D-GARA: A Dynamic Benchmarking Framework for GUI Agent Robustness in Real-World Anomalies

Sen Chen[1*], Tong Zhao[1*], Yi Bin[1†], Fei Ma[2], Wenqi Shao[3], Zheng Wang[1]
[1]Tongji University
[2]Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ)
[3]Shanghai AI Laboratory
cs.sen@hotmail.com, zt.tong@hotmail.com, yi.bin@hotmail.com

Abstract

Developing intelligent agents capable of operating a wide range of Graphical User Interfaces (GUIs) with human-level proficiency is a key milestone on the path toward Artificial General Intelligence. While most existing datasets and benchmarks for training and evaluating GUI agents are static and idealized, failing to reflect the complexity and unpredictability of real-world environments, particularly the presence of anomalies. To bridge this research gap, we propose D-GARA, a dynamic benchmarking framework, to evaluate Android GUI agent robustness in real-world anomalies. D-GARA introduces a diverse set of real-world anomalies that GUI agents commonly face in practice, including interruptions such as permission dialogs, battery warnings, and update prompts. Based on D-GARA framework, we construct and annotate a benchmark featuring commonly used Android applications with embedded anomalies to support broader community research. Comprehensive experiments and results demonstrate substantial performance degradation in state-of-the-art GUI agents when exposed to anomaly-rich environments, highlighting the need for robustness-aware learning. D-GARA is modular and extensible, supporting the seamless integration of new tasks, anomaly types, and interaction scenarios to meet specific evaluation goals.

Code — https://sen0609.github.io/D-GARA
Extended version — https://arxiv.org/abs/2511.16590

1 Introduction

A GUI agent is designed to operate a wide variety of Graphical User Interfaces (GUIs), simulating human interaction through visual interface understanding, task planning, and action execution. This capability represents a crucial milestone toward the vision of Artificial General Intelligence. In early approaches, GUI agents relied on predefined templates, empirical rules, or heuristic search trees to accomplish simple tasks, which are inflexible and difficult to generalize to real-world scenarios. Powered by Vision-Language Models (VLMs), recent GUI agents have made significant progress in executing complex tasks, demonstrating strong visual understanding and task planning abilities, particularly on a range of static benchmarks. However, such success in ideal environments often obscures their underlying fragility in dynamic and realistic settings. Real-world tasks are inherently more challenging, and execution environments are more complex, usually involving unpredictable events and interruptions, which significantly limits their practical deployment and applicability.

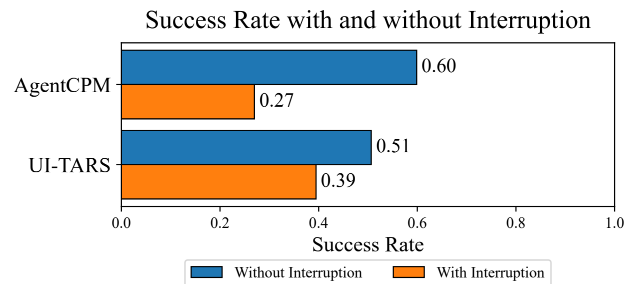

Figure 1: Comparison of task success rates for UI-TARS and AgentCPM under normal conditions and real-world interruptions.

However, current GUI agents largely ignore such interruptions and are typically trained on idealized datasets, *i.e.*, sequences of screenshots paired with corresponding actions to accomplish given tasks. To quantitatively investigate the impact of real-world interruptions on GUI agents, we evaluate two representative state-of-the-art models specifically designed for GUI interaction: UI-TARS-72B and AgentCPM-GUI-8B. As shown in Figure 1, both models experience a significant decline in task success rate when subjected to real-world anomalies, with performance degradation reaching up to 33%. Obviously, existing static and idealized benchmarks, *e.g.*, Android Control (Li et al. 2024), fail to expose the vulnerability of GUI agents or to assess their robustness in the presence of interruptions. In practice, application interfaces are dynamic, and actions of an agent may trigger unforeseen system events which would lead to different ways to reach the task goal. Although recent work has begun to introduce "anomalies" into evaluation settings (Yang et al. 2025), these efforts remain confined to static paradigms and fall short of simulating sud-

[*]These authors contributed equally.
[†]Corresponding author.
Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

den and process-altering events that arise during real-time interactions. AndroidWorld (Rawles et al. 2025) provides a dynamic benchmarking environment while the included applications and tasks are pure and simple, failing to simulate the complex real-world operation process. Therefore, a standardized framework is urgently needed to comprehensively evaluate agent robustness against anomalies in dynamic and real-world environments.

