four non-mutually exclusive subtypes: counting, arithmetic calculation, numeric comparison, and threshold filtering. This design empowers AppSim-Bench to benchmark how effectively GUI agents perform multi-step interface navigation with quantitative reasoning.

Simulated App Construction We construct the simulated apps through a human-led, coding-agent-assisted process. Human developers specify the task requirements, backend data schemas, page hierarchies, and task-path screenshots for each app. Given these specifications, coding agents generate candidate implementations, including page layouts, widget bindings, navigation logic, and frontend behaviors. Human developers then inspect, execute, and correct the generated apps until the task-relevant UI flows and backend states match the benchmark specification.

The resulting apps reproduce task-relevant UI flows from real-world applications while using controllable backend data. This design guarantees that every task initiates from a deterministic state, and remains unaffected by environmental stochasticity induced by personalization, advertisements, recommendation updates, and dynamic online content. Complete construction details are provided in Appendix A, and the quality-control procedure in Appendix D.

Automated Evaluation AppSim-Bench uses deterministic outcome evaluation rather than predefined action-path matching. Following AndroidWorld(Rawles et al., 2025), the evaluator checks whether the final outcome satisfies the task goal after an agent completes an interaction. We use three verification types: **(1) State tasks** require a visible or backend state change, such as sending a message or placing an order. **(2) Answer tasks** require the agent to retrieve information and return a structured answer, such as a count, price, or selected option. **(3) Hybrid tasks** require both a state change and a structured answer. The evaluator applies the corresponding deterministic check for each task type.

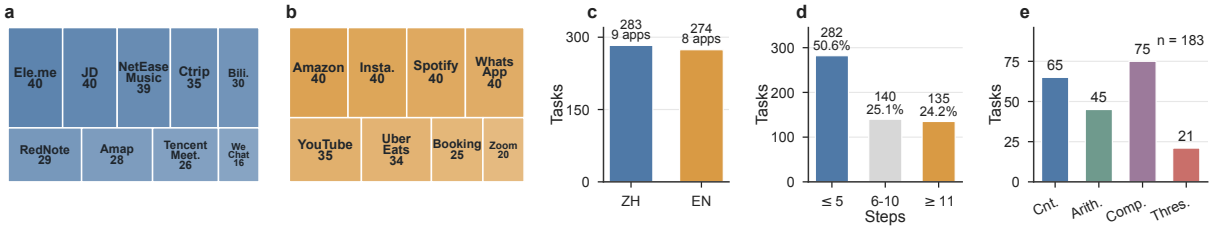


Figure 2: Statistics of AppSim-Bench. (a–b) App-level task coverage within the Chinese and English app ecosystems, where each rectangle denotes a simulated app and its task count. (c) Aggregate task coverage across both ecosystems. (d) Task distribution by human-annotated reference steps. (e) Distribution of numerical reasoning subtypes (subtypes are not mutually exclusive), here, "Cnt.", "Arith.", "Comp.", and "Thres." denote counting, arithmetic calculation, numeric comparison, and threshold filtering tasks.

Detailed schemas and examples are provided in Appendix B, and a representative agent trajectory on an answer task is shown in Appendix K.

Statistics of AppSim-Bench As summarized in Figure 2, AppSim-Bench comprises 557 tasks across 17 simulated apps from Chinese and English app ecosystems. These two ecosystems are approximately balanced, with 283 tasks from 9 Chinese apps and 274 tasks from 8 English apps. We categorize tasks by the human-annotated reference steps required for completion: 282 tasks require at most 5 steps, 140 require 6–10 steps, and 135 require at least 11 steps. Among all tasks, 183 are annotated as numerical-reasoning tasks, with subtype labels for counting, arithmetic calculation, numeric comparison, and threshold filtering. Full statistics are reported in Appendix H.3.

4 Experiments

4.1 Experimental Setup

Baselines. We evaluate 15 powerful general-purpose agents from five model families on AppSim-Bench: GPT-(5.5/5.4/5), Claude-(Opus-4.7/Sonnet-4.6/Sonnet-4.5/Haiku-4.5), Gemini-(3.1/3/2.5)-Pro, Qwen-3.6-(Plus/Flash), and Doubao-Seed-(2.0/1.8/1.6). Qwen and Doubao are evaluated with their official GUI agent prompts and tool schemas. For Claude, GPT, and Gemini, we adapt M3A (Rawles et al., 2025), retaining its mobile observation and action interface, but use a single-model version to match the setting of Qwen and Doubao. We further evaluate 4 GUI-specialized agents that are post-trained for GUI interaction: MobileAgent-v3.5 (Ye et al., 2025), UI-TARS-1.5-7B (Qin et al., 2025), AgentCPM-GUI (Zhang et al., 2025), and V-Droid (Dai et al., 2026), each integrated with its own official GUI prompt and action schema. Among them, MobileAgent-v3.5 is the only multi-agent system. This yields 19 evaluated agents in total.

Evaluation Protocol. All experiments are conducted on devices running Android 13 (API 33) at a screen resolution of 1080×1920. Before each task begins, we clear the app state, restore the task-specific initial state, and reset the agent history. At each step, the agent receives the task instruction, current screen observation, and a concise action history summarizing previous operations within the same task. We set the temperature to 0 for all evaluated agents, and cap each run at 50 steps. More details are reported in Appendix J.

Metrics. We evaluate all baseline agents' performance using a single binary metric: the agent receives a score of 1 when the final goal condition is satisfied, and 0 otherwise. For state tasks, this score is produced by checking whether the final app state satisfies the predefined goal condition. For the answer and hybrid evaluation cases reported in this paper, we manually audited the structured-answer extraction outputs used for scoring and found no extraction errors in the audited cases. For each model evaluated on $\mathcal{T}$, let $y_i \in \{0, 1\}$ denote the score on task i, a_i denote the number of valid actions, and h_i denote the human-annotated reference steps. We define accuracy (ACC) and average actions (AvgAct) as,

$$\mathrm{ACC} = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} y_i, \mathrm{AvgAct} = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} a_i.$$

To measure efficiency of successfully completed tasks, we define Action Overhead (ActOH) as,

$$\mathrm{ActOH} = \frac{1}{\sum_{i \in \mathcal{T}} y_i} \sum_{i \in \mathcal{T}} y_i \frac{a_i}{h_i},$$

where lower values indicate trajectories closer to the human reference length.

Table 2: Accuracy summary on AppSim-Bench across overall performance, task length, and numerical-reasoning subtype. “ZH Apps” and “EN Apps” denote task-weighted accuracy over the Chinese and English app groups. Task-length groups are based on the human-annotated task step count $\#S$. “Cnt.”, “Arith.”, “Comp.”, and “Thres.” denote counting, arithmetic calculation, numeric comparison, and threshold filtering tasks. **Bold** and underlining indicate the highest and second-highest accuracies in each column. ↓ and ↑ indicate accuracy variations relative to the adjacent shorter task-length bucket. † highlights the most substantial performance degradation within each adjacent-bucket comparison. Note that numerical reasoning subtypes are not mutually exclusive.

Agent Model	Overall Benchmark			Task Length			Numerical Reasoning Subtype			
	Overall	ZH Apps	EN Apps	$\#S_{\leq 5}$	$\#S_{6\text{-}10}$	$\#S_{\geq 11}$	Cnt.	Arith.	Comp.	Thres.
General-purpose Agents										
Claude-Opus-4.7	**50.27%**	**60.07%**	40.15%	58.87%	**54.29%** ↓4.58%	28.15% ↓26.14%†	**52.31%**	55.56%	**52.00%**	**42.86%**
Gemini-3-Pro	49.91%	**60.07%**	39.42%	56.38%	52.86% ↓3.52%	**33.33%** ↓19.53%	46.15%	42.22%	49.33%	23.81%
GPT-5.5	49.73%	58.66%	**40.51%**	**59.22%**	52.14% ↓7.08%	27.41% ↓24.73%	44.62%	**57.78%**	**52.00%**	33.33%
Gemini-3.1-Pro	49.55%	59.36%	39.42%	56.74%	52.86% ↓3.88%	31.11% ↓21.75%	43.08%	42.22%	50.67%	23.81%
Doubao-Seed-1.6	39.14%	50.88%	27.01%	45.74%	43.57% ↓2.17%	20.74% ↓22.83%	23.08%	22.22%	36.00%	9.52%
GPT-5	33.39%	37.81%	28.83%	43.62%	31.43% ↓12.19%	14.07% ↓17.36%	27.69%	35.56%	40.00%	4.76%
Doubao-Seed-1.8	32.14%	34.63%	29.56%	43.26%	27.86% ↓15.40%	13.33% ↓14.53%	23.08%	31.11%	28.00%	4.76%
Qwen-3.6-Plus	31.42%	39.58%	22.99%	39.01%	31.43% ↓7.58%	15.56% ↓15.87%	23.08%	13.33%	28.00%	9.52%
Doubao-Seed-2.0	29.26%	36.40%	21.90%	37.94%	28.57% ↓9.37%	11.85% ↓16.72%	27.69%	28.89%	20.00%	14.29%
GPT-5.4	29.08%	34.63%	23.36%	38.30%	25.71% ↓12.59%	13.33% ↓12.38%	24.62%	24.44%	30.67%	9.52%
Qwen-3.6-Flash	28.19%	38.87%	17.15%	38.30%	25.71% ↓12.59%	9.63% ↓16.08%	20.00%	24.44%	26.67%	4.76%
Claude-Sonnet-4.6	27.29%	31.10%	23.36%	36.17%	27.86% ↓8.31%	8.15% ↓19.71%	29.23%	33.33%	26.67%	14.29%
Claude-Haiku-4.5	22.80%	22.61%	22.99%	34.04%	15.71% ↓18.33%†	6.67% ↓9.04%	20.00%	22.22%	20.00%	9.52%
Claude-Sonnet-4.5	21.01%	24.03%	17.88%	31.91%	14.29% ↓17.62%	5.19% ↓9.10%	23.08%	26.67%	21.33%	14.29%
Gemini-2.5-Pro	17.24%	21.20%	13.14%	26.24%	14.29% ↓11.95%	1.48% ↓12.81%	23.08%	17.78%	17.33%	9.52%
GUI-specialized Agents										
MobileAgent-v3.5	31.42%	35.34%	27.37%	41.49%	26.43% ↓15.06%	15.56% ↓10.87%	18.46%	24.44%	32.00%	4.76%
UI-TARS-1.5-7B	22.62%	31.10%	13.87%	30.14%	22.86% ↓7.28%	6.67% ↓16.19%	20.00%	8.89%	16.00%	4.76%
AgentCPM-GUI	10.05%	9.19%	10.95%	17.02%	5.71% ↓11.31%	0.00% ↓5.71%	3.08%	2.22%	5.33%	4.76%
V-Droid	9.87%	13.43%	6.20%	15.25%	7.86% ↓7.39%	0.74% ↓7.12%	15.38%	11.11%	9.33%	0.00%

