MobileBench-OL: A Comprehensive Chinese Benchmark for Evaluating Mobile GUI Agents in Real-World Environment

Qinzhuo Wu[1*], **Zhizhuo Yang**[12*], **Hanhao Li**[13], **Pengzhi Gao**[1†], **Wei Liu**[1], **and Jian Luan**[1]

[1]MiLM Plus, Xiaomi Inc [2]Peking University [3]Chinese University of Hong Kong

{wuqinzhuo, gaopengzhi, liuwei40, luanjian}@xiaomi.com,

{yzz2022}@stu.pku.edu.cn, {lihanhao}@link.cuhk.edu.hk

Abstract

Recent advances in mobile Graphical User Interface (GUI) agents highlight the growing need for comprehensive evaluation benchmarks. While new online benchmarks offer more realistic testing than offline ones, they tend to focus on the agents' task instruction-following ability while neglecting their reasoning and exploration ability. Moreover, these benchmarks do not consider the random noise in real-world mobile environments. This leads to a gap between benchmarks and real-world environments. To addressing these limitations, we propose MobileBench-OL, an online benchmark with 1080 tasks from 80 Chinese apps. It measures task execution, complex reasoning, and noise robustness of agents by including 5 subsets, which set multiple evaluation dimensions. We also provide an auto-eval framework with a reset mechanism, enabling stable and repeatable real-world benchmarking. Evaluating 12 leading GUI agents on MobileBench-OL shows significant room for improvement to meet real-world requirements. Human evaluation further confirms that MobileBench-OL can reliably measure the performance of leading GUI agents in real environments. Our data and code will be released upon acceptance.

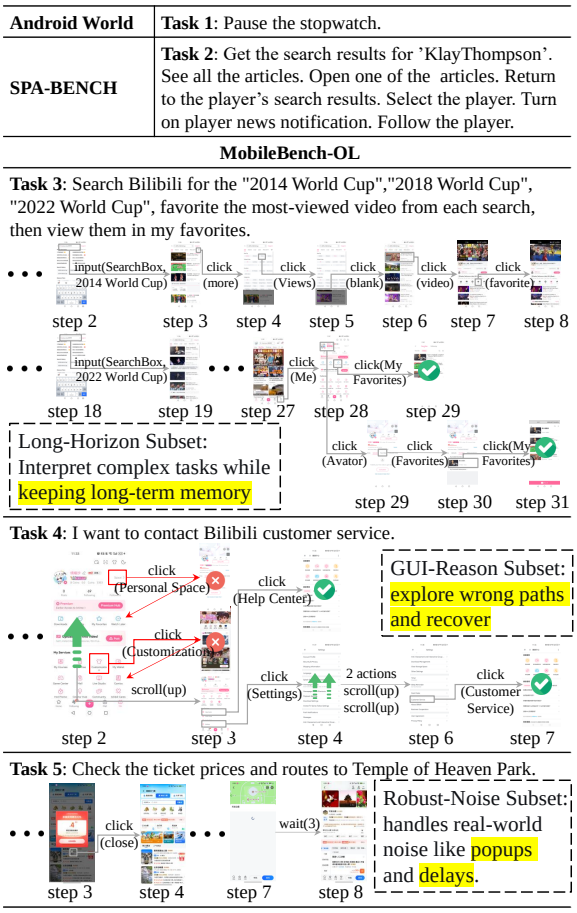


Figure 1: In addition to instruction-following ability, MobileBench-OL also measures long-horizon reasoning, exploration, and real-world noise handling.

1 Introduction

In recent years, the research community has shown growing interest in developing mobile GUI agents (Hong et al., 2023; Wu et al., 2025b), which have demonstrated remarkable performance in controlling mobile devices. As GUI agents continue to emerge, the fair evaluation of their performance becomes essential, leading to the growing need for benchmarks. Previous offline benchmarks (Chen et al., 2024a; Rawles et al., 2023a; Li et al., 2024) assess agents in static GUI environments. Actions are generated from static screenshots and then compared against predefined standard answers. These methods ignore the impact of unexpected factors in real-world environments, such as pop-ups, and cannot handle scenarios where a single task may have multiple valid GUI trajectories. Therefore, some studies (Xie et al., 2024; Lee et al., 2024; Zhang et al., 2023b) have introduced online evaluation benchmarks that deploy apps in device emulators to simulate real-world environments. They use device state (Rawles et al., 2024) or LLM

[*] Equal contribution.

[†] Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 13659–13693

Benchmark	Tasks	Apps	Dynamic	Third-party Apps	Chinese Apps	Difficulty Levels	Long-Horizon	GUI-Reasoning	Noise-Robust	Reset
AITW (Rawles et al., 2023a)	30378	357	✗	✓	✗	✗	✗	✗	✗	✗
AITZ (Zhang et al., 2024a)	2504	70	✗	✓	✗	✗	✗	✗	✗	✗
AndroidControl (Li et al., 2024)	15283	833	✗	✓	✗	✗	✗	✗	✗	✗
MobileAgentBench (Wang et al., 2024b)	100	10	✗	✗	✗	✓	✗	✗	✗	✗
AppAgent (Zhang et al., 2023a)	50	10	✗	✓	✗	✗	✗	✗	✗	✗
LearnGUI-Offline (Liu et al., 2025a)	2253	73	✗	✓	✗	✗	✗	✗	✗	✗
GUI-Robust (Yang et al., 2025)	5318	392	✗	✓	✓	✗	✗	✗	✓	✗
AndroidArena (Xing et al., 2024)	221	4	✓	✗	✗	✗	✗	✗	✗	✗
LLamaTouch (Zhang et al., 2024b)	496	57	✓	✓	✗	✗	✗	✗	✗	✗
AndroidWorld (Rawles et al., 2024)	116	20	✓	✓	✗	✓	✗	✗	✗	✗
AndroidLab (Xu et al., 2024)	138	9	✓	✓	✗	✗	✗	✗	✗	✗
A3 (Chai et al., 2025)	201	20	✓	✓	✗	✓	✗	✗	✗	✗
B-Moca (Lee et al., 2024)	131	15	✓	✗	✗	✗	✗	✗	✗	✗
LearnGUI-Online (Liu et al., 2025a)	101	23	✓	✗	✗	✗	✗	✗	✗	✗
D-GARA (Chen et al., 2025)	152	7	✓	✓	✓	✗	✗	✗	✓	✗
SPA-Bench (Chen et al., 2024b)	340	66	✓	✓	✓	✓	✓	✗	✗	✗
MobileBench-OL	1080	80	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of different datasets and environments for benchmarking Mobile GUI agents.

scores (Sun et al., 2025) to evaluate the generated GUI trajectories, offering a more accurate measure of performance in dynamic environments.

Despite significant progress, previous online benchmarks still have several limitations: First, most benchmarks focus on simple, atomic tasks or rigid, step-by-step instructions. They do not assess an agent's ability to reason and explore autonomously in dynamic environments. In contrast, a comprehensive benchmark should include complex, open-ended tasks that require long-term planning and GUI reasoning. Long-horizon tasks like "sequentially search for and favorite popular videos related to three World Cups" measure the ability to maintain focus on a high-level goal while executing multiple subtasks, allowing for diverse solution paths. Meanwhile, tasks like "contact customer service" examine GUI reasoning abilities. Since customer service functionality is hidden deep within the interface, the agent needs to first explore semantically related areas (e.g., Personal Space and Customization), then recover from incorrect paths. Beyond simply following instructions, such a benchmark also tests an agent's ability to interpret tasks and autonomously explore interfaces to complete them. Furthermore, existing benchmarks overlook the unpredictability of real-world mobile environments, such as unexpected pop-ups, interface lag, or operation errors. In Figure 1, an ad pop-up requires clicking accurately on the close icon to proceed, and a network delay page requires agent to wait a few seconds. Current benchmarks fail to measure how well agents handle such real-world variability, creating a gap between academic evaluation and real-world deployment.

To address these limitations, we propose MobileBench-OL, an online benchmark with 1080 real-world GUI tasks from 80 Chinese apps. It evaluates agents across three core dimensions via five subsets: (1) Base Capabilities, the Base and Long-Tail subsets; (2) Complex Reasoning, Long-Horizon ($\geq$20 steps) and GUI-Reasoning (requires exploration) subsets; (3) Robustness, Noise-Robust subset (4 artificial noises). We also introduce an automated evaluation (Auto-Eval) framework that assesses trajectories without manual intervention or system data. A fine-grained, instruction-based Reset Mechanism returns devices to their initial state, ensuring stability and repeatability.

Our contributions can be summarized as follows:

- We propose MobileBench-OL, a comprehensive benchmark of 1080 real-world tasks across 80 apps. It includes five subsets and evaluates multiple dimensions, such as basic capabilities, complex reasoning, and robustness, to thoroughly assess agent performance in real-world environments.
- We propose an Auto-Eval framework with a Reset Mechanism that ensures stable and reproducible task generation and execution.
- Evaluations of 12 leading GUI Agents reveal significant room for improvement in real-world performance. Human evaluation further confirms the benchmark provides stable and reliable measurements.

2 Related Work

Table 1 shows the comparison of MobileBench-OL with other benchmarks. With the rapid ad-

Subset	Task Examples	Golden Steps	Functional Point	Weight	Difficulty	Challenges
Base	Search for Pokemon on Bilibili.	4	Search	-	Easy (step:4)	Covering vairous functional points
Long-Tail	Set my main team to Premier League‘s Arsenal in Dongqiudi.	8	Personal Center	-	Medium (step:8)	Basic tasks with novel UIs in long-tail apps.
Long-Horizon	Open Tomato Novel, search for "War and Peace","The Lady of the Camellias","The Ordinary World","Tallow Ball" in order, follow the authors of these four books.	31	Search	-	Hard (step:31)	Hard tasks require more than 20 steps to complete.
GUI-Reasoning	Navigate to Toutiao’s scan function.	6	Channel	0.5	Easy (weight:0.5)	Tasks require exploration and reasoning, e.g. Icon Understanding, Hidden Function Discovery.
	I want to contact Bilibili customer service.	4	Settings	3	Hard (weight:3)	
	Set a 15-minute turn-off timer in Bilibili.	11	Settings	4	Hard (weight:4)	
Noise-Robust	Go to DingTalk and send a picture from the album to the DingTalk assistant.	9	Message	-	Medium (step:9)	Randomly introduce noise page during inference, e.g. pop-ups.

Table 2: Example tasks for the five subsets. Note that Base, Long-Tail, Long-Horizon, and Noise-Robust subsets use the number of golden steps to classify difficulty levels, while the GUI-Reasoning subset uses exploration weight.

vancement of GUI Agents, traditional static benchmarks (Wu et al., 2024a; Zhan and Zhang, 2023; Deng et al., 2023) have become inadequate. Recently proposed dynamic environments such as AndroidWorld (Rawles et al., 2024), LlamaTouch (Zhang et al., 2024b), and Mobile-Env (Zhang et al., 2023b) evaluate task completion via device state or LLM judgments rather than verifying trajectory–goal consistency. However, many benchmarks rely on offline, static apps that differ from mainstream designs and therefore do not represent real-world usage (Liu et al., 2025b). While SPA-Bench (Chen et al., 2024b) incorporates popular open-source apps, it cannot support apps and tasks that require account login or have high memory usage. Tasks in existing benchmarks are either too simple or overly rigid, neglecting the reasoning and exploration abilities in dynamic environments. Regarding real-world anomalies, GUI-Robust (Yang et al., 2025) proposes a static benchmark with seven anomaly scenarios. D-GARA (Chen et al., 2025) introduces a dynamic benchmark in which anomaly interruptions are manually configured per specific task. We categorize four common noise types and randomly introduce them during inference, thereby evaluating the agent’s robustness against unpredictable environmental noise. See Appendix F for more related works.

3 MobileBench-OL

3.1 Apps Selection

To ensure authenticity and comprehensiveness, we divided apps into 12 categories based on the App Store’s classification. For the four subsets excluding the Long-Tail, we selected the most popular app from each category. For the Long-Tail subset, we selected 5–6 apps per category with diverse core functions, totaling 68 apps. All categories and icons are listed in Figure 6 and Table 14. Unlike earlier studies, we did not exclude apps that require account login or high memory usage, as our benchmarks were run on real devices.

3.2 Benchmark Subsets

3.2.1 Base Subset and Long-Tail Subset

The Base and Long-Tail Subsets assess the agent’s basic abilities. The Base Subset covers varied difficulty and function points from popular apps, while the Long-Tail Subset focuses on 68 less common apps. Figures 8 and 9 in the Appendix show two example tasks. They both involve finding a product in a shopping app and navigating to reviews or the store homepage. Though the high-level steps are similar, their UI designs and app-specific elements differ significantly. Thus, the Long-Tail Subset evaluates whether the agent can apply general skills to novel interfaces, which tests its true robustness and adaptability.

Difficulty Levels Based on Golden Steps: The difficulty of four subsets (excluding GUI-Reasoning) is defined by golden steps, as in Table 2: Easy (<8 steps), Medium (8–19 steps), Hard ($\geq$ 20 steps). Golden steps reflect the minimal GUI pages a human needs to complete the task. Base, Long-Tail, and Noise-Robust subsets contain Easy or Medium tasks; Long-Horizon tasks are all Hard.

Functional Point Coverage: In Base subset, the tasks cover all of the app’s key functions. In Figure 2, while Bilibili’s primary function is keyword-based video "search," secondary functions like Channels, Personal Center and Settings are also essential. Including these secondary points allows a more comprehensive evaluation of the GUI agent.

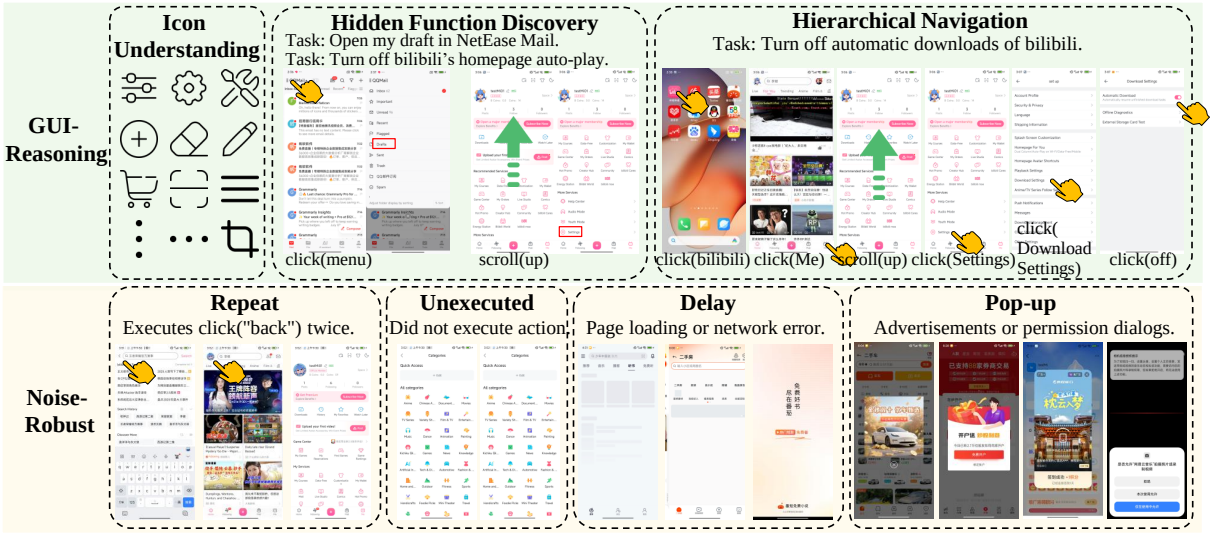


Figure 2: Examples of three exploration abilities (above) and four noise types (bottom).

