

# ProBench: Benchmarking GUI Agents with Accurate Process Information

Leyang Yang<sup>1,2</sup>, Ziwei Wang<sup>1,2</sup>, Xiaoxuan Tang<sup>3</sup>, Sheng Zhou<sup>1\*</sup>,  
Dajun Chen<sup>3</sup>, Wei Jiang<sup>3</sup>, Yong Li<sup>3\*</sup>

<sup>1</sup>Zhejiang Key Laboratory of Accessible Perception and Intelligent Systems, Zhejiang University

<sup>2</sup>College of Computer Science and Technology, Zhejiang University

<sup>3</sup>Ant Group

{yangleyang, wangziwei98, zhousheng\_zju}@zju.edu.cn

{tangxiaoxuan.txx, chendajun.cdj, liyong.liy, jonny.jw}@antgroup.com

## Abstract

With the deep integration of artificial intelligence and interactive technology, Graphical User Interface (GUI) Agent, as the carrier connecting goal-oriented natural language and real-world devices, has received widespread attention from the community. Contemporary benchmarks aim to evaluate the comprehensive capabilities of GUI agents in GUI operation tasks, generally determining task completion solely by inspecting the final screen state. However, GUI operation tasks consist of multiple chained steps while not all critical information is presented in the final few pages. Although a few research has begun to incorporate intermediate steps into evaluation, accurately and automatically capturing this process information still remains an open challenge. To address this weakness, we introduce **ProBench**, a comprehensive mobile benchmark with over 200 challenging GUI tasks covering widely-used scenarios. Remaining the traditional State-related Task evaluation, we extend our dataset to include Process-related Task and design a specialized evaluation method. A newly introduced Process Provider automatically supplies accurate process information, enabling precise assessment of agent’s performance. Our evaluation of advanced GUI agents reveals significant limitations for real-world GUI scenarios. These shortcomings are prevalent across diverse models, including both large-scale generalist models and smaller, GUI-specific models. A detailed error analysis further exposes several universal problems, outlining concrete directions for future improvements.

## 1 Introduction

With the deep integration of artificial intelligence (Achiam et al. 2023; OpenAI 2023; Wang et al. 2024) and interactive technology, GUI Agent, as the carrier connecting natural language and real-world devices, has received widespread attention from the community (Yao et al. 2024; Hong et al. 2024; Bai et al. 2024; You et al. 2025; Wang et al. 2025). Early studies (Bai et al. 2021; Li et al. 2020; Hsiao et al. 2022) concentrated on single-image problems, such as grounding and single-screen reasoning. As planning, reasoning and memory capabilities have advanced, recent works (Wu et al. 2024; Lin et al. 2025) have explored more complex operation tasks with multiple steps.

\*Corresponding Authors.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

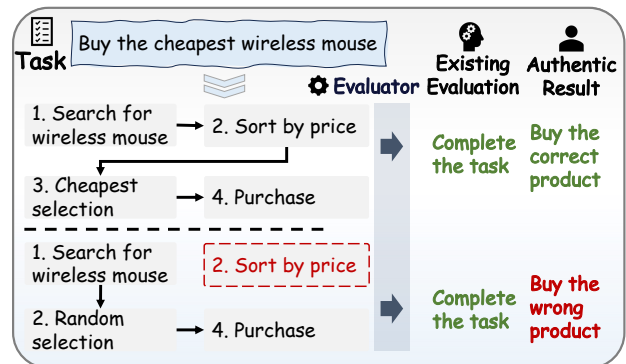


Figure 1: Illustration of a false outcome under existing evaluation. Red border indicates the ignored critical action.

However, by overlooking the inherently multi-step character of operation tasks, current GUI agents are typically evaluated only on the final screen state, leading to sub-optimal performance in practice. Specifically, existing evaluations (Lee et al. 2024; Rawles et al. 2024; Xu et al. 2024) generally determine the task completion exclusively relying on the final screen state, while not all critical information is contained in the final state. Consider the illustrative task “Buy the cheapest wireless mouse” in Figure 1, a crucial step is whether the agent explicitly complete “Sort by price from low to high” before selecting. If the sorting step is omitted, the subsequent selection is effectively random, unlikely to choose the lowest-priced product. Under existing evaluation, both the properly sorted trajectory and the unsorted, erroneous trajectory are judged equally successful, because of the final screen showing a wireless mouse in the purchase confirmation view in either case.

