MMBENCH-GUI: HIERARCHICAL MULTI-PLATFORM EVALUATION FRAMEWORK FOR GUI AGENTS

Xuehui Wang[2,1*], **Zhenyu Wu**[2,1*], **JingJing Xie**[3,1*], **Zichen Ding**[1*], **Bowen Yang**[4,1*], **Zehao Li**[4,1*], **Zhaoyang Liu**[5,1*], **Qingyun Li**[6,1], **Xuan Dong**[7], **Zhe Chen**[8,1], **Weiyun Wang**[9,1], **Xiangyu Zhao**[2,1], **Jixuan Chen**[8,1], **Haodong Duan**[1], **Tianbao Xie**[10], **Chenyu Yang**[1], **Shiqian Su**[7], **Yue Yu**[7], **Yuan Huang**, **Yiqian Liu**, **Xiao Zhang**, **Yanting Zhang**[11] **Xiangyu Yue**[12], **Weijie Su**[1], **Xizhou Zhu**[7], **Wei Shen**[2✉], **Jifeng Dai**[7], **Wenhai Wang**[12,1✉]

[1]Shanghai AI Laboratory, [2]Shanghai Jiao Tong University, [3]Xiamen University,
[4]University of Science and Technology of China,
[5]The Hong Kong University of Science and Technology, [6]Harbin Institute of Technology,
[7]Tsinghua University, [8]Nanjing University, [9]Fudan University, [10]University of Hong Kong
[11]Donghua University, [12]The Chinese University of Hong Kong

ABSTRACT

We introduce MMBench-GUI, a hierarchical benchmark for evaluating GUI automation agents across Windows, macOS, Linux, iOS, Android, and Web platforms. It comprises four levels—GUI Content Understanding, Element Grounding, Task Automation, and Task Collaboration—covering essential skills for GUI agents. In addition, we propose a novel Efficiency–Quality Area (EQA) metric to assess GUI agent execution efficiency in online automation scenarios. Through MMBench-GUI, we identify accurate visual grounding as a critical determinant of overall task success, emphasizing the substantial benefits of modular frameworks that integrate specialized grounding modules. Furthermore, to achieve reliable GUI automation, an agent requires strong task planning and cross-platform generalization abilities, with long-context memory, a broad action space, and long-term reasoning playing a critical role. More important, task efficiency remains a critically underexplored dimension, and all models suffer from substantial inefficiencies, with excessive redundant steps even when tasks are ultimately completed. The integration of precise localization, effective planning, and early stopping strategies is indispensable to enable truly efficient and scalable GUI automation. Our benchmark code, evaluation data, and running environment will be publicly available at https://github.com/open-compass/MMBench-GUI.

1 INTRODUCTION

With the rapid advancement of Vision-Language Models (VLMs) (Wang et al., 2024; Chen et al., 2024b; Bai et al., 2025; Zhu et al., 2025; Team et al., 2025; Xiaomi, 2025), the capability of agents to perform complex interactions within Graphical User Interfaces (GUIs) has significantly improved (Wu et al., 2024a; Cheng et al., 2024; Hong et al., 2024; Zheng et al., 2024; Gou et al., 2024). These GUI agents have shown great potential in automating complex, repetitive tasks across various domains, substantially enhancing productivity (Xu et al., 2024b; Wu et al., 2024b; Lin et al., 2024; Qin et al., 2025; Yang et al., 2024).

However, widely adopted evaluation benchmarks (Zhou et al., 2023; Cheng et al., 2024; Xie et al., 2024; Li et al., 2024; Chang et al., 2024; Rawles et al., 2024; Li et al., 2025; Nayak et al., 2025; Sun et al., 2025; Xie et al., 2025) currently face several critical limitations that hinder the further development of GUI agents: (1) existing benchmarks predominantly evaluate isolated capabilities and do not comprehensively analyze the agents' overall capabilities and the relationships between multiple capabilities (Deng et al., 2023a; Cheng et al., 2024; Xie et al., 2025; Li et al., 2025); (2) current evaluation metrics primarily emphasize task accuracy and success rate, overlooking