More details are provided in Appendix B.1.1.

3.2.2 Long-Horizon Subset

The Long-Horizon subset tests whether an agent can maintain a high-level goal, correctly decompose it into subtasks, and execute them sequentially without losing track. Each task requires at least 20 steps. As shown in Figures 11 and 12, tasks such as instant delivery (filling in multiple recipient and sender fields) or adjusting font size (repeatedly clicking the reduce button) evaluate the agent's ability to manage, sequence, and precisely control subtasks while tracking their completion status.

3.2.3 GUI-Reasoning Subset

The GUI-Reasoning subset replicates real-world scenarios requiring active exploration and complex reasoning. Instead of just interacting with obvious elements, tasks involve interpreting visual context, learning through trial and error, and recovering from dead ends. These tasks assess three advanced exploration abilities, as shown in Figure 2:

• Icon Understanding: Interpreting visual elements without text labels (weight: 0.5).

• Hidden Function Discovery: Finding non-visible features, like "Draft" and "Settings" hidden in a submenu or an off-screen area (weight: 1).

• Hierarchical Navigation: Efficiently navigating deep menu structures to find a specific function, e.g., "Auto Downloads" via "Settings → Download Settings → Auto Downloads" (weight:2). Hidden Function Discovery is page-level, while Hierarchical Navigation involves app-level menu navigation.

Difficulty Levels Based on Exploration Weight: Each exploration ability has an assigned weight. Task difficulty is calculated by summing the required ability weights: Easy (weight$\leq$1), Medium (1<weight$\leq$2), and Hard (weight>2).

Table 2 shows several examples. An easy task, like navigating to a scan function (weight: 0.5), requires only recognizing the Scan icon. A hard task like contacting customer service (weight: 3) has fewer golden steps but requires more complex exploration, including understanding the app layout and scrolling to find the customer service function. Similarly, setting a timer (weight: 4) demands navigating hierarchical menus and locating two hidden functions, Settings and Timer.

3.2.4 Noise-Robust Subset

In real-world environments, app actions can lead to unexpected GUI states. The Noise-Robust subset tests an agent's ability to autonomously recover from disruptions. Following Yang et al. (2025), we categorize noise into four types (see Figure 2):

• Repeat: The action executes multiple times, often due to insufficient waiting time after execution.

• Unexecuted: The agent generates an action, but the system does not execute this action.

• Delay: The GUI page remains in a prolonged loading state, blocking access to updated elements.

• Pop-up: An unexpected dialog interrupts the workflow (e.g., a permission dialog or an ad).

Noise Injection: We use Base Subset tasks as seeds and simulate noise during trajectory generation. At each step, one of four noise types occurs with a 20% probability. For Repeat and Unexecuted noise, the action is either repeated or skipped. For Delay

Tasks	Task Success Conditions
Open Bilibili and change my profile avatar to a random one.	`1.//*[(@text="Avatar" or @text="Change Avatar") and bbox_contains_point(../@bounds, $point)]` `2.//*[@text="Shuffle"and bbox_contains_point(../@bounds,$point) and contains(@package,"bili")]`
Find the subway route from South Railway Station to Fengtai Station on Amap.	`//*[contains(@text,"Public Transportation") and @selected="true" and contains(@package,"map")]` `and //*[contains(@text,"South Railway Station") and contains(@resource-id, "summary_start")]` `and //*[contains(@text,"Fengtai Station") and contains(@resource-id,"summary_end")].`

Table 3: Example tasks and their corresponding success conditions.

Subset	Base	Long-Tail	Long-Horizon	GUI-Reasoning	Noise-Robust
Tasks	310	340	60	60	310
Avg Steps	5.61	5.38	22.73	6.22	5.61
Easy	196	269	-	13	196
Medium	114	80	-	24	114
Hard	-	-	60	23	-
Tasks Per App	20-30	5	5	5	20-30
Apps	12	68	12	12	12
Apps Per Category	1	5-6	1	1	1
Functional Points	28	11	8	18	28

Table 4: Dataset statistics of MobileBench-OL.

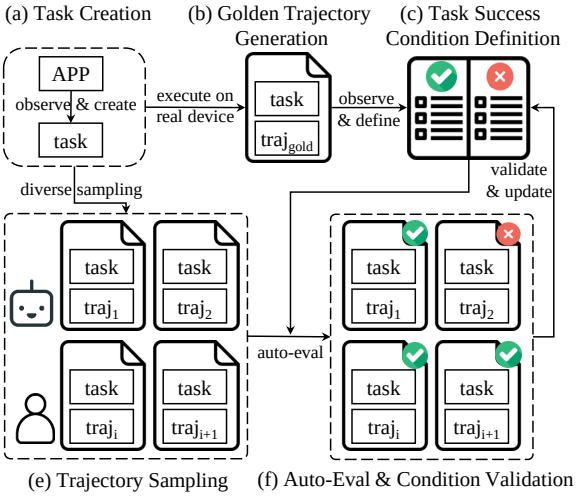


Figure 3: Data construction pipeline.

and Pop-up noise, we predefine five GUI pages per app and randomly select one to display. For Delay noise, if the agent acts instead of waiting, the action executes on the next page. For Pop-up noise, the GUI remains blocked by the pop-up until the correct close element is clicked. Further details are provided in Appendix B.1.4.

3.3 Data Construction

Table 4 shows the statistics for MobileBench-OL. See Appendix B.3 for more details.

Data construction pipeline: Tasks are manually created from scratch to avoid LLM data contamination. The pipeline (Figure 3) includes five steps: (1) Annotators design and run tasks on real devices to obtain golden trajectories. (2) Domain experts define task success conditions based on the task-trajectory pairs. (3) UI-TARS-1.5 and annotators sample multiple trajectories for diverse completion scenarios. (4) An auto-eval framework assesses trajectories against the success conditions. (5) Experts verify the auto-eval results, updating the task success conditions based on any discrepancies. Each task has $\geq$5 samples for reliability.

Annotation of Task Success Conditions: Task success in MobileBench-OL is determined by rule-based conditions using UI elements and actions observed only in successful trajectories. These conditions ensure: (1) *Completeness*: every successful trajectory meets the condition; (2) *Soundness*: no failed trajectory fully meets it. This design can match various valid trajectories. For example, task "Change to a random Bilibili avatar" is satisfied simply by clicking the "Change Avatar" and then the "Shuffle", regardless of the navigation path.

Standardized Implementation: For scalability, conditions are uniformly defined using XPath-like rules based on GUI attributes like text, resource-id, and package (see Appendix C). In Table 3, one sub-condition uses `contains(@package,"bili")` and `text="Avatar"` to locate the Avatar button, while `bbox_contains_point` confirms the interaction occurred within its bounds. Another sub-condition locates an element with `text="Shuffle"`. Combining these sub-conditions allows for reliable identification of successful trajectories.

4 Auto-Eval Pipeline

Auto-Eval Framework: The Auto-Eval framework consists of GUI Trajectory Generation and Auto Evaluation, as shown in Figure 4. The single-step trajectory loop iteratively (1) observes the device state and interaction history to form multimodal input, (2) generates actions from this input, (3) translates actions into a unified format and executes them to update the device. This loop repeats until the agent triggers a complete action or the trajectory reaches the max length limit.

Once the trajectory ends, the framework evaluates it against the task success conditions and

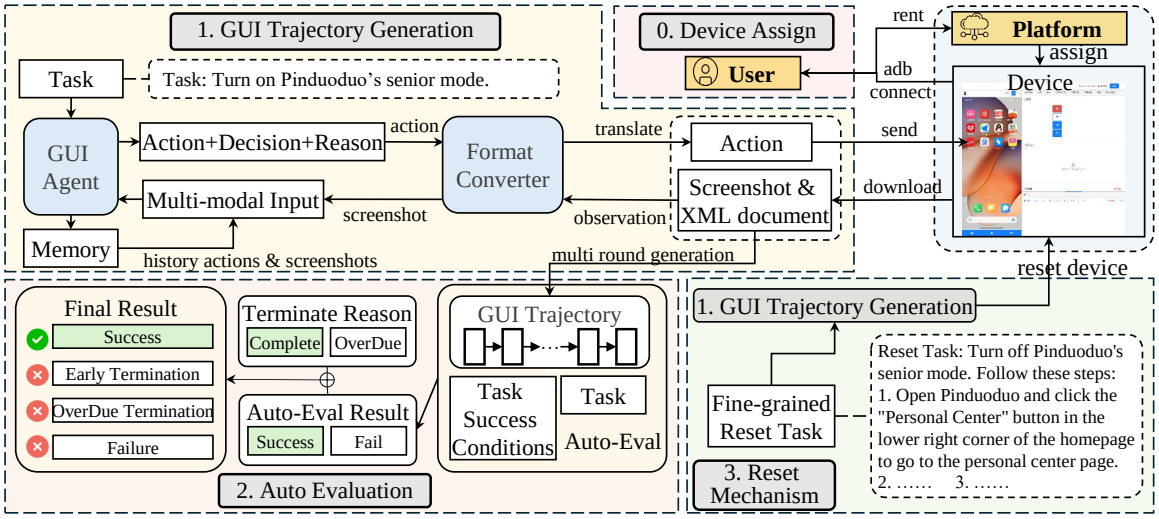


Figure 4: Pipeline of the Auto-Eval framework.

generates an auto-eval result: A trajectory meeting all conditions and ending with a "Complete" action is marked Success. If it meets all conditions but reaches the step limit, it is Overdue Termination. If it ends with a "Complete" action but does not meet all conditions, it is Early Termination. Otherwise, the trajectory is marked Failure.

Reset Mechanism: In real-world environments, device states constantly change during task execution, and some tasks cause persistent changes that can affect later evaluations. For example, "Open Senior Mode" changes the UI style permanently, while "Setting Company Address" allows users to skip future reconfigurations after one successful set. To address this, we propose a reset mechanism that uses a reliable agent to execute fine-grained inverse tasks. As shown in Figure 4, the inverse task of "Turn on Pinduoduo's senior mode" is defined as a step-by-step fine-grained description of how to "turn off Pinduoduo's senior mode", including the shape and position of interactive elements.

Reset tasks fall into four categories: (1) Task-level reset: Adjusting app state or user data for a single task. (2) App-level reset: A shared reset (e.g., Clear Shopping Cart) applicable to multiple tasks in an app. (3) No reset needed: Tasks that only require viewing or navigation. (4) Infeasible: Tasks with server interaction or irreversible actions. For infeasible tasks, we expand task success conditions to ensure reliable evaluation. For example, a sign-in task succeeds if either the "Sign In" button is clicked or a "Signed in today" element appears.

After each round of benchmark evaluation, 255 task-level and app-level reset tasks are executed to restore the device state. Experiments have shown that the reset mechanism has a success rate exceeding 90%. More details refer to Appendix D.

5 Experiment

5.1 Experiments Setting

We define four key metrics for agent performance: (1) Success Rate (SR): The primary metric, indicating a task is successfully completed and ends with a "Complete" action. (2) Sub-condition Success Rate (Sub-SR): Measures the ratio of completed sub-conditions, regardless of how the task ended. (3) Step Ratio: Compares the agent's steps to the ideal human golden steps, with a max limit of three times the golden steps. (4) Failure Reasons: Categorized into three types: Early Termination, Overdue Termination, and Failure.

We test 12 GUI agents: 4 closed-source (GPT-4o (Yan et al., 2023), M3A (Rawles et al., 2024), T3A (Rawles et al., 2024), Mobile-Agent-V2 (Wang et al., 2024a)) and 8 open-source (InternVL2 (Wang et al., 2024e), CogAgent (Hong et al., 2023), UGround-V1 (Gou et al., 2025), OS-Atlas (Wu et al., 2024b), Qwen2-VL (Wang et al., 2024c), Qwen2.5-VL (Bai et al., 2025b), UI-TARS (Qin et al., 2025), and UI-TARS-1.5 (Qin et al., 2025)). Besides, we have supplemented the evaluation results with newer agents such as Qwen3-VL (Bai et al., 2025a), Mobile-Agent-V3 (Ye et al., 2025) (based on GUI-OWL-8B), and MAI-UI (Zhou et al., 2025). Most open-source models are of comparable capacities(7B,8B,9B). The step limit is set to three times the golden steps, with a 3-second wait after each action. Details of metrics and implementation refer to Appendix G.

Model	Base		Long-Tail		Long-Horizon		GUI-Reasoning		Noise-Robust	
	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)
GPT-4o	30.32%	34.75%	22.35%	24.95%	0.00%	7.27%	21.67%	25.67%	27.10%	31.10%
T3A	21.61%	23.74%	8.24%	9.56%	1.67%	9.84%	13.33%	16.67%	5.81%	9.87%
M3A	40.65%	43.88%	25.88%	28.09%	3.33%	18.12%	25.00%	33.06%	16.45%	23.47%
Mobile-Agent-V2	46.45%	48.66%	33.53%	35.88%	3.33%	19.26%	41.67%	44.17%	37.42%	38.23%
InternVL2-8B	0.32%	0.59%	0.29%	0.92%	0.00%	0.94%	0.00%	0.00%	0.00%	0.43%
CogAgent-9B	4.19%	4.30%	1.47%	2.21%	0.00%	2.25%	0.00%	0.00%	2.26%	2.53%
UGround-V1-7B	0.97%	1.77%	0.88%	2.12%	0.00%	3.44%	0.00%	0.83%	0.00%	1.41%
OS-Atlas-Pro-7B	2.58%	13.33%	0.00%	11.76%	0.00%	8.64%	1.67%	7.83%	0.65%	8.76%
Qwen2-VL-7B	10.32%	19.46%	5.29%	17.28%	0.00%	9.50%	0.00%	1.67%	4.84%	16.24%
Qwen2.5-VL-7B	31.61%	37.63%	25.88%	27.65%	0.00%	16.30%	16.67%	23.44%	30.97%	35.01%
UI-TARS-7B	46.77%	57.06%	32.94%	41.32%	1.67%	19.74%	23.33%	33.17%	40.97%	49.06%
UI-TARS-1.5-7B	**60.97%**	**64.84%**	**41.76%**	**46.32%**	**15.00%**	**45.25%**	**38.33%**	**40.83%**	**56.77%**	**62.18%**

Table 5: The overall performance of GUI agents on MobileBench-OL.

Model	Base		Long-Horizon		GUI-Reasoning		Noise-Robust	
	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)	SR (↑)	Sub SR (↑)
UI-TARS-1.5-7B	60.97%	64.84%	15.00%	45.25%	38.33%	40.83%	56.77%	62.18%
Mobile-Agent-V2	46.45%	48.66%	3.33%	19.26%	41.67%	44.17%	37.42%	38.23%
Mobile-Agent-V3	49.03%	53.99%	0.00%	24.36%	35.00%	40.33%	38.71%	42.77%
MAI-UI-8B	65.16%	72.42%	8.33%	39.66%	43.33%	52.22%	53.57%	64.20%
Qwen3-VL-8B	63.23%	67.39%	11.67%	35.25%	36.67%	48.33%	50.32%	58.79%
Qwen3-VL-32B	72.58%	77.42%	20.00%	48.75%	43.33%	51.67%	66.13%	69.97%

Table 6: The overall performance of recent GUI agents on MobileBench-OL.