Recently, a few research has begun to consider intermediate process information into assessments. SPA-Bench (Chen et al. 2024) downplays the importance of process information by explicitly decomposing high-level instructions into a rigid sequence of steps and guiding the agent to follow the pre-defined trajectory. Such a design neglects the evaluation of task planning capabilities. Furthermore, manually annotating inspectable intermediate states limits the task’s scalability. A3 (Chai et al. 2025) calls an LLM to decompose tasks into several essential states and judges the completion

of subtasks through step-level comparisons, declaring the task complete only when all subtasks are fulfilled. However, the accuracy of this approach is constrained by the inherent limitations of existing LLM task decomposition. Thus, accurately and automatically providing process information remains an open challenge.

To address this weakness, we introduce **ProBench**, a comprehensive mobile benchmark with over 200 challenging GUI tasks. ProBench covers widely-used scenarios of 34 mainstream Chinese and English online applications across media, news, social, shopping, etc. For comprehensive evaluation, we retain the essential State-related Task, where all necessary information appears on the final screenshot. Therefore, performance can still be evaluated solely from the final screen state. More importantly, to better reflect real-world scenarios, we include Process-related Task that place higher demand on the operation process. These tasks cannot be judged by the final screen alone. Accurate evaluation must consider both the final state and the critical operation steps (e.g., applying a price filter or specifying a delivery address). To supply this information automatically, we introduce a Process Provider. Including Structure Description Converter which parses page hierarchy information while MLLM-based Summarizer employs MLLM to detect and summarize the changes by comparing screens, the two optional components provide process information accurately. With the above design, ProBench enables to evaluate the ability of GUI agents to rigorously capture and execute the necessary operation process.

Our evaluation of advanced GUI agents on ProBench reveals significant limitations for real-world GUI scenarios. These shortcomings are prevalent across diverse models, including both large-scale generalist models and smaller, GUI-specific models. Besides, we figure out that GUI agents struggle with applications of social and lifestyle categories. An in-depth error analysis further uncovers several universal problems, offering directions for future improvements. Our contributions are summarized as follows:

- We introduce ProBench, a comprehensive mobile benchmark with over 200 challenging GUI tasks, covering widely-used scenarios across bilingual online apps.
- We design an automated evaluation pipeline, which provides critical process information through Process Provider, achieving efficient and accurate evaluation.
- The evaluation on ProBench reveals significant limitations in real-world scenarios, both large-scale generalist models and smaller, GUI-specific models. We also provide further analysis to uncover universal problems.

## 2 Related Work

### 2.1 Static GUI Benchmark

RICO (Deka et al. 2017) marked an important milestone in GUI-related research by providing a basic dataset for GUI element classification and detection. Afterwards, UGIF (Venkatesh, Talukdar, and Narayanan 2022) introduced instruction-based GUI control task. AITW (Rawles et al. 2023) expanded the field with a large-scale dataset,

Benchmark	Online	Chinese	MV Free	AP Process
AndroidArena	✗	✗	✗	✗
AndroidWorld	✓	✗	✗	✗
B-MoCA	✓	✗	✗	✗
AndroidLab	✗	✗	✗	✗
SPA-BENCH	✓	✓	✓	✗
A3	✗	✗	✓	✗
<b>ProBench</b>	✓	✓	✓	✓

Table 1: **Comparison of ProBench with other dynamic GUI benchmarks.** Online means whether includes third-party online apps and Chinese means whether includes Chinese apps. MV Free means whether the evaluation phase is free of manual verification participation. AP Process means whether capture precise process information automatically.

while suffering from instruction redundancy and frequent mislabeling. AITZ (Zhang et al. 2024) refined AITW by applying Chain-of-Action-Thought reannotation and got a cleaner but smaller dataset. AndroidControl (Li et al. 2024) further introduced a dataset with simpler tasks and unique action space compared to AITW. However, these static benchmarks evaluate agents by one-step operation based on single screenshot and correct action history. Without realistic historical actions, the method ignores the inherently dynamic and interactive feature of real-world environment. In addition, one-step error directs to task failure, leaving no opportunity to assess the capability of reflection or recovery.