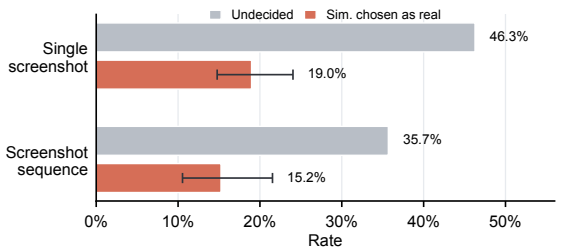


Figure 3: Blind human evaluation of visual realism. Participants compared real-app content with the corresponding simulated-app content under two settings: single screenshots and screenshot sequences from continuous interactions. “Undecided” is computed over all judgments. “Sim. chosen as real” is computed over judgments with an explicit A/B choice; error bars denote Wilson 95% confidence intervals.

4.2 Human Validation of Visual Realism

To verify whether AppSim-Bench preserves sufficient visual realism for agent evaluation, seven annotators conducted a blind A/B study over 11 sessions, comparing matched real and simulated assets across both static screenshots and dynamic interaction sequences. For each trial, evaluators identified the authentic interface or selected "undecided."

The study demonstrates the high visual fidelity of our simulated applications. For static screenshots (510 judgments), 46.3% were deemed "undecided," and the simulation was mistaken for the real app in 19.0% of the 274 decided cases. For dynamic sequences (255 judgments), the "undecided" rate was 35.7%, with the simulation selected as real in 15.2% of the 164 decided cases. These results provide empirical validation that our simulated environments are visually non-trivial to distinguish from real-world services.

Figure 7 in the Appendix compares the page-transition topologies of a real app and its simulator. The simulator faithfully preserves task-relevant pages and transitions, while strategically pruning extraneous real-app states that fall outside the scope of our benchmark tasks. Appendix E quantifies the residual effect of this pruning: under a pessimistic upper bound it accounts for at most 4.28% of measured success.

4.3 Overall Performance

Table 2 consolidates three complementary perspective of model performance: overall accuracy, task-length robustness, and subtype accuracy in

Table 3: Action efficiency and budget-exhaustion behavior. Arrows indicate the preferred performance direction. AvgAct is the average number of valid actions across all tasks. ActOH (Action Overhead) is computed on successful tasks as the ratio of valid actions to human-annotated reference steps. Budg. Exh. counts trajectories that exhaust the predefined 50-step interaction budget. Budg. ACC reports accuracy among budget-exhausted trajectories. Rep.≥3 and Rep.≥5 specify the count and percentage of budget-exhausted trajectories that contain at least three or five consecutive repetitions of the same normalized action signature.

Agent Model	ACC ↑	AvgAct ↓	ActOH ↓	Budg. Exh. ↓	Budg. ACC ↑	Rep.≥3 ↓	Rep.≥5 ↓
General-purpose Agents							
Claude-Opus-4.7	50.27%	14.01	1.98	33	0.00%	27 (81.82%)	20 (60.61%)
Gemini-3-Pro	49.91%	10.79	1.69	14	0.00%	5 (35.71%)	3 (21.43%)
GPT-5.5	49.73%	15.04	2.09	45	0.00%	27 (60.00%)	15 (33.33%)
Gemini-3.1-Pro	49.55%	10.69	1.80	19	10.53%	7 (36.84%)	3 (15.79%)
Doubao-Seed-1.6	39.14%	15.59	1.73	85	1.18%	74 (87.06%)	52 (61.18%)
GPT-5	33.39%	23.15	2.94	136	0.00%	105 (77.21%)	61 (44.85%)
Doubao-Seed-1.8	32.14%	9.71	1.60	44	0.00%	36 (81.82%)	27 (61.36%)
Qwen-3.6-Plus	31.42%	12.66	1.62	23	0.00%	11 (47.83%)	9 (39.13%)
Doubao-Seed-2.0	29.26%	10.76	1.81	46	0.00%	33 (71.74%)	28 (60.87%)
GPT-5.4	29.08%	10.88	1.56	16	0.00%	12 (75.00%)	7 (43.75%)
Qwen-3.6-Flash	28.19%	13.30	2.10	38	0.00%	27 (71.05%)	23 (60.53%)
Claude-Sonnet-4.6	27.29%	29.80	3.11	253	0.40%	222 (87.75%)	171 (67.59%)
Claude-Haiku-4.5	22.80%	31.63	4.07	221	0.00%	208 (94.12%)	162 (73.30%)
Claude-Sonnet-4.5	21.01%	23.31	3.30	93	3.23%	81 (87.10%)	55 (59.14%)
Gemini-2.5-Pro	17.24%	22.10	2.38	106	0.00%	74 (69.81%)	40 (37.74%)
GUI-specialized Agents							
MobileAgent-v3.5	31.42%	16.79	2.02	83	1.20%	69 (83.13%)	40 (48.19%)
UI-TARS-1.5-7B	22.62%	24.76	1.67	226	0.00%	156 (69.03%)	138 (61.06%)
AgentCPM-GUI	10.05%	24.82	1.71	242	0.00%	203 (83.88%)	198 (81.82%)
V-Droid	9.87%	33.30	4.60	313	0.32%	313 (100.00%)	295 (94.25%)
Summary	30.76%	18.58	2.15	2036	0.44%	1690 (83.01%)	1347 (66.16%)

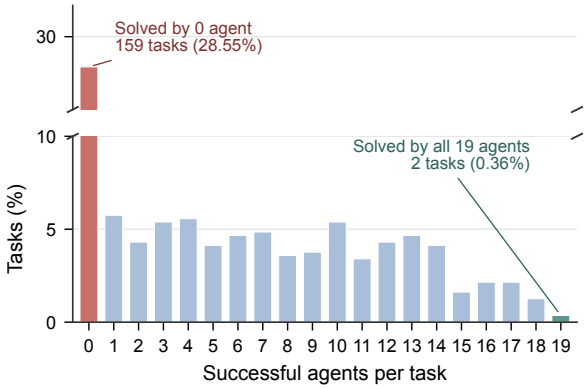


Figure 4: Cross-agent task solvability. The chart shows the proportion of tasks solved by k agents.

numerical-reasoning. The overall results underscore that reliable autonomous GUI task execution remains an elusive goal for state-of-the-art agents. The frontier model, Claude-Opus-4.7, reaches a ceiling of 50.27% overall accuracy. Gemini-3-Pro, GPT-5.5, and Gemini-3.1-Pro follow closely, yet all remain below 50%. The remaining models form a substantially lower tier. For example, GPT-5 achieves 33.39% overall accuracy, whereas Gemini-2.5-Pro reaches only 17.24%. This stratification demonstrates that AppSim-Bench captures both granular deltas among frontier models and wider capability gaps among weaker baselines. GUI-specialized post-training does not close this gap. The strongest such agent, MobileAgent-v3.5, reaches 31.42%, and the other three remain below 23%. The ceiling on AppSim-Bench is therefore set by the task demands rather than by the absence of GUI-specific post-training. The group-level evaluations further demonstrate that agent performance is non-uniform across application ecosystems. Leading models reach approximately 59–60% accuracy on the Chinese applications, yet only about 39–41% on the English applications. Superior performance within one linguistic ecosystem does not guarantee comparable reliability in the other. The app-level results in Tables 9 and 10 further demonstrate substantial variation across different applications.