To fill this research gap, we propose to design a novel **D**ynamic benchmarking framework for **G**UI **A**gent **R**obustness in real-world **A**nomalies, dubbed **D-GARA**, tailored for Android GUI agents. The core idea of D-GARA is to transform the evaluation setting from static screenshots to a dynamic environment to simulates real-world interactions. Specifically, D-GARA integrates an Android simulator to construct a realistic Android environment for executing tasks. While a GUI agent performs tasks in this environment, D-GARA injects a series of high-frequency anomalies through a controllable injection system, including unexpected modal pop-ups, disruptive system alerts, and intermittent application crashes. Although GUI Robust (Yang et al. 2025) also provides a small portion of anomalous samples, its static nature makes it only capable of simulating simple anomalies by inserting a screenshot of an alert or pop-up into a normal task trajectory. These anomalies are easy to simulate and straightforward for agents to bypass. Agents can simply click through them and continue along the original execution path. Even in the case of a mistake, the benchmark provides a gold-standard screenshot for the next action prediction, essentially following a teacher-forcing paradigm similar to next-token prediction in language models. In contrast to these static settings, D-GARA steps further by introducing more complex and realistic anomalies that may lead to entirely different execution trajectories. For example, in real-world scenarios, certain anomalies may redirect the agent to unexpected screens outside the intended task flow. If the agent does not actively maintain awareness of the task goal, it can easily become misled by these unanticipated diversions. Such anomalies pose a significantly greater threat to task completion, as they can completely derail the agent from its intended plan. D-GARA is designed to simulate these severe anomalies, providing a more comprehensive evaluation of robustness and decision-making capabilities for GUI agent.

Besides, D-GARA facilitates the collection and annotation of dynamic benchmarking samples for Android GUI agents. It provides a dedicated tool named `datacollector`, which assists users in gathering human trajectories, including screenshots and XML files—for their tasks, while offering the flexibility to inject interruptions at any desired point. We also include several common interruptions by default, such as Low Battery Dialogs and Location Permission. Moreover, users can easily define and configure custom interruptions for specific Apps or to meet particular requirements. Leveraging these capabilities, we contribute a benchmark collected from several commonly used applications, each embedded with diverse interruptions, to benefit the research community. In addition, we propose a robustness metric to evaluate how well GUI agents perform when exposed to interruptions, and we assess several state-of-the-art agents to identify their limitations and potential areas for improvement.

In summary, the main contributions of this paper are as follows:

- We first explicitly consider interruptions during the real-world execution of GUI agents. To simulate such scenarios, we introduce D-GARA, a novel benchmarking framework for assessing GUI agent robustness under real-world anomalies, which also supports dynamic evaluation and real-world data collection.
- We collect and annotate one of the first benchmarks involving commonly used applications, intentionally triggering a wide range of real-world interruptions to examine how agents respond to unexpected events. This benchmark will be open-sourced to support the future research in the GUI agent community.
- We conduct extensive experiments on state-of-the-art GUI agents using D-GARA, aiming to comprehensively evaluate the impact of anomalies and assess the robustness of agents when faced with such interruptions. We also propose a robustness metric for this purpose. Our experimental results show that all current SOTA methods suffer from significant performance degradation, highlighting the urgent need for more robust solutions.

2 Related Work

2.1 GUI Agent

The rapid development of Multimodal Large Language Models (MLLMs) has made the GUI agent paradigm both feasible and increasingly popular.These models can be broadly categorized into two types: closed-source models, such as GPT-4 (Team 2024), Gemini 2.5 (Team 2025) and open-source models, including Qwen2.5-VL (Bai et al. 2025), InternVL (Chen et al. 2024) and MiniCPM (Yao et al. 2024).

Closed-source models typically exhibit superior general-purpose reasoning and planning capabilities, enabling them to better understand complex tasks and follow instructions. As a result, many GUI agent systems leverage these models as planners while relying on separate modules for grounding. Notable examples include GUI Actor (Wu et al. 2025) and U-Ground (Gou et al. 2025), both of which use powerful language models for high-level decision making while incorporating specialized grounding mechanisms to interact with the interface.

To address privacy concerns from handling sensitive data, recent work favors compact open-source models for local deployment, aiming to improve grounding and planning while ensuring practical usability. For example, UI-TARS-1.5 (Qin et al. 2025), an open-source multimodal agent built on top of Qwen2.5-VL (Bai et al. 2025), which incorporates reinforcement learning to enhance planning and reasoning capabilities during inference. Another representative model is AgentCPM-GUI (Zhang et al. 2025), developed by THUNLP, Renmin University of China, and ModelBest. It is optimized for on-device deployment and is built upon the

lightweight MiniCPM-V model (Yao et al. 2024). With reinforcement fine-tuning and pre-training on a large bilingual Android dataset named CAGUI (Zhang et al. 2025), it excels in grounding Chinese mobile UI elements and executing real-world tasks across more than 30 popular applications. These works reflect a growing emphasis on building GUI agents that are not only capable of understanding user interfaces, but also practical for real-world deployment through targeted post-training and efficient architectures.