### 2.2 Dynamic GUI Benchmark

Researchers have developed a range of dynamic evaluation systems to better approximate real-world environment. AndroidArena (Xing et al. 2024) focuses exclusively on Google and built-in system apps, limiting its capacity to evaluate the general performance of third-party apps. B-Moca (Lee et al. 2024) incorporates applications such as Walmart, while the tasks are overly simplistic and lack diversity. AndroidWorld (Rawles et al. 2024) increases task diversity by selecting open-source applications from F-Droid, while discrepancies in design formula remain between these applications and mainstream software. AndroidLab (Xu et al. 2024) introduces additional variety by leveraging offline and static applications. Concurrently, these benchmarks predominantly depend on element matching (Lee et al. 2024) or pre-defined detection code (Xu et al. 2024) on the final screen, omitting the evaluation of the crucial intermediate steps. Realizing the shortcoming, SPA-BENCH (Chen et al. 2024) and A3 (Chai et al. 2025) consider intermediate process information into assessments, while introducing MLLMs to assist the evaluation. Nevertheless, these evaluations cannot capture process information accurately and automatically. ProBench addresses this gap by building a dedicated evaluation pipeline while capturing process information using Process Provider. Table 1 compares our work with existing

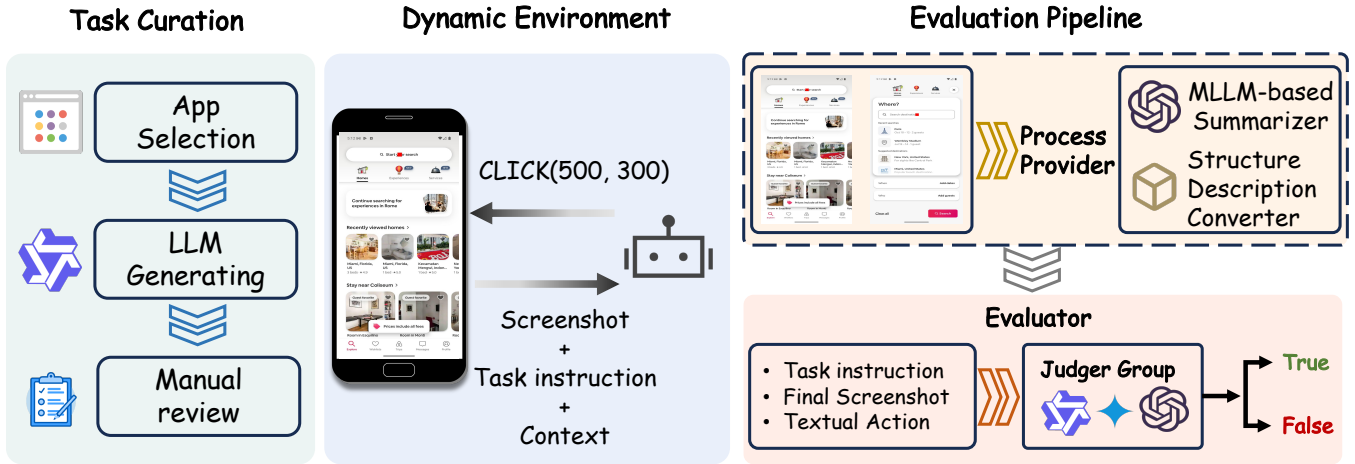


Figure 2: **Overview of our ProBench benchmark.** ProBench is a comprehensive mobile benchmark, comprising three key modules: (i) **Task Curation:** We select 34 mainstream bilingual applications, generate candidate tasks using LLMs, and refine them through manual review. (ii) **Dynamic Environment:** Agents complete the specified tasks by controlling the device. (iii) **Evaluation Pipeline:** For Process-related Task, we optionally choose either the Structure Description Converter or the MLLM-based Summarizer of Process Provider to supply process information. The final evaluation is performed by the judger selected from the judger group.

dynamic GUI agent benchmarks, highlighting our innovations and unique contributions.

### 3 ProBench

#### 3.1 Overview

Serving as the communication bridge between GUI agent and device, ProBench is implemented based on adbutils, an open-source python Android Debug Bridge(ADB) library, as illustrated in Figure 2.

ProBench acquires the current screen state by capturing the real-time screenshot. The screenshot, together with the task instruction and context information (such as historical operation records), is transmitted to the agent. Upon receiving these inputs, the agent predicts the subsequent specific operation. Textual operations are relayed back to ProBench, which parses and converts them into device control commands. These commands are executed via adbutils to manipulate the device. This iterative process continues until the agent confirms task completion or the system-defined maximum step limit is reached.

During the evaluation phase, evaluator receives the task instruction and final screenshot of the execution process. Concurrently, if the task is Process-related, users are afforded the flexibility to choose Structure Description Converter or MLLM-based Summarizer from Process Provider to provide accurate process information. The evaluator make final judgments under the input assistance.

#### 3.2 Task Curation

Starting from the application list provided by SPABENCH (Chen et al. 2024), we remove (i) English applications that cannot be executed on virtual devices and (ii) applications that embed bot-detection or other anti-automation

mechanisms.