5.2 Main Results

Table 5 summarizes the performance of all GUI Agents on MobileBench-OL. We can see that:

• UI-TARS-1.5-7B achieved the highest SR and Sub-SR across all five subsets.

• Performance is the lowest in the Long-Horizon subset due to its comprehensive assessment of task interpretation, decomposition, multi-step execution, and long-term memory.

• Results drop in Long-Tail and GUI-Reasoning subsets because agents struggle with uncommon app structures and lack strong exploration abilities.

• Performance also drops in the Noise-Robust subset, as most agents are unfamiliar with handling interface noise, such as unexpected pop-ups.

• Agents like UGround and InternVL2 have very low SR (<5%), likely due to poor grounding. They can generate appropriate intents during execution, but often fail to produce the correct actions.

• Agents that accept multiple images as input (e.g., GPT-4o) achieve higher SR. This allows them to observe state changes caused by previous actions, thus aiding in error correction and loop recovery.

• Qwen2-VL-7B shows a large gap between its SR and Sub-SR because it often fails to understand when to terminate a task.

• The Sub-SR metric closely aligns with the SR in most subsets except Long-Horizon. This is because Long-Horizon tasks are complex and have many completion conditions, making it difficult for the agent to finish the entire task even when some sub-tasks are completed.

With the rapid development of GUI agents and foundation models, more capable agents featuring stronger basic capabilities and better reasoning have emerged. After supplementing the evaluation with newer agents (Qwen3-VL, Mobile-Agent-V3, and MAI-UI), Table 6 reveals additional findings:

• At similar parameter sizes, performance ranking is roughly: MAI-UI-8B > Qwen3-VL-8B > Mobile-Agent-V3 (based on GUI-OWL-8B). MAI-UI-8B matches Qwen3-VL-32B. Mobile-Agent-V3 performs weakly because GUI-OWL is misaligned with our benchmark. It prioritizes pressing the home button even when the target app is already open, and fails to swipe when the app is not visible.

• Improvements are most evident on the Base subset. Compared to UI-TARS-1.5, newer agents show stronger grounding and task understanding.

• For more challenging reasoning tasks, improvements remain limited. Compared to UI-TARS-1.5-7B, other agents with similar parameter sizes exhibit lower SR and Sub-SR on the Long-Horizon subset, with SR lower by 4–7%. This indicates that their long-range task capabilities still lag behind.

	Base			Long-Tail			Long-Horizon			GUI-Reasoning			Noise-Robust		
	Early	Overdue	Failure	Early	Overdue	Failure	Early	Overdue	Failure	Early	Overdue	Failure	Early	Overdue	Failure
M3A	31.29%	1.61%	26.45%	41.76%	0.59%	31.76%	50.00%	0.00%	46.67%	35.00%	5.00%	35.00%	14.84%	5.48%	63.23%
T3A	73.87%	0.00%	4.52%	86.76%	0.00%	5.00%	95.00%	0.00%	3.33%	85.00%	0.00%	1.67%	37.74%	2.90%	53.55%
Mobile-Agent-V2	53.55%	0.00%	0.00%	66.47%	0.00%	0.00%	96.67%	0.00%	0.00%	58.33%	0.00%	0.00%	62.58%	0.00%	0.00%
Qwen2.5-VL	39.68%	2.90%	25.81%	31.18%	0.59%	42.35%	28.33%	0.00%	71.67%	45.00%	3.33%	35.00%	40.00%	1.94%	27.10%
UI-TARS	18.39%	7.42%	27.42%	25.59%	6.47%	35.00%	13.33%	3.33%	81.67%	23.33%	5.00%	48.33%	21.29%	6.77%	30.97%
UI-TARS-1.5	17.42%	2.26%	19.35%	21.76%	2.65%	33.82%	25.00%	3.33%	56.67%	21.67%	0.00%	40.00%	19.35%	3.55%	20.32%

Table 7: Statistics of failure reasons (Early Termination/Overdue Termination/Failure) on all 1080 tasks.

Long-Horizon	Total Tasks	Failed Tasks
SR Tasks	15.0%	–
Reasoning & Planning Failure	15.0%	17.6%
Subtask Omission	6.7%	7.8%
Function Navigation Failure	16.7%	19.6%
Attribute Omission/Error	25.0%	29.4%
Visual Grounding Failure	21.7%	25.5%

Table 8: Error Distribution of UI-TARS-1.5 across 60 tasks in Long-Horizon Subset.

GUI-Reasoning	Easy	Medium	Hard
M3A	38.46%	37.50%	4.35%
T3A	23.08%	16.67%	4.35%
Mobile-Agent-V2	53.85%	50.00%	26.09%
Qwen2.5-VL	30.77%	16.67%	8.70%
UI-TARS	30.77%	25.00%	17.39%
UI-TARS-1.5	61.54%	41.67%	21.74%

Table 9: Success Rate across GUI-Reasoning difficulty.

Noise-Robust	Repeat	Unexecuted	Delay	Pop-Up
M3A	22.78%	16.67%	22.37%	3.90%
T3A	10.13%	3.85%	3.95%	5.19%
Mobile-Agent-V2	49.37%	43.59%	40.79%	15.58%
Qwen2.5-VL	32.91%	37.18%	25.00%	28.57%
UI-TARS	43.04%	37.18%	46.05%	37.66%
UI-TARS-1.5	60.76%	60.26%	60.53%	41.56%

Table 10: Success Rate across different noise types.

• On noise-robust tasks, perform comparably to or even slightly worse than UI-TARS, indicating that robustness to real-world noise remains a bottleneck.

• SR improvements are small, but Sub-SR improvements are substantial. For example, MAI-UI-8B vs UI-TARS-1.5-7B shows only +0.27% overall SR but +4.49% overall Sub-SR, as difficult full-task completion remains unsolved while simpler subtasks are handled better.

• Qwen3-VL-32B significantly outperforms Qwen3-VL-8B, especially on Base and Noise-Robust subsets, but less so on Long-Horizon and GUI-Reasoning. This shows parameter scaling mainly benefits basic capabilities, not complex reasoning or planning.

5.3 Ablation Study

Long-Horizon Error Analysis Table 8 classifies errors in the Long-Horizon subset into four categories. Attribute errors (25.0%) were most frequent, showing difficulty in tracking attributes over long-term trajectories. The other three categories, which relate to the agent's grounding, task intent understanding, and function navigation abilities, also need improvement. Subtask omissions (6.7%) were least common but highlight the risk of missing subtasks steps in complex tasks. See Appendix H.1 for detailed definitions and examples.

GUI-Reasoning Difficulty Analysis Table 9 shows that SR decreases as exploration difficulty increases in GUI-Reasoning subset. The agent performs well on Easy tasks, which only require simple icon recognition or finding one hidden function. Performance drops sharply on Hard tasks, which require locating multiple hidden functions or deeper navigation.

Noise Type Analysis Table 10 shows that Pop-Up noise has the strongest negative effect in the Noise-Robust subset. While other noises mainly test the ability to handle unexpected action results or recover from an error state, Pop-Up noise specifically requires understanding pop-up windows and accurately locating the close icon. Many agents fail to close the pop-up window, causing them to become stuck on the pop-up page, terminating the task early, or exceeding step limits.

Failure Reasons Table 7 summarizes failure reasons for top-performing GUI agents. Failure is most common, followed by Early Termination, where the agent misinterprets the task or stops prematurely. Overdue Termination occurs less frequent, typically when the agent fails to recognize task completion and enters an action loop. Closed-source agents like Mobile-Agent-V2 have a higher rate of Early Termination, indicating a more aggressive stopping threshold and higher confidence.

Difficulty Analysis Table 11 shows SR across difficulty levels based on golden steps. Performance declines with increasing task length in all

	Base		Long-Tail		Long-Horizon	Noise-Robust	
	Easy	Medium	Easy	Medium	Hard	Easy	Medium
M3A	46.94%	29.82%	28.08%	18.75%	-	19.90%	10.53%
T3A	26.53%	13.16%	9.23%	5.00%	-	8.16%	1.75%
Mobile-Agent-V2	53.06%	35.09%	37.31%	21.25%	-	43.88%	26.32%
Qwen2.5-VL	42.35%	13.16%	31.54%	7.50%	-	41.84%	12.28%
UI-TARS	52.55%	36.84%	36.15%	22.50%	1.67%	44.39%	35.09%
UI-TARS-1.5	65.31%	53.51%	46.15%	27.50%	15.00%	62.76%	46.49%

Table 11: SR across different Length Difficulty Levels.

	Num				Success Rate		Auto-Eval
	TP	FP	FN	TN	Auto-Eval	Human	Correct
Base	185	3	12	110	60.97%	63.54%	95.16%
Long-Tail	145	0	5	190	41.76%	44.12%	98.53%
Long-Horizon	9	0	0	51	15.00%	15.00%	100.00%
GUI-Reasoning	22	1	2	35	38.33%	40.00%	95.00%
Noise-Robust	173	1	3	133	56.77%	56.77%	98.71%
Overall	534	5	22	519	49.91%	51.48%	97.50%

Table 12: Human evaluation of Auto-Eval results. TP/FP/FN/TN follow the confusion matrix definition. Last column shows Auto-Eval's accuracy relative to human judgment.

	Base	Long-Tail	GUI-Reasoning	Noise-Robust
Reset Tasks	65	102	23	65
Auto-Eval SR	95.38%	92.86%	91.30%	95.38%
Human Eval SR	96.92%	94.35%	95.65%	92.31%

Table 13: Human evaluation of Reset Mechanism.

four subsets, especially in the Long-Horizon tasks, as the agent struggled to complete these complex multi-subtask scenarios. Overall, MobileBench-OL presents a considerable challenge and leaves room for further improvement.

Human evaluation of Auto-Eval Framework To verify the effectiveness of Auto-Eval Framework, we conducted a human evaluation of UI-TARS-1.5 results, as shown in Table 12. False Negatives (FN) show it sometimes classifies correct GUI trajectories as fail, while False Positives (FP) confirm that Auto-Eval rarely mislabels incorrect trajectories as successful, proving its soundness. Overall, Auto-Eval achieves more than 95% accuracy, indicating that it can reliably label GUI trajectories.

Human evaluation of Reset Mechanism We also verified the Reset Mechanism's effectiveness via human evaluation. In Table 13, we executed 255 fine-grained reset tasks with UI-TARS-1.5 and evaluated the result using both Auto-Eval and human evaluation. Human evaluation show an accuracy of more than 90%, indicating the effectiveness of reset mechanism. In addition, Auto-Eval results is close to human evaluations, further verifying the reliability of Auto-Eval.

More Analyze We further present three supplementary experiments in Appendix H: a fine-grained error categorization on the Long-Horizon subset, step-ratio analysis, and pass@k evaluation. We further analyze the main causes of agent failures and provide corresponding improvement suggestions, as detailed in Appendix I.

6 Conclusion

This paper introduces MobileBench-OL, an online benchmark for evaluating GUI agents in real-world environments. It contains 1080 tasks from 80 Chinese apps, integrating 5 subsets that measure not only task execution but also complex reasoning, exploration ability, and robustness to noise. To ensure stable and reproducible evaluation, we provide an auto-eval framework with a remote mobile control platform and reset mechanism. Experimental results evaluating 12 leading agents show significant room for improvement to meet real-world requirements. Human evaluation further confirms the benchmark's stability and reliability.

MobileBench-OL assesses multiple critical dimensions for real-world agent performance. We hope that MobileBench-OL can serve as a real, fair and scalable benchmark, and contribute to future research in the community.

Limitations

While MobileBench-OL offers a comprehensive online benchmark, it still has some limitations that may be addressed in future updates. While the benchmark already includes long-horizon tasks requiring multiple subtasks and GUI-reasoning tasks requiring exploration, it currently lacks complex cross-app tasks. These would require the combined use of multiple apps to complete several subtasks. We plan to address these more complex scenarios in future updates.

Ethics Statement

This paper is conducted in accordance with the ACM Code of Ethics. We strictly filtered the benchmark, removing any data that could potentially expose personal privacy, thereby ensuring the highest level of protection for personal data. All data labeling was completed by crowdsourced workers, whom we paid at least $0.5 for each step and provided with necessary training. The human evaluation of our work was carefully conducted by IT experts. We ensured the reviewers had gender balance and diverse educational backgrounds, reflecting a wide range of perspectives and experiences.

References

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. 2025a. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025b. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Yue Cao, Yingyao Wang, Pi Bu, Jingxuan Xing, Wei Jiang, Zekun Zhu, Junpeng Ma, Sashuai Zhou, Tong Lu, Jun Song, et al. 2025. Androidlens: Long-latency evaluation with nested sub-targets for android gui agents. *arXiv preprint arXiv:2512.21302*.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. 2024. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149*.

Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Liuyi Chen, Yilin Bai, Zhigang He, Chenlong Wang, Huichi Zhou, Yiqiang Li, et al. 2024a. Gui-world: A video benchmark and dataset for multimodal gui-oriented understanding. *arXiv preprint arXiv:2406.10819*.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, et al. 2024b. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*.

Sen Chen, Tong Zhao, Yi Bin, Fei Ma, Wenqi Shao, and Zheng Wang. 2025. D-gara: A dynamic benchmarking framework for gui agent robustness in real-world anomalies. *arXiv preprint arXiv:2511.16590*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

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.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2023. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. 2025. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive, and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2024. Benchmarking mobile device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*.

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

Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, et al. 2025b. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. 2025c. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *Preprint*, arXiv:2501.04575.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv 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, et al. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

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

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

Jiahui Sun, Zhichao Hua, and Yubin Xia. 2025. Autoeval: A practical framework for autonomous evaluation of mobile agents. *arXiv preprint arXiv:2503.02403*.

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

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

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024c. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. 2024d. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *arXiv preprint arXiv:2410.14803*.

Weiyun Wang, Zhe Chen, Wenhai Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Jinguo Zhu, Xizhou Zhu, Lewei Lu, Yu Qiao, and Jifeng Dai. 2024e. Enhancing the reasoning ability of multimodal large language models via mixed preference optimization. *arXiv preprint arXiv:2411.10442*.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *Preprint*, arXiv:2501.11733.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025a. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. *arXiv preprint arXiv:2505.20660*.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2025b. Reachagent: Enhancing mobile agent via page reaching and page operation. *arXiv preprint arXiv:2502.02955*.

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

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. 2024b. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

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. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6061–6072.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2024. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *Preprint*, arXiv:2311.07562.

Jingqi Yang, Zhilong Song, Jiawei Chen, Mingli Song, Sheng Zhou, Xiaogang Ouyang, Chun Chen, Can Wang, et al. 2025. Gui-robust: A comprehensive dataset for testing gui agent robustness in real-world anomalies. *arXiv preprint arXiv:2506.14477*.

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

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *arXiv preprint arXiv:2404.05719*.

Zhuosheng Zhan and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023a. 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, et al. 2023b. Mobile-env: Building qualified evaluation benchmarks for llm-gui interaction. *arXiv preprint arXiv:2305.08144*.

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

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. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–13.