2.2 GUI Benchmark

Benchmark datasets are essential for evaluating GUI agents, and can be broadly divided into two categories based on the core agent capabilities they target: grounding benchmarks and task-completion benchmarks. Grounding-focused benchmarks, such as ScreenSpot (Cheng et al. 2024), ScreenSpot-v2 (Wu et al. 2024) and ScreenSpot-Pro (Li et al. 2025), evaluate an agent's ability to localize UI elements within static screenshots. Task-completion benchmarks, such as Mind2Web (Deng et al. 2023) and GUI-Odyssey (Lu et al. 2024a), assess the agent's capability to perform multi-step interactions and complete realistic tasks using screenshots. Recently, several benchmarks have been proposed for dynamic GUI environments, such as Android World (Rawles et al. 2025) is a dynamic benchmarking environment which provide 116 tasks,but the problem is it is most of its tasks is pure and simple,and it don't support researchers to add their own tasks. and also it is a anomaly-free benchmark. Recent efforts such as GUI-Robust (Yang et al. 2025) exposes agent limitations under irregular GUI states but relies on static overlays (e.g. pop-ups, login pages) that are shallow and reversible. Real-world anomalies are dynamic and non-deterministic, often requiring real-time recovery and replanning. To overcome this, we propose D-GARA, a dynamic framework that injects anomalies during live Android task execution. It enables realistic, high-fidelity robustness evaluation beyond what static benchmarks can offer.

3 The D-GARA Framework

D-GARA incorporates interruption injection and success validation into the execution trajectory of agent to better simulate real-world disturbances and evaluate task completion. Figure 2 provides an overview of the framework, and the following section details each component.

3.1 Framework Overview

As shown in Figure 2, D-GARA is a modular execution framework that coordinates agent interaction, real-time anomaly injection, and success validation. The process begins by capturing a screenshot of the initial state and the corresponding UI layout (XML hierarchy) from the Android device. The framework then decides whether to inject an interruption based on the current state. If triggered, a new screenshot and XML file are collected and passed to the agent. With this multimodal input, the agent produces an action command, e.g., a tap or text entry, which is sent to the device via `ADB`. After a brief pause for the UI to stabilize, the framework evaluates the updated UI state. The above process forms an "execution cycle" in which two core modules operate sequentially: 1) *the anomaly trigger mechanism*, which determines whether a context-aware interruption is needed, and 2) *the success validation mechanism*, which checks whether the agent has reached the goal. Together, these mechanisms support D-GARA's integrated interruption triggering and success validation.

3.2 Interruption Trigger Mechanism

To make the interruption injection more natural and reasonable, we propose **Semantic Anomaly Triggering Mechanism**, which inspects the textual content of the XML file for specified keywords using a lightweight rule-based evaluator. Each rule defines a target condition through a set of keywords and an associated matching threshold. An anomaly is triggered only when the current XML file meets all required conditions. For example, as shown in Figure 2, a navigation-related rule may specify keywords such as Drive, Nearby, Metro, and Mine (translated from Chinese) in the UI, and it is triggered when the XML file contains at least three of these terms. Once the condition is met, the framework injects a predefined interruption, such as a permission dialog, system alert, or other interruptions. A concrete rule specification for this case is shown in Listing 1.

Listing 1: Anomaly injection rule for the Drive page.

```
1  - id: rule_drive_permission
2    conditions:
3      all:
4        - type: semantic_element_exists
5          keywords: ["Drive", "Nearby", "
               Metro", "Mine"]
6          threshold: 0.75
7    actions:
8      - type: inject_interference
9        interference_id: "
               location_permission_dialog"
10       follow_up:
11         accept: "redirect_to_settings"
12         deny: "terminate_app"
```

Building on this basic trigger mechanism, we design a two-stage anomaly handling pipeline that models both the interruption pop-up displayed on the screen and the follow-up actions after the agent responds. In Stage 1, the agent detects and handles a real dialog in the foreground, such as a location permission prompt. In Stage 2, D-GARA uses ADB commands to trigger the follow-up action based on the agent's choice, simulating the dynamic state transition and executing the corresponding system action. For example, when opening Amap, the system may display a location permission dialog with two options: Accept or Deny. If the agent accepts, the workflow redirects to the settings interface to enable the permission. If denied, the application terminates, since it cannot work without the location permission. These follow-up outcomes illustrate how a single interaction can lead to different execution paths, making anomaly han-