The filtering process yields 34 target applications contained 14 English and 20 Chinese, covering a wide spectrum of categories for potential GUI scenarios. For every application we first compose a handful of seed tasks manually. Leveraging the generative capacity of Qwen3 (Yang et al. 2025), we then expand these seeds into a larger candidate pool. All automatically generated tasks undergo human filter and editing to guarantee correctness and diversity. In particular, we make efforts to reduce lexical and structural overlap between instructions so that agents must exhibit genuine capabilities rather than memorize recurring patterns. We define these operation tasks into two complementary types: State-related Task and Process-related Task, to ensure the diversity of our benchmark.

**State-related Task** focuses exclusively on the final screen state. The task is deemed successful if the requested information is visible on the screen in the end, regardless of the intermediate action sequence. For instance, in the task *"Query the current balance in Alipay"*, the task is successful once the correct balance amount is shown on the screen. The specific sequence of actions or operations employed is immaterial if the final state meets the task instruction.

**Process-related Task** demands a more comprehensive consideration: the agent must execute some specific operation process, not just reach the end state. Considering the instruction *"Find the highest-rated sushi restaurant in Tokyo and check its full menu"*, merely displaying the menu is insufficient. The agent must also perform a series of implicit operations that are not directly observable from the final screen state, such as locating Tokyo, sorting restaurants by ratings, and selecting sushi restaurants. Both the final screen and the intervening critical steps are therefore evaluated. Figure 3 shows the statistics of ProBench.

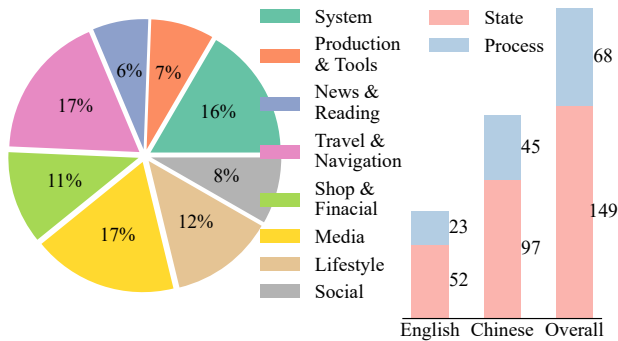


Figure 3: **Statistics of ProBench.** The left illustrates the diverse application categories included. The right displays the amount of each type tasks within different languages.

### 3.3 Action Space

In alignment with the action space defined in AITW, ProBench inherits the primitive actions, including `CLICK`, `SWIPE`, `TYPE`, `ENTER`, `BACK`, `COMPLETE`. Given that all tasks are confined to single application, we remove the `HOME` action from the action space, disallowing navigation to other applications. Considering the inherent challenges posed by network latency in online services, we additionally introduce a `WAIT` action to handle situations where network resources are slow to load, allowing the system to pause execution.

### 3.4 Evaluation Pipeline

We define three possible final outcomes in our evaluation pipeline: (i) **Uncompleted** (the agent reaches the step budget without emitting the completion signal), (ii) **Failure** (the agent does emit completion, but the completed actions does not satisfy the task requirements), and (iii) **Success** (the agent emits completion and achieves the task goal).

For State-related Task, we retain the evaluation method by relying solely on the final screen state. Once the agent issues the task completion, the evaluator determine whether the task instruction is fulfilled via the last screenshot.

As noted in Section 3.2, Process-related Task relies on both the final state and the intermediate operations which is called process information. To provide precise process information, we develop the Process Provider composed with two optimal components:

- Structure Description Converter** As shown in Algorithm 1, after every click, we parse the `ally` tree and locate the smallest clickable node enclosing the click coordinates. We record its `text` and `content_desc` attributes. If both are empty, we extract the `resource_id`, which often indicates the functional hint such as `filter_icon`, then supplement it with details of relevant child nodes. The resulting string serves as a compact, human-readable description of the action.
- MLLM-based Summarizer** Because the structure information can be noisy or missing arising from software design variances, we include MLLM-based Summarizer,

#### Algorithm 1: Convert Structure Information to Description

**Input:** `ally_tree`, `click_coordinate`

**Output:** `desc`

```

1: structure ← parse_tree(ally_tree)
2: node ← get_minimum(structure, click_coordinate)
3: desc ← parse_description(node)
4: if desc is empty then
5:   desc ← parse_resourceid(node)
6:   for child in node do
7:     desc ← desc + parse_description(child)
8:   end for
9: end if
10: return desc