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

The appendix provides supplementary details to further describe the content of the main section. Section A introduces MobileBench-OL environment, including the running devices, the apps and APKs used, the defined action space, and the format of the environment data. Section B details all 5 subset, providing examples and statistics. This includes definitions of metrics such as feature coverage, exploration difficulty, noise classification, and noise injection methods. Section C provides an analysis of GUI trajectories and Task Success Conditions. Section D introduces the Reset Mechanism. Section E provides the prompts used for the different baselines. Section F provides more related works. Section G details the experimental setup, including the Evaluation Metrics, Implementation Details, and Baselines. Section H presents ablation experiments not included in the main text. Section I analyzes the main causes of agent failures and provides corresponding improvement suggestions.

A MobileBench-OL Environment

A.1 Real-World Environments

To ensure the benchmark runs in real-world environments, we use an online mobile control platform. We offer three physical smartphones connected to the platform, each pre-installed with 80 benchmark apps and logged in with real, verified user accounts to ensure authentic app behavior and network conditions. Researchers interact with these devices through our online control platform. The platform provides ADB access; the GUI Agent can send actions to the devices for execution, as well as download environmental data from them, with all operations conducted under real network conditions. In addition, researchers can monitor the device screens in real time through the platform.

Figure 5 shows the platform interface. The web interface allows real-time screen viewing, ADB connection via device IP for data download and command execution, and direct screen interaction through clicking and dragging.

Moreover, we will release all APK files, as shown in Figure 6 and Table 14, allowing users to reproduce the experimental benchmark environment on their own devices. This ensures the benchmark remains accessible beyond the lifecycle of any specific platform. Researchers can simply run the benchmark on their own mobile phones. All 80 benchmark APK files can be installed via ADB. While our platform uses verified accounts, logging in with any random account during local reproduction does not affect evaluation outcomes. This approach with physical devices is chosen because virtual emulators often fail to run real-world apps stably, support account login, and may crash frequently.

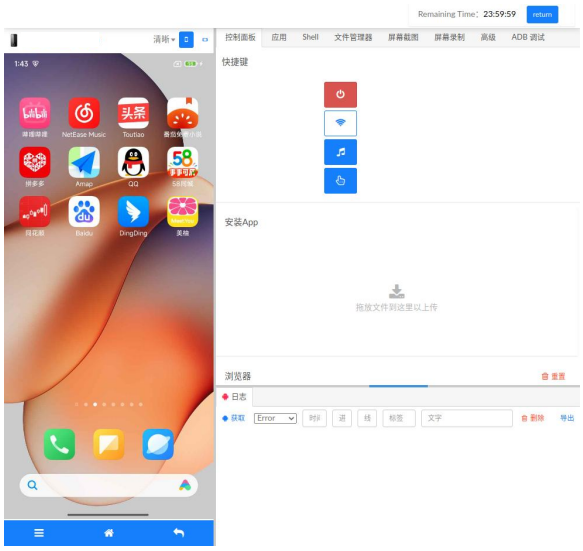


Figure 5: Online platform interface.

Figure 6: List of all app icons.

Base					
App Name	Category	APK	App Name	Category	APK
BiliBili	Video	bili_8.39.0.apk	NeteaseMusic	Music	netease_7.23.1.apk
ArticleNews	News	articlenews_11.6.0.apk	FanqieRead	Reading	fanqieread_6.6.7.32.apk
Pinduoduo	Shopping	pinduoduo_7.53.0.apk	Minimap	Transportation	minimap_15.11.1.2030.apk
QQ	Social	qq_9.1.91.apk	Wuba	Lifestyle	wuba_13.29.5.apk
Tonghuashun	Finance	tonghuashun_11.30.02.apk	Baidubrowser	Tools	baidubrowser_15.11.0.10.apk
DingDing	Office	dingding_7.6.55.apk	Seeyou	Sports	seeyou_8.87.1.0.apk
Long-Tail					
App Name	Category	APK	App Name	Category	APK
idxyer	Sports	idxyer_10.3.0.apk	keep	Sports	keep_8.5.30.apk
xiaoxun	Sports	xiaoxun_1.2.19.apk	dwbtime	Sports	dwbtime_11.6.8.apk
dongqiudi	Sports	dongqiudi_8.4.2.apk	hupu	Sports	hupu_8.0.50.apk
kugoumusic	Music	kugoumusic_20.2.2.apk	qtradio	Music	qtradio_10.8.7.apk
dreamina	Music	dreamina_1.5.8.apk	qqmusic	Music	qqmuisc_14.6.0.8.apk
migumusic	Music	migumusic_7.47.0.apk	theater	Video	theater_2.7.1.2.apk
haokan	Video	haokan_7.83.0.10.apk	huajiao	Video	huajiao_9.6.4.2030.apk
xiguavideo	Video	xiguavideo_9.6.4.apk	tencentvideo	Video	tencentvideo_9.01.60.30228.apk
yangcong345	News	yangcong_7.87.0.apk	huatu	News	huatu_7.4.350.apk
xunfeiai	News	xunfeiai_3.0.4.apk	jiakao	News	jiakao_8.80.0.apk
zuoyebang	News	zuoyebang_14.30.0.apk	wukong	News	wukong_13.0.0.apk
icredit	Office	icredit_19.2.0.apk	workschedule	Office	workschedule_2.0.91.apk
netease	Office	netease_7.23.1.apk	kdweibo	Office	kdweibo_10.8.12.apk
mobilemcloud	Office	mobilemcloud_12.1.0.apk	camscanner	Office	camscanner_6.91.5.apk
weread	Reading	weread_9.3.4.apk	dmzj	Reading	dmzj_3.9.14.apk
fcatfreader	Reading	fcatfreader_7.76.apk	jinjiang	Reading	jinjiang_6.6.7.apk
qidian	Reading	qidian_7.9.417.apk	mtxx	Reading	mtxx_11.12.0.apk
smart360	Tools	smart360_2.12.1.apk	zhejiang	Tools	zhejiang_7.26.0.apk
weitdy	Tools	weitdy_1.0.73.apk	zhipuai	Tools	zhipuai_3.1.3.apk
cbn	Tools	cbn_2.0.2.apk	newhope	Tools	newhope_4.0.6.apk
vipshop	Shopping	vipshop_9.53.7.apk	youpin	Shopping	youpin_5.31.0.apk
yonghui	Shopping	yonghui_11.6.0.1.apk	vmall	Shopping	vmall_1.25.5.300.apk
xiaomishop	Shopping	xiaomishop_5.41.0.apk	motorfans	Transportation	motorfans_3.65.00.apk
zeekrlife	Transportation	zeekrlife_4.9.11.apk	autohome	Transportation	autohome_11.75.5.apk
aima	Transportation	aima_5.3.2.apk	htinns	Transportation	htinns_9.34.0.apk
szzc	Transportation	szzc_9.2.0.apk	ivwen	Social	ivwen_11.0.6.apk
momo	Social	momo_9.17.6.apk	hsj	Social	hsj_2.9.8.apk
maimai	Social	maimai_6.6.72.apk	douban	Social	douban_7.104.0.apk
cc5	Lifestyle	cc5_10.15.0.apk	calendar	Lifestyle	calendar_7.2.6.apk
moviepro	Lifestyle	moviepro_8.8.4.apk	wsgw	Lifestyle	wsgw_3.1.6.apk
starbucks	Lifestyle	starbucks_10.11.0.apk	xiachufang	Lifestyle	xiachufang_8.8.65.apk
lottery	Finance	lottery_3.6.6.apk	miaomiao	Finance	miaomiao_4.1.7.apk
lifecircle	Finance	lifecircle_2.75.1.apk	cailianshe	Finance	cailianshe_8.6.2.apk
creditcard	Finance	creditcard_6.2.9.apk	gszq	Finance	gszq_9.09.000.apk

Table 14: The apks of 80 MobileBench-OL Apps.

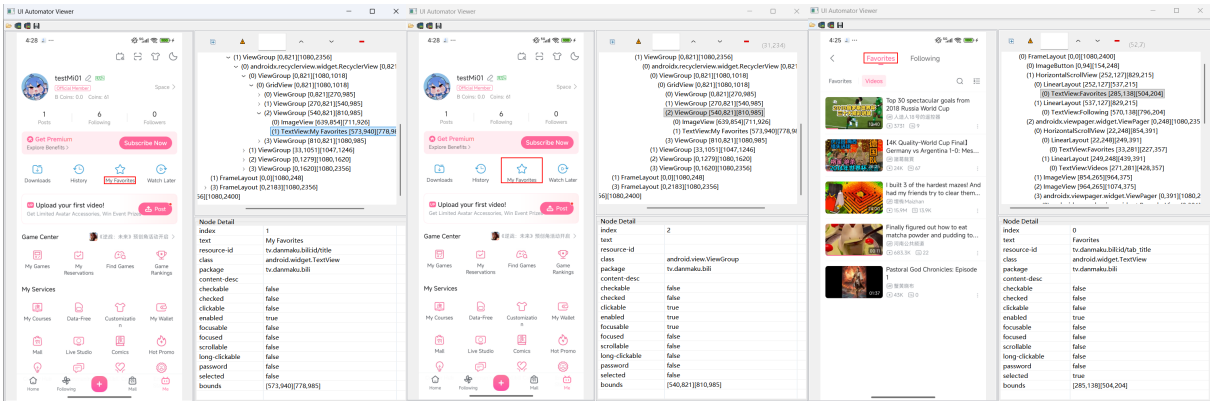
A.2 APP Selection

The distribution and categories of 80 apps are presented in Figure 6. Table 14 listed the APK version for all apps. Researchers can find the corresponding version in the app's version history on the App Store to replicate our test environment. They can also try installing the latest version of the corresponding app. Two months after building our benchmark environment, we tested the latest versions and found that UI changes caused fluctuations in agent performance, but the range of fluctuation did not exceed 5%. Therefore, if researchers wish only to compare the performance of two agents, they can use the latest version to test both simultaneously. However, to reproduce the exact results in the paper, we recommend using the corresponding version of the APK.

A.3 Action Space

Different GUI agents generate actions in different formats, which we will translate into a unified action format. Our action space includes:

```
click(point='<point>x1 y1</point>')
type(content='...')
scroll(point='<point>x1 y1</point>',
       direction='...')
press_home()
press_back()
```



(a) The TextView of "My Favorites" (b) The ViewGroup of "My Favorites" (c) The Page of "My Favorites"

Figure 7: Examples of XML files.

```
wait()
long_press(point='<point>x1 y1
        </point>')
finished(content='xxx')
```

A.4 Environmental Data

The GUI Agent gathers environmental data from mobile devices, which includes screenshots and corresponding XML files. This XML provides a structured hierarchy of the GUI elements, where each node corresponds precisely to a visual element in the screenshot and contains attributes such as text, resource-id, and class. As shown in the Figure 7, an element with the text `My Favorite`, its resource-id is `tv.danmaku.bili:id/title`, class is `android.widget.TextView`, content-desc is empty, package is `tv.danmaku.bili`, selected is `false`, clickable is `true`, bounding box is `[565,1056][786,1105]`.

When constructing the task success conditions, if we want to represent the "My Favorites" caption element, the importance of each attribute varies. For completeness and soundness, we need to ensure that the attribute does not change every time this element is selected; and that the attributes combined uniquely identify this specific element. Therefore, the text attribute is the most important; the resource-id and package attributes can only filter a subset of elements and are of less important; class, content-desc, and clickable are useless; bounds cannot be used because it cannot be guaranteed that "My Favorites" will be in the same position every time the user enters the profile page. Therefore, the condition format of this element can be:

- `@text="My Favorites" and @resource-id = "tv.danmaku.bili:id/title"`
- `contains(@text, "My Favorites") and contains(@resource-id, "id/title")`
- `contains(@text, "My Favorites") and @clickable="true" and contains(@package, "bili") and contains(@resource-id, "title")`

For a task such as "Navigate to My Favorites," there are intuitively two ways to define its success conditions: (1) Determine whether the My Favorites page appears in the trajectory, and (2) Determine whether the My Favorites button is clicked on pages such as the Personal Center or System Settings.

For identifying the My Favorites page, the method is to locate elements that are guaranteed to appear on that page. Use multiple elements to uniquely identify the page and avoid similar elements on other pages from triggering false positives. For example, if there is a `Favorites` element on the page with a resource-id of type `tab_title` and selected set to `true`, and a `Following` element with a resource-id of type `tab_title` and selected set to `false`, then we can consider this page to be the My Favorites page. The task success condition is:

- `//*[contains(@text, "Favorites") and @selected="true" and contains(@resource-id, "tab_title")] and //*[contains(@text, "Following") and @selected="false" and contains(@resource-id, "tab_title")]`

For detecting a click on the My Favorites button, the approach is to first identify the page preceding the My Favorites page, uniquely distinguishing it using the My Favorites button and other elements on that page. Then, add an action position con-

Subsets	Task Examples	Steps Difficulty	Exploration Difficulty
Base	Open Bilibili's animation channel to see what children's animations are available.	Easy	-
	On Bilibili, find the updates from the people I follow and like their posts.	Easy	-
	Check my pending payment orders on Bilibili's membership store.	Easy	-
	Find and follow singer Jay Chou's personal homepage on NetEase Cloud Music.	Medium	-
	Open NetEase Cloud Music and post a note saying "Good luck with exams."	Medium	-
Long-Tail	Check on Dingxiang Garden which medication should be used for allergic rhinitis.	Easy	-
	Look up the lottery results for traditional football lottery in China Sports Lottery.	Easy	-
	Turn on the "Honk to Find Car" feature in Aima.	Easy	-
	Set a shift alarm for the day shift at 19:00 in the scheduling calendar.	Medium	-
	Select the first two photos from the album and create a collage using the first template style in Meitu	Medium	-
Long-Horizon	In the Amap app, find the car rental service. Set the pick-up city to Qingdao, the pick-up and drop-off location to May Fourth Square, and the pick-up and return time from 10:00 to 16:00 tomorrow. Sort the available car models by price in ascending order and select the Comfort type. Stop before filling in the order information. The task ends after confirming all details are correct.	Hard	-
	Go to Tonghuashun to view the A-share stock rankings. Add the following as filter criteria: highlights, industry, and announcements. Filter for stocks with excellent profitability, in the medical equipment industry, and restructured within the past month. After filtering, browse the stock with the highest trading volume.	Hard	-
	In Baidu Browser, search for images of "mango pudding" and "matcha pudding" separately. Download one image from each search result and open the source of the images. Go to "My Pictures" to view the downloaded images.	Hard	-
GUI-Reasoning	Go to Toutiao and clear the search history.	-	Easy
	View the details of the first product in the men's clothing tops and jackets category on Pinduoduo.	-	Medium
	Play all the news for the watchlist stocks on Tonghuashun.	-	Hard
	Go to Jinri Toutiao to change the profile picture, selecting one from the default artistic avatars.	-	Hard

Table 15: Task examples and their corresponding difficulty levels

straint for the current step to determine whether the action involved interacting with the My Favorites button. Note that accessing the My Favorites page may involve clicking either the corresponding text or the star icon next to it. Therefore, the position constraint is not the TextView bounding box shown in Figure 7(a), but the ViewGroup bounding box containing both elements, as illustrated in Figure 7(b). The task success condition is:

- `//*[contains(@text, "My Favorites") and contains(@resource-id, "id/title") and bbox_contains_point (../@bounds,`

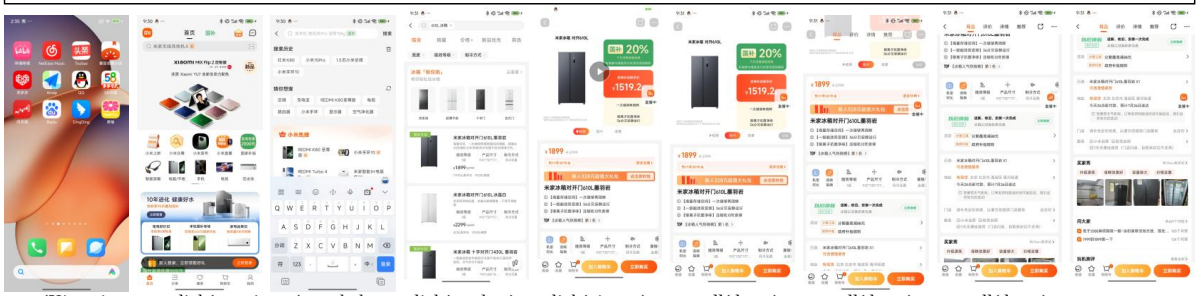
Task: Open the store homepage for baby wipes on Pinduoduo.

open(Pinduoduo) | click(search_box) | input(serach_box, baby wipe) | click(filter) | click(OK) | click(product) | click(store)

```
Task Success Condition:
//*[ contains(@content-desc, "baby wipe")  and contains(@resource-id,
"id/tv_title")] and //*[ contains(@text, “store") and contains(@resource-id,
"id/pdd")] ,
//*[(contains(@text, "store") or contains(@text, “enter brand"))  and
bbox_contains_point (../@bounds,$point)].
```

Figure 8: A task example from Base subset and its corresponding success condition.