Figure 4 further examines task-level coverage across models. The distribution shows that 28.55%

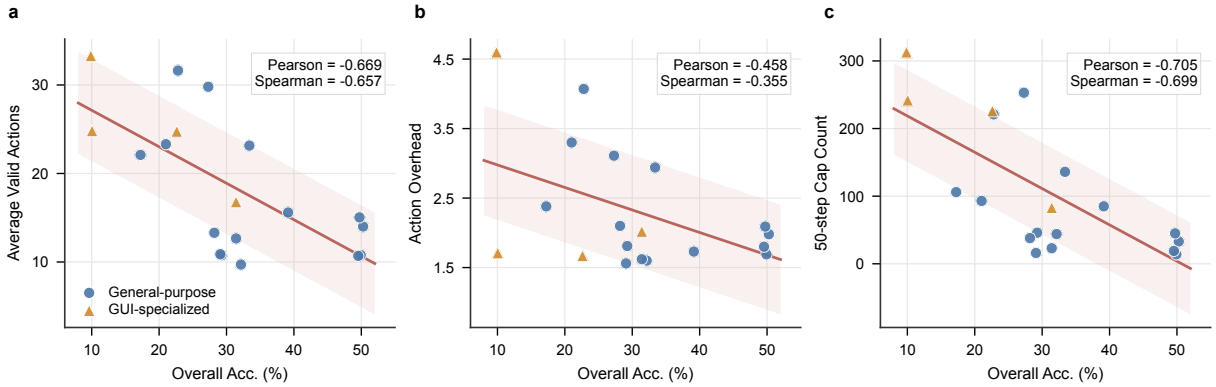


Figure 5: Accuracy-efficiency relationships across agents. Each point denotes one agent. Panels show overall accuracy vs. (a) average valid actions, (b) Action Overhead, and (c) budget-exhausted trajectories. Pearson and Spearman correlations are shown in each panel.

of tasks are not solved by any model, indicating a substantial set of shared failure cases. At the other end, only 0.36% of tasks are solved by all models included in this analysis. This pattern suggests that current GUI agents do not merely differ in overall accuracy. They also share systematic blind spots on a non-trivial portion of AppSim-Bench. Appendix G categorizes these shared failures into three capability gaps.

4.4 Task Length as a Long-horizon Bottleneck

The middle block of Table 2 groups tasks by the human-annotated number of reference steps required for completion. It reports model accuracy on three task-length buckets, $\#S_{\leq 5}$, $\#S_{\text{6-10}}$, and $\#S_{\geq 11}$. A clear pattern emerges: accuracy consistently decreases as the required number of reference steps increases. Every evaluated model performs worse on $\#S_{\text{6-10}}$ tasks than on $\#S_{\leq 5}$ tasks, and worse again on $\#S_{\geq 11}$ tasks. For example, GPT-5 drops from 43.62% to 31.43% and then to 14.07%. The top-performing model also varies across task lengths: GPT-5.5 leads on $\#S_{\leq 5}$ tasks, Claude-Opus-4.7 leads on $\#S_{\text{6-10}}$ tasks, and Gemini-3-Pro leads on $\#S_{\geq 11}$ tasks. However, the strongest long-task result is only 33.33%, indicating that no evaluated model reliably handles the longest workflows.

The magnitude of the drop further shows where degradation is most severe. Claude-Haiku-4.5 has the largest decrease from $\#S_{\leq 5}$ to $\#S_{\text{6-10}}$, falling by 18.33%. Claude-Opus-4.7 has the largest decrease from $\#S_{\text{6-10}}$ to $\#S_{\geq 11}$, falling by 26.14%. Overall, longer interaction consistently reduces model reliability. In longer workflows, agents must track completed operations, interpret the current screen against the original instruction, and prevent intermediate errors from accumulating. The consistent degradation indicates that current agents still struggle to maintain coherent progress over extended mobile interactions. The same monotonic decline holds for the GUI-specialized agents, and is even steeper: AgentCPM-GUI and V-Droid fall to 0.00% and 0.74% on $\#S_{\geq 11}$ tasks. The long-horizon bottleneck is thus independent of model class.

4.5 Task Type Analysis

Sections 4.3 and 4.4 show two sources of difficulty in AppSim-Bench: aggregate reliability gaps and long-horizon degradation. We next examine whether performance also varies with the type of reasoning required by the task. The right block of Table 2 focuses on numerical reasoning tasks and breaks them down by subtype. The leading models in the overall benchmark remain strong, but their relative advantages become more subtype-specific. Claude-Opus-4.7 leads on counting and threshold filtering, while GPT-5.5 leads on arithmetic calculation. Gemini-3-Pro and Gemini-3.1-Pro remain competitive on numeric comparison, but they are weaker on threshold filtering. Thus, task type exposes capability differences that are partly hidden by aggregate benchmark scores.

The subtype breakdown further shows that the difficulty is not limited to arithmetic alone. On arithmetic calculation, GPT-5.5 reaches 57.78%, and Claude-Opus-4.7 reaches 55.56%. Counting and numeric comparison are also relatively stable for the leading models, with best accuracies around 52%. Threshold filtering is the main exception. Claude-Opus-4.7 reaches 42.86%, GPT-5.5 reaches

33.33%, and most other models remain below 15%. This indicates that applying a numeric cutoff during UI search or selection is substantially harder than computing or comparing an isolated value.

Lower-tier models show a sharper collapse on the same subtype, with several systems reaching only 4.76% or 9.52%. Overall, numerical reasoning in AppSim-Bench is better understood as a composite capability. It requires value extraction, arithmetic manipulation, comparison, and constraint maintenance under interactive GUI conditions. A manual audit reported in Appendix F attributes these failures almost entirely to the reasoning stage: only 1 of 1,287 audited execution steps contained a perception or extraction error.

4.6 Efficiency Analysis

We next examine whether final task success is accompanied by efficient execution. Table 3 reports three complementary indicators: average valid actions, Action Overhead, and budget exhaustion. These metrics distinguish accurate agents that act efficiently from agents that rely on long or repetitive trajectories.

Figure 5 shows that accuracy is negatively associated with all three inefficiency indicators. Models with higher accuracy tend to use fewer actions on average, with Pearson and Spearman correlations of -0.669 and -0.657. The same pattern holds for Action Overhead, with correlations of -0.458 and -0.355. Budget exhaustion is also negatively correlated with accuracy, with correlations of -0.705 and -0.699. Thus, low accuracy is not merely a final-state failure. It is also associated with longer action sequences, less human-like successful trajectories, and more frequent budget exhaustion. The association is weakest for Action Overhead, because the GUI-specialized agents produce short trajectories on the tasks they do solve while solving far fewer of them.

At the model level, the strongest systems combine relatively high accuracy with limited budget exhaustion. Claude-Opus-4.7 reaches 50.27% accuracy with 33 budget-exhausted trajectories, and the lower-accuracy model Claude-Haiku-4.5 reaches 22.80% accuracy with 221. Budget-exhausted trajectories are almost never successful, with an accuracy of only 0.44%. This makes budget exhaustion a strong empirical marker of failed execution.

Repeated-action statistics provide a finer-grained view of budget-exhausted runs. Across all models, 1,690 such trajectories contain at least three consecutive repetitions of the same normalized action signature, and 1,347 contain at least five. This pattern is especially pronounced in lower-accuracy models. These statistics should not be interpreted as exact loop counts, because repeated actions may be valid when the screen state changes. Nevertheless, consecutive repetition within budget-exhausted trajectories indicates limited progress despite continued interaction.

5 Conclusion

We present AppSim-Bench, a reproducible benchmark designed to evaluate mobile GUI agents within realistic and controllable mobile environments. AppSim-Bench leverages controllable backend data and deterministic outcome verification to mitigate the environmental volatility inherent in live commercial applications. Extensive evaluations across 19 prominent GUI agents underscore that reliable autonomous mobile execution remains an unresolved challenge. The leading model completes only 50.27% of the tasks, while 28.55% remain entirely unsolved by any agent. Performance drops sharply within longer workflows and numerical reasoning tasks. Elevated action overhead and frequent interaction budget exhaustion also present as a major inefficiency. The evaluation highlights a critical need for improved long-horizon state tracking, robust numerical constraint handling, and sophisticated recovery mechanisms from stalled interactions.

Limitations

AppSim-Bench prioritizes reproducibility by evaluating GUI agents in controllable simulated environments rather than live commercial apps. This design removes major sources of evaluation noise, including personalization, advertisements, network variation, account-specific content, and continuously changing online services. These factors, however, remain important for understanding agent behavior in open-ended mobile ecosystems. Thus, AppSim-Bench provides a stable basis for comparative evaluation, while complementary real-world studies are still needed to assess deployment-time robustness. The current benchmark covers a broad but incomplete subset of mobile interaction scenarios. Although it includes 557 tasks across 17 Chinese and English apps, future work should further expand task coverage, especially toward cross-app workflows, real-time interaction, accessibility-

oriented tasks, and security-sensitive scenarios.

Realism is approximated per dimension rather than wholesale. We reproduce the visual appearance, in-page interaction, and navigation logic of the retained pages, but deliberately simplify four others: functional breadth, since we cover the task-relevant subset rather than an app's full feature set; dynamic and real-time content, such as live feeds and personalized recommendations; system-level factors, such as network latency and loading failures; and data scale, since manually constructed backend data need not reflect the long-tail histories of real users. These are trade-offs made in exchange for deterministic verification, and they are largely orthogonal to whether an agent can bring about a target state change.

