SIMUWOB: SIMULATING REAL-WORLD MOBILE APPS FOR FAST AND FAITHFUL GUI AGENT BENCHMARKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mobile GUI agents powered by large language models have progressed rapidly, creating a growing need for realistic, challenging, and efficient evaluation. Existing interactive benchmarks often rely on open-source apps, file-centric tasks, or manually engineered reward functions, limiting their coverage of real-world app semantics and complex, long-horizon workflows. To address these limitations, we introduce SimuWoB, a fully synthetic benchmark comprising 120 challenging tasks across 63 simulated mobile applications. We develop a two-stage environment synthesis framework that constructs interactive mobile-style applications and injects task-specific logic together with executable state-based validators. Each environment is deployed as a backend-free webpage accessible via URL, enabling lightweight, reproducible, and parallelizable evaluation. Across five recent mobile GUI agents, the average success rate is only 27.92%, dropping to 17.82% on long-horizon tasks, indicating substantial room for improvement on complex workflows. Evaluation result comparison with real-world sample tasks demonstrate that agent assessments based on our synthetic environment generalize well. We further provide diagnostic insights across key capability dimensions and discuss implications for future mobile GUI agent development.

1 INTRODUCTION

Mobile GUI agents powered by large language models (LLMs) and vision-language models (VLMs) have advanced rapidly in recent years (Qin et al., 2025; Xu et al., 2025; Liu et al., 2024; Ye et al., 2025; Zhou et al., 2025). Given a natural-language instruction and an interactive interface, an agent must interpret visual observations, track task progress, execute actions, and adapt to environment feedback until completion. A growing body of benchmarks has been developed to measure these capabilities across mobile and general computer-use settings (Rawles et al., 2025; Kong et al., 2025; Xu et al., 2024; Wang et al., 2024; Xie et al., 2024). However, evaluating increasingly capable mobile agents remains challenging along three dimensions: **(1) Limited coverage of realistic workflows.** Reliable interactive evaluation typically requires controllable application state and robust task validation. Consequently, many rely on open-source applications with accessible backends (Rawles et al., 2025; Kong et al., 2025), constrained task domains such as file operations (Xue et al., 2026), or manually engineered execution checks (Xu et al., 2024; Xie et al., 2024). These design choices improve reproducibility, but make it difficult to scale evaluation toward the diversity of applications and workflows encountered in everyday mobile use. **(2) Limited coverage of complex interactions.** As GUI agents improve, evaluation increasingly needs to stress capabilities beyond visual grounding and short navigation chains. Existing benchmarks and datasets extensively evaluate grounding and basic interaction (Cheng et al., 2024; Li et al., 2025), while long-horizon state tracking, repeated information extraction, aggregation, and multi-step reasoning remain less systematically covered. Such tasks are particularly important for exposing failures that may not appear in short interaction sequences. **(3) High evaluation overhead.** Interactive benchmarks commonly rely on emulators, virtual machines, or containerized environments to recover application state and support reproducible execution (Rawles et al., 2025; Xie et al., 2024; Zhou et al., 2024). Although effective, these systems introduce non-trivial setup, reset, and runtime costs, which complicate large-scale parallel evaluation and downstream uses such as iterative agent development or online training. To address these limitations, we introduce SimuWoB.

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

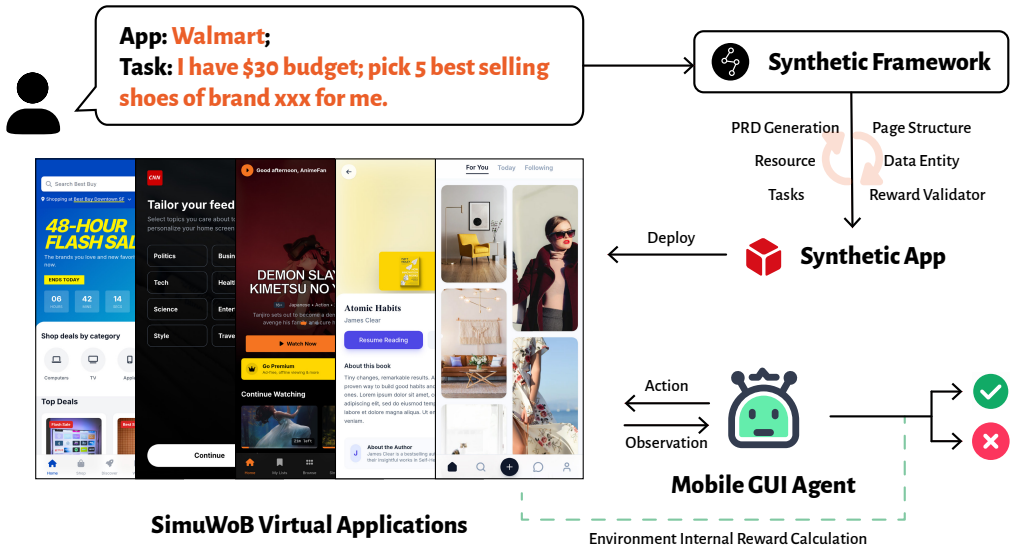


Figure 1: An overview of SimuWoB. After collecting real-world tasks along with related applications, our framework iteratively builds and refines the environment until the task and reward validator are ready. Mobile GUI agents can then directly interact with the environments and perform tasks.

SimuWoB is a fully synthetic benchmark for mobile GUI agents built on a two-stage *simulated environment generation framework*, as illustrated in Figure 1. Given a natural-language task description and optional app screenshots, the framework iteratively generates and refines a backend-free webpage that simulates a real-world mobile application with aligned interaction logic over mock data. During the generation process, it also validates task solvability and produces executable and valid reward functions for each task. This design directly improves **environment coverage**, **reward reliability**, and **evaluation speed**, circumventing the challenge of constructing rewards inherent in real-world apps, enabling more faithful benchmarking at a much lower operational cost. Every environment is able to be served via a URL, avoiding heavyweight application installation and device-state management while enabling lightweight, reproducible, and parallel evaluation. The framework further includes an iterative validation-and-repair process to identify task or environment defects before an environment enters the benchmark. Built with this framework, SimuWoB contains 120 tasks across 63 simulated applications, covering 20 of the 33 Google Play Store application categories. The released benchmark tasks originate from a user-study pipeline involving 86 participants across 77 industries. To probe different capability dimensions, we organize the benchmark into *simple*, *long-horizon*, and *math-related* tasks. Half of the tasks require more than 20 interactions, with the longest requiring over 50 steps. An anonymous interactive demo is available here.

Five recent mobile GUI agents are evaluated on SimuWoB. The average success rate is 27.92%, dropping to 17.82% on the long-horizon subset, revealing substantial remaining difficulty in complex mobile workflows. We further evaluate the same agents on 30 tasks across 23 real applications. Their overall ranking is preserved between simulated and real settings, and performance at the agent-by-category level shows strong correlation (Spearman $\rho = 0.912$, 95% bootstrap CI [0.71, 0.98]). These results provide evidence that SimuWoB captures relative capability differences that are also observed on real mobile applications. Our contributions are as follows:

1. We develop a scalable two-stage framework for synthesizing interactive mobile-style applications together with executable state-based task validators, enabling lightweight and reproducible GUI-agent evaluation.
2. Using this framework, we construct SimuWoB, a benchmark of 120 challenging tasks across 63 simulated applications and 20 Google Play Store categories, grounded in real-world user needs and covering simple, long-horizon, and math-related workflows.
3. We benchmark five recent mobile GUI agents and analyze their failures across key capability dimensions. A 30-task evaluation on 23 real applications further shows strong agreement in relative agent and category-level performance trends between simulated and real settings.