Task: Check out the buyer shows for the 610L refrigerator at Xiaomi Mall.



open(Xiaomi Shop) | click(search_box) | input(serach_box, 610L refrigerator) | click(product) | click(picture) | scroll(down) | scroll(down) | scroll(down)

```
Task Success Condition:
//*[contains(@text, "refrigerator") and contains(@text, "610L") and
contains(@package, “xiaomi.shop")],
//*[contains(@text, “buyer show")]  and //*[contains(@text, “comment")]
```

Figure 9: A task example from Long-Tail subset and its corresponding success condition.

`$point)]` and `//*[contains(@text, "History")` and `contains(@resource-id, "id/title")]`

B MobileBench-OL Benchmark Details

B.1 Benchmark Subsets

Evaluating a GUI agent requires more than completing page navigation tasks on a popular app; it also demands handling long-horizon tasks, exploration, and robustness to noise. We categorize the evaluation into three core dimensions: 1- Base Capabilities (the Base and Long-Tail subsets), 2- Complex Reasoning (the Long-Horizon and GUI-Reasoning subsets), and 3- Robustness (the Noise-Robust subset). MobileBench-OL leverages this structure to offer a comprehensive evaluation of GUI agents.

Table 15 shows several examples for the Base, Long-Tail, Long-Horizon and GUI-Reasoning subsets. Figure 8 and Figure 9 show examples from the Base subset and Long-Tail subset, respectively.

B.1.1 Functional Point Coverage

In Base subset, when constructing tasks for an app, we expect the benchmark to comprehensively cover all key functional points, not just the core functions. As illustrated in Figure 10, the core

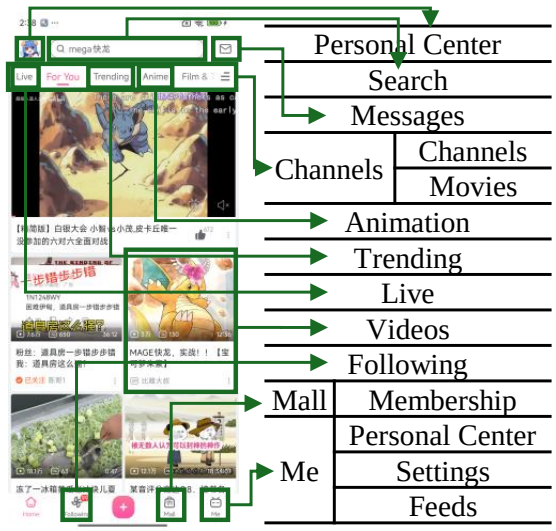


Figure 10: Homepage and 13 key functional points of bilibili. Note that some functions are only accessible via secondary pages.

function of the Bilibili app is keyword-based video search. However, the app also includes additional functions, such as Channels (movies, videos, trending), Personal Center, and Settings. Ensuring coverage of these secondary functions enables a more comprehensive evaluation of the GUI Agent's universality and adaptability.

The key functions were identified based on a large number of tasks requested by real users of mobile assistants and are primarily located on the app's first and second-level pages.

Bilibili's functions include:

```
|-- Homepage
|   |-- Search
|   |-- Live
|   |-- Animation
|   |-- Movies
|   `-- Sports
|-- Following
|-- Posting Works
|-- Member Shopping
|-- My Profile
|   |-- My Profile
|   |-- Settings
|   |-- Offline Cache
|   |-- History
|   |-- My Favorites
|   |-- Watch Later
|   |-- Scan
|   |-- Comics
|   `-- Member Center
`-- Video Playback Page
    `-- One-Click Three-Fold
```

NetEase Cloud Music's functions include:

```
|-- Recommendations
|   |-- Search
|   |-- Song Recognition
|   |-- Daily Recommendations
|   |-- Hot Songs Chart
|   |-- Audiobooks
|   |-- Playback
|   |-- Personal Radar
|   |-- Play Cards
|   |-- Podcasts
|   `-- Scan
|-- Personal Roaming
|-- My Profile
|   |-- My Profile
|   |-- Heartbeat Mode
|   |-- Favorites
|   `-- Local
`-- Extended Menu
    |-- Member Center
    |-- Settings
    `-- Timed Shutdown
```

In the Base subset, the annotator first writes at least one query for each function in an app. Then, for a selection of popular and frequently used tasks, they complete all associated queries for that app. This ensures that all of the app's key functionalities are covered.

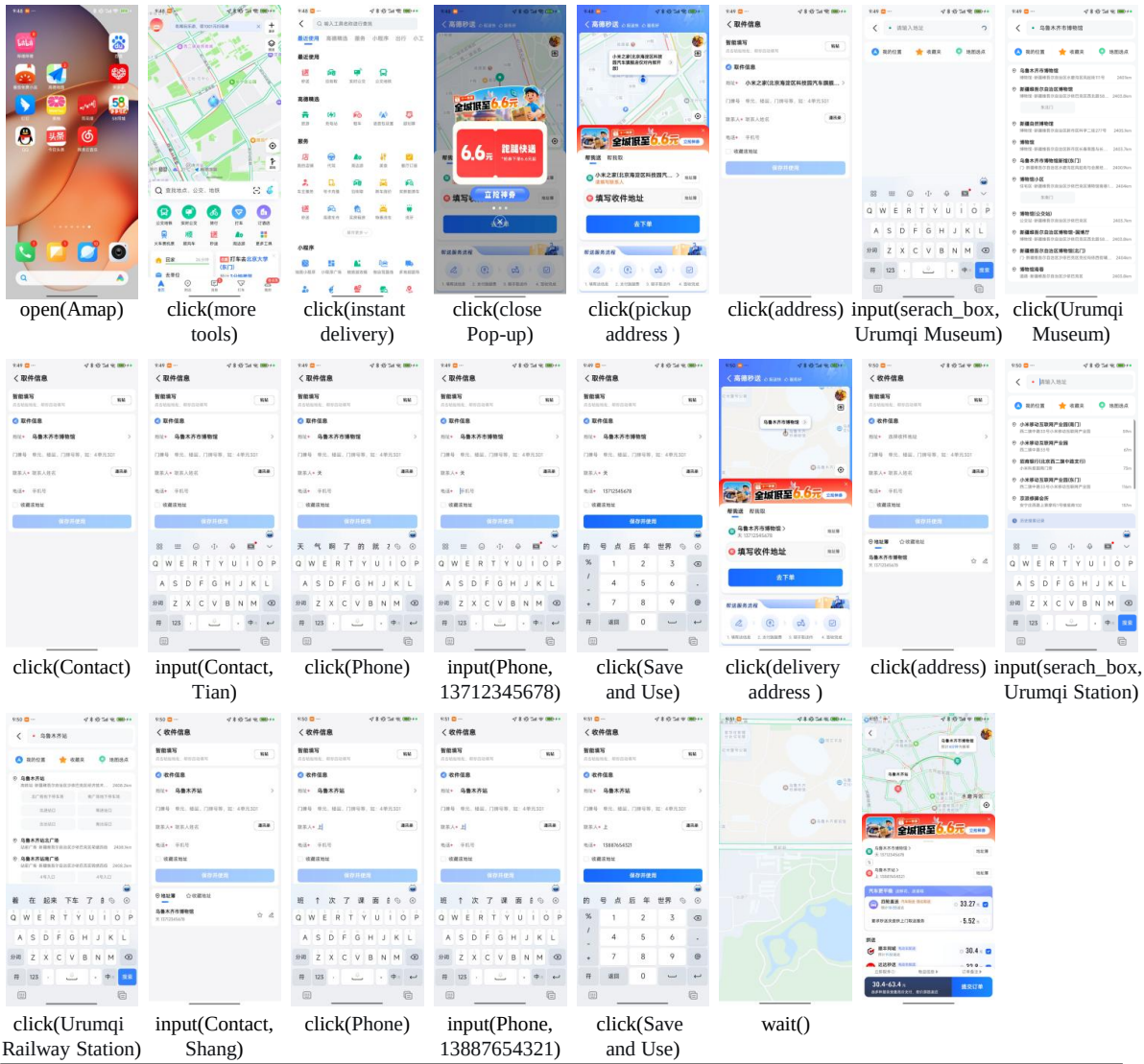
To reduce the number of functionality categories in the dataset, some functions were renamed and merged. However, each consolidated functionality is still guaranteed to have at least one query. For example, "Personal Center" and "My Account" are now collectively referred to as "Personal Center."

B.1.2 Long-Horizon Subset

The Long-Horizon subset evaluates the agent's ability to maintain a high-level goal, decompose it into a correct sequence of sub-tasks, and execute them without losing track of the objective. Each task requires at least 20 steps to complete.

We include two examples in Figure 11 and Figure 12. The task of instant delivery requires filling in multiple fields of information for both the recipient and the sender, testing the ability to

Task: Open Amap and book an instant delivery. Set the pickup address as Urumqi Museum, with the recipient's name as Tian and phone number as 13712345678. Set the delivery address as Urumqi Railway Station, with the recipient's name as Shang and phone number as 13887654321. Check the required price (stay on the interface before submitting the order).




```
Task Success Condition:
//*[contains(@text, "Urumqi Museum")] and //*[contains(@text, "Tian")] and
//*[contains(@text, "13712345678")] and //*[contains(@text, "Save and Use") and
bbox_contains_point(../@bounds,$point)],
//*[contains(@text, "Urumqi Railway Station")] and //*[contains(@text, "Shang")] and
//*[contains(@text, "13887654321")] and //*[contains(@text, "Save and Use") and
bbox_contains_point(../@bounds,$point)]
```

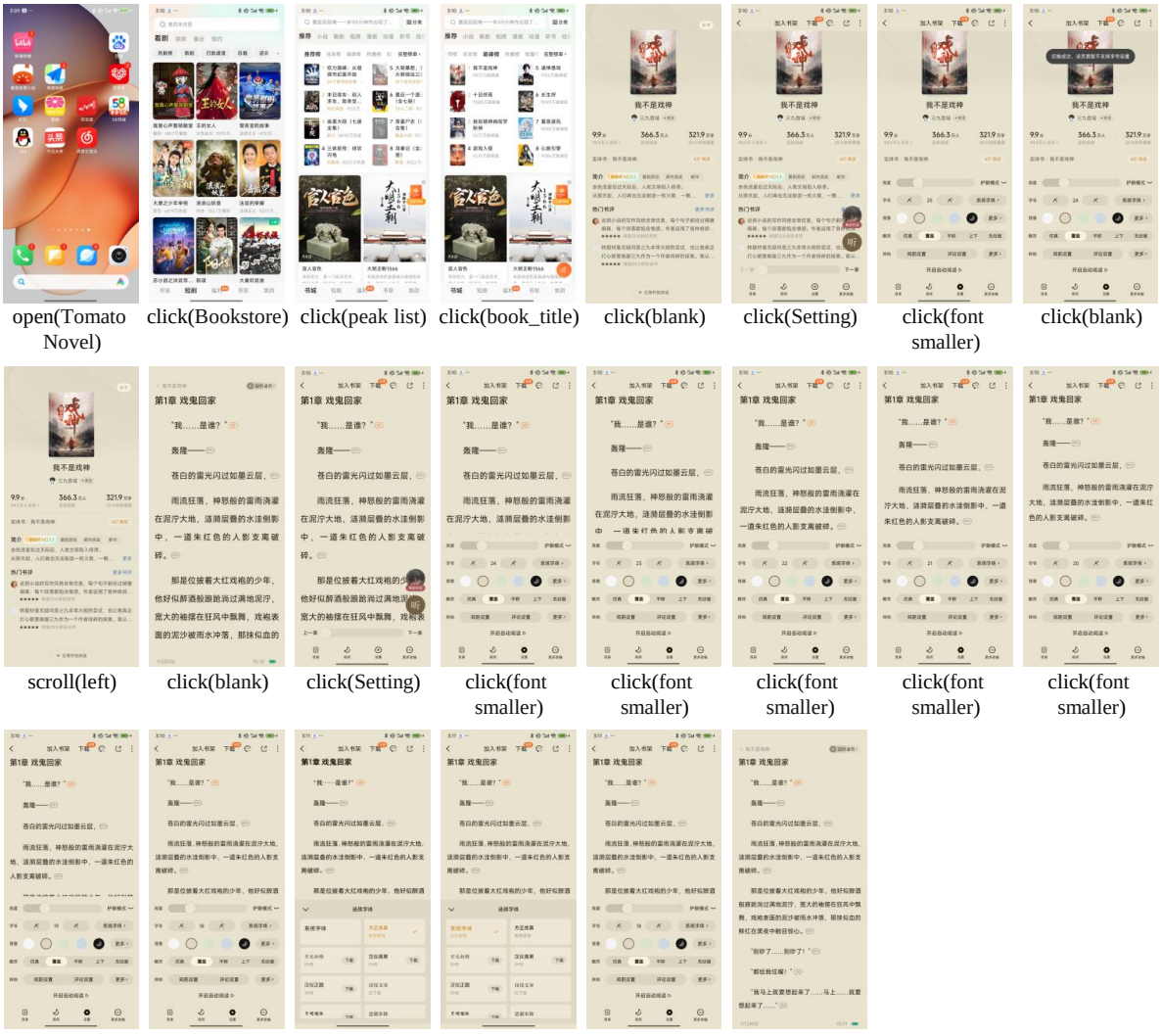
Figure 11: A 22-step long-horizon task and its corresponding success condition.

manage and sequence multiple subtasks without omission. Adjusting the font size in a novel reader requires repeatedly clicking the reduce button, testing the agent's precise control over components. Other components requiring precise control include date selection, alarm settings, and speed adjustments. In summary, long-horizon tasks require agents to accurately decompose and prioritize subtasks, while keeping track of which subtasks have been completed, which are in progress, and which are still pending.