Our fidelity and quality evidence is likewise bounded. Behavioral fidelity is measured indirectly, through the placeholder residual of Appendix E rather than a direct coverage ratio of implemented interactive elements. Construction quality rests on deterministic verification and multi-round human inspection (Appendix D) rather than standardized inter-annotator agreement or a residual bug rate. The error decomposition of Appendix F is a deep audit of a single app, so its near-zero extraction rate remains to be confirmed across ecosystems.

Ethics Statement

AppSim-Bench is intended for non-commercial research on mobile GUI agent evaluation. Its simulated apps are built for controlled benchmarking and are not connected to commercial services, user accounts, payment systems, or private user data. All backend data are manually defined or synthetic, and no real user records are used. The benchmark does not imply endorsement by, affiliation with, or substitution for any commercial product. App names are used only to describe the simulated interaction scenarios. The goal is reproducible research evaluation, not reproduction of proprietary services, redistribution of commercial app functionality, or automation of real-world services. AppSim-Bench is extensible, allowing researchers to customize tasks, app logic, and backend data. Such extensions should include privacy and safety safeguards. Custom backend data should not contain personal information or other sensitive records. Future releases and derivative uses should remain restricted to sandboxed research and evaluation settings.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62302186) and the Fundamental Research Funds for the Central Universities (Grant Nos. CCNU25ai017 and 2026CXGR089).

We sincerely thank Agentic Labs at Acrab AI for its support of this work. We also thank Luyao Chang, Xinzhu Du, Sirui Tang, and De Wu for their time and effort in developing the simulated apps.

References

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s2: A compositional generalist-specialist framework for computer use agents. *Preprint*, arXiv:2504.00906.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. *Preprint*, arXiv:2409.08264.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. 2025. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. *Preprint*, arXiv:2410.15164.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. 2026. Advancing mobile gui agents: A verifier-driven approach to practical deployment. *Preprint*, arXiv:2503.15937.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. 2024. Assistgui: Task-oriented desktop graphical user interface automation. *Preprint*, arXiv:2312.13108.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, Jingji Chen, Jingjia Huang, Kang Lei, Liping Yuan, Lishu Luo, Pengfei Liu, Qinghao Ye, Rui Qian, Shen Yan, and 178 others. 2025. Seed1.5-vl technical report. *Preprint*, arXiv:2505.07062.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao.

2025. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *Preprint*, arXiv:2509.26490.

Sayash Kapoor, Emily Cantrell, Kenny Peng, Thanh Hien Pham, Christopher A. Bail, Odd Erik Gundersen, Jake M. Hofman, Jessica Hullman, Michael A. Lones, Momin M. Malik, Priyanka Nanayakkara, Russell A. Poldrack, Inioluwa Deborah Raji, Michael Roberts, Matthew J. Salganik, Marta Serra-Garcia, Brandon M. Stewart, Gilles Vandewiele, and Arvind Narayanan. 2023. Reforms: Reporting standards for machine learning based science. *Preprint*, arXiv:2308.07832.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand. Association for Computational Linguistics.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pages 5295–5306, New York, NY, USA. Association for Computing Machinery.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2025. Benchmarking mobile device control agents across diverse configurations. *Preprint*, arXiv:2404.16660.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, Xiaoyu Liang, WenHao Wang, Tianze Wu, Zhengxi Lu, Siheng Chen, LiLinghao, Hao Wang, Guanjing Xiong, and 2 others. 2025a. Llm-powered gui agents in phone automation: Surveying progress and prospects. *Preprint*, arXiv:2504.19838.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. 2025b. Learnact: Few-shot mobile gui agent with a unified demonstration benchmark. *Preprint*, arXiv:2504.13805.

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

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, and 11 others. 2025. Gui agents: A survey. *Preprint*, arXiv:2412.13501.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *Preprint*, arXiv:2501.12326.

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. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *Preprint*, arXiv:2307.10088.

Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel J. Kochenderfer. 2024. Betterbench: Assessing ai benchmarks, uncovering issues, and establishing best practices. *Preprint*, arXiv:2411.12990.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. 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)*, pages 16022–16076, Bangkok, Thailand. Association for Computational Linguistics.

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

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025. Opencua: Open foundations for computer-use agents. *Preprint*, arXiv:2508.09123.

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

Qinzhuo Wu, Weikai Xu, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, and Shuo Shang. 2024. Mobilevlm: A vision-language model for better intra- and inter-ui understanding. *Preprint*, arXiv:2409.14818.

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. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Preprint*, arXiv:2404.07972.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the weakness of large language model agents within a complex android environment. *Preprint*, arXiv:2402.06596.

Pei Yang, Hai Ci, and Mike Zheng Shou. 2025. macosworld: A multilingual interactive benchmark for gui agents. *Preprint*, arXiv:2506.04135.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *Preprint*, arXiv:2508.15144.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *Preprint*, arXiv:2312.13771.

Danyang Zhang, Zhennan Shen, Rui Xie, Situo Zhang, Tianbao Xie, Zihan Zhao, Siyuan Chen, Lu Chen, Hongshen Xu, Ruisheng Cao, and Kai Yu. 2024a. Mobile-env: Building qualified evaluation benchmarks for llm-gui interaction. *Preprint*, arXiv:2305.08144.

Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024b. Llamatouch: A faithful and scalable testbed for mobile ui task automation. *Preprint*, arXiv:2404.16054.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, Jie Xie, Wei Zhou, Wang Xu, Yuanheng Zhang, Zhou Su, Zhongwu Zhai, Xiaoming Liu, Yudong Mei, Jianming Xu, and 6 others. 2025. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning. *Preprint*, arXiv:2506.01391.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. *Preprint*, arXiv:2401.01614.

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. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

A Simulated App Construction Details

Figure 6 summarizes the construction process for the simulated apps used in AppSim-Bench. The process has four main steps.

- **Schema Design**: Define task-aware backend schemas that contain the entities and attributes required for execution and evaluation (Appendix A.1).
- **UI Modeling**: Specify task-relevant UI flows and page hierarchies for each app (Appendix A.2).
- **Agent-assisted Developing**: Human developers use coding agents to generate candidate implementations, then inspect and correct the apps (Appendix A.3).
- **Instrumentation**: Add verification scripts, structured logs, and state records for deterministic evaluation (Appendix A.4).

A.1 Schema Design

For each simulated app, we define backend data schemas according to the benchmark tasks it must support. We identify task-relevant entities and attributes, and omit fields that are not needed for task execution or evaluation. For example, in modeling the "User" class for WeChat, we retain attributes such as *name*, *avatar*, and *status*, while excluding fields such as *last_login_time* and *device_info*. This selective design keeps the app state sufficient for benchmark evaluation without adding unrelated backend complexity.

A.2 UI Modeling

For each app, we specify a task-relevant page hierarchy and interaction paths based on the app's functional structure. For instance, when simulating WeChat, we organize the app into four main tabs, *Chat*, *Contact*, *Discover*, and *Me*, and define the subpages and navigation paths needed by the benchmark tasks. The page hierarchy constrains the simulated UI flows and helps ensure that the implemented interactions support the intended tasks.

A.3 Agent-assisted Developing

Coding agents are used as implementation assistance, not as an autonomous benchmark-construction method. In our implementation, developers provide task specifications, page hierarchies,

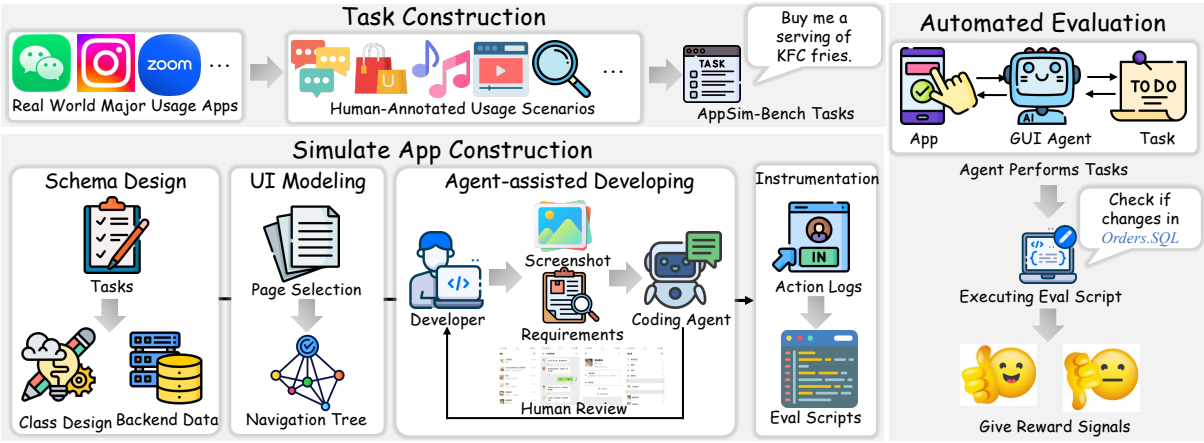


Figure 6: Supplementary overview of simulated app construction in AppSim-Bench. The process specifies task requirements, backend data schemas, page structures, and verification signals for each simulated app. Coding agents are used as implementation assistance under human specification and inspection.

and representative screenshots to guide candidate app implementations. The generated layouts, widget bindings, navigation logic, and frontend behaviors are then inspected and corrected by human developers. Before evaluation, each app is executed and checked to ensure that the task-relevant UI flows and backend states match the benchmark specification.