2 Related Work

GUI Agents. In recent years, GUI agents have evolved from systems that primarily rely on textual or structured UI representations (Wen et al., 2024; Deng et al., 2023) to vision-based agents that operate directly from screenshots and predict low-level interface actions (Qin et al., 2025; Xu et al., 2025; Liu et al., 2024; Ye et al., 2025; Zhou et al., 2025). Recent surveys provide broader overviews of computer-use agents and highlight persistent challenges in long-horizon planning, memory, grounding, and robust execution (Sager et al., 2025; Hu et al., 2025). Our work focuses on the complementary problem of how to evaluate these capabilities efficiently and reproducibly.

GUI Agent Benchmarks. GUI-agent benchmarks broadly fall into *static datasets* and *interactive environments*. Static benchmarks such as AITW, Mind2Web, and ScreenSpot-Pro (Rawles et al., 2023; Deng et al., 2023; Li et al., 2025) support scalable offline evaluation of grounding, instruction understanding, and action prediction, but cannot fully capture closed-loop behaviors such as error recovery, delayed feedback, or stateful dependencies across long trajectories. Interactive benchmarks instead execute agent actions in controllable environments and measure end-to-end task success. Representative ones include AndroidWorld, MobileWorld, AndroidLab, A3, and MobileAgentBench (Rawles et al., 2025; Kong et al., 2025; Xu et al., 2024; Chai et al., 2026; Wang et al., 2024), while OSWorld provides a related interactive setting for desktop agents (Xie et al., 2024).

Reliable interactive evaluation requires both recoverable environment state and task-specific success checking. Existing mobile benchmarks commonly achieve this through controllable applications running in emulators and manually constructed task evaluators (Rawles et al., 2025; Kong et al., 2025; Xu et al., 2024). This design provides reliable and reproducible evaluation, but expanding to additional applications and workflows requires corresponding environment integration and evaluation logic. Moreover, emulator- or container-based execution introduces non-trivial deployment and state-reset overhead. Long-horizon information tracking, repeated extraction, and aggregation are also not consistently isolated as explicit evaluation dimensions across existing benchmarks.

Table 1: Comparison with representative GUI-agent benchmarks. "Reward construction" describes how task-completion criteria are constructed. Average step counts are taken from the corresponding benchmark statistics when available. "–" denotes unavailable or inapplicable information.

Benchmark	Interactive	Execution	Reward Construction	Task Source	# Tasks	# Apps	Avg. Steps
AndroidWorld (Rawles et al., 2025)	Yes	Emulator	Human-written	Human	116	20	∼5
MobileWorld (Kong et al., 2025)	Yes	Emulator	Human-written	Human	116	20	27.8
A3 (Chai et al., 2026)	Yes	Emulator	Human-written	Human	201	20	∼10
AndroidLab (Xu et al., 2024)	Yes	Emulator	Human-written	Human	138	9	∼8.5
MobileAgentBench (Wang et al., 2024)	Yes	Emulator	Human-written	Human	100	10	∼4.5
PixelHelp (Li et al., 2020)	No	Offline	–	Human	187	4	4.2
AITW (Rawles et al., 2023)	No	Offline	–	Human	660K	159	6.5
MoTIF (Burns et al., 2021)	No	Offline	–	Human	4,707	125	4.5
AndroidControl (Li et al., 2024)	No	Offline	–	Human	15,283	833	–
SimuWoB	**Yes**	**Web simulation**	**LLM-synthesized Human audited**	**User study**	**120**	**63**	**19.1**

SimuWoB occupies a complementary point in this design space. Rather than instrumenting existing apps, it synthesizes mobile-style interactive environments together with executable state-based validators. This allows app semantics, task logic, and evaluation functions to be constructed jointly, while backend-free URL deployment enables lightweight state reset and parallel execution. Compared with static benchmarks, SimuWoB remains fully interactive; compared with emulator-based mobile benchmarks, it trades native-system fidelity for greater control over app coverage, task complexity, and evaluation infrastructure. Table 1 summarizes representative interactive mobile benchmarks.

3 Methodology

In this section, we describe the construction of SimuWoB. Starting from real-world user needs, we derive executable task specifications (Section 3.1), then synthesize mobile-style environments and

task-specific validators through a two-stage generation pipeline (Section 3.2). We subsequently validate and repair the generated environments through a human-in-the-loop quality-control process (Section 3.3). Finally, we summarize the benchmark and its evaluation interface (Section 3.4).

3.1 Task Collection and Specification

SimuWoB is constructed through a user-need-driven pipeline. To ground benchmark tasks in realistic mobile use cases, we conducted a user study that collected open-ended requests describing tasks participants would like mobile agents to perform. After filtering malformed responses, the study contains 260 valid requests from 86 participants across 77 industries. Each record includes a natural-language request, its motivating use case, and an annotator judgment of whether the request can be operationalized as a mobile-agent task. Additional details of the study are provided in Appendix E.

We transform these requests into executable task specifications in four steps. *(i)* Collected requests are normalized by merging semantically equivalent intents and removing near-duplicates. *(ii)* We remove requests that require unavailable permissions, unsupported cross-application dependencies, or otherwise non-executable conditions. *(iii)* We operationalize the remaining requests into task specifications with explicit execution intent and completion criteria. *(iv)* The resulting pool is balanced across language, application domain, and task difficulty. This process yields 197 candidate task specifications that enter environment construction and validation. Each candidate task is associated with a target application or application domain, and therefore directly determines which environments are constructed by the synthesis pipeline below. All tasks in the final benchmark originate from this user-study pipeline. During environment synthesis, we additionally ask an LLM to propose auxiliary tasks that exercise related application functionality; these auxiliary tasks are used only to improve environment completeness and are *not* included in benchmark evaluation.

3.2 Two-Stage Environment and Validator Synthesis

Given a task specification and its associated app, our goal is to construct an executable environment that reproduces the relevant mobile interaction semantics while exposing controllable internal state for reliable evaluation. We formulate this process as two stages, illustrated in Figure 2: the first constructs a functional application independent of a particular benchmark trajectory, and the second injects task-specific data, logic, and validators.

Stage 1: Application construction. We first collect publicly available application metadata, including the application name, visual style, feature descriptions, and core interaction logic. When available, screenshots are used as transient references for layout, iconography, and information hierarchy; these reference screenshots are not included in the released benchmark. Given these inputs, a code-generation LLM iteratively constructs a minimal working environment (MWE). Each iteration consists of three operations: *(i)* drafting or updating a product requirements document (PRD) describing pages, features, visual conventions, and interaction flows; *(ii)* implementing or revising UI components, data schemas, and interaction logic; and *(iii)* reviewing the current implementation for missing or inconsistent functionality. The review is incorporated into the PRD and used to guide the next iteration. The resulting MWE contains executable interface logic, persistent application state, and initial mock data. An example of the PRD and iterative refinements is provided in Appendix B.

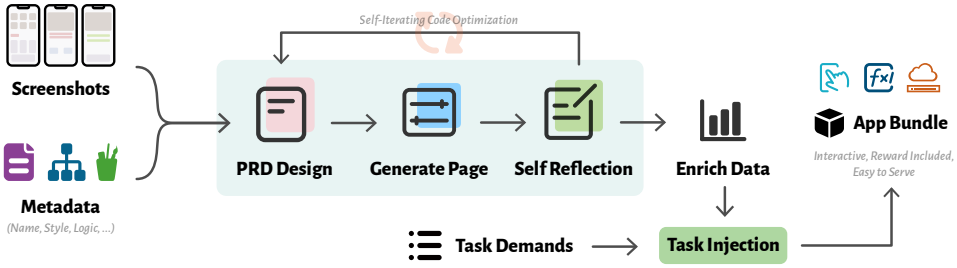


Figure 2: Two-stage environment synthesis pipeline of SimuWoB. Stage 1 constructs a functional mobile-style application from application metadata and optional visual references. Stage 2 enriches its data and injects benchmark-specific logic together with executable state-based validators.