B.1.3 GUI-Reasoning Subset

The GUI-Reasoning subset emulate real-world scenarios that require active exploration and sophisticated reasoning. In these tasks, the goal

Task: Open the top-ranked novel on the Tomato Novel's peak list, set the font size to 18, and the font to the system font.




```
Task Success Condition:
//*[contains(@text, "peak list") and bbox_contains_point(@bounds,$point) ],
//*[contains(@text, "font size")  ] and //*[contains(@text, "18")],
//*[contains(@text, "system font")  and bbox_contains_point(../@bounds,$point)]  or
(//*[contains(@text, " font size")  ] and //*[contains(@text, "system font") ]) ''' ]}
```

Figure 12: A 21-step long-horizon task and its corresponding success condition.

is not directly achievable through a single, obvious action; instead, the agent must infer how to proceed by understanding the visual context, learning through trial and error, and recovering from dead ends, much like a human user. For example, to "disable messages from strangers," a user wouldn't find this option directly on the main page. They would logically navigate to the "Settings" or "Message" section and browse through sub-menus to locate it. Similarly, an agent need to explore the interface, testing different paths and recovering from dead ends, which directly tests its exploratory and reasoning abilities.

These tasks test advanced exploration capabilities, which we categorize into three key areas: Icon Understanding, Hidden Function Discovery, and Hierarchical Navigation. Table 16 shows the descriptions and weights of each ability.

• Icon Understanding: Interpreting the meaning of visual elements without explicit text labels.

Task	Icon	Hidden Function	Hierarchical	Weight	**Exploration Difficulty**	Golden Steps	**Length Difficulty**	**Functional Point**
Search for Pokemon on Bilibili.	❌	❌	❌	0	-	4	Easy	Search
Open the second episode of Naruto on bilibili	❌	❌	❌	0	-	6	Easy	Animation
Navigate to Toutiao's scan function.	✅	❌	❌	0.5	Easy	6	Easy	Channel
Open Bilibili, change my profile avatar to a random one.	✅	✅	❌	1.5	Medium	6	Easy	Personal Center
I want to contact Bilibili customer service.	❌	✅	✅	3	Hard	4	Easy	Settings
Open Bilibili and set a 15-minute timer to turn it off.	❌	✅ ✅	✅	4	Hard	11	Medium	Settings

Figure 13: 6 MobileBench-OL's example tasks.

Ability	Weight	Description
Icon Understanding	0.5	Identify and understand icons, graphics, or other visual elements.
Hidden Function Discovery	1	Find hidden or advanced features on the current page.
Hierarchical Navigation	2	Understand app structure and hierarchy to find specific functions.

Table 16: The descriptions of exploration abilities.

• Hidden Function Discovery: Locating and accessing functionalities that are not immediately visible on the screen (e.g., finding a "Draft" channel hidden in the submenu, or swiping up to reveal a "Settings" option that is off-screen).

• Hierarchical Navigation: Efficiently traversing deep and complex menu structures to find a specific setting or page. (e.g., finding the "Auto Downloads" option within "Download Settings" in the Settings menu.)

Here, Hidden Function Discovery is page-level, meaning the function is hidden within the current page. Examples include the "More Channels" list displayed by clicking a dropdown button, a pop-up window that appears after clicking "More," or a settings button revealed by swiping up. These functions can be discovered through observation and inference based on the current page. Hierarchical Navigation is app-level, requiring an understanding of the app's structure. For example, finding the "Timer" function might require navigating to User Center → Settings → General Settings → Timer. Similarly, finding "Stranger Message Notifications" could involve going to User Center → Messages → Options → Notifications. In many cases, the agent might select the wrong element on the User Center page, making the corresponding function difficult to locate. In this case, Hierarchical Navigation is more difficult; therefore, we assign it a higher difficulty weight.

Exploration Difficulty Levels: The abilities of Icon Understanding, Hidden Function Discovery, and Hierarchical Navigation have assigned weights of 0.5, 1, and 2, respectively. A task's exploration difficulty is calculated by summing the weights of all abilities required for its completion, as defined by:

$$\text{Level} = \begin{cases} \text{Easy} & 0 < \text{total weight} \leq 1 \\ \text{Medium} & 1 < \text{total weight} \leq 2 \\ \text{Hard} & 2 < \text{total weight} \end{cases}$$

Figure 13 shows several task examples. Easy-level task like navigating to scan function (Weight: 0.5), require Scan icon recognition. Medium-level task, like changing an avatar (Weight:1.5), involve recognizing avatar picture and managing pop-up windows. Hard-level task, like contacting customer service (Weight:3), has fewer golden steps comparing to change an avator (4<6), but has a higher exploration difficulty because it requires to understand the app structure to locate the customer service area, and scroll up the page to find a hidden feature. Similarly, the task of setting a timer (Weight:4) requires an understanding of the hierarchical structure and finding two hidden functions, Setting and Timer.

B.1.4 Noise-Roubst Subset

The Noise-Roubst subset introduces four distinct types of environmental noise to evaluate agent robustness:

Noise Cause: Each noise type is triggered by a specific failure or interruption in the inference process.

• Repeat noise means the agent incorrectly assumes a previous action has failed. This is typically caused by insufficient waiting time after action execution or by fetching the device status too quickly, leading the agent to generate and execute the same action repeatedly.

• Unexecuted noise means the agent's previous action fails to trigger the expected UI response (e.g., a button click that has no effect on the interface).

• Delay noise arises from extended interruptions that prevent state progression. This includes Page Loading Delays, temporary Network Disconnection, or Ad-delay pages that hold the interface in a loading state.

• Pop-up noise is triggered by unexpected overlay pages that interrupt the main workflow. These include Login Pages requiring authentication, Captcha Pages with unsolvable challenges, Ad Pop-ups obscuring the UI, and Cookie Consent Pop-ups that must be dismissed.

Handling Method: The required agent response and the potential negative outcomes of an incorrect response are closely linked for each noise type.

• For Repeat noise, the agent must return to the previous page. If mishandled, the page will have already advanced after the second execution, forcing the agent to restart from a new, unintended state. This may be misinterpreted as the initial action causing two page transitions.

• For Unexecuted noise, the agent must re-execute the intended action. Failure to do so results in an unchanged state, requiring the agent to re-observe and retry. This can be mistaken for the action itself being invalid.

• For Delay noise, the correct response is to wait. If the agent waits correctly, the true result page loads normally. However, if the agent generates any other action during the delay, that action will be performed on the real result state (which is hidden), often leading to errors or navigation to an incorrect page.

• For Pop-up noise, the agent must close the pop-up. A correct click on the close button will reveal the true underlying page. If the agent performs any other action, the pop-up will persist in the agent's view, and the action will have no effect on the real device state, creating a deadlock.

Noise Injection Method: The noise injection method and the page state presented to the agent are as follows.

For Repeat noise, the injection method is to execute the same agent action twice consecutively. The agent is then shown the page that results from this double execution.

For Unexecuted noise, the injection method is to block the agent's action from being passed to the device for execution. Consequently, the agent continues to see the unchanged previous page.

For Delay and Popup noises, a special simulation approach is used. Because it is impossible to reliably source random real-world waiting or pop-up pages, pre-prepared template pages are used. The noise injection method involves copying and presenting these pre-prepared pages to the agent. Meanwhile, the device continues to process the action in the background and reaches the real result state. Therefore, the agent is shown a simulated Delay (waiting) page or a simulated Popup page, while the device displays the true outcome of the action.

B.2 Constuction Process

We manually design tasks from scratch for each app, rather than modifying existing datasets, to reduce the risk of large language model (LLM) data contamination. During this process, we focus on the agent's task execution and exploration abilities, while ensuring diversity in tasks in terms of difficulty levels and function coverage.

Three human annotators constructed tasks based on the apps from scratch. When constructing tasks, we begin by analyzing the app's hierarchical structure, page layout, and distribution of function points. For the high-frequency "Base" subset, we annotate 20–30 tasks per app, ensuring comprehensive coverage of all function points. For the "Long-Tail" subset (with lower frequency), we annotate 5 tasks per app. Since their core functions differ significantly from those of the Base apps, we mainly focus on their core functions to enhance the benchmark's comprehensiveness. For the 'Long-Horizon' subset, we ensure that the tasks within it require at least 20 steps to be completed. For the 'GUI-Reasoning' subset, we ensure that the tasks within it require at least one exploration task to be completed. Noise-Robust use the same test set as the Base subset.

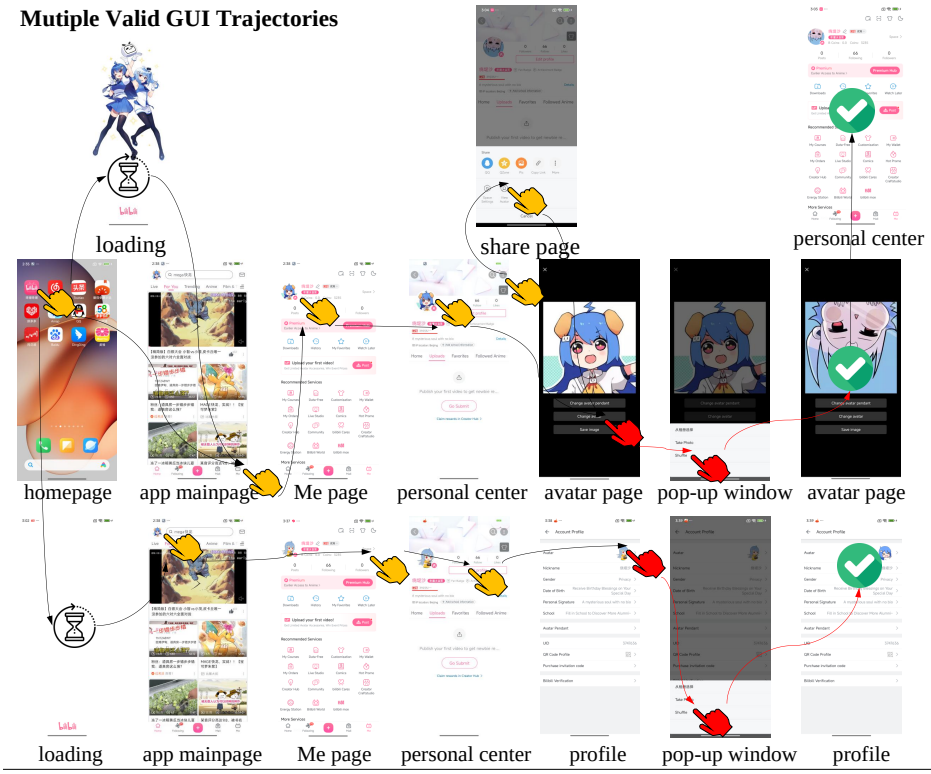
B.3 Dataset statistics

Table 4 shows the dataset statistics for MobileBench-OL.

Task Distribution: MobileBench-OL is divided into five subsets. The Base subset contains 310 tasks covering 12 mainstream apps, with 20–30 tasks per app. The Long-Tail subset contains 340 tasks covering 68 less popular apps, with 5 tasks per app. The Long-Horizon subset contains 60 tasks, with 5 tasks for each of the 12 top apps. The GUI-Reasoning subset also contains 60 tasks, with 5 tasks per app. The Noise-Robust subset follows the same task distribution as the Base

Task: Open Bilibili and change my profile avatar to a random one.

Mutiple Valid GUI Trajectories



Conditions for completing the task

1. Click the "Avatar" or "Change Avatar" button on the profile or avatar page.
2. Select the "Shuffle" option in the pop-up window.
3. The above process must be completed within the Bilibili app.

Task Success Condition

//*[(@text="Avatar" or @text="Change Avatar") and bbox_contains_point(../@bounds,$point) and contains(@package, "bili")],
//*[@text="Shuffle" and bbox_contains_point(../@bounds,$point) and contains(@package, "bili")].

Figure 14: An example task and its corresponding success condition. Red arrows indicate the definitive indicators for completing the task.

subset but explicitly introduces noise only during inference. Due to significant fluctuations in golden steps caused by noise, the average steps are not recalculated after noise is introduced.

Difficulty Distribution: The difficulty of the Base, Long-Tail, Long-Horizon, and Noise-Robust subsets is categorized based on their golden steps. In Long-Horizon, all tasks require more than 20 steps to complete and are classified as Hard. The GUI-Reasoning subset rates task difficulty according to the required exploration ability and contains contains more Medium and Hard tasks.

Functional Point Distribution: The Base subset covers all 28 core functional points of the top apps. The Long-Tail and GUI-Reasoning subsets cover fewer functional points, focusing on core functions of the long-tail apps and exploratory tasks, respectively.

C Data Analysis

C.1 GUI Trajectory

In real-world environments, a task may correspond to multiple valid GUI trajectories, as shown in Figure 14. We have human annotators label a golden trajectory for each task. Here, annotators remove loading screens, ad pop-ups, permission request pages, etc., and focus only on completing the task itself. The middle row of trajectories in Figure 14 can be regarded as the golden trajectory for this task. It is worth noting that the golden trajectory is not necessarily always the shortest valid path. For example, to find Customer Service in Figure 1, the shortest path requires only 4 steps, but people usually do not realize that the Help

Tasks	Task Success Conditions
Open Bilibili and change my profile avatar to a random one.	//*[(@text="Avatar" or @text="Change Avatar") and bbox_contains_point (../@bounds, $point) and contains(@package, "bili")] and //*[@text="Shuffle" and bbox_contains_point(../@bounds,$point) and contains(@package, "bili")].
I want to contact Bilibili customer service.	//*[(@text="Customer Service" or @text="Help Center") and bbox_contains_point(../@bounds, $point) and contains(@package, "bili")].
Find the subway route from Beijing South Railway Station to Beijing Fengtai Station on Amap.	//*[contains(@text, "Public Transportation") and @selected="true" and contains (@package, "map")] and //*[contains(@text, "Beijing South Railway Station") and contains(@resource-id, "route_edit_summary_start")] and //*[contains(@text, "Beijing Fengtai Station") and contains(@resource-id,"route_edit_summary_end")]

Table 17: Example tasks and their corresponding success conditions.

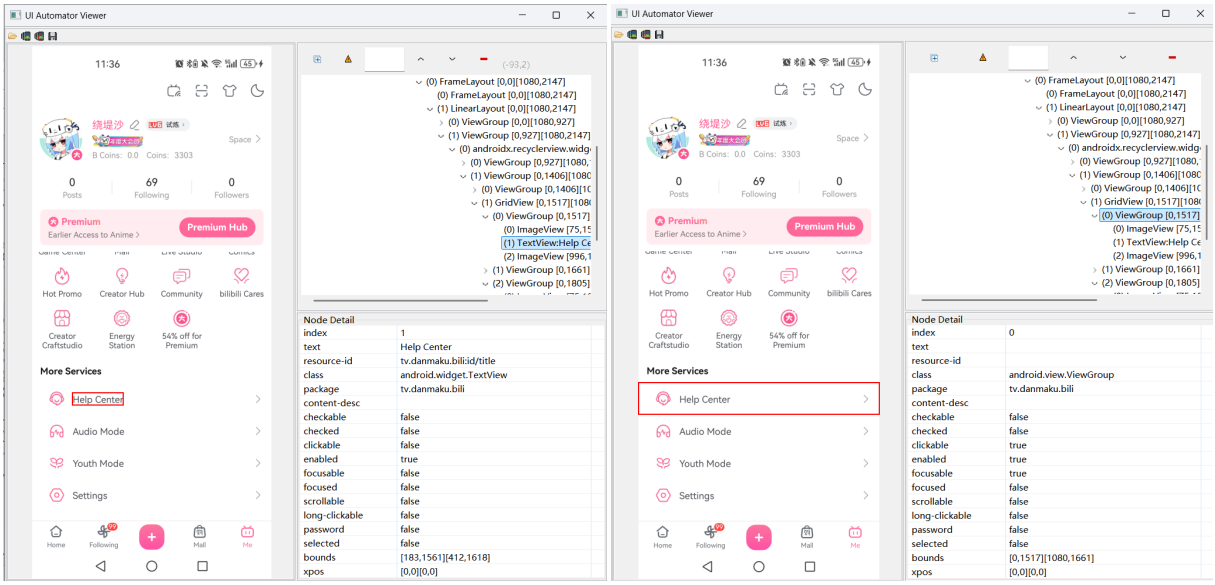


Figure 15: The bounding box of the 'Help Center' and the parent node of 'Help Center'.