A.4 Instrumentation

Each simulated app records task-relevant state changes through structured logs or backend fields. For example, WeChat maintains chat logs for message-sending tasks, while Ele.me records click logs and order states for ordering tasks. These records allow the evaluator to verify task outcomes from a controlled initial state. Because the backend data are predefined, the evaluation is not affected by recommendation algorithms, advertisements, account-specific content, or changing online services.

B Evaluation Schemas and Examples

For state tasks, each task is represented as (S_0, G, S_t), where S_0 is the initial state, G is the goal condition, and S_t is the final state after execution. During execution, the app records structured logs or backend fields. The evaluator compares S_t with G and marks success only when target fields show the expected changes, such as updated chat history, order status, or like count.

For answer tasks, each task defines an answer schema with required fields, data types, and format constraints. If an agent returns free-form text, a shared reference executor extracts a structured answer under this schema. The evaluator then performs schema validation, canonicalization, and deterministic matching against the gold answer. This scoring stage contains no LLM calls.

For hybrid tasks, both the state-transition check and the answer-schema check must pass. All reported baselines use the same reference executor. Users may also bypass it by making agents directly output schema-conformant JSON.

To verify that the answer extraction step does not introduce evaluation noise into the reported results, we manually checked the structured-answer extraction outputs used to score the answer and hybrid evaluation cases reported in this paper. For each case, we compared the agent's final response, the structured answer extracted by the shared reference executor, and the gold answer under the predefined schema. We found no extraction errors in the audited cases (0/3933).

C Comparison between Real Apps and Simulators

Figure 7 shows a page-transition comparison between a real app and its simulator. The simulator preserves the matched task-relevant pages and transitions, while excluding unmatched real-app states that are not required for the benchmark tasks.

D Benchmark Construction Quality Control

Each simulated app was implemented by a single lead developer and then independently inspected by several quality-control reviewers, whose reported issues were fixed before the app entered

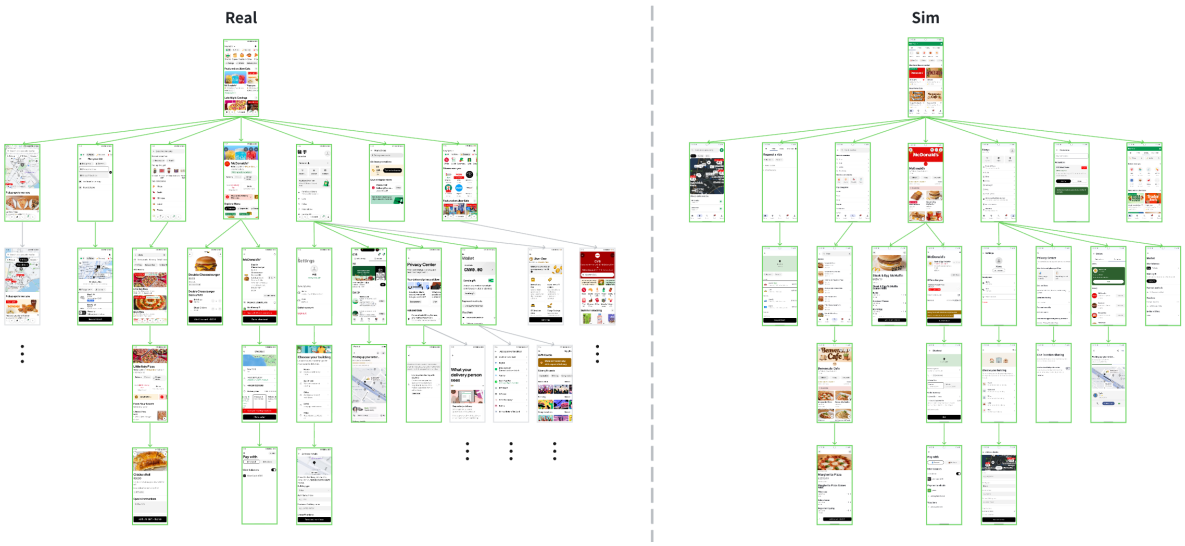


Figure 7: Page-transition comparison between a real app and its simulator. Green boxes indicate matched task-relevant pages in both apps, while black boxes mark real-app pages not retained in the simulator.

Table 4: Human review–correction rounds during the construction of the Uber Eats simulator. "Rounds" counts the iterations of human inspection and correction before a page was accepted.

Page	Rounds	Representative correction
Home	3	Realigned top icons, added real images
Cart	3	Split action buttons, linked store page
Profile	2	Unified icon style
Trips	1	Added ride-request form
Grocery	2	Added real images
Location	2	Made stores clickable, live filter
Search	2	Live input, clear, and results
Store Home	2	Real store data, dish-card entry
Product Detail	3	Add-to-cart return path
View Cart	3	Corrected quantities and images
Total	23	≈2.3 rounds per page, 2h22m

the benchmark. Reviewers executed both "correct completion" and "deliberate failure" trajectories and checked the True/False verdict of the evaluator case by case; no misjudged case was found.

To make the iterative nature of this process concrete, we recorded the full construction log of one app. We selected Uber Eats because it is the English app with the widest functional span in AppSim-Bench, covering food ordering, grocery, ride requests, map-based store selection, and cart management, so its pages exercise most of the interaction patterns encountered elsewhere. Table 4 reports, per page, the number of human review–correction rounds and the wall-clock construction time. No page was accepted on the first attempt: each required about 2.3 rounds on average, and the corrections span both visual presentation and interaction correctness.

E Behavioral Fidelity of Task-aware Pruning

AppSim-Bench reproduces the interactive elements and navigation logic of the real app on every retained page, and implements multiple valid paths per task. Fully task-irrelevant pages are not implemented; they render an explicit "page under development" placeholder. Such a placeholder is the one behavioral element that differs from a real app, since it signals early that the agent has left the task scope, whereas a real off-track page might let it drift further.

We therefore quantify the residual effect of pruning as the fraction of executions that entered a placeholder and still succeeded. This is a deliberately pessimistic upper bound: it credits every such success entirely to the placeholder, even though the agent may have recovered on its own.

Because this measure requires manually replaying trajectories to determine whether a placeholder was reached, we audited a subset rather than all 19 agents. We selected six apps spanning both ecosystems and four categories (video, food delivery, social, e-commerce, music), and three agents chosen to bracket the accuracy range observed in Table 2: Claude-Opus-4.7 (highest), GPT-5 (intermediate), and Gemini-2.5-Pro (lowest among general-purpose agents). If the residual were driven by agent capability, this spread would expose it.

Table 5 shows that the residual is at most 7.34% for any single agent and 4.28% over all 654 executions, concentrated in a few app–agent combi-

Table 5: Tasks that entered a "page under development" placeholder and still succeeded, reported as a pessimistic upper bound on the effect of task-aware pruning. Percentages are over each app's tasks.

App	Tasks	Claude-Opus	GPT-5	Gemini-2.5
Bilibili	30	2 (6.7%)	2 (6.7%)	1 (3.3%)
Uber Eats	34	1 (2.9%)	7 (20.6%)	1 (2.9%)
Instagram	40	1 (2.5%)	2 (5.0%)	1 (2.5%)
NetEase Music	39	2 (5.1%)	1 (2.6%)	0 (0.0%)
JD	40	2 (5.0%)	3 (7.5%)	0 (0.0%)
YouTube	35	0 (0.0%)	1 (2.9%)	1 (2.9%)
Total	218	8 (3.67%)	16 (7.34%)	4 (1.83%)

Table 6: Manual audit of perception/extraction errors on the numerical-reasoning tasks of JD. Every execution step of each failed task was inspected.

Agent	Failed	Steps	Err.	Rate
Claude-Opus-4.7	17/27	284	0	0%
GPT-5	20/27	443	0	0%
Gemini-2.5-Pro	25/27	560	1	0.18%
Total	—	1,287	1	0.08%

nations and near zero elsewhere. Replacing the placeholders with fully implemented pages would therefore shift the measured accuracies by at most a few points, and the pruned regions do not constitute an exploitable shortcut.

F Error Decomposition on Numerical-reasoning Tasks

A numerical-reasoning task is executed in two stages: the agent first locates and reads the values that enter the computation from the interface (perception and extraction), then applies counting, arithmetic, comparison, or threshold filtering to them (arithmetic and logical reasoning). Failures can arise in either stage, and the two carry different implications for what the benchmark measures.

Separating them requires comparing, value by value, what the agent read against what the interface displayed. We did not automate this with a judge VLM, because a model prone to misreading cannot reliably decide whether another model misread. We therefore inspected screenshots manually, which bounds how much we could audit. We selected the numerical-reasoning tasks of JD as the audit target because it is the most numerically dense app in the benchmark (prices, discounts, quantities, ratings) and therefore places the heaviest load on perception; if extraction errors were common anywhere, they would appear here. We audited every execution step of the failed tasks for the same three agents used above.

Across 1,287 execution steps, only one step exhibited a perception or extraction error. Almost all observed failures therefore arise in the reasoning stage rather than the perception stage. Notably, the agent with the most failed tasks (Gemini-2.5-Pro, 25/27) also shows a near-zero extraction rate, which indicates that the discriminating factor is reasoning rather than reading. This holds for the state-of-the-art agents audited here; for weaker agents, perception errors may become more frequent.