Stage 2: Task and validator injection. Starting from the MWE, we first enrich the environment with task-relevant mock content, including text, images, and structured records, while preserving the data schemas and interaction conventions established in Stage 1. The task-injection agent then receives the task specification, including its intended outcome and verification criteria, and modifies task-relevant application logic when necessary. For each task, the agent also synthesizes an executable *state-based validator*. Because the simulator exposes its internal data state to the evaluation harness, the validator can directly check task-relevant state variables and, when required, returned structured content. This avoids relying solely on trajectory matching or an LLM judge for task completion. The validator is not exposed to the evaluated GUI agent. Separating application construction from task injection also reduces direct coupling between the environment and a single target trajectory: the application's basic pages and interaction logic are constructed before task-specific modifications are introduced. During synthesis, auxiliary LLM-proposed tasks are additionally used to exercise related functionality and discourage construction of an environment that supports only one scripted execution path. We empirically examine the effect of the two-stage design in Appendix F.

3.3 Validation, Repair, and Quality Audit

LLM-generated environments may contain incomplete interaction logic, inconsistent state transitions, or task-specific implementation errors. We therefore apply a validation-and-repair procedure before an environment is admitted to the benchmark, as illustrated in Figure 3. For each generated app bundle, a validation GUI agent interactively attempts every candidate task, after which the synthesized validator checks whether the required final state has been reached. The resulting trajectory and environment state are then inspected during quality control. When an environment- or task-side defect is identified, a human expert provides a targeted description of the issue, which is returned to the synthesis framework for automatic repair and re-validation. Only tasks that pass this process enter the validated candidate pool.

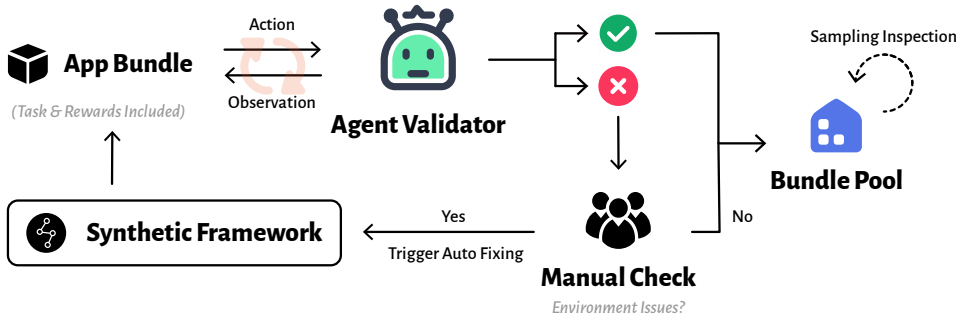


Figure 3: Automatic issue inspection and correction workflow. A validation agent executes each candidate task. Human experts inspect failures and provide targeted feedback for environment-side defects, after which the environment is repaired and re-validated.

Across the 197 candidate tasks, 120 require no repair after the initial synthesis, corresponding to a first-pass acceptance rate of 60.9%. The remaining 77 tasks require one repair cycle based on expert feedback. Two expert annotators independently inspect the candidate tasks; their initial judgments agree on 189/197 cases (95.9%), corresponding to Cohen's $\kappa = 0.91$. The eight disagreements are resolved through reconciliation. For tasks requiring repair, preparing the diagnosis and repair instruction takes approximately 0.3 human-hours per task on average. The recorded repair effort is therefore approximately 23 hours, with approximately two additional hours used for final filtering.

After validation, we perform a final filtering step to remove tasks that are insufficiently realistic or too simple to provide useful evaluation signal, resulting in the released set of 120 tasks. All released tasks are manually checked for executability, instruction clarity, and validator consistency. We additionally audit validator correctness using actual agent evaluation logs. 150 trajectories are sampled, balanced between 75 validator-positive and 75 validator-negative examples, and manually determine task completion while blinded to the original validator verdict. Human inspection agrees with the synthesized validator on all 150 sampled trajectories, confirming the quality of the validators.

3.4 Benchmark Composition and Evaluation Interface

Following the construction and filtering procedure above, SimuWoB contains 120 executable tasks across 63 simulated applications, covering 20 of the 33 Google Play Store application categories. The benchmark includes both Chinese and English interfaces and instructions, and covers a broad range of daily-use domains. A detailed category distribution is provided in Appendix B.

We characterize tasks along two orthogonal dimensions. First, 46 tasks require the agent to return structured information in addition to interacting with the application, while the remaining 74 are evaluated solely from environment state. For return-value tasks, the agent receives a JSON schema together with the instruction; task success depends on both the relevant environment state and the returned schema-compliant content. Second, we group tasks according to their primary source of difficulty: *simple*, *long-horizon*, and *math-related*. Simple tasks primarily require navigation and direct application operations. Long-horizon tasks involve extended action sequences, repeated navigation, or maintaining information across multiple interface states. Math-related tasks require extracting and aggregating information before producing an answer. The final benchmark contains 54 simple, 55 long-horizon, and 11 math-related tasks. Their average reference trajectory lengths are 14.96, 24.73, and 11.27 steps, respectively, with individual tasks ranging from roughly 10 to more than 50 interactions. Representative examples are shown in Table 2.

Table 2: Example tasks from different categories. Step counts refer to reference task executions.

Task Type	**Task**	**# Steps**
Simple	Add xxx item to the cart.	14.96
Long-Horizon	View the first 15 images recommended in "For You", summarize their titles, posters, and the posters' follower counts.	24.73
Math-Related	During my purchase history, how much did I spend on shipping in total?	11.27

Although SimuWoB environments are implemented using web technologies, agents interact with them through a mobile-style visual interface. The agent receives screenshots rather than DOM elements or accessibility trees, and its action space consists of mobile-oriented operations such as tap, swipe, long press, text input, and wait. Web-specific operations such as URL navigation, DOM access, or hover are not exposed. For standardized evaluation, the environment additionally exposes three lightweight DOM-level functions to the evaluation harness: *(i)* `window.getTasks`, which returns task identifiers, instructions, and optional return schemas; *(ii)* `window.evaluateTask`, which checks the task-specific environment state and optional returned content; and *(iii)* `window.reset`, which restores the initial environment state before each trajectory. These evaluator-side APIs are not visible to the GUI agent. Implementation examples are provided in Appendix B. Because each environment is backend-free and served directly through a URL, multiple evaluation instances can be launched without emulator installation or device snapshot management. Our experiments use parallel workers for evaluation; depending on hardware resources, the system can typically run 8–16 concurrent workers or more.

4 Experiments

4.1 Experimental Setup

Evaluated agents. We evaluate five recent mobile GUI agents that operate directly from visual observations: UI-TARS-1.5 (Qin et al., 2025), Seed-1.8 (ByteDance, 2026), Gemini 3 Pro (Google, 2025), GUI-Owl-1.5-8B-Instruct (Xu et al., 2026), and MAI-UI-8B (Zhou et al., 2025). The first three are API-based systems, while the latter two are officially released open-source checkpoints; we do not perform additional fine-tuning. For API-based systems, we use `doubao-ui-tars-250428`, `doubao-seed-1.8-251228`, and `gemini-3-pro-preview`. All agents receive only screenshots and interact through the same mobile-style action interface. We use the prompts and inference settings provided by their official implementations whenever applicable.