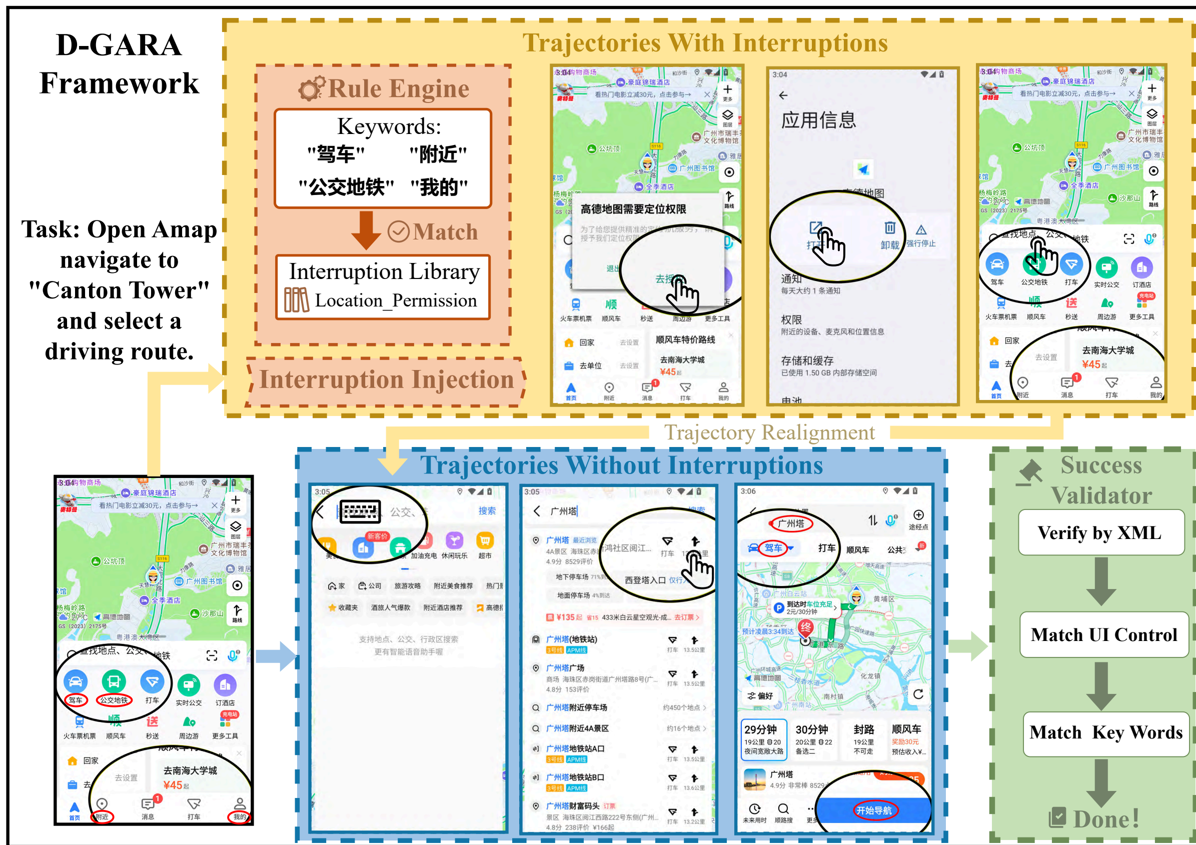

Figure 2: Pipeline of D-GARA Framework for Real-World GUI Agent Robustness Evaluation, Including Rule-Based Interruption Injection and Goal-State Validation

dling more realistic compared to static pop-ups.

To support flexible anomaly definitions, D-GARA places all trigger logic in external configuration files instead of hard coding it. Each anomaly rule prototype specifies the target activity, the triggering condition, the content to display, and the follow up actions. This separation makes the logic easy to modify while keeping the framework stable. Researchers can create or adjust anomaly scenarios by editing the configuration files, providing a modular and practical foundation for evaluating agent robustness.

3.3 Success Validation Mechanism

Traditional evaluations of GUI agents on static datasets assume a fixed action sequence: each screenshot corresponds to a specific action, and the task concludes once all frames are processed. In dynamic environments, however, these assumptions no longer apply, because actions may not be unique, states can evolve unpredictably, and there is often no clear signal for task completion or goal achievement. While the action space typically includes a ”done” option to indicate task completion, this signal is unreliable. Small models may fail to produce it even after completing the task, while larger models may misuse it and terminate prematurely. As a result, relying on model-generated signals does not guarantee consistent task verification across different models or task types. These limitations highlight the need for a state-centered validation strategy, where completion is judged based on the actual UI state rather than the agent’s self-reported signals.

Listing 2: A success validation rule for a video-like task.

```
1  task_drive_like_video:
2    conditions:
3      - type: element_property_contains
4        selector: "resource-id=tv.app:id/
             like_button"
5        attribute: "content-desc"
6        value: "Liked"
```

To handle these issues, D-GARA introduces the *Success Validator*, a state centered evaluation module that verifies progress at every state update. After each agent action, the framework records a new screenshot and the corresponding XML view hierarchy. Before passing this state back to the agent, the validator inspects the XML content to determine whether it satisfies a declaratively defined goal condition. Listing 2 provides an example rule for a video related task. In this case, the validator checks whether the property of a specific UI element contains the value “Liked”, which indicates that the task has been successfully completed.

In general, the validator performs this check after every agent action. When the agent explicitly outputs a completion signal such as "done", the validator immediately verifies the result. If the goal condition is not satisfied at that moment, the task is stopped and counted as a failure.