G Failure Modes of Unsolved Tasks

We examined the 159 tasks that no evaluated agent solved. Their difficulty concentrates in three capability gaps. The first is **long-horizon multi-step workflows**, where the agent must track completed actions, interpret the current screen against the original instruction, and suppress the accumulation of intermediate errors. The second is **incomplete information gathering**, where the agent stops exploring too early, for example by inspecting only the visible portion of a list, and then decides on partial information. The third is **numerical reasoning chains on the interface**, where the agent must enumerate the relevant values before comparing or aggregating them, so that a break at any link causes failure.

The hardest tasks typically stack several of these gaps. In AMAZON#21, the agent must find the three most expensive items in a long shopping cart, sum their prices, and return the total; the deterministic verifier expects 3118.98. No agent produced the correct value, and the outputs clustered at lower totals such as 1698.98, 1848.98, and 2868.98. Every reported total falls below the correct one, which is the signature of deciding before the list was fully enumerated. This case combines incomplete information gathering with a broken numerical-reasoning chain.

H App Info in AppSim-Bench

H.1 Overview of the Apps in AppSim-Bench

We present the list of applications (Apps) included in AppSim-Bench in Table 7. These Apps span a diverse range of categories, including video sharing, travel services, food delivery, navigation, e-commerce, music streaming, social media, communication, ride hailing, and video conferencing, ensuring a comprehensive evaluation environment.

Table 7: Overview of Apps in AppSim-Bench. This table lists the Chinese and English names, a brief description, and the number of tasks for each App.

ID	App Name (ZH)	App Name (EN)	Description	#Tasks
1	哔哩哔哩	Bilibili	A video-sharing app with ACG-oriented community content.	30
2	\	Booking.com	A travel booking app for hotels, flights, cars, taxis, and attractions.	25
3	携程旅行	Ctrip	A travel-service app for hotels, flights, trains, trips, and messages.	35
4	饿了么	Ele.me	A local-life and food-delivery app for restaurants, orders, coupons, and addresses.	40
5	高德地图	Amap	A map and navigation app with nearby places, routes, favorites, and travel tools.	28
6	网易云音乐	NetEase Cloud Music	A music-streaming app with songs, playlists, charts, lyrics, and user profiles.	39
7	京东	JD	An e-commerce app for product search, carts, orders, addresses, and store services.	40
8	小红书	RedNote	A lifestyle social-media app for notes, creators, comments, messages, and profiles.	29
9	腾讯会议	Tencent Meeting	A video-conferencing app for meetings, contacts, messages, and personal rooms.	26
10	\	Uber Eats	A food-delivery and local-services app with restaurants, carts, orders, and rides.	34
11	微信	WeChat	A messaging and social app with chats, contacts, groups, and Moments.	16
12	\	YouTube	A video platform for search, playback, comments, subscriptions, Shorts, and settings.	35
13	\	Amazon	An e-commerce app for product discovery, shopping lists, carts, orders, and support.	40
14	\	WhatsApp	A messaging app with chats, calls, statuses, channels, communities, and contacts.	40
15	\	Zoom	A video-meeting app for instant meetings, scheduled meetings, chat, contacts, and settings.	20
16	\	Spotify	A music and podcast app with playback, search, playlists, lyrics, and subscriptions.	40
17	\	Instagram	A social-media app for posts, reels, messages, profiles, search, and publishing.	40

H.2 Screenshots of the Apps in AppSim-Bench

In this section, we present screenshots for all 17 apps included in AppSim-Bench. As shown in Figures 8–10, these screenshots provide an intuitive overview of the interface designs and functional layouts covered by the benchmark.

H.3 Statistics of AppSim-Bench

Table 8 reports the complete app-level task statistics of AppSim-Bench, including Chinese/English app membership, task-length buckets, and numerical-reasoning subtype counts.

I More Experimental Results

Tables 9 and 10 report the complete app-level accuracy results for Chinese and English apps, respectively. The compact main-text summary in Table 2 is derived from these tables.

J Agent Interface and Prompt Details

This section reports the agent interfaces used in the experiments. The goal is to make the evaluation protocol reproducible without expanding the main text. All agents are evaluated under the same device setup, app reset protocol, and 50-step budget described in Section 4.1.

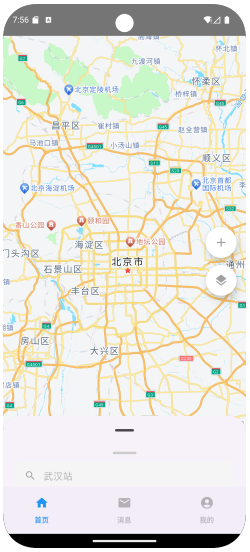
Qwen and Doubao are evaluated with their official mobile/GUI agent prompts and tool schemas. For Claude, GPT, and Gemini, we adapt the M3A agent from AndroidWorld (Rawles et al., 2025). The adapted interface keeps M3A's mobile observation design, action schema, Set-of-Mark grounding, and action-history mechanism. We use a single-model version rather than the original multi-agent M3A setting, so that all evaluated model families are compared under a single-model agent setup.

For all evaluated agents, we set the temperature to 0. The action history provided at each step is a concise textual record of previous operations within the same task, including prior actions and their observed outcomes when available. It does not include past screenshots; the visual input is supplied by the current screen observation. All prompt templates are provided in Appendix J.1.

(a) Bilibili

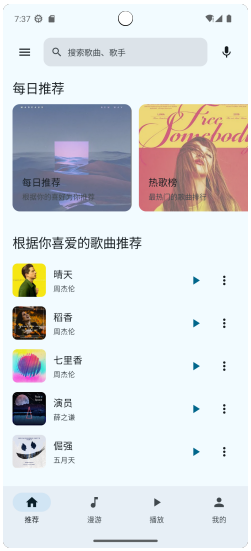
(b) Ctrip

(c) Ele.me



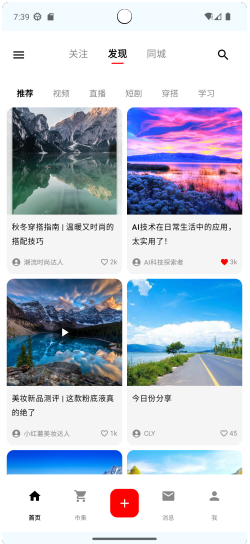
(d) Amap

(e) JD



(f) NetEase Cloud Music

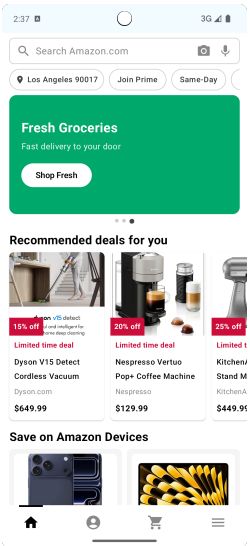
Figure 8: Screenshots of the Apps in AppSim-Bench (1/3).



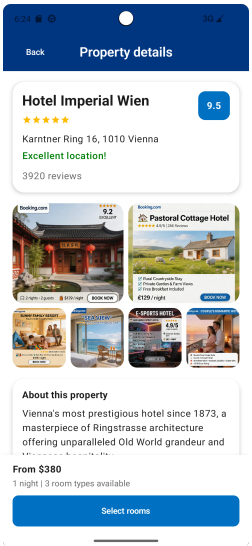
(a) RedNote

(b) Tencent Meeting

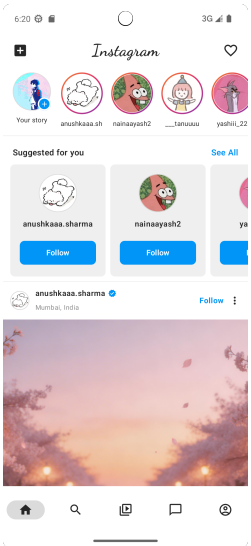
(c) WeChat



(d) Amazon

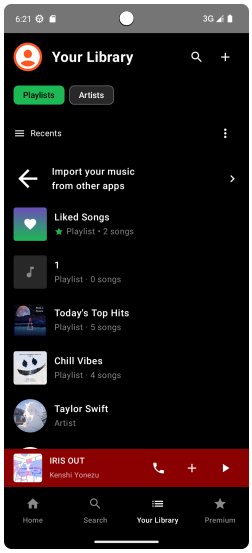


(e) Booking.com

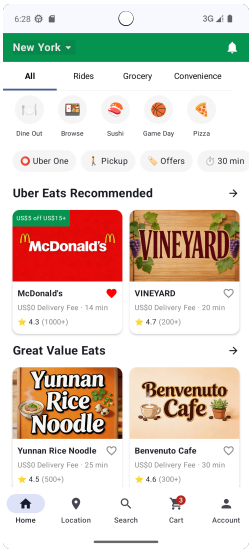


(f) Instagram

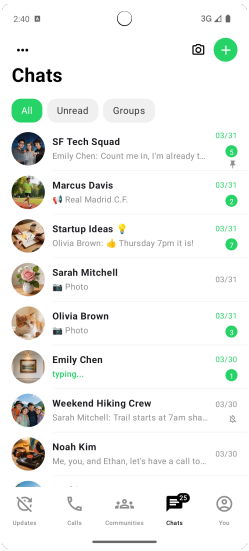
Figure 9: Screenshots of the Apps in AppSim-Bench (2/3).