Evaluation protocol. Our primary metric is task success rate (SR). Each trajectory is limited to 100 interaction steps, and all reported SimuWoB results are averaged over two runs. Experiments

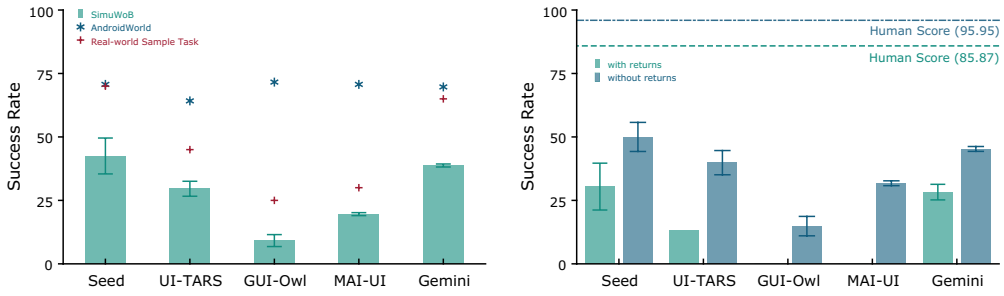
are executed with eight parallel workers. Human performance is evaluated under the same task instructions and success criteria. Among the 120 tasks in SimuWoB, 46 additionally require a structured return value, while 74 can be evaluated solely from environment state. The two local models do not reliably produce schema-compliant structured outputs under their released inference interfaces, so we report their results only on tasks without return-value requirements.

Real-world validation. To examine whether capability differences measured by SimuWoB could also be observed on real applications, we construct an independent validation set of 30 tasks across 23 real mobile apps. The set contains 15 Chinese and 15 English tasks, including 12 simple, 13 long-horizon, and 5 math-related tasks, following the same high-level category definitions as SimuWoB, but use different concrete applications and task instances. All real-world executions are manually judged and do not use SimuWoB validators.

4.2 Main Benchmark Results

The main results are shown in Figure 4. Overall, `seed-1.8` achieves the best SR on SimuWoB, followed by `Gemini 3 Pro`, `UI-TARS-1.5`, `MAI-UI` and `GUI-Owl-1.5`. For tasks without return-value requirements, `seed-1.8` reaches 50.00%, `Gemini 3 Pro` reaches 45.27% and `UI-TARS-1.5` reaches 39.86%; for tasks with return-value requirements, their SR drops to 30.43%, 28.26% and 13.04%, respectively; this phenomenon may stem from the fact that tasks requiring return values are relatively more difficult. For local models, `MAI-UI` outperforms `GUI-Owl-1.5`. Although all agents report much higher scores on AndroidWorld (64.2%–71.6%), their SR on SimuWoB is substantially lower, indicating the higher challenge posed by SimuWoB. While humans earn an average score of 92.08%, which proves there is still huge ability gap between current mobile GUI agents and humans.

Notably, the performance trends observed on sample tasks from real apps are largely consistent with those of SimuWoB, meaning the relative performance rankings among models remain the same. This demonstrates that evaluation results obtained in synthetic environments can generalize to real-world apps, thereby validating the feasibility and generalization ability of this approach. Detailed statistical correlation analysis are placed at Appendix C.



(a) GUI agent SR on different benchmarks

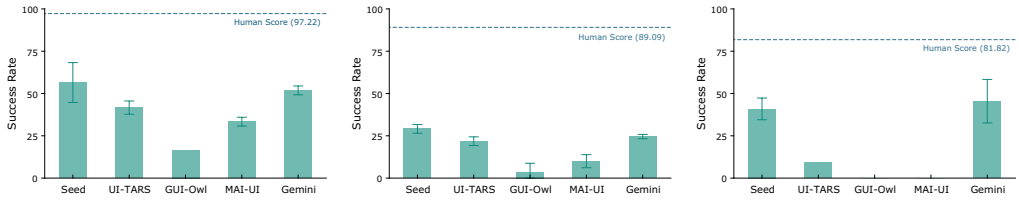
(b) SR on w/ & w/o return-value subsets

Figure 4: Experimental results of different agents on SimuWoB. For local models, we evaluate only tasks without return-value requirements.

4.3 Analysis

Tasks in SimuWoB are challenging for mobile GUI agents. As shown in Figure 4a, all evaluated agents achieve much higher SR on AndroidWorld (64.2%–71.6%) than on SimuWoB. On SimuWoB, overall SR drops to 42.50% for the best model (`seed-1.8`) and to as low as 9.17% for `GUI-Owl-1.5-8B`; even on the easier subset without return-value requirements, the best SR is only 50.00%. At the aggregate level, the average SR across all models is 27.92%, versus 69.38% on AndroidWorld, and this drop is consistent for all agents. In addition, SimuWoB exhibits clear discriminative power: the best–worst spread in overall SR reaches 33.33 points (42.50% vs 9.17%),

and even within stronger API models the spread remains substantial (42.50% vs 29.58%). These results show that SimuWoB is substantially more challenging for current mobile GUI agents and provides stronger separation across capability levels. From Figure 4b, we can tell that tasks with return values are relatively harder, leading to an obvious SR drop for agents.



(a) Simple Subset (b) Long-horizon Subset (c) Math-related Subset

Figure 5: Success rate of different task categories across evaluated agents in SimuWoB.

Mobile GUI agents fall short in *long-horizon* tasks. Results in Figure 5 show a clear drop on *long-horizon* tasks compared with *simple* tasks for all agents (e.g., 56.48% vs 29.09%), with an average SR of only 17.82%. This gap suggests the core bottleneck is not UI operation, but state management across many steps: agents must identify and preserve critical intermediate information while discarding low-value ones under limited context budgets. The case in Figure 6 illustrates this failure mode. The task asks the agent to traverse historical trips and aggregate service fees and taxes. The agent executes each UI step correctly (opening "Trips", entering history, opening each trip invoice), but fails to write extracted values into a persistent internal state. It then moves to the next trip assuming previously seen details are still available, and eventually produces an incorrect final response. Because most agents cannot keep all prior screenshots and observations in context at every step, *long-horizon* performance depends on explicit memory strategies (what to store, when to update, and how to reuse it), which current models still handle poorly. Similar issues also exist for *math-related* tasks because agents also need to take down key information for further calculations. SR of human also falls on these subsets.

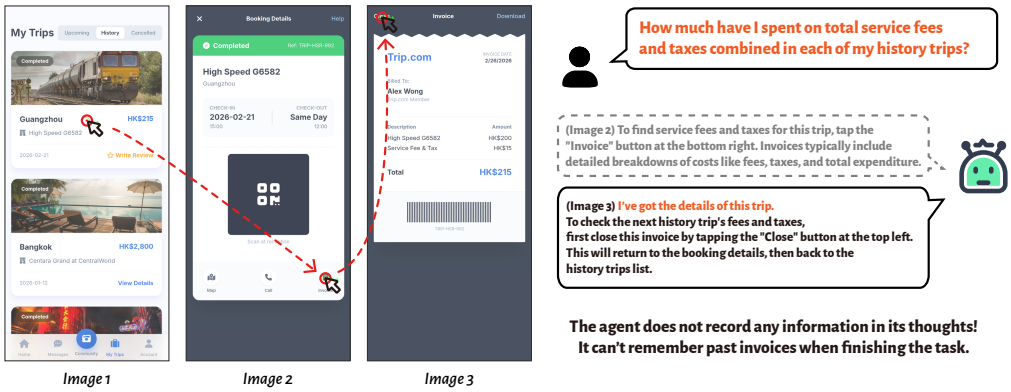


Figure 6: Case study of a *long-horizon* failure: the agent executes UI operations correctly but does not persist key information in context, leading to an incorrect final answer.