A task is considered successful once the *Success Validator* confirms that the stabilized UI state matches the specified goal schema. These schemas describe the expected final interface, such as the presence of particular elements or the values of their properties. With this design, task success depends only on the final interface state rather than on the correctness of every intermediate action. In practice, this allows an agent to take detours, make temporary mistakes, or backtrack, as long as it eventually reaches the required goal state. This outcome centered design ensures that task completion is evaluated consistently across applications and environments. As a final safeguard, all automatically validated results are reviewed by human evaluators to ensure reliability in cases where automatic rules may be insufficient. In future work, we will explore more to involve large foundation model for this verification.

4 Benchmark Construction

Building upon the D-GARA framework, we construct **D-GARA-152**, a benchmark that includes 152 real world tasks across 7 widely used Android applications. While Section 3 introduced the general execution framework, this section explains how D-GARA is instantiated as a concrete benchmark through task selection, interruption design, goal specification, and evaluation metrics.

4.1 Benchmark Overview

D-GARA-152 focuses on applications that exhibit functional complexity, high user activity, and diverse interaction patterns, including e-commerce (e.g., JD.com, Amazon), social media (e.g., Weibo, Facebook), content consumption (e.g., Bilibili), navigation (e.g., Amap, Google Maps), and travel services (e.g., Ctrip). Task distribution is intentionally non-uniform, prioritizing high usage platforms to better reflect realistic agent workloads.

To emulate real-world disturbance patterns, each task is paired with one or more injected interruptions drawn from several representative categories. The interruption classification and injection strategy are detailed in Section 4.2. This strategy provides broad coverage of interruption types and keeps the scenarios aligned with real usage patterns.

4.2 Interruption Design Strategy

Our interruption classification consists of five categories that cover common disruption sources in mobile environments:

- **System Resource**: low-battery warnings, thermal throttling alerts
- **System Network**: Wi-Fi disconnection, mobile data switches
- **App Malfunction**: crashes, freezes
- **Permission Control**: runtime permission dialogs
- **UX Disruption**: update prompts, feedback forms

Interruption injection follows a template based approach. We design general dialog layouts in Android Studio and compile them into a standalone APK. At runtime, D-GARA fills these templates with values defined in the external configuration files, which allows a single layout to support many interruption scenarios. Using a dedicated APK keeps interruption logic isolated from the target applications and avoids the inconsistencies of native system dialogs across devices and Android versions. Because both the layout and the textual content are parameterized, new interruptions can be added by extending the configuration library without modifying the core framework, keeping the mechanism portable, reusable, and easy to extend.

4.3 Success Validation Design

Building on the `Success Validation` module (Section 3.3), adapting it to the 152 benchmark tasks required a systematic process supported by human review.

We used a lightweight `DataCollector` tool to record the target screenshot and XML file for each application after a human operator completed a representative task. The designer then examined this final XML file to identify stable and unambiguous indicators of success. These XML attributes are usually more stable than visual appearance, because they reflect the underlying structure of the interface rather than transient graphical elements. Many applications use the same interface to mark completion for all tasks of a given type. For example, the rule in Listing 2 captures the success condition for a "like" action, and this condition applies to every task in that category for the same application. Once such indicators are extracted, this extracted success condition can be reused across all related tasks without additional manual effort. This reuse greatly reduces the cost of building large task suites, because the designer does not need to craft separate conditions for individual tasks.

4.4 Evaluation Metrics

With task success defined through declarative rules, we next introduce the metrics used to evaluate agent performance under both baseline and interrupted conditions as follows:

Success Rate (SR) measures the proportion of tasks successfully completed under a given condition.

Robust Success Rate (RSR) quantifies robustness over tasks solvable in baseline conditions:

$$\text{RSR} = \frac{\left|\left\{i \mid SR^{(i)}_{\text{baseline}} = 1 \land SR^{(i)}_{\text{interruption}} = 1\right\}\right|}{\left|\left\{i \mid SR^{(i)}_{\text{baseline}} = 1\right\}\right|}.$$

RSR isolates interruption robustness from overall task-solving competence, enabling fair comparison among agents with different baseline capabilities. For instance, an agent may achieve a high SR yet exhibit a low RSR, indicating fragility under disturbances, whereas an agent with moderate SR but high RSR demonstrates stronger adaptability. Together, SR and RSR provide a comprehensive view of agent performance under real-world disruptions.

5 Results and Analysis

To assess and analyze the robustness of GUI agents under real-world interruptions, we conduct extensive experiments and in-depth evaluations using several state-of-the-art GUI agents and advanced multimodal large language models (MLLMs), *e.g.*, GPT-4o and Gemini2.5.

5.1 Overall Performance Impact

We first investigate the robustness of GUI agents, such as UI-TARS-1.5-72B (Qin et al. 2025) and AgentCPM-GUI-8B (Zhang et al. 2025), focusing on their ability to maintain task success in the presence of common interruptions and anomalies. In addition to such agents explicitly designed and trained for GUI tasks, we also evaluate several general MLLMs, including the powerful GPT-4o (Team 2024), Gemini-2.5 (Team 2025) , and Qwen2.5-VL-7B (Bai et al. 2025), as many agent frameworks adopt these models as their backbone, where they have demonstrated strong performance across a variety of tasks.