```

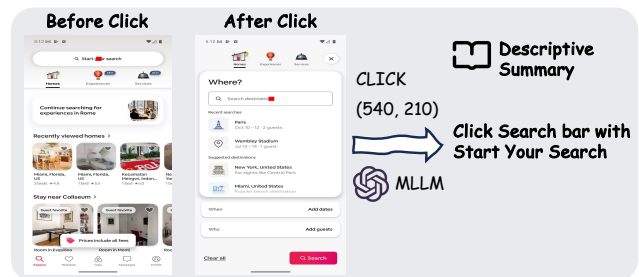


Figure 4: The implementation example of MLLM-based Summarizer. Take clicking the search box on the homepage of Airbnb as the example.

employing MLLM to compare the screenshots before and after the action to generate a descriptive textual summary. Specifically, we merge the two screenshots before and after the `CLICK` action and mark the click coordinate in the image. MLLM-based Summarizer compare the screenshots and generates a descriptive textual summary of the performed operation, as shown in Figure 4.

The evaluator receives the complete sequence of textualized actions together with the final screenshot to judge whether the key procedure has been performed and the task completed correctly.

A comparison with human annotations confirms the reliability of our evaluation method. Detailed accuracy are reported in Table 2.

Evaluation Method	Task Type	Correct
Evaluator	State-related	96.0%
Evaluator + Structure Description Converter	Process-related	89.7%
Evaluator + MLLM-based Summarizer	Process-related	94.1%

Table 2: **The correctness of our evaluation pipeline.** We choose Gemini 2.5 Pro (Comanici et al. 2025) as the MLLM for both the Evaluator and the Summarizer. "Correct" represents the correctness determined by human.

Model	English			Chinese			Overall		
	ST	PT	Avg.	ST	PT	Avg.	ST	PT	Avg.
<i>Proprietary Models</i>									
GPT-4o	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Claude 4 Sonnet	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Gemini 2.5 Pro	44.2	21.7	37.3	<b>46.4</b>	<b>31.1</b>	<b>41.5</b>	<b>45.6</b>	<b>27.9</b>	<b>40.1</b>
<i>General Open-source Models</i>									
Qwen2.5-VL-7B	7.7	0.0	5.3	6.2	2.2	4.9	6.7	1.5	5.1
Qwen2.5-VL-32B	26.9	8.7	21.3	14.4	13.3	14.1	18.8	11.8	16.6
Qwen2.5-VL-72B	<b>63.5</b>	<b>30.4</b>	<b>53.3</b>	28.9	26.7	28.2	40.9	<b>27.9</b>	36.9
InternVL3-8B	13.5	0.0	9.3	3.1	2.2	2.8	6.7	1.5	5.1
<i>GUI-specific Models</i>									
UI-TARS-1.5-7B	26.9	8.7	21.3	3.1	0.0	2.1	11.4	2.9	8.8
UI-R1-E-3B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
GUI-R1-3B	15.4	0.0	10.7	10.3	0.0	7.0	12.1	0.0	8.3

Table 3: Evaluation results of advanced GUI agents. The highest model performance in each column is highlighted in pink. ST is short for State-related Task and PT is short for Process-related Task.

## 4 Experiments

### 4.1 Models

In order to comprehensively evaluate the performance of current GUI agents, we select several representative agents covering three categories: proprietary models, general open-source models and GUI-specific models. For proprietary models, we choose GPT-4o (Hurst et al. 2024), Claude 4 Sonnet (Drolet et al. 2023) and Gemini 2.5 Pro (Comanici et al. 2025). For general open-source models, we choose Qwen2.5-VL (Bai et al. 2025) with 7B, 32B, 72B and InternVL3-8B (Zhu et al. 2025). For GUI-specific models, we choose UI-TARS-1.5-7B (Qin et al. 2025), UI-R1-E-3B (Lu et al. 2025) and GUI-R1-3B (Luo et al. 2025). The template of each model will be shown in Appendix. We do not incorporate Set-of-Mark (Yang et al. 2023), so the model output coordinates directly.

### 4.2 Evaluation Setup

English applications are run on the Android emulator<sup>1</sup> and Chinese applications are executed on a physical Android phone equipped with AdbKeyboard<sup>2</sup> to enable Chinese-character input. Before every execution, we manually clear all in-app histories to guarantee a consistent initial state. Each task is allotted a maximum of 15 interaction steps. For evaluation, we employ the Structure Description Converter to supply process information, and use Gemini 2.5 Pro as the judge in evaluator.