Agents struggle to explore implicit entry points. Some tasks require discovering a functional entry point that is not immediately visible (e.g., deeply nested in the UI and not exposed on the home screen). In these settings, the key bottleneck is weak proactive exploration after an initial strategy fails. The case in Figure 7 illustrates this behavior. The agent is asked to find a "Client Sync" event in a simulated Gmail app and return its start time and participants. Because the instruction does not explicitly include whether the event should be searched in email or calendar views (the latter requires opening the side menu), the agent first searches in email but finds no result. Instead of switching hypotheses and exploring alternative entry points, it continues with low-yield scrolling and eventually

fails. This pattern suggests that robust exploration, trial-and-error adaptation, and online experience reuse remain underdeveloped in current agents, especially when interface affordances are not explicit.

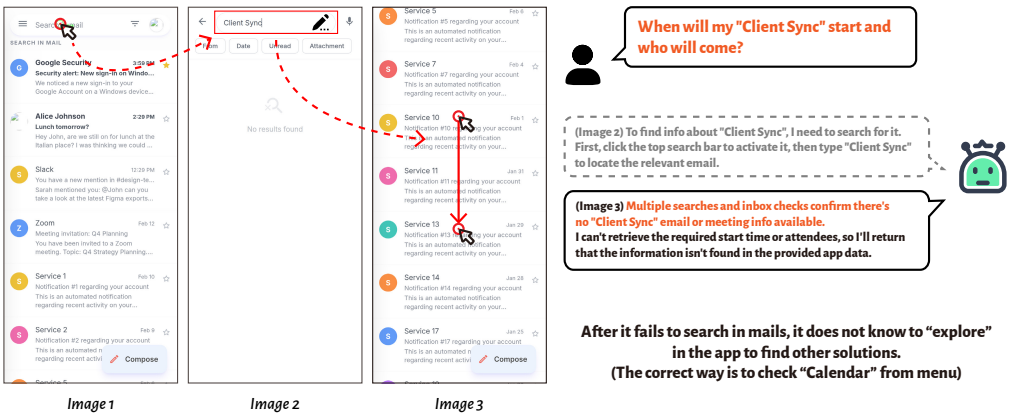


Figure 7: Case study of a *vague-description* failure: the agent fails to locate the task entry point due to a lack of proactive exploration capabilities following initial failures.

Agents perform poorly on tasks that require fine-grained control. Fine-grained control is a common requirement in real-world mobile tasks, including dragging a slider to a target position, setting date/time values with pickers, confirming payment via drag gestures, and invoking context menus through long presses. Compared with standard click-and-type tasks, these operations impose much stricter spatial and temporal constraints: small coordinate errors or incorrect interaction timing are often enough to cause failure. We curated 11 such tasks in SimuWoB and summarize model performance in Figure 8. While this subset is quite easy for humans to finish (full score on all tasks), the results on agents show a clear weakness: Gemini 3 Pro completes 5/11 tasks (still below 50%), while the other agents complete only about 0.5 to 1.5 tasks on average.

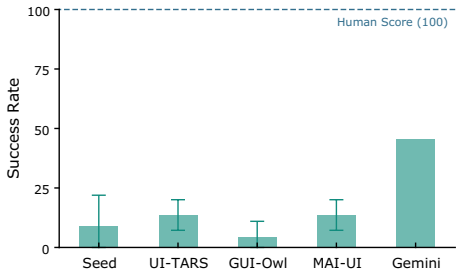


Figure 8: Fine-grained control results.

Inspection of failed trajectories reveals two recurring patterns. *(1) Coordinate-sensitive execution.* For drag-and-drop interactions, many predicted coordinates are not accurate enough, leading to repeated unsuccessful attempts to reach the target state. *(2) Interaction-policy discovery.* Control semantics vary across interfaces: some scrollbars respond to taps, others require dragging, and some hidden actions are exposed only by long press. Without robust UI priors or effective online trial-and-error adaptation, agents struggle to discover and execute the correct control policy.

5 Conclusion

In this paper, we present SimuWoB, a fully synthetic benchmark for evaluating mobile GUI agents. SimuWoB uses a two-stage LLM-driven framework to synthesize backend-free, interactive mobile-style environments together with executable state-based validators. The resulting benchmark contains 120 tasks across 63 simulated applications, covering simple, long-horizon, and math-related workflows grounded in real-world user needs. Evaluation of five recent mobile GUI agents reveals substantial remaining headroom, especially on long-horizon and information-intensive tasks. A 30-task study on 23 real applications further shows strong agreement in relative agent and category-level performance trends between simulated and real settings. Together with our analysis of information management, exploration, and fine-grained control failures, these results suggest that SimuWoB provides a scalable testbed for measuring and diagnosing progress in mobile GUI agents.

AI USE STATEMENT

Generative AI tools were used as an integral component of the benchmark construction method. Specifically, code-generation language models were used to synthesize application code, mock data, interaction logic, and task-specific state-based validators from human-derived task specifications and application metadata, as described in Section 3. So we admit the usage of AI for research ideation and the synthesis of datasets. The resulting environments and validators were subjected to the validation, human inspection, and repair procedures described in the paper.

We also used AI to assist with manuscript editing and polishing, including improving wording, clarity, and organization. All AI-assisted writing and generated research artifacts were reviewed by the authors. The authors take full responsibility for the final manuscript, scientific claims, experimental results, and artifacts produced with the aid of generative AI.

REFERENCES

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments, 2021. URL https://arxiv.org/abs/2104.08560.

ByteDance. ByteDance Seed 1.8. https://seed.bytedance.com/en/seed1_8, 2026.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, Shuai Ren, Rongduo Han, Haining Zhang, Siyuan Huang, and Hongsheng Li. A3: Android agent arena for mobile gui agents with essential-state procedural evaluation, 2026. URL https://arxiv.org/abs/2501.01149.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents, 2024. URL https://arxiv.org/abs/2401.10935.

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

Google. A new era of intelligence with Gemini 3. https://blog.google/products-and-platforms/products/gemini/gemini-3/, November 2025.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, Jiwei Li, Yuchen Eleanor Jiang, Wangchunshu Zhou, Guoyin Wang, Keting Yin, Zhou Zhao, Hongxia Yang, Fan Wu, Shengyu Zhang, and Fei Wu. Os agents: A survey on mllm-based agents for general computing devices use, 2025. URL https://arxiv.org/abs/2508.04482.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments, 2025. URL https://arxiv.org/abs/2512.19432.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use, 2025. URL https://arxiv.org/abs/2504.07981.

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, 2024. URL https://arxiv.org/abs/2406.03679.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences, 2020. URL https://arxiv.org/abs/2005.03776.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, and Jie Tang. Autoglm: Autonomous foundation agents for guis, 2024. URL https://arxiv.org/abs/2411.00820.

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, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

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

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

Pascal J. Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F. Grewe, and Thilo Stadelmann. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions, 2025. URL https://arxiv.org/abs/2501.16150.

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents, 2024. URL https://arxiv.org/abs/2406.08184.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android, 2024. URL https://arxiv.org/abs/2308.15272.

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

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-agent-v3.5: Multi-platform fundamental gui agents, 2026. URL https://arxiv.org/abs/2602.16855.

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, 2024. URL https://arxiv.org/abs/2410.24024.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction, 2025. URL https://arxiv.org/abs/2412.04454.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. Evocua: Evolving computer use agents via learning from scalable synthetic experience, 2026. URL https://arxiv.org/abs/2601.15876.