Model	SR (NoInt)	SR (WithInt)	RSR
Gemini2.5-flash	80.26%	68.42%	73.77%
GPT-4o	69.08%	60.53%	66.67%
Qwen2.5-VL-7B	69.08%	46.05%	53.33%
UI-TARS-1.5-72B	50.66%	39.47%	48.05%
AgentCPM-GUI-8B	59.87%	26.97%	39.56%

Table 1: Performance of models under interruption and non-interruption conditions

As the results shown in Table 1, all models exhibit a significant decrease in task success rates under interruption (comparison between the first two columns), with an average drop exceeding 17.5%. This indicates that none of the agents can effectively handle unforeseen events during execution. These findings also support our hypothesis that performance on static benchmarks does not reliably translate to robustness in dynamic real-world conditions. We observe that larger models, such as GPT-4o and Gemini-2.5, demonstrate stronger robustness and greater ability to withstand interruptions. This phenomenon may be attributed to their powerful planning capabilities, as most interruptions can be mitigated through effective planning, a skill in which larger models have been shown to excel in other domains, such as mathematical reasoning (Shi et al. 2024; Lin et al. 2023). From the RSR values in the last column, we observe that even AgentCPM-GUI and UI-TARS-72B, trained specifically for GUI interaction, still exhibit the weakest robustness when exposed to interruptions. This suggests that their training may primarily adapt the model to visually perceive the UI interface, while their planning ability remains heavily dependent on the underlying base model.

5.2 Robustness Across Interruption Types

As we know, different anomalies can lead to different execution paths, potentially disrupting perception, UI structure, or overall task flow. In other words, different types of interruptions vary in difficulty. To better analyze the effects

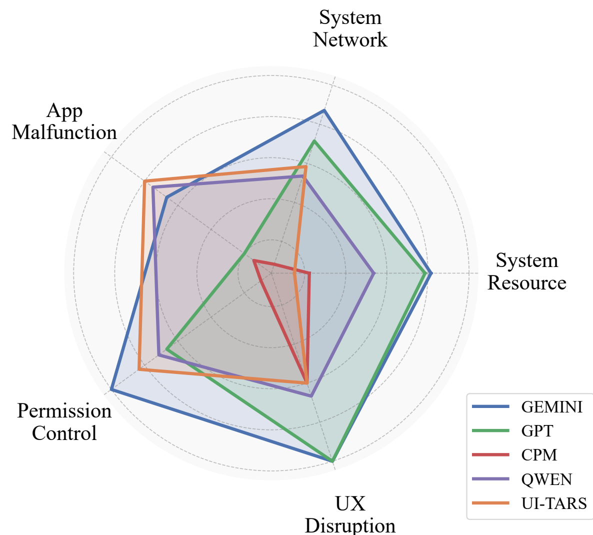

Figure 3: Robust success rate across five categories of GUI interruptions.

of different interruption types, as aforementioned, we group them into five representative categories based on their functional characteristics and system scope: *System Resource*, *System Network*, *App Malfunction*, *Permission Control*, and *UX Disruption*. Each category represents a distinct dimension of operational disruption, ranging from low-level system constraints to high-level user experience interruptions. The robust success rates (RSR) for each category are illustrated in Figure 3. As we can see, Gemini and GPT-4o significantly outperform other methods on *Permission Control* and *System Network*, even achieving 100% for *Permission Control*, likely because these types of interruptions can be resolved with common knowledge, a strength of both models. For APP-level interruptions, which often require more specialized knowledge of UI interactions, UI-TARS achieves notably higher performance than in other interruption types, as it is trained for such tasks. Even AgentCPM demonstrates strong performance peak in this category, although it ranks lowest overall.

Model	Dual-button RSR	Single-button RSR
Qwen2.5-VL-7B	96.15%	41.27%
AgentCPM-GUI-8B	82.35%	9.30%

Table 2: Robustness (RSR) comparison across models and interaction modes

During our experiments, we observed that many interruptions present multiple response options, *e.g.*, “Install Now” and “Close” in an APP update prompt, and agents typically choose “Close” to skip the interruption and continue the assigned task. Although this behavior demonstrates that agents can bypass interruptions, it makes us difficult to assess whether they can handle more complex paths. For example, selecting “Install Now” to complete the installation, and then return to the original task flow. To investigate this,

we designed a specific test mode in which only the complex choice is available, *e.g.*, “Install Now” here, forcing agents to follow the more challenging execution path. As shown in Table 2, the performance drops sharply when agents are required to handle interruptions through the complex path. This finding suggests that agents need to strengthen their ability to truly manage complex interruptions, rather than simply skipping them by choosing “Close”.