### 4.3 Main Results

**Overall performance** Table 3 reports the evaluation results on our ProBench. Gemini 2.5 Pro ranks first with

<sup>1</sup><https://developer.android.com/studio/run/emulator>

<sup>2</sup><https://github.com/senzhk/ADBKeyboard>

an average accuracy of 40.1%, confirming its strong comprehensive capability. Regardless of model size or training paradigm, performance is consistently higher on State-related Task than on Process-related Task, indicating that the latter which require dynamic attention to process, place higher demand on GUI agent. However, even for the simpler State-related Task, current agents still struggle: only Qwen2.5-VL-72B surpasses 60 % accuracy on English State-related Task, highlighting the substantial room for improvement in GUI agents for real-world GUI scenarios.

**General open-source models exhibit a significant scaling effect.** As the scale of model parameters increases, the Qwen2.5-VL series improves steadily, reaching an overall accuracy of 36.9%, closing the gap with the level of top proprietary model. In English operation tasks, the 72B variant achieves an average accuracy of 53.3%, outperforming every other model. These results underline that, in GUI scenarios, a larger model scale can yield substantial performance gains.

**GUI-specific models exhibit limited generalization.** UI-R1-E-3B fails to complete even a single task, related to the lack of COMPLETE action examples in its training phase. The agent never issues a completion signal even if it has completed the task correctly. UI-TARS-1.5-7B, while markedly outperforming its base model Qwen2.5-VL-7B on English tasks, still lags far behind large-parameter general models with good grounding capability. These observations suggest that task-specific fine-tuning or reinforcement learning on static datasets has a natural gap with the real environment and cannot bridge deficiencies in basic vision language understanding. Although domain-specific data promotes performance improvement, robust and strong general foundational capabilities remain the decisive factor.

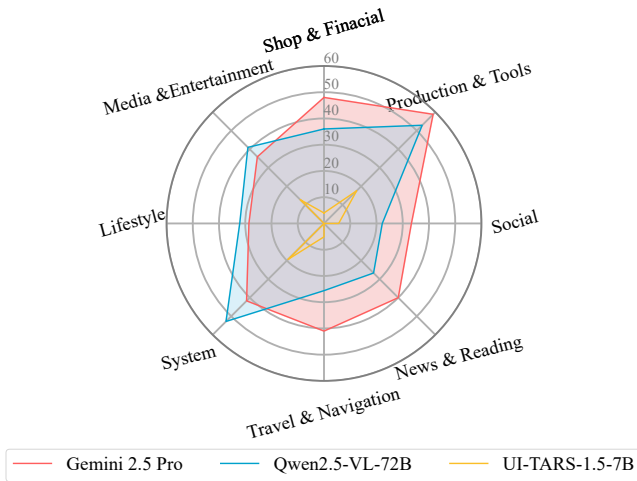


Figure 5: Accuracies of GUI agents on ProBench among different application categories. We demonstrate the best proprietary, general open-source and GUI-specific models.

#### 4.4 Performance on Application Categories

Figure 5 reports accuracy by application category. The agents generally perform better on production tools and system applications, while struggling with social-networking and lifestyle applications, the very domains that most closely match daily information-retrieval requests. Typical user tasks include "show the latest direct message from user X", "how long until my food arrives", or "where is my package right now". Several interface properties explain the gap. Social and lifestyle applications refresh content frequently and display information in a highly fragmented manner than relatively static pages like shopping or system settings: icon-only buttons, deeply nested folding cards, and advertising pop-ups crowd the limited screen estate. Efficient signals are sparse, while visual distractors abound, making it difficult for current GUI agents to complete instructions, ultimately depressing the overall accuracy.

### 5 Error Analysis

To illustrate the obstacles encountered by agents during execution, we present several error cases from our evaluation and distill them into universal problems.

#### 5.1 Insufficient Grounding Capability

Grounding capability, the ability to accurately interpret and locate GUI elements, serves as the foundation for successful completion of various GUI tasks. However, most proprietary models lack pre-training on GUI-related data and consequently show obvious deficiencies in grounding, seriously constraining the potential for performance improvement.