Center also navigates to Customer Service, and may take 7 steps to find the Customer Service function through the Settings.

C.2 Task Success Conditions

To determine whether a GUI trajectory successfully completes a task in MobileBench-OL, we define task success conditions. These conditions may consist of multiple sub-conditions involving element matching (verifying the presence of specific GUI elements) and action matching (checking for critical actions). Each condition is annotated using a rule-based approach to ensure robustness.

Since a single task may have multiple valid GUI trajectories, we construct success conditions using only those elements and actions that appear exclusively in successful trajectories. This ensures: (1) Completeness: Every successful trajectory satisfies the entire condition. (2) Soundness: No unsuccessful trajectory fully meets the condition. This approach guarantees that the evaluation is both precise (only correct trajectories pass) and flexible (matching diverse but valid paths).

For example, in Figure 14, the success condition of task "Change to a random Bilibili avatar" is clicking the Avatar button and then selecting Shuffle, regardless of the navigation path. This design provides precise evaluation while tolerating diverse valid paths.

To ensure scalability and stability, task success conditions are uniformly defined using XPath-like rules. Each GUI element is characterized by attributes such as text, resource-id, class, package, and checked. These attributes are used to describe the required elements and actions in the rules. Finally, task success conditions are converted into normalized format.

As shown in Figure 14, the first sub-condition uses "contains(@package, "bili")" to confirm the current page belongs to bilibili, text="Avatar" ensures the presence of the "Avatar" element, and bbox_contains_point verifies that the interaction

Original Task	Category	Reset Task
Turn on Pinduoduo's senior mode.	Task-level Reset	Turn off Pinduoduo's senior mode. Follow these steps: 1. Open Pinduoduo and click the "Personal Center" button in the lower right corner of the homepage to go to the personal center page. 2. If you see a "Close Large Text" button in the upper right corner, it means Senior Mode is enabled, click it. If the button is not there, Senior Mode is already off, and no further action is needed. 3. On the Senior Mode page, click "Close Senior Mode" to complete the process. Follow the above steps to turn off Pinduoduo's senior mode.
Turn on Bilibili private message intelligent filtering.	Task-level Reset	I need to turn off the private message intelligent filtering on Bilibili. Follow these steps: 1. Open Bilibili and tap the "My" button at the bottom of the home page. 2. Once on the "My" page, if the background is black, tap the sun icon in the top right to switch to light mode. If the background is already white, proceed to step 3. 3. On the "My" page, swipe up until you see the "Settings" button at the bottom of the More Services section. 4. Tap the "Settings" button to enter the Settings page. 5. On the Settings page, tap "Message Settings" to go to the Message Settings page. 6. In the Message Settings page, if the toggle next to "Private Message Intelligent Filtering" is turned on, tap it to turn it off. If the toggle is already off, the intelligent filtering feature is disabled, and the task is complete.
Search for today's stock price of Wuliangye on Toutiao.	App-level Reset	Clear Toutiao search history with the following steps: 1. Open Toutiao and tap the search box at the top of the home page. 2. In the search page, there is a trash can icon on the right side of the search history section—this represents clearing search history. Tap this trash can button. 3. In the expanded options, tap the "Delete All" button. 4. In the pop-up window asking "Are you sure you want to clear the search history?", tap the "Confirm" button. Follow the steps above to clear the search history.
Search for today's gold price on Toutiao.		
Check out the first news article in the Toutiao Picture Channel and give it a like.	No reset needed	-
Open Bilibili messages and mark all as read with one click.	Infeasible	-

Table 18: Example reset tasks from different Reset Categories.

occurs within the bounding box of the "Avatar" button. Similarly, the second sub-condition checks for an element with the text "Shuffle" and the package "bili", and the current action occurs within the bounding box of this element.

Table 17 shows more examples. For the task of accessing customer service, clicking on the Help Center or Customer Service can accomplish it. It is worth noting that '../@bounds' refers to the parent node of this element. The three attribute constraints of this element in the Task Success Conditions are described in natural language as follows: the text of this element contains 'Help Center' or 'Customer Service', the package includes 'bili', and the interaction point for the next action on this GUI page falls within the bounding box of this element's parent node. As shown in Figure 15, the bounding box of the element with the text 'Help Center' covers only the text box, while the actual boundary for accessing this function should also include the headphone icon on the left and the blank area on the right. Therefore, we need the interaction point to fall within the parent node of 'Help Center'.

D Reset Mechanism

Table 18 illustrates several reset tasks from different reset categories. Given an original task such as "Turn on Bilibili private message intelligent filtering," the reset task is a step-by-step inversion designed to turn off "private message intelligent filtering." At each step, we prompt the agent by specifying which element it should interact with, along with the element's possible location, shape, and the current GUI's likely state. In this way, we guide the agent to reset the device state as

You are an agent who can operate an Android phone on behalf of a user. Based on user's goal/request, you may - Complete some tasks described in the request by performing actions on the phone using visual understanding.
At each step, you will be given the history path, current screenshot (before action) and the task goal. You must analyze the screen and output your action decision:
1. A brief reasoning in Chinese: Why and where to take the next action.
2. A structured action command in format below.
Supported Actions:
- Click/tap a position on screen: ‘{{"click(start_point=(x1,y1))"}}‘
- Scroll the screen: ‘{{"scroll(start_box=(x1,y1), end_box=(x2,y2))"}}‘
- Type text into an input field when searching: ‘{{"type(content=...)}}‘
- Press home button: ‘{{press_home()}}‘
- Press back button: ‘{{press_back()}}‘
- Wait for UI update: ‘{{wait()}}‘
- The task is finished: ‘{{finished(content=)}}‘
You must only use the above 7 actions.
Use coordinates based on your visual understanding of the screenshot.
The current user goal/request is: {goal}
the size of the screenshot is 1080 * 2400
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 (notice that some elements in this list may not be visible in the current screen and so you can not interact with it, can try to scroll the screen to reveal it first), the numeric indexes are consistent with the ones in the labeled screenshot:
{ui_elements}
Here are some useful guidelines you need to follow:
General:
...
Action Related:
...
Text Related Operations:
...
Now output your decision: Thought: Please infer your intention and location in Chinese. Action: (structured action selected from above 7 actions)...

Figure 16: The prompt of GPT-4o.

completely as possible, regardless of the original task's complexity.

For Task-level resets, each original task is assigned a separate reset task. For App-level resets, multiple original tasks share a single reset task. For example, an original task like "Search for today's stock price of Wuliangye on Toutiao" has the reset task "Clear Toutiao search history." Thus, various search tasks on Toutiao can all use this single reset task, which only needs to be performed once per benchmark test epoch

For tasks that require no reset, such as "Check out the first news article in the Toutiao Picture Channel and give it a like," the process is purely observational. These tasks leave no lasting environmental changes or shortcuts, and the navigation process in one trial does not interfere with the trajectory generation of the next.

For infeasible tasks, such as "Open Bilibili messages and mark all as read with one click," once all unread messages on the device are marked as read, we can no longer accurately identify and

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 (including cases like you don't have enough information or can not perform some necessary actions), 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. We have added marks (bounding boxes with numeric indexes on their TOP LEFT corner) to most of the UI elements in the screenshot, use the numeric index to indicate which element you want to click: ‘{{"action_type": "click", "index": <target_index>}}‘.
- Long press on an element on the screen, similar with the click action above, use the numeric label on the bounding box to indicate which element you want to long press: ‘{{"action_type": "long_press", "index": <target_index>}}‘.
- Type text into a text field (this action contains clicking the text field, typing in the text and pressing the enter, so no need to click on the target field to start), use the numeric label on the bounding box to indicate the target text field: ‘{{"action_type": "input_text", "text": <text_input>, "index": <target_index>}}‘
- Press the Enter key: ‘{{"action_type": "keyboard_enter"}}‘ - Navigate back: ‘{{"action_type": "navigate_back"}}‘
- Scroll the screen or a scrollable UI element in one of the four directions, use the same numeric index as above if you want to scroll a specific UI element, leave it empty when scroll the whole screen: ‘{{"action_type": "scroll", "direction": <up, down, left, right>, "index": <optional_target_index>}}‘
- Wait for the screen to update: ‘{{"action_type": "wait"}}‘
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 (notice that some elements in this list may not be visible in the current screen and so you can not interact with it, can try to scroll the screen to reveal it first), the numeric indexes are consistent with the ones in the labeled screenshot: {ui_elements}
Here are some useful guidelines you need to follow: ... {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:

Figure 17: The prompt of M3A.

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, a list of descriptions for most UI elements on the current screen will be given to you (each element can be specified by an index), together with a history of what you have done in previous steps. 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 (including cases like you don't have enough information or can not perform some necessary actions), 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 a UI element (specified by its index) on the screen: ‘{{"action_type": "click", "index": <target_index>}}‘.
- Long press on a UI element (specified by its index) on the screen: ‘{{"action_type": "long_press", "index": <target_index>}}‘.
- Type text into an editable text field (specified by its index), this action contains clicking the text field, typing in the text and pressing the enter, so no need to click on the target field to start: ‘{{"action_type": "input_text", "text": <text_input>, "index": <target_index>}}‘
- Press the Enter key: ‘{{"action_type": "keyboard_enter"}}‘
- Navigate back: ‘{{"action_type": "navigate_back"}}‘
- Scroll the screen or a scrollable UI element in one of the four directions, use the same numeric index as above if you want to scroll a specific UI element, leave it empty when scroll the whole screen: ‘{{"action_type": "scroll", "direction": <up, down, left, right>, "index": <optional_target_index>}}‘
- Wait for the screen to update: ‘{{"action_type": "wait"}}‘
The current user goal/request is: {goal}
Here is a history of what you have done so far: {history}
Here is a list of descriptions for some UI elements on the current screen: {ui_elements_description}
Here are some useful guidelines you need to follow: General ... {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:

Figure 18: The prompt of T3A.

revert them to an unread state. Therefore, for such tasks, we consider two success scenarios and define the corresponding success conditions for each: successfully clicking the one-click read button within the trajectory, or arriving at the message page and finding no unread messages.

E Prompt

The prompt for GPT-4o is shown in Figure 16. The prompts used in M3A and T3A are provided in Figures 17 and 18, respectively. The prompt used in Mobile-Agent-V2 can be seen in Figure 19. The prompt for UI-TARS is in Figure 20. The prompts for Qwen2-VL, Qwen2.5-VL and UI-TARS-1.5 are shown in Figure 21.

Background
This image is a phone screenshot. Its width is {width} pixels and its height is {height} pixels. The user's instruction is: {instruction}.
Screenshot information
In order to help you better perceive the content in this screenshot, we extract some information on the current screenshot through system files. This information consists of two parts: coordinates; content. The format of the coordinates is [x, y], x is the pixel from left to right and y is the pixel from top to bottom; the content is a text or an icon description respectively. The information is as follow: {clickable_infos}
Please note that this information is not necessarily accurate. You need to combine the screenshot to understand.
Keyboard status
We extract the keyboard status of the current screenshot and it is whether the keyboard of the current screenshot is activated. The keyboard status is as follow: {keyboard_status}
Hint
There are hints to help you complete the user's instructions. The hints are as follow:
The required app is already open. Please do not use the "Open app" or "Home" actions.
History operations
Before reaching this page, some operations have been completed. You need to refer to the completed operations to decide the next operation. These operations are as follow: {action_history}
Progress
After completing the history operations, you have the following thoughts about the progress of user's instruction completion: Completed contents: {completed_content}
Memory
During the operations, you record the following contents on the screenshot for use in subsequent operations: Memory: {memory}
Response requirements
Now you need to combine all of the above to perform just one action on the current page. You must choose one of the six actions below:
Open app (app name): If the current page is desktop, you can use this action to open the app named "app name" on the desktop.
Tap (x, y): Tap the position (x, y) in current page.
Swipe (x1, y1), (x2, y2): Swipe from position (x1, y1) to position (x2, y2).
{type_action}
Back: Return to the previous page.
Home: Return to home page.
Stop: If you think all the requirements of user's instruction have been completed and no further operation is required, you can choose this action to terminate the operation process.
Output format
Your output consists of the following three parts:
Thought
Think about the requirements that have been completed in previous operations and the requirements that need to be completed in the next one operation.
Action
You can only choose one from the six actions above. Make sure that the coordinates or text in the "()".
Operation
Please generate a brief natural language description for the operation in Action based on your Thought.

Figure 19: The prompt of Mobile-Agent-V2.

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(start_box='<|box_start|>(x1,y1)<|box_end|>')
type(content='')
scroll(direction='down or up or right or left')
press_back()
press_home()
wait()
finished() # Submit the task regardless of whether it succeeds or fails.
Note
- Use English in Thought part.
- Summarize your next action (with its target element) in one sentence in Thought part.
User Instruction

Figure 20: The prompt of UI-TARS.

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>')
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')
press_home()
press_back()
wait()
finished(content='xxx') # Use escape characters ', ", and \n in content part to ensure we can parse the content in normal python string format.
Note
- Use Chinese in ‘Thought‘ part.
- Write a small plan and finally summarize your next action (with its target element) in one sentence in ‘Thought‘ part.
User Instruction

Figure 21: The prompt of Qwen2-VL, Qwen2.5-VL and UI-TARS-1.5.

F More Related Work

GUI Agents. The field of GUI agents has made significant progress recently, driven by the rapid development of Multimodal Large Language Models (MLLMs) (OpenAI, 2023; Liu et al., 2023) and large-scale GUI datasets (Zhan and Zhang, 2023; Wu et al., 2024a; Deng et al., 2023). Some current research (Wang et al., 2024a; Lu et al., 2024) leverages powerful closed-source models like GPT-4o (Yan et al., 2023) and Gemini (Comanici et al., 2025), using prompt engineering to build GUI

agents. Other work (Bai et al., 2024; Wang et al., 2024d) employs supervised fine-tuning (SFT) and reinforcement learning from human feedback (RLHF) on open-source base models trained on GUI datasets. These studies have benefited from the support of growing large-scale GUI datasets. For example, AITW (Rawles et al., 2023b) and Android Control (Li et al., 2024) support GUI navigation tasks, while AMEX (Chai et al., 2024) and Ferret-UI (You et al., 2024) enable deeper understanding of mobile UI hierarchies and elements. To enhance GUI agents' ability to handle complex tasks, recent studies (Wu et al., 2025a; Wang et al., 2025; Liu et al., 2025c) have introduced advanced features such as task planning, memory storage, execution reflection and error correction. Rapidly advancing GUI agents also require comprehensive benchmarks to evaluate their performance across various tasks and environments.