5.3 The Effects of Action Coordinate

Current GUI agents typically use both screenshots and XML element coordinates to ensure precise actions. While humans, as natural agents, rely solely on visual content to understand and execute tasks, as also applied in (Lu et al. 2024). To examine the impact of these modalities, we disentangle them and test two settings. In the *screenshot-only* setting (as shown in Table 3), agents receive only screenshots and are required to infer both the action and its coordinates. In the *screenshot*+XML setting, agents predict the action from screenshots but obtained coordinates directly from the XML file.

Results show that even the strong model Gemini2.5 performs poorly without XML input, exhibiting an almost 35% drop in success rate, indicating that current agents struggle to infer accurate spatial coordinates from vision alone. In other words, the agents know *what* action to take but not *where* to execute it. In contrast, AgentCPM-GUI, which is trained specifically for GUI interaction, shows a smaller performance drop, suggesting it may have partially learned coordinate prediction. These findings highlight that enhancing visual perception and coordinate prediction could therefore be crucial to improving GUI agent performance.

Model	Modality	SR (NoInt)	SR (WithInt)
AgentCPM	Screenshot+XML	59.87%	26.97%
	Screenshot-only	56.58%	19.74%
Gemini2.5	Screenshot+XML	80.26%	68.42%
	Screenshot-only	45.33%	41.33%

Table 3: Comparison of AgentCPM-GUI and Gemini 2.5 Flash under modality settings

5.4 Perception Drift After App Crash

We observe an interesting phenomenon, in which GUI agents often fail to fully restart a task after encountering an app crash. As shown in Figure 4, recovering from a GUI anomaly is not the same as recovering from its disruptive effects, such as blocking actions or redirecting execution to unpredictable screens. Even after addressing the anomaly, for example by reopening the crashed app and returning to the previous page, residual effects may persist. In this case, GPT-4o correctly identifies the crash, relaunches the App, and appears to recover. However, upon returning to the search page, it deviates from the correct path, and then clicks the search button directly instead of re-entering the required search term, resulting in task failure. This behavioral drift highlights a key issue that the decision-making of a model can be overly influenced by historical actions in the prompt, even when they no longer match the current visual state. In this case, GPT-4o has previously used the search interface, entered the target keyword, and logged that step in its action history. After the app crashes, upon returning to the same interface, it assumes that the keyword was still present and ignores that the search bar now contains recommended keywords. Improving the ability of an agent to retain useful memory while discarding misleading history could make it more intelligent and robust.

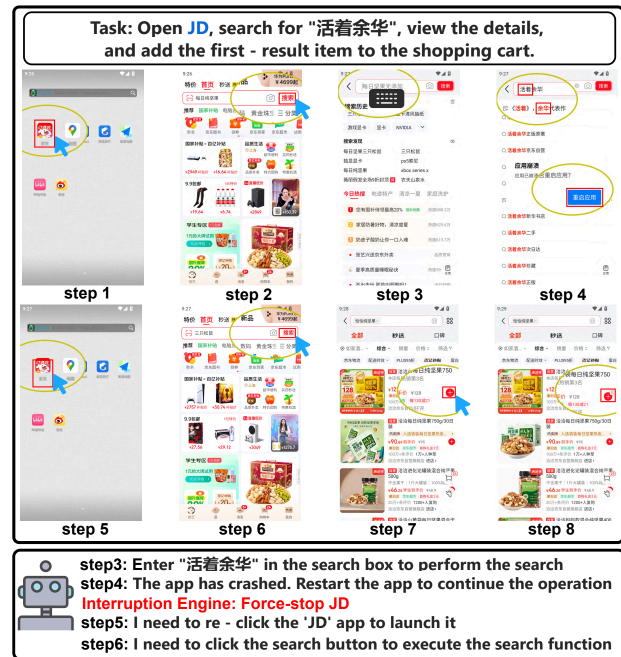

Figure 4: A case of perceptual drift after App crash recovery. The black and red dash indicate normal and interruption path, respectively.

6 Conclusion

In this paper, we introduced D-GARA, a dynamic benchmarking framework designed to evaluate the robustness of GUI agents under real-world anomalies. Unlike prior static benchmarks that fall short in simulating the complexity of interactive environments, D-GARA enables active anomaly injection and state-centric validation in live Android settings. Our framework allows for flexible, extensible, and realistic robustness testing. Experimental results reveal significant performance degradation in existing state-of-the-art agents when exposed to dynamic interruptions, underscoring the need for robustness-aware learning and evaluation. By open-sourcing D-GARA, we hope to facilitate broader research into building more resilient and generalizable GUI agents that can operate effectively in unpredictable real-world environments. We encourage the community to build upon D-GARA by contributing new tasks, interruption types, and evaluation strategies to further advance robustness research in GUI agent development.