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. Mobile-agent-v3: Fundamental agents for gui automation, 2025. URL https://arxiv.org/abs/2508.15144.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. Mai-ui technical report: Real-world centric foundation gui agents, 2025. URL https://arxiv.org/abs/2512.22047.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

A LIMITATIONS

While our benchmark is built on LLM-synthesized environments and automatically constructed rewards, and already provides informative signals for agent evaluation, several limitations remain. (1) Observation space. Since our environments are web-based, they currently expose only visual observations and interaction affordances available on the page, rather than richer structured signals (e.g., accessibility trees) that can be accessed on real Android systems. (2) Cross-app task support. At present, SimuWoB focuses on single-app workflows and does not yet cover tasks that require coordinated operations across multiple applications. However, these constraints should be interpreted as scope boundaries rather than invalidating factors. They do not undermine the benchmark's practical value, and they do not affect the main empirical conclusion of this work: current mobile GUI agents still show substantial gaps on realistic, long-horizon tasks. More broadly, the field continues to face objective bottlenecks in benchmark construction, including environment selection, reliable reward design, and evaluation efficiency. Our synthesis pipeline provides a technically viable foundation to address both limitations above, and we leave these to future work.

B SIMUWOB ENVIRONMENT SYNTHESIZING & STATISTICS

PRD Iterations. Following the pipeline of Figure 2, we first had the model draft a detailed PRD document based on the given metadata, then asked it to write code based on the document. Here follows an example of PRD document when generating simulated application iQIYI, which consists of many parts that defines the workflow, UI transition logic, data entity, styles, etc.

Product Overview

Product Positioning

iQIYI is a leading high-quality video entertainment service platform. Its core positioning is to provide long-form video streaming services (movies, TV series, variety shows, anime), while gradually integrating short-form video feeds, pan-entertainment communities, and IP-derived consumption (literature, e-commerce, ticketing).

Core Value

- *Enjoy Quality*: Emphasizes high-definition picture quality, Dolby audio, and exclusive original content.
- *Immersive Experience*: Enhances viewing focus with dark mode and distraction-free playback design.
- *VIP Privileges*: Establishes a comprehensive membership tier and benefits system, providing differentiated services.

Design Philosophy & Style

Overall Style

- *Browsing Mode (Light)*: On pages such as "Home", "Discovery", and "Profile", a large area of white and light gray backgrounds is used. The design pursues breathability, reduces the use of dividing lines, and uses card shadows and spacing to distinguish content, reducing visual pressure caused by information density.
- *Immersive Mode (Dark/Gold)*: On the "Playback" page and "VIP Member" page, the interface switches to dark or black-gold color schemes. The dark background highlights the video, creating a cinema-like atmosphere; the black-gold tones reinforce the sense of premium status and privileges for VIP users.

Visual Metaphors

- *Cards*: Simulate physical posters, carrying content covers.
- *Flow*: Waterfall flow layout, metaphorically representing an endless supply of content.

Main Interface Architecture

Home - Content Distribution Center

- *Layout Logic*: Top search + navigation → Header banner carousel → Double-column waterfall recommendation feed.
- *Key Elements*:
  - Top Function Area: Contains search box (displaying trending search terms), history entry, and upload/publish entry.
  - Navigation Bar: Horizontally scrollable tabs (Recommendations, TV Series, Movies, Anime, etc.).
  - Content Cards: Uses 16:9 cover images, with the drama title, subtitle (e.g., "All 36 Episodes"), and badges (e.g., "Exclusive," "Free for a Limited Time") displayed at the bottom.
- *Design Purpose*: Maximize content exposure and guide user clicks through algorithmic recommendations.

Free / Suike (Short Video) - Immersive Short Videos (...)

VIP (Membership) - Monetization Center (...)

Discovery - Community & Topics (...)

Core User Flows

Long-Form Video Watching Flow

1. *Trigger*: User clicks on a drama/movie cover on the [Home] page.
2. *Transition*: The page slides right to enter the [Playback Details Page].
3. *Loading*:
   - Non-member: Pre-roll ad plays before the video (showing countdown and "Skip ad with membership" button).
   - Member: Enters the main video directly, with a prompt saying "Exclusive VIP, ads have been skipped for you".
4. *Actions*: Single-tap on the video area to evoke the control layer; use gravity sensor or tap the button to switch to [full-screen landscape mode].

Membership Subscription Flow (...)

Short Video Browsing Flow (...)

Visual Interface Guidelines

Color Palette

- *Primary Brand Color*: iQIYI Green (#00CC36). Represents vitality and youthfulness. Used for the logo, selected states, and primary action buttons.
- *Secondary Colors*:
  - VIP Gold (#E2B376): Used for membership identifiers and paid buttons.
  - Alert Red (#FF3B30): Used for notification badges and limited-time flash sales.
- *Neutral Colors*:
  - Backgrounds: #FFFFFF, #F5F5F5, #1A1A1A
  - Text: #333333, #666666, #999999

Typography: Chinese Fonts: ..., English Fonts: ..., Hierarchy Specifications: ...

Iconography & Components

- Icon Style: Primarily line icons, transitioning to solid style with the brand color when selected. Lines are rounded and approachable.
- Badges: Located in the top-right or bottom-right corner of the cover, using semi-transparent backgrounds or brand-colored backgrounds, with extremely small font sizes, used to convey status.
- Buttons: Full pill-shaped buttons, emphasizing tactile comfort.

Design Summary

The design of the iQIYI App successfully balances the tension between massive content distribution and user experience.

1. Clear Information Hierarchy: Through the card-style design featuring "large images with minimal text", users can quickly capture the visual focus while scrolling rapidly.
2. Contextual Design: Strictly distinguishes between the "browsing for content" scenario (bright, efficient) and the "watching content" scenario (dark, immersive), aligning with user mental models.
3. Monetization Integration: The VIP membership design is not just a functional entry point but an independent visual system that effectively stimulates users' desire to pay through color psychology.
4. Ecosystem Loop: Cleverly embeds short videos (Suike) and community (Discovery) into the bottom navigation, forming a content consumption loop of "long-form video attracts → community discussion → short-form video kills time".

```
window.getTasks = function () {
  return [
    {
      taskId: 0,
      task: "Search for the TV series The Knockout and add it to your collections.",
      params: {
        type: "object",
        properties: {
          videoId: { type: "integer", const: 1 },
        },
        required: ["videoId"],
      },
    },
    {
      taskId: 1,
      task: ```Purchase the "The Knockout Official Artbook" from the Mall. During checkout,
      create and use a new shipping address with the recipient name 'TestUser'.```,
      params: {
        type: "object",
        properties: {
          recipientName: { type: "string", const: "TestUser" },
        },
        required: ["recipientName"],
      },
    },
    {
      taskId: 2,
      task: ```Customize the app appearance by changing the theme to Pink and set the
      default playback quality to '4K' in Settings.```,
      params: {
        type: "object",
        properties: {
          targetTheme: { type: "string", const: "#FF3366" },
          targetQuality: { type: "string", const: "4K" },
        },
        required: ["targetTheme", "targetQuality"],
      },
    },
    {
      taskId: 3,
      task: ```Go to the Wallet's Points Mall and redeem the 1-Day VIP Pass using
      your points.```,
      params: {
        type: "object",
        properties: {
          productTitle: { type: "string", const: "1-Day VIP Pass" },
        },
        required: ["productTitle"],
      },
    },
  ];
};
```

Listing 1: Example of function `window.getTasks`.

After writing, it reviewed the existing codebase, proposed a series of items to be added or modified, updated the PRD document accordingly, and iterated repeatedly. During iterations, we collect 2 examples of self-reflection outputs of the model to reveal this progress:

- *(Iteration 1)* I have enhanced the 'Store' to handle more complex user states (search history, following, daily check-in), updated the 'Router' to handle view cleanup (essential for timers like carousels), and significantly upgraded the 'HomeView', 'PlayerView', 'SearchView', 'ShortView', and 'WalletView' to be fully interactive and logic-driven.
- *(Iteration 2)* I have added the missing 'Collections', 'About', and 'Feedback' views, updated the 'Player' to handle episodes dynamically, and ensured the 'Profile' and 'Settings' flows are fully connected. I also updated the 'Store' to seed some initial collection data for a better first-run experience.