GPT-4o and Claude 4 Sonnet perform poorly in our evaluation, primarily due to their limited grounding capability. A critical barrier to task progression is their inability to accurately locate GUI elements, resulting in difficulties generating precise coordinates. As illustrated in Figure 6, Claude attempts to click the search box on the Airbnb homepage

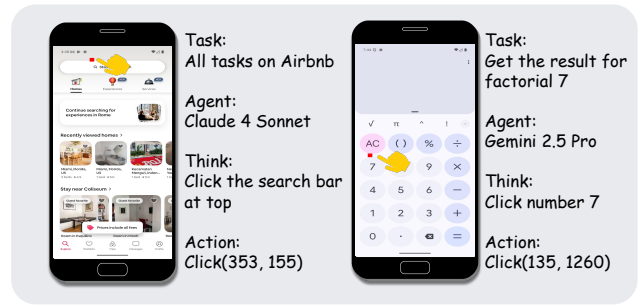


Figure 6: The lack of grounding capability in GUI agents. We show the accurate screen location where the agent actually clicked after coordinate conversion.

but fails to accurately locate it. Unable to complete the initial step ultimately leads to the failure of all tasks on Airbnb. Gemini 2.5 Pro also experiences shortcomings in grounding capability. When assigned the task "Get the result for factorial 7", Gemini is unable to correctly select the number 7, directly causing the task to fail. These cases demonstrate that current mainstream proprietary models are generally insufficient in grounding ability, largely due to the absence of pre-training on specialized GUI-related knowledge.

#### 5.2 Insensitive to Historical Operation

Compared to State-related Task, Process-related Task imposes higher demands on the tracking of historical operations, where the final screen state usually fails to fully represent all the requirements of the task. Agents not only need to comprehend the current screen state, but also integrate previous actions to accurately determine whether the task is truly completed. In other words, only by thoroughly considering the historical operations can agents make correct judgments regarding task completion.

However, most models fail to adequately attend to historical operations. As shown in Figure 7, we use the example of Qwen2.5-VL-72B performing the task "Find apartments in New York City and filter with 3 bedrooms", focusing particularly on its attempts to complete "filter with 3 bedrooms". In practice, due to Qwen's inability to effectively utilize historical operation information, it does not recognize that the filtering operation has already been completed after applying the filtering conditions. As a result, the model repeatedly clicks the filter button, leading to an execution loop and eventual task failure.

During our experiments, we implement an **early stop** mechanism: if the model outputs the same operation five consecutive times, the task is marked as failure in advance. We record the proportion of uncompleted tasks among all failed tasks, as well as the proportion of early-stopped tasks among all uncompleted tasks. The detailed results are presented in Table 4. Across all models, the early stop ratio remains high. This phenomenon indicates that when the model repeatedly performs the same action, it lacks the ability to recognize and reflect on its ineffective behavior and make appropriate adjustments.

Additionally, we observe that GUI-specific models fre-



## Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No.62372408). This work was supported by Ant Group Research Fund.