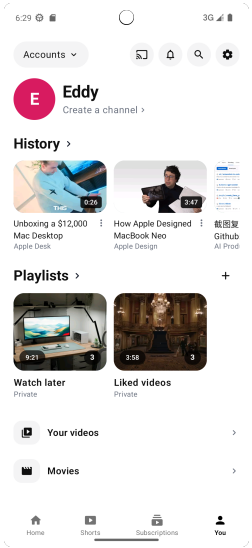
(a) Spotify



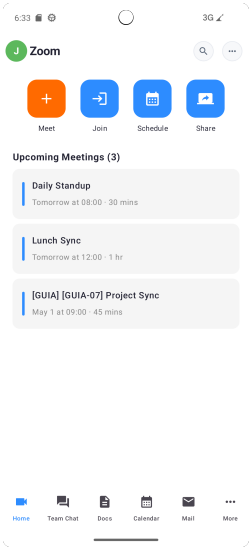
(b) Uber Eats



(c) WhatsApp



(d) YouTube



(e) Zoom

Figure 10: Screenshots of the Apps in AppSim-Bench (3/3).

Table 8: Task statistics of AppSim-Bench by app. ZH and EN indicate whether an app belongs to the Chinese or English app group. The task-length buckets are based on the human-annotated reference steps $\#S$ required to complete a task. "Cnt.", "Arith.", "Comp.", and "Thres." denote counting, arithmetic calculation, numeric comparison, and threshold filtering, respectively. Numerical-reasoning subtypes are not mutually exclusive.

App	Overall Benchmark			Task Length			Numerical Reasoning Tasks				
	Total	ZH	EN	$\#S_{\leq 5}$	$\#S_{6\text{-}10}$	$\#S_{\geq 11}$	Total	Cnt.	Arith.	Comp.	Thres.
Amazon	40	×	✓	10	15	15	21	5	4	14	2
Bilibili	30	✓	×	20	5	5	17	4	8	6	0
Booking	25	×	✓	5	8	12	9	0	1	6	2
Ctrip	35	✓	×	15	8	12	8	0	3	6	4
Ele.me	40	✓	×	25	9	6	21	8	6	9	2
Gaode	28	✓	×	14	9	5	11	1	1	8	2
Instagram	40	×	✓	30	5	5	2	2	0	0	0
JD	40	✓	×	10	20	10	27	7	6	12	6
NetEase Cloud Music	39	✓	×	26	8	5	3	2	0	1	0
RedNote	29	✓	×	17	7	5	6	4	1	1	0
Spotify	40	×	✓	30	5	5	3	3	0	0	1
Tencent Meeting	26	✓	×	15	5	6	10	8	1	1	0
Uber Eats	34	×	✓	19	6	9	15	2	9	5	0
WeChat	16	✓	×	6	5	5	5	5	0	0	0
WhatsApp	40	×	✓	10	15	15	7	3	0	3	1
YouTube	35	×	✓	25	5	5	13	8	5	0	0
Zoom	20	×	✓	5	5	10	5	3	0	3	1
ZH Apps	283	✓	×	148	76	59	108	39	26	44	14
EN Apps	274	×	✓	134	64	76	75	26	19	31	7
All Apps	557	✓	✓	282	140	135	183	65	45	75	21

Table 9: Accuracy on Chinese apps. The first numeric column reports task-weighted accuracy over the Chinese app group. NetEase Music denotes the NetEase Cloud Music simulation. **Bold** and underlining indicate the best and second-best values in each column.

Agent Model	ZH Apps	Bilibili	Ctrip	Ele.me	Gaode	JD	RedNote	Tencent Meeting	WeChat	NetEase Music
General-purpose Agents										
Gemini-3-Pro	**60.07%**	80.00%	77.14%	55.00%	67.86%	55.00%	24.14%	**57.69%**	**81.25%**	53.85%
Claude-Opus-4.7	**60.07%**	63.33%	65.71%	**72.50%**	**78.57%**	52.50%	27.59%	**57.69%**	68.75%	56.41%
Gemini-3.1-Pro	59.36%	73.33%	**80.00%**	55.00%	57.14%	**60.00%**	24.14%	**57.69%**	68.75%	**58.97%**
GPT-5.5	58.66%	**90.00%**	71.43%	65.00%	53.57%	**60.00%**	27.59%	46.15%	43.75%	56.41%
Doubao-Seed-1.6	50.88%	56.67%	74.29%	55.00%	46.43%	32.50%	41.38%	65.38%	18.75%	53.85%
Qwen-3.6-Plus	39.58%	20.00%	62.86%	52.50%	64.29%	20.00%	**37.93%**	38.46%	18.75%	33.33%
Qwen-3.6-Flash	38.87%	26.67%	60.00%	45.00%	53.57%	27.50%	31.03%	53.85%	6.25%	33.33%
GPT-5	37.81%	46.67%	17.14%	37.50%	50.00%	32.50%	20.69%	42.31%	62.50%	46.15%
Doubao-Seed-2.0	36.40%	43.33%	48.57%	45.00%	42.86%	25.00%	31.03%	42.31%	25.00%	23.08%
GPT-5.4	34.63%	50.00%	62.86%	37.50%	25.00%	25.00%	17.24%	19.23%	18.75%	41.03%
Doubao-Seed-1.8	34.63%	26.67%	57.14%	35.00%	50.00%	27.50%	20.69%	46.15%	6.25%	30.77%
Claude-Sonnet-4.6	31.10%	50.00%	22.86%	45.00%	21.43%	30.00%	24.14%	30.77%	12.50%	30.77%
Claude-Sonnet-4.5	24.03%	26.67%	22.86%	25.00%	14.29%	17.50%	10.34%	26.92%	31.25%	41.03%
Claude-Haiku-4.5	22.61%	30.00%	45.71%	25.00%	10.71%	17.50%	13.79%	7.69%	0.00%	33.33%
Gemini-2.5-Pro	21.20%	36.67%	17.14%	37.50%	3.57%	10.00%	13.79%	23.08%	12.50%	28.21%
GUI-specialized Agents										
MobileAgent-v3.5	35.34%	30.00%	68.57%	32.50%	50.00%	27.50%	20.69%	30.77%	12.50%	33.33%
UI-TARS-1.5-7B	31.10%	16.67%	51.43%	35.00%	53.57%	15.00%	27.59%	26.92%	12.50%	33.33%
V-Droid	13.43%	20.00%	25.71%	15.00%	0.00%	17.50%	3.45%	15.38%	6.25%	10.26%
AgentCPM-GUI	9.19%	0.00%	34.29%	5.00%	7.14%	5.00%	6.90%	7.69%	0.00%	10.26%

Table 10: Accuracy on English apps. The first numeric column reports task-weighted accuracy over the English app group. **Bold** and underlining indicate the best and second-best values in each column.

Agent Model	EN Apps	Amazon	Booking	Instagram	Spotify	UberEats	WhatsApp	YouTube	Zoom
General-purpose Agents									
GPT-5.5	**40.51%**	**55.00%**	48.00%	**40.00%**	**27.50%**	**47.06%**	**12.50%**	57.14%	45.00%
Claude-Opus-4.7	40.15%	47.50%	44.00%	37.50%	22.50%	**47.06%**	**12.50%**	**65.71%**	60.00%
Gemini-3.1-Pro	39.42%	52.50%	**56.00%**	27.50%	22.50%	41.18%	10.00%	57.14%	**75.00%**
Gemini-3-Pro	39.42%	52.50%	**56.00%**	27.50%	25.00%	41.18%	**12.50%**	57.14%	65.00%
Doubao-Seed-1.8	29.56%	27.50%	48.00%	27.50%	20.00%	**47.06%**	5.00%	42.86%	30.00%
GPT-5	28.83%	45.00%	28.00%	35.00%	17.50%	29.41%	5.00%	37.14%	40.00%
Doubao-Seed-1.6	27.01%	20.00%	32.00%	30.00%	17.50%	29.41%	5.00%	54.29%	40.00%
GPT-5.4	23.36%	25.00%	36.00%	32.50%	15.00%	17.65%	10.00%	31.43%	25.00%
Claude-Sonnet-4.6	23.36%	17.50%	36.00%	30.00%	15.00%	23.53%	10.00%	45.71%	10.00%
Qwen-3.6-Plus	22.99%	27.50%	40.00%	20.00%	12.50%	29.41%	7.50%	34.29%	20.00%
Claude-Haiku-4.5	22.99%	32.50%	12.00%	30.00%	15.00%	20.59%	7.50%	45.71%	15.00%
Doubao-Seed-2.0	21.90%	10.00%	36.00%	20.00%	22.50%	29.41%	7.50%	37.14%	20.00%
Claude-Sonnet-4.5	17.88%	12.50%	24.00%	25.00%	20.00%	14.71%	7.50%	31.43%	5.00%
Qwen-3.6-Flash	17.15%	15.00%	12.00%	15.00%	15.00%	32.35%	5.00%	25.71%	20.00%
Gemini-2.5-Pro	13.14%	10.00%	12.00%	20.00%	17.50%	11.76%	5.00%	22.86%	0.00%
GUI-specialized Agents									
MobileAgent-v3.5	27.37%	27.50%	52.00%	20.00%	17.50%	32.35%	7.50%	48.57%	25.00%
UI-TARS-1.5-7B	13.87%	7.50%	12.00%	12.50%	15.00%	17.65%	2.50%	28.57%	20.00%
AgentCPM-GUI	10.95%	5.00%	4.00%	22.50%	12.50%	11.76%	2.50%	20.00%	5.00%
V-Droid	6.20%	7.50%	8.00%	2.50%	7.50%	8.82%	5.00%	8.57%	0.00%