After 15 iterations, we get the minimal working example of the application, then the data will be further enriched, and task requirements will be provided to the model to inject to the app as well. Finally, all provided tasks will be bundled into the environment through three DOM-based API functions, as we introduced in Section 3.4.

App Category Statistics. Against public Play Store statistics (non-game categories, apps >50K downloads), SimuWoB's 20 categories cover 81.2% of apps, including 8 of the top 10 and 16 of the top 20 categories; the 13 uncovered ones are each <5.2% and some (Personalization, Comics) offer little task surface. Internally the 63 apps spread as Social 8, Travel & Local 8, Shopping 7, Music & Audio 6, Communication 5, Entertainment 4, News/Food/Business 3 each, tail of 1–2.

```
window.evaluateTask = function (params) {
  if (!params || params.taskId === undefined) return { success: false };
  const user = window.AppStore.getUser();

  switch (params.taskId) {
    case 0: {
      // Task: Collect video 1
      const isCollected = window.AppStore.isCollected(params.videoId);
      return { success: isCollected, score: isCollected ? 100 : 0 };
    }
    case 1: {
      // Task: Buy product with specific address name
      const orders = user.orders || [];
      // Find order for the specific book
      const order = orders.find((o) => o.title.includes("The Knockout Official Artbook"));
      if (!order) return { success: false, score: 0 };

      // Check if address contains the required name
      const addressMatch =
        order.address && order.address.includes(params.recipientName);
      return { success: addressMatch, score: addressMatch ? 100 : 0 };
    }
    case 2: {
      // Task: Theme Pink and Quality 4K
      const currentTheme = user.settings.themeColor;
      const currentQuality = user.settings.playback
        ? user.settings.playback.defaultQuality
        : "";
      const success =
        currentTheme === params.targetTheme &&
        currentQuality === params.targetQuality;
      return { success: success, score: success ? 100 : 0 };
    }
    case 3: {
      // Task: Redeem VIP Card
      const transactions = user.transactions || [];
      const redeemed = transactions.some((t) =>
        t.title.includes(params.productTitle),
      );
      return { success: redeemed, score: redeemed ? 100 : 0 };
    }
  }
};
```

Listing 2: Example of function `window.evaluateTask`.

DOM Function Examples. Listing 1 shows an example of `window.getTasks` function in the environment. This function returns a series of dictionary objects, each containing information about a task supported by each environment, including the task ID, description, and the JSON schema of the expected return value. Note that the model tends to generate schemas that only contain 'const' fields; during evaluation, we automatically filter out such fields to avoid confusing the agent being evaluated.

Listing 2 shows an example of `window.evaluateTask` function. For each task, the environment has corresponding low-level code that can directly access its relevant data state; we use local storage to manage the underlying data and ensure data consistency. This code is very concise and intuitive, with correct logic, so no additional manually written pattern matching or similar logic is needed to complete the evaluation.

Listing 3 shows an example of `window.reset` function. Using this function, we can reset various states in the environment, ensuring that state modifications from previous task executions do not interfere with subsequent task executions. Additionally, this function also resets the page position, ensuring the page returns to its initial state.

```
window.reset = function () {
  localStorage.clear();
  window.location.reload();
};
```

Listing 3: Example of function `window.reset`.

C Comprehensive Experimental Results

Detailed results on benchmark. Table 3 and table 4 displays the full experimental results on SimuWoB. During evaluation, all GUI agents were tested using the prompts, code, and other resources from their official implementations. The experiment is run with 8 parallel workers. The full benchmark dataset is available at this link. SimuWoB provides unified interfaces for agent-environment interactions. The observation channel currently exposes screenshots of the environment state. The action interface maps model outputs (action type and parameters) to executable simulator operations. Supported operations include click, swipe, long press, type, clear text, enter, wait, etc. This design is compatible with most existing mobile GUI agents.

Table 3: Experimental results of different agents on SimuWoB.

Agent	**SR**	**SR** (*w/ returns*)	**SR** (*w/o returns*)	**SR** (*AW*)
# of Tasks	120	46	74	116
`seed-1.8`	42.50 (±7.07)	30.43 (±9.22)	50.00 (±5.73)	70.7
`Gemini 3 Pro`	38.75 (±0.59)	28.26 (±3.07)	45.27 (±0.96)	> 69.7
`UI-TARS-1.5`	29.58 (±2.95)	13.04 (±0)	39.86 (±4.77)	64.2
`GUI-Owl-1.5-8B`	9.17 (±2.35)	-	14.86 (±3.83)	71.6
`MAI-UI-8B`	19.58 (±0.59)	-	31.76 (±0.95)	70.7
Average	27.92	23.91	36.35	69.38
Human	92.08 (±0.59)	85.87 (±1.54)	95.95 (±1.92)	-

To verify that our benchmark effectively simulates real-world app usage scenarios, we select a total of 20 tasks across 17 apps in a real mobile environment and evaluate these GUI agents on them. The test results are manually verified to ensure the accuracy of the evaluation. These tasks still cover the categories of *simple*, *long-horizon*, and *math-related*, as shown in Table 6. As demonstrated in the main text of our paper, we found that the performance trends of GUI agents on these sample tasks are largely consistent with those observed on our benchmark. This indicates that our benchmark's evaluation of agent capabilities generalizes well to real-world usage scenarios.

Table 4: Success rate of different task categories across evaluated agents. We use the first 4 letters to represent the corresponding category: *simple* (simp), *long-horizon* (long), and *math-related* (math).

Agent	**SR** (*simp.*)	**SR** (*long.*)	**SR** (*math.*)
# of Tasks	54	55	11
`seed-1.8`	56.48 (±11.7)	29.09 (±2.57)	40.91 (±6.42)
`Gemini 3 Pro`	51.85 (±2.62)	24.55 (±1.28)	45.45 (±12.86)
`UI-TARS-1.5`	41.67 (±3.92)	21.82 (±2.57)	9.09 (±0)
`GUI-Owl-1.5-8B`	16.67 (±0)	3.64 (±5.14)	0.00 (±0)
`MAI-UI-8B`	33.33 (±2.62)	10.00 (±3.86)	0.00 (±0)
Average	40.00	17.82	19.09
Human	97.22 (±3.93)	89.09 (±2.57)	81.82 (±0)

Statistical Correlation Analysis. Here we list the results of correlation analysis between SimuWoB and the real-world validation set. On the task level, the order remains the same; so we move on and perform a more fine-grained analysis on agent × category level, as shown in Table 5.

Table 5: Performance comparison across different task categories.

Agent	**Simp.**		**Long.**		**Math**	
	simu	**real**	**simu**	**real**	**simu**	**real**
Seed 1.8	56.5	75.0	29.1	61.5	40.9	80.0
Gemini 3 Pro	51.9	91.7	24.6	46.2	45.5	60.0
UI-TARS-1.5	41.7	66.7	21.8	30.8	9.1	20.0
MAI-UI-8B	33.3	66.7	10.0	0.0	0.0	0.0
GUI-Owl-1.5-8B	16.7	58.3	3.6	0.0	0.0	0.0

Based on the results, the Spearman ρ is 0.912, bootstrap 95% CI is [0.71, 0.98] (exclude 0), and Pearson r is 0.917. The regression slope is 1.58, intercept is 3.3 and R^2 equals to 0.84. Consequently, we have the following findings: *(1)* The within-agent difficulty ordering is preserved. Agents do better on simple tasks and worse on long-horizon or math tasks. *(2)* The capability-frontier is preserved. Two local models score 0% on real long-horizon and math tasks, matching their near-zero SimuWoB scores (3.6%/10.0% and 0.0%/0.0%), which means SimuWoB is not manufacturing a weakness that real apps do not show. *(3)* It demonstrates that on a finer-grained level, the real-world validation result matches the one from SimuWoB, further verifying the generalization of our simulation environments.