## References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Bai, C.; Zang, X.; Xu, Y.; Sunkara, S.; Rastogi, A.; Chen, J.; and y Arcas, B. A. 2021. UIBert: Learning Generic Multimodal Representations for UI Understanding. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 1705–1712. ijcai.org.
- Bai, H.; Zhou, Y.; Pan, J.; Cemri, M.; Suhr, A.; Levine, S.; and Kumar, A. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 12461–12495.
- Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.
- Chai, Y.; Li, H.; Zhang, J.; Liu, L.; Liu, G.; Wang, G.; Ren, S.; Huang, S.; and Li, H. 2025. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149*.
- Chen, J.; Yuen, D.; Xie, B.; Yang, Y.; Chen, G.; Wu, Z.; Yixing, L.; Zhou, X.; Liu, W.; Wang, S.; et al. 2024. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*.
- Comanici, G.; Bieber, E.; Schaekermann, M.; Pasupat, I.; Sachdeva, N.; Dhillon, I.; Blistein, M.; Ram, O.; Zhang, D.; Rosen, E.; 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*.
- Deka, B.; Huang, Z.; Franzen, C.; Hibschan, J.; Afergan, D.; Li, Y.; Nichols, J.; and Kumar, R. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, 845–854.
- Drolet, L.; Caron, P.-O.; Forget, J.; and Turcotte Jean-Robert, G. 2023. Claude4. *CERTIFICATS D’ABSENCE POUR RAISONS MÉDICALES: QUELS SONT LES RÔLES ET LES DÉFIS DU MÉDECIN TRAITANT? UN PORTRAIT DESCRIPTIF DE LA RÉALITÉ AU QUÉBEC*, 19.
- Hong, W.; Wang, W.; Lv, Q.; Xu, J.; Yu, W.; Ji, J.; Wang, Y.; Wang, Z.; Dong, Y.; Ding, M.; et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14281–14290.
- Hsiao, Y.-C.; Zubach, F.; Baechler, G.; Carbune, V.; Lin, J.; Wang, M.; Sunkara, S.; Zhu, Y.; and Chen, J. 2022. Screenqa: Large-scale question-answer pairs over mobile app screenshots. *arXiv preprint arXiv:2209.08199*.
- Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Lee, J.; Min, T.; An, M.; Hahm, D.; Lee, H.; Kim, C.; and Lee, K. 2024. Benchmarking Mobile Device Control Agents across Diverse Configurations. *arXiv:2404.16660*.
- Li, W.; Bishop, W.; Li, A.; Rawles, C.; Campbell-Ajala, F.; Tyamagundlu, D.; and Riva, O. 2024. On the effects of data scale on computer control agents. *arXiv e-prints*, arXiv–2406.
- Li, Y.; Li, G.; He, L.; Zheng, J.; Li, H.; and Guan, Z. 2020. Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 5495–5510.
- Lin, K. Q.; Li, L.; Gao, D.; Yang, Z.; Wu, S.; Bai, Z.; Lei, S. W.; Wang, L.; and Shou, M. Z. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 19498–19508.
- Lu, Z.; Chai, Y.; Guo, Y.; Yin, X.; Liu, L.; Wang, H.; Xiao, H.; Ren, S.; Xiong, G.; and Li, H. 2025. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.
- Luo, R.; Wang, L.; He, W.; and Xia, X. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.
- OpenAI. 2023. GPT-4V(ision) system card. *CoRR*.
- Qin, Y.; Ye, Y.; Fang, J.; Wang, H.; Liang, S.; Tian, S.; Zhang, J.; Li, J.; Li, Y.; Huang, S.; et al. 2025. UI-TARS: Pioneering Automated GUI Interaction with Native Agents. *arXiv preprint arXiv:2501.12326*.
- Rawles, C.; Clinckemahillie, S.; Chang, Y.; Waltz, J.; Lau, G.; Fair, M.; Li, A.; Bishop, W.; Li, W.; Campbell-Ajala, F.; et al. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.
- Rawles, C.; Li, A.; Rodriguez, D.; Riva, O.; and Lillicrap, T. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36: 59708–59728.
- Venkatesh, S. G.; Talukdar, P.; and Narayanan, S. 2022. Ugif: Ui grounded instruction following. *arXiv preprint arXiv:2211.07615*.
- Wang, P.; Bai, S.; Tan, S.; Wang, S.; Fan, Z.; Bai, J.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Fan, Y.; Dang, K.; Du, M.; Ren, X.; Men, R.; Liu, D.; Zhou, C.; Zhou, J.; and Lin, J. 2024. Qwen2-VL: Enhancing Vision-Language Model’s Perception of the World at Any Resolution. *arXiv preprint arXiv:2409.12191*.
- Wang, Z.; Chen, W.; Yang, L.; Zhou, S.; Zhao, S.; Zhan, H.; Jin, J.; Li, L.; Shao, Z.; and Bu, J. 2025. Mp-gui: Modality perception with mllms for gui understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 29711–29721.

Wu, Z.; Wu, Z.; Xu, F.; Wang, Y.; Sun, Q.; Jia, C.; Cheng, K.; Ding, Z.; Chen, L.; Liang, P. P.; et al. 2024. OS-ATLAS: A Foundation Action Model for Generalist GUI Agents. *arXiv preprint arXiv:2410.23218*.

Xing, M.; Zhang, R.; Xue, H.; Chen, Q.; Yang, F.; and Xiao, Z. 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*, 6061–6072.

Xu, Y.; Liu, X.; Sun, X.; Cheng, S.; Yu, H.; Lai, H.; Zhang, S.; Zhang, D.; Tang, J.; and Dong, Y. 2024. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, J.; Zhang, H.; Li, F.; Zou, X.; Li, C.; and Gao, J. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Yao, Y.; Yu, T.; Zhang, A.; Wang, C.; Cui, J.; Zhu, H.; Cai, T.; Li, H.; Zhao, W.; He, Z.; et al. 2024. MiniCPM-V: A GPT-4V Level MLLM on Your Phone. *arXiv preprint arXiv:2408.01800*.

You, K.; Zhang, H.; Schoop, E.; Weers, F.; Swearngin, A.; Nichols, J.; Yang, Y.; and Gan, Z. 2025. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, 240–255. Springer.

Zhang, J.; Wu, J.; Teng, Y.; Liao, M.; Xu, N.; Xiao, X.; Wei, Z.; and Tang, D. 2024. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Zhu, J.; Wang, W.; Chen, Z.; Liu, Z.; Ye, S.; Gu, L.; Tian, H.; Duan, Y.; Su, W.; Shao, J.; et al. 2025. Internv13: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.