Acknowledgments

This work is partially supported by the Central Guidance on Local Science and Technology Development Fund of Shanghai City (No.YDZX20253100002004), and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM116). We would also like to thank Haoxuan Li, Junrong Liao, Jian Liu, Wenhao Shi, and Haiyue Zhang for their insightful comments and valuable suggestions during the internal review of this manuscript.

References

Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; Zhong, H.; Zhu, Y.; Yang, M.; Li, Z.; Wan, J.; Wang, P.; Ding, W.; Fu, Z.; Xu, Y.; Ye, J.; Zhang, X.; Xie, T.; Cheng, Z.; Zhang, H.; Yang, Z.; Xu, H.; and Lin, J. 2025. Qwen2.5-VL Technical Report. arXiv:2502.13923.

Chen, Z.; Wu, J.; Wang, W.; Su, W.; Chen, G.; Xing, S.; Zhong, M.; Zhang, Q.; Zhu, X.; Lu, L.; Li, B.; Luo, P.; Lu, T.; Qiao, Y.; and Dai, J. 2024. InternVL: Scaling up Vision Foundation Models and Aligning for Generic Visual-Linguistic Tasks. arXiv:2312.14238.

Cheng, K.; Sun, Q.; Chu, Y.; Xu, F.; Li, Y.; Zhang, J.; and Wu, Z. 2024. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. arXiv:2401.10935.

Gou, B.; Wang, R.; Zheng, B.; Xie, Y.; Chang, C.; Shu, Y.; Sun, H.; and Su, Y. 2025. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents. arXiv:2410.05243.

Li, K.; Ziyang, M.; Lin, H.; Luo, Z.; Tian, Y.; Ma, J.; Huang, Z.; and Chua, T.-S. 2025. ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use. In *Workshop on Reasoning and Planning for Large Language Models*.

Li, W.; Bishop, W.; Li, A.; Rawles, C.; Campbell-Ajala, F.; Tyamagundlu, D.; and Riva, O. 2024. On the Effects of Data Scale on UI Control Agents. arXiv:2406.03679.

Lin, Z.; Liu, C.; Zhang, R.; Gao, P.; Qiu, L.; Xiao, H.; Qiu, H.; Lin, C.; Shao, W.; Chen, K.; et al. 2023. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models. *arXiv preprint arXiv:2311.07575*.

Lu, Y.; Yang, J.; Shen, Y.; and Awadallah, A. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

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.; Clinckemaillie, S.; Chang, Y.; Waltz, J.; Lau, G.; Fair, M.; Li, A.; Bishop, W.; Li, W.; Campbell-Ajala, F.; Toyama, D.; Berry, R.; Tyamagundlu, D.; Lillicrap, T.; and Riva, O. 2025. AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents. arXiv:2405.14573.

Shi, W.; Hu, Z.; Bin, Y.; Liu, J.; Yang, Y.; Ng, S. K.; Bing, L.; and Lee, R. 2024. Math-LLaVA: Bootstrapping Mathematical Reasoning for Multimodal Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 4663–4680.

Team, G. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Team, G. 2025. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805.

Wu, Q.; Cheng, K.; Yang, R.; Zhang, C.; Yang, J.; Jiang, H.; Mu, J.; Peng, B.; Qiao, B.; Tan, R.; Qin, S.; Liden, L.; Lin, Q.; Zhang, H.; Zhang, T.; Zhang, J.; Zhang, D.; and Gao, J. 2025. GUI-Actor: Coordinate-Free Visual Grounding for GUI Agents. arXiv:2506.03143.

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

Yang, J.; Song, Z.; Chen, J.; Song, M.; Zhou, S.; linjun sun; Ouyang, X.; Chen, C.; and Wang, C. 2025. GUI-Robust: A Comprehensive Dataset for Testing GUI Agent Robustness in Real-World Anomalies. arXiv:2506.14477.

Yao, Y.; Yu, T.; Zhang, A.; Wang, C.; Cui, J.; Zhu, H.; Cai, T.; Li, H.; Zhao, W.; He, Z.; Chen, Q.; Zhou, H.; Zou, Z.; Zhang, H.; Hu, S.; Zheng, Z.; Zhou, J.; Cai, J.; Han, X.; Zeng, G.; Li, D.; Liu, Z.; and Sun, M. 2024. MiniCPM-V: A GPT-4V Level MLLM on Your Phone. arXiv:2408.01800.

Zhang, Z.; Lu, Y.; Fu, Y.; Huo, Y.; Yang, S.; Wu, Y.; Si, H.; Cong, X.; Chen, H.; Lin, Y.; Xie, J.; Zhou, W.; Xu, W.; Zhang, Y.; Su, Z.; Zhai, Z.; Liu, X.; Mei, Y.; Xu, J.; Tian, H.; Wang, C.; Chen, C.; Yao, Y.; Liu, Z.; and Sun, M. 2025. AgentCPM-GUI: Building Mobile-Use Agents with Reinforcement Fine-Tuning. *arXiv preprint arXiv:2506.01391*.