D Broader Impacts

The broader impact of SimuWoB is primarily tied to its role as an evaluation and development testbed for mobile GUI agents. On the positive side, a fast, reproducible, and controlled benchmark can help researchers identify failures in long-horizon reasoning, memory, exploration, and fine-grained control before such agents are deployed in real applications. Because the environments are synthetic and backend-free, evaluation can be conducted without exposing real user accounts, private records, payments, messages, or other sensitive app data. This may lower the barrier to safer and more transparent analysis of mobile-agent capabilities. At the same time, stronger GUI agents may also enable harmful automation, such as unauthorized account operations, spam, fraud, scraping, or privacy-invasive workflows, if deployed without appropriate safeguards. Benchmarks like SimuWoB could indirectly accelerate these capabilities by making agent weaknesses easier to diagnose and improve. We therefore encourage future users of SimuWoB to treat it as a controlled research benchmark rather than evidence of readiness for unsupervised real-world deployment, and to pair progress on task success with safeguards such as permission checks, human confirmation for sensitive actions, rate limits, and auditing.

Table 6: Real-World Validation Sample Tasks

App	Task Description
RedNote	Find Lei Jun's homepage and like the second post.
Douyin	Search for "ice cream" in the group-buying section and add the first item to favorites.
Weibo	Like the latest Weibo post by the user "Cherry".
AMap	Find public transportation routes from the current location to Wangfujing, and start navigation for the route with the shortest travel time.
Settings	Without using the search function, open the "Show pointer location" setting in Developer Options.
Wikipedia	Open the article explaining Europa Universalis V.
Instagram	Scroll through Instagram Reels until you find one with over 2M likes, and like it.
Reddit	Search for news about the new model from Anthropic and upvote to a relevant post with more than 1k upvotes.
Google Play	Check the most relevant ratings and reviews of app "YouTube Kids".
Spotify	Search for Jay Chow and start playing his album "Opus 12" in random order.
RedNote	Browse the first 10 posts recommended on the homepage, and record the author, number of likes, and number of comments for the post with the highest number of likes.
Pinduoduo	Browse the first 10 items in the "Billions Subsidies" channel, and output the store name, price, and number of product reviews.
Netease Music	Browse the songs in "Daily Recommendations" and output the title, artist, and album of the top three songs with the longest duration.
Hema Fresh	In the categories (excluding "Recommended for You"), view the subcategories under each category, and output which category has the most subcategories along with the count.
Trip	Search for flights from Beijing to Shanghai, browse the top 5 flights for each of the next 3 days, record the flight with the lowest price for each day, and output the date, departure time, airline, and ticket price.
Booking	Search for hotels in Xiamen from May 15th to May 16th at Siming District. View the top 15 hotels and output the ones with free cancellation and over 1000 reviews.
Tiktok	Watch 15 videos and list the creator, like count, and comment count for the top 3 most-liked videos.
Spotify	Search for Vae and check all his releases. Output the name, number of songs and total time of each album.
eBay	Search for iPhone and check the top 20 items. Tell me the names, shop names and maximum prices of Brand New items.
Google Play	Check top 15 Top Charts Games, and return the app with the most raters. Tell me its name, rating score, and numbers of downloads.
Douyin	Search for "Iran", review the first 15 videos in the search results, and record the author, number of likes, number of favorites, and number of comments for the three videos with the highest number of favorites.
Ximalaya	Search for "Water Margin", review the first 15 albums, and record the album name, play count, and recommendation score for albums with a recommendation score above 90%.
Zhihu	Browse the top 12 questions on the trending list, filter for questions with more than 300 followers, and output the question title, topic tags, and total view count.
Jingdong	Search for "apple" and find the first tablet product in the product list, and add it to the shopping cart.
CSDN	Browse blog posts related to "Kimi" and like the first post with more than 8,000 views.
YouTube	Search for Python tutorial videos (excluding shorts or ads), and check the top 20 items. Report the video titles and authors for those whose release dates are within 3 years, and views are over 1M.
Airbnb	Search for room in Paris from Apr 30th to May 1st with 1 adult. Check the top 20 items, and output the descriptions of rooms whose ratings are higher than 4.85 and prices are lower than $100.
Reddit	Look into the current maximum number of achievements a Reddit user can unlock.
Instagram	Search for OpenAI on Instagram and like the first post that has over 5 million views.
Wikipedia	Open the first feature article, and set the theme to sans-serif, 130% font and Sepia color.

E User Study

We conducted a volunteer-based user study to collect realistic mobile-agent task requests. Participants were asked to describe daily mobile-app tasks they would like an agent to help with, along with brief

background pain points. Participation was voluntary and uncompensated. The study did not ask participants to provide account credentials, private messages, payment records, or other sensitive personal data. Collected requests were manually filtered, normalized, and transformed into synthetic benchmark tasks; no real user accounts or real personal records are included in SimuWoB. All examples reported in the paper are anonymized. The detailed questions are displayed in Table 7. After the information was collected, we organized the feedback provided by the volunteers and designed corresponding tasks based on these actual needs, as shown in Table 8. This helps us to better understands the actual pain points of daily usage on mobile applications, thus designing more appropriate tasks to evaluate GUI agents on a benchmark which is much closer to the real world scenario.

Table 7: Questions we asked for volunteers in our user study.

No.	Question
1	What are the main apps you typically use, and what activities do you usually perform with them?
2	While using these apps, are there any aspects you find unsatisfactory or overly cumbersome?
3	Are there any needs in your daily life or work that you wish your smartphone could support but currently cannot?
4	Are there any mobile phone features or usage methods that you know others around you use, which you feel could be helpful to you, but you don't know how to use yourself?
5	Apart from your current smartphone, do you use any other smart devices regularly (e.g., a second phone, computer, tablet, etc.)? What do you use each of these devices for?
6	On a scale of 1 to 9, how would you rate your current smartphone user experience? (1 = "Completely unable to meet my needs, useless"; 9 = "Meets all my needs, no need for improvement"). Please explain your reason.

Table 8: Example of requests from user study and actual tasks included in SimuWoB.

ID	Category	Original Request	Verified Task
118	Cost	Help me record all online expenses, including credit card payment records.	During my history purchases, how much did I spend on shipping in total?
14	Work	Help me check which students in the WeChat group have not submitted their assignments and remind them to complete their work.	Check the "2025 Student" chat, identify students who have not completed the "Completed Assignment Relay" and send each of them a message.

F Ablations on 2-stage Synthesis Pipeline

As in Section 3, we apply a 2-stage pipeline to produce environments. In order to further demonstrate its advantages, we conduct corresponding ablation experiments on this point. This experiment compare two approaches. The first is the two-stage generation scheme mentioned in our paper, which first constructs the app MWE and then introduces specific task logic to ensure that the app's basic functionalities are complete. The second is a one-stage approach that constructs the app and its corresponding tasks simultaneously. We use both methods to synthesize 20 different apps, and employ LLM-as-judge to evaluate the two sets of synthesized apps in terms of whether non-golden-path flows are functional, and whether there're placeholder/dead-controls. Then a score out of 100 is assigned. The evaluation results are presented below; the generation model and the judge model remain the same across these 2 methods.

Table 9: Ablation experiment on 2-stage synthesis pipeline.

Method	2-stage (SimuWoB)	1-stage
Score	**85.0**	69.2

From the comparison result, we could clearly see that 2-stage generation produces much better results. 1-stage method leads to an environment that leaves much functions as static placeholders, while two-stage one gets richer pages, more complicated and higher-fidelity components and better interaction experience.