GUI Benchmarks. With the rapid advancement of GUI Agents, traditional static benchmarks have become inadequate for comprehensively evaluate their full abilities. Recently, dynamic benchmark environments have been proposed (Kong et al., 2025; Cao et al., 2025), such as AndroidWorld (Rawles et al., 2024), LlamaTouch (Zhang et al., 2024b), and Mobile-Env (Zhang et al., 2023b). Unlike static benchmarks, these environments evaluate task completion based on device state detection or LLM judgments rather than requiring user actions to precisely match predefined answers. However, these dynamic benchmarks mainly rely on offline static apps (e.g., Google Suite apps, F-Droid apps, and built-in system apps) (Liu et al., 2025b), which are significantly different from mainstream app designs and cannot represent real-world usage scenarios. While SPA-Bench (Chen et al., 2024b) has made progress by incorporating popular open-source apps, it remains limited. It cannot support apps and tasks that require logging into an account or have high memory usage. A key limitation of all of these benchmarks is that they rely on simulators rather than real devices, which both significantly limits the scope of tasks and apps and may not accurately reflect performance in real-world environments. Additionally, the tasks in existing benchmarks are often too simple or too rigid, neglecting the evaluation of the exploration and generalization capabilities of GUI Agents, which are crucial in dynamic environments.

G Experiments Settings

G.1 Evaluation Metrics

We define four key metrics to evaluate agent performance:

• Success Rate (SR): A GUI trajectory is considered a successful completion if it meets the entire task success condition and ends with Complete action. This is the core metric of MobileBench-OL.

• Sub-condition Success Rate (Sub SR): Sub-SR calculates the ratio of matched sub-conditions to the total number of sub-conditions, regardless of the termination reason. As discussed earlier, each task's condition may consist of multiple sub-conditions involving element matching and action matching. We evaluate whether a GUI trace meets these sub-conditions to provide fine-grained insight into task execution.

• Step Ratio: This metric compares the number of steps taken by the agent to the golden steps operated by human ideally. The agent's max step limit is set to triple the golden steps. Golden steps excludes redundant steps such as pop-ups or loading waits.

• Failure Reasons: This metric is categorized into three types, see Figure 4: (1) Early Termination: The agent reports task finished without completing the task. (2) Overdue Termination: The agent completes the task but keeps operating until max step. (3) Failure: The agent neither completes the task nor reports finished.

G.2 Implementation Details.

During the evaluation process, all test phones have all apps pre-installed and account logins completed. After each round of MobileBench-OL evaluation, we will perform reset tasks to restore the phone to its initial state.

The specific evaluation process is as follows: The GUI agent will cyclically obtain environmental information from the device platform, extract prompts, generate actions, and finally send them back to the platform for execution. This loop will continue until the task is completed or the max step limit is reached. The max step limit for the task is 3 times the human annotated trajectory's steps, and there is a 3-second waiting interval after each action is executed. We found that most agents could not correctly start the target app (even if the app has been placed on the main screen of the phone). For

this reason, we set all tasks to start directly from the app homepage.

G.3 Baselines

We evaluate 8 open-source models and 4 closed-source model under unified benchmark setting. All apps are kept in consistent versions and reset to an initial state before testing to ensure repeatability and fair comparisons. All open source agents were deployed on an 80GB NVIDIA A100 GPU, while closed source models were accessed through their respective APIs. All open-source models are of comparable capacities(7B,8B,9B).

For close-source models:

• GPT-4o: We deliberately avoid auxiliary inputs such as SOM or element lists—unlike many previous works—to ensure a strictly screenshot-only evaluation baselines.

• M3A: Following the setup in *AndroidWorld* (Rawles et al., 2024), M3A uses GPT-4o as the backbone model, taking UI element information and SoM-enhanced screenshots as input to generate actions. It further includes a reflection step—comparing pre- and post-action UI states—and incorporates the reflection summary into the action history for subsequent decisions.

• T3A: T3A adopts a similar pipeline and also uses GPT-4o as the backbone model, but operates in text-only mode: it relies solely on UI element textual information, without any visual input (e.g., screenshots or SoM).

• Mobile-Agent-V2 (Wang et al., 2024a): Mobile Agent V2 employs a multi-agent pipeline comprising a Planning Agent, a Decision Agent, and a Reflection Agent, all built upon GPT-4o as the backbone model. To support icon understanding, it integrates an external icon captioning model via the Qwen-VL-Plus API. We adapt it to our architecture.

For open-source models:

• Qwen2.5-VL and UI-TARS are fine-tuned from Qwen2-VL; the remaining baselines (e.g., InternVL2-8B, CogAgent-9B) are run at a comparable capacity, primarily 7B with occasional 8B or 9B variants, to ensure fair comparison.

• We standardize prompting as follows: all Qwen-based variants use the official UI-TARS prompt; models with vendor-provided prompts/templates (e.g., CogAgent, OS-Atlas) use their official settings; all remaining baselines use the AndroidWorld generic prompt.

H More Experiments

H.1 Long-Horizon Error Analysis

We classify the errors of UI-TARS-1.5 on the Long-Horizon subset into five categories. For each category, we provide a detailed definition and examples below. If a task contains multiple errors, we mark it according to the first error that occurs.

• Reasoning & Planning Failure: The agent has a biased understanding of semantics or has formulated an incorrect execution plan. This includes inconsistencies between the trajectory intent and the task intent, or an incorrect execution order of subtasks.

Reasoning & Planning Failure

Error Case 1:
Task: I want to use Amap to view nearby coffee shops, filter them to those listed for more than 3 years, view the details of the most popular coffee shops, plan public transportation routes to those coffee shops, add them to my favorites, and then go to my favorites to view the routes.
Result: The agent did not add the selected route to my favorites or go to my favorites; instead, they clicked “Start Navigation” directly to view the real-time navigation status.

Error Case 2:
Task: Send "Good morning" to DingTalk assistant on the DingTalk, and then search for emojis for "french fries," "hamburger," "egg tart," "cookie," and "chocolate," select one of each, and send them to the agent.
Result: The agent does not search for emojis; instead, it directly sends "french fries," "hamburger," etc., as plain text.

• Subtask Omission: The agent completely forgets to complete a subtask within the complex task.

Reasoning & Planning Failure

Error Case 1:
Task: Go to NetEase Cloud Music and add "Selling Newspapers," "Planting the Sun," and "The Barber" to my favorite music list. Then go to my favorite music list to view

Model	Base		Long-Tail		Long-Horizon		GUI-Reasoning		Noise-Robust	
	All Tasks	SR Tasks	All Tasks	SR Tasks	All Tasks	SR Tasks	All Tasks	SR Tasks	All Tasks	SR Tasks
GPT-4o	2.16	1.55	2.22	1.46	0.88	-	1.97	1.66	2.40	1.75
M3A	1.95	1.23	2.10	1.39	1.15	0.98	2.06	**1.05**	2.74	1.65
T3A	1.53	1.56	1.40	1.37	**0.54**	**0.29**	**1.40**	1.43	2.41	1.60
Mobile-Agent-V2	2.28	1.48	2.33	1.63	1.11	0.47	1.97	1.50	2.53	1.71
InternVL2-8B	1.83	2.01	2.04	2.13	1.41	-	1.89	-	3.23	-
CogAgent-9B	**1.43**	**1.38**	1.24	**1.41**	1.12	-	1.48	-	**1.76**	1.59
UGround-V1-7B	2.54	2.42	2.01	2.07	1.41	-	2.49	-	3.11	-
OS-Atlas-Pro-7B	2.62	1.81	**1.00**	2.07	1.35	-	2.45	2.17	3.22	2.00
Qwen2-VL-7B	1.98	1.51	1.08	1.76	0.93	-	2.21	-	2.77	**1.49**
Qwen2.5-VL-7B	1.98	1.49	2.25	1.47	0.82	-	1.95	1.39	2.00	1.57
UI-TARS-7B	1.81	1.37	2.24	1.54	1.51	1.55	1.87	1.41	2.19	1.55
UI-TARS-1.5-7B	1.73	1.41	2.13	1.51	2.11	1.31	1.86	1.43	1.97	1.51

Table 19: The Step Ratio (↓) performance of GUI agents across all tasks and SR tasks on MobileBench-OL. A dash ("-") indicates that the agent has no successful tasks. "All Tasks" represents the overall step ratio across all tasks, while the "SR Tasks" represents the step ratio for successfully completed tasks.

them.
Result: The agent did not complete the subtask of searching for "The Barber" and adding it to my favorite music list.

• Function Navigation Failure: The agent knows which function it needs, but cannot navigate to the entry point to activate that function within the app.

Reasoning & Planning Failure

Error Case 1:
Task: Go to Flush to check the A-share stock rankings. Browse the gainers list, losers list, and the stock with the highest trading volume respectively, and view their financial disclosures in the "Important Event Reminders" section of their announcements and information sequentially.
Result: The agent could not find the functional entry points for "Important Event Reminders" and financial disclosures.

• Attribute Omission/Error: The agent attempts to complete a subtask, but the trajectory is incomplete or incorrect. Specifically, when completing a subtask that requires multiple forms or attributes, the agent misses one or two attributes or does not fully follow the task instructions.

Reasoning & Planning Failure

Error Case 1:
Task: Open Bilibili and search for "Minecraft". Sort the search results by view count and filter for videos published in the past week. Select a video, like it, and add it to my favorites. On its comment page, sort the comments by time, like the top comment, then edit a comment with "haha" and a smiling emoji and post it.
Result: The agent did not favorite the video, did not sort the comments on the comment page by time, and could not find a smiling emoji.

• Visual Grounding Failure: The agent fails to locate or interact with the correct UI elements, including errors such as being unable to locate small elements and incorrectly using sliding components.

Reasoning & Planning Failure

Error Case 1:
Task: Open Bilibili and search for "gameplay live streams." Select a live stream from the live channel to enter. In the player settings of that live stream room, set it to stop playback after 20 minutes. In the bullet chat settings, set the display area to 3/4 of the screen and the bullet chat speed to fast.
Result: When setting the time to 20 minutes, the agent did not correctly locate the starting point for the swipe action and was unable to successfully set the time element to 20.

Error Case 2:
Task: In Baidu Browser, search for pictures of Huskies, Samoyeds, and Alaskan dogs

Model	pass@k		
	k=1	k=3	k=5
Qwen2.5-VL	17.10%	31.29%	39.35%
UI-TARS	46.77%	56.45%	69.35%
UI-TARS-1.5	60.97%	69.03%	73.87%

Table 20: Pass@k ($k \in \{1, 3, 5\}$).

respectively, download them, and go to "My Pictures" to view them.
Result: When the agent went to "My Pictures," the app popped up a cloud storage recommendation ad. The agent's ability to locate the small component was insufficient, and it was unable to successfully click the close button, continuing to fail the operation until the trajectory steps were exhausted.

H.2 Step Ratio

In Table 19, a lower step-ratio indicates higher efficiency. For agents with SR below 5%, their Step Ratio is better because they only handle simple tasks and terminate complex tasks prematurely. Better-performing agents have better Step Ratio. This is because they are less prone to loops or aimless exploration and thus finish with step counts closer to the golden path.

H.3 Pass@K Experiment

As shown in Table 20, pass@k evaluation (at least one success out of k attempts) on the Base subset reveals that UI-TARS-1.5 achieves 69.03% (a 1.21x gain) with 3 retries and 73.87% (a 1.29x gain) with 5 retries. Overall, while all models benefit from retrying, the gains diminish as the number of retries increases.

I Error Analysis and Suggestions

In this section, we analysis the reasons for the Agent's errors, along with improvement suggestions.

1. Lack of Prior Knowledge of App Function Layout

Observation: We found that the Agent often searches for functions in the wrong places when performing tasks. For example, when it needs to "post a note," the Agent doesn't look for the posting entry on the homepage or notes page, but instead repeatedly searches in the personal center.

Suggestion: This indicates that the Agent lacks a common-sense understanding of the general layout of App functions. Future development requires a more comprehensive training trajectory collection scheme that covers more diverse page layouts, allowing the model to learn the general rules of "where a certain function is usually located."

2. Insufficient Exploration Ability, Single Path

Observation: The Agent is heavily influenced by high-frequency paths in the pre-training data. When faced with unfamiliar interfaces or complex tasks, it often only tries 1-2 of the most common paths. Once these paths fail, it doesn't try clicking on various possible UI elements and gradually eliminate incorrect options based on page feedback, unlike a human.

Suggestion: Reinforcement learning needs to be introduced, or a framework that simulates the human "trial and error elimination" process needs to be built to improve the agent's autonomous exploration ability in unknown environments.

3. Poor ability to recover from errors

Observation: This manifests in two ways: First, the agent often doesn't realize that a step might be wrong, causing it to stray further down the wrong path; second, when an operation fails to reach the expected page (e.g., an invalid click), the agent easily gets stuck in a loop of repeatedly trying the same invalid operation until it runs out of steps.

Suggestion: In addition to introducing reinforcement learning, a more effective state tracking mechanism needs to be built, allowing the agent to learn the error-correction logic of "if the current state does not match the expectation, then backtrack to the previous step."

4. "Approximately correct" bias in understanding the text of the task objective

Observation: Sometimes the agent only "roughly" understands the task, lacking the ability to verify precise constraints, leading to results that appear close but are actually incorrect. For example, when asked to search for "Trending List," the agent considers the task complete after finding the "Top Trending List"; when asked to enter a "Comic" interface, the agent mistakenly enters the corresponding novel interface and stops operating.

Suggestion: This is partly due to text understanding issues and partly due to a lack of fine-grained target validation. This requires fine-tuning the cue words, or introducing more "similar

but incorrect" negative samples during the SFT stage for refined training, teaching the agent to distinguish easily confused concepts, and adding a secondary validation of key elements before the task ends.

5. Insufficient Semantic Understanding of Specific UI Elements

Observation: We found that the agent has problems understanding certain UI icons, such as the "Scan" or "One-Click Cleanup" icons. Without textual cues, the agent may not be able to associate the icon with its function.

Suggestion: Semantic annotation of these key icons may be needed followed by specialized fine-tuning training.

6. Easily Remembers "Modular Patterns" and Ignores Dynamic Interface Changes

Observation: The agent mechanically applies fixed operating paradigms, ignoring the real-time state of interface components. For example, in the Tomato Novel's personal preference settings, the "Confirm" button is grayed out and unclickable if the preference hasn't been changed. Even if the preference has been changed, clicking "Confirm" once will make the button grayed out again (indicating an update). However, the agent didn't understand this logic; it only remembered the pattern of "clicking 'Confirm' at the end of setting attributes," so it repeatedly clicked the grayed-out button. Occasionally, it would accidentally change the preference and turn the button back on, and after turning it grayed out again, it would continue clicking the grayed-out button until it ran out of steps.

Suggestion: This case reveals the agent's blind spot in perceiving the dynamic states (enabled/disabled) of UI components. Future training needs to strengthen the model's understanding of "button state changes" and the underlying "system feedback," rather than simply imitating fixed click sequences.