J.1 Prompt Templates

J.1.1 M3A Single-model GUI Prompt

M3A Single-model GUI Prompt

```
You are an agent who can operate an Android phone on
      behalf of a user. Based on user's goal/request,
      you may
- Answer back if the request/goal is a question (or a
      chat message), like user asks "What is my
      schedule for today?".
- Complete some tasks described in the requests/goals
      by performing actions (step by step) on the
      phone.

When given a user request, you will try to complete it
       step by step. At each step, you will be given
      the current screenshot (including the original
      screenshot and the same screenshot with bounding
       boxes and numeric indexes added to some UI
      elements) and a history of what you have done (
      in text). Based on these pieces of information
      and the goal, you must choose to perform one of
      the action in the following list (action
      description followed by the JSON format) by
      outputing the action in the correct JSON format.
- If you think the task has been completed, finish the
       task by using the status action with complete
      as goal_status: {"action_type": "status", "
      goal_status": "complete"}
- If you think the task is not feasible, finish by
      using the status action with infeasible as
      goal_status: {"action_type": "status", "
      goal_status": "infeasible"}
- Answer user's question: {"action_type": "answer", "
      text": "<answer_text>"}
- Click/tap on an element on the screen: {"action_type
      ": "click", "index": <target_index>}
- Long press on an element on the screen: {"
      action_type": "long_press", "index": <
      target_index>}
- Type text into a text field: {"action_type": "
      input_text", "text": <text_input>, "index": <
      target_index>}
- Press the Enter key: {"action_type": "keyboard_enter
      "}
- Navigate to the home screen: {"action_type": "
      navigate_home"}
- Navigate back: {"action_type": "navigate_back"}
- Scroll the screen or a scrollable UI element: {"
      action_type": "scroll", "direction": <up, down,
      left, right>, "index": <optional_target_index>}
- Open an app: {"action_type": "open_app", "app_name":
       <name>}
- Wait for the screen to update: {"action_type": "wait
      "}

Guidelines:
- Usually there will be multiple ways to complete a
      task, pick the easiest one. If something does
      not work as expected, sometimes a simple retry
      can solve the problem, but if it does not,
      switch to other solutions.
- For requests that are questions, remember to use the
       answer action to reply to user explicitly
      before finish.
- If the desired state is already achieved, you can
      just complete the task.
- Use open_app whenever you want to open an app.
- Use input_text whenever you want to type something.
- For click, long_press and input_text, the index
      parameter must be visible in the screenshot and
      in the UI element list.
- Consider exploring the screen by using scroll
      actions.

The current user goal/request is: {goal}
Here is a history of what you have done so far:
{history}
The current screenshot and the same screenshot with
      bounding boxes and labels added are also given
      to you.
Here is a list of detailed information for some of the
       UI elements:
{ui_elements}
{additional_guidelines}
Now output an action from the above list in the
      correct JSON format, following the reason why
      you do that. Your answer should look like:
Reason: ...
Action: {"action_type":...}

Your Answer:
```

J.1.2 Qwen-style GUI Prompt

Qwen-style GUI Prompt

```
# Tools
You may call one or more functions to assist with the
      user query.

You are provided with function signatures within <
      tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "mobile_use
      ", "description": "Use a touchscreen to interact
       with a mobile device, and take screenshots.
      This is an interface to a mobile device with
      touchscreen. You can perform actions like
      clicking, typing, swiping, etc. 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. The screen's resolution is 999x999
      .", "parameters": {"properties": {"action": {"
      description": "The action to perform. Available
      actions: click, long_press, swipe, type, answer,
       system_button, wait, terminate.", "enum": ["
      click", "long_press", "swipe", "type", "answer",
       "system_button", "wait", "terminate"], "type":
      "string"}, "coordinate": {"description": "
      Required by click, long_press, and swipe.", "
      type": "array"}, "coordinate2": {"description":
      "Required by swipe.", "type": "array"}, "text":
      {"description": "Required by type and answer.",
      "type": "string"}, "time": {"description": "
      Required by long_press and wait.", "type": "
      number"}, "button": {"description": "Required by
       system_button.", "enum": ["Back", "Home", "Menu
      ", "Enter"], "type": "string"}, "status": {"
      description": "Required by terminate.", "enum":
      ["success", "failure"], "type": "string"}}, "
      required": ["action"], "type": "object"}}}
</tools>

For each function call, return a json object with
      function name and arguments within <tool_call></
      tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-
      object>}
</tool_call>

# Response format
Response format for every step:
1) Thought: one concise sentence explaining the next
      move (no multi-step reasoning).
2) Action: a short imperative describing what to do in
       the UI.
3) A single <tool_call>...</tool_call> block
      containing only the JSON.

Rules:
- Output exactly in the order: Thought, Action, <
      tool_call>.
- Be brief: one sentence for Thought, one for Action.
- Do not output anything else outside those three
      parts.
- If finishing, use action=terminate in the tool call.

The user query: {instruction}.
Task progress (You have done the following operation
      on the current device): {stage2_history}.
```

J.1.3 Doubao-style GUI Prompt

Doubao-style GUI Prompt

```
You are a GUI agent. You are given a task and your
      action history, with screenshots. You need to
      perform the next action to complete the task.

## Output Format
Thought: ...
Action: ...

## Action Space
click(point='<point>x1 y1</point>')
long_press(point='<point>x1 y1</point>')
type(content='') # If you want to submit your input,
      use "\n" at the end of content.
scroll(point='<point>x1 y1</point>', direction='down
      or up or right or left')
open_app(app_name='')
drag(start_point='<point>x1 y1</point>', end_point='<
      point>x2 y2</point>')
press_home()
press_back()
finished(content='xxx')

## Note
- Use {language} in Thought part.
- Write a small plan and finally summarize your next
      action (with its target element) in one sentence
       in Thought part.
- The coordinate should be integer between 0~1000. Do
      not output float numbers.

## User Instruction
{instruction}
```

K Example Agent Trajectory

Table 2 and the analyses above report aggregate accuracy; here we walk through one solved task step by step to make the interaction pattern concrete. The task targets the Amazon simulator: *Find the cheapest electronics item in the cart and report its name and price.* The agent must open the app, navigate to the cart, scroll to the bottom to see every item before comparing prices, and report the cheapest electronics item by name and price. The trajectory below uses five action types: `open_app`, `click`, `scroll`, `answer`, and `status`.

Agent Trajectory: AMAZON, Find the Cheapest Electronics Item in the Cart

```
APP: Amazon
Goal: Find the cheapest electronics item in the cart
      and report its name and price.

Step 1  Reasoning : Open Amazon app first to enter the
       cart.
        Action    : {"action_type": "open_app", "
      app_name": "Amazon"}
        Screen    : App home page.

Step 2  Reasoning : Open the cart; the list is long,
      so only its top is visible.
        Action    : {"action_type": "click", "index":
      <cart_tab>}
        Screen    : Cart page, top of the list.
      Visible items:
                    iPhone 17 Pro Max $1369.99 ;
      Marshall Acton III $199.99 ;
                    MacBook Air M4 $1099.00.

Step 3  Reasoning : More items may be below; scroll
      down to reveal them.
        Action    : {"action_type": "scroll", "
      direction": "down"}
        Screen    : Two more items come into view:
                    Sony WH-1000XM5 $399.99 ; Nintendo
       Switch OLED $349.99.

Step 4  Reasoning : Keep scrolling to continue reading
       the cart.
        Action    : {"action_type": "scroll", "
      direction": "down"}
        Screen    : Two more items come into view:
                    Samsung Galaxy Watch7 $299.99 ;
      Dyson V15 Detect $649.99.

Step 5  Reasoning : The end is not yet known; keep
      scrolling to see more.
        Action    : {"action_type": "scroll", "
      direction": "down"}
        Screen    : Two more items come into view:
                    Lodge Cast Iron Cookware Set $129
      .99 ; KitchenAid Mixer $449.99.

Step 6  Reasoning : The end-of-list marker is now
      visible, so the bottom of the
                    cart is reached; stop scrolling.
        Action    : {"action_type": "scroll", "
      direction": "down"}
        Screen    : Last items, followed by the end of
       the list:
                    Nike Pegasus 41 $139.99 ; Vitamix
      Explorian $349.95 ;
                    Coca-Cola Zero (24-pack) $16.99.
      --- Bottom of cart ---

Step 7  Reasoning : All 12 items have been seen. Among
       the electronics
                    (iPhone $1369.99, MacBook Air
      $1099.00, Sony $399.99,
                    Nintendo Switch $349.99, Galaxy
      Watch7 $299.99, Marshall
                    $199.99), the Marshall Acton III
      is the cheapest. Report its name and price.
        Action    : {"action_type": "answer", "text":
      "Marshall Acton III Bluetooth Home Speaker, $199
      .99"}

Step 8  Reasoning : The answer has been returned; the
      task is complete.
        Action    : {"action_type": "status", "
      goal_status": "complete"}

Verify Result: agent answer contains gold keys "
      Marshall Acton III" and "199.99"  ->  True (task
       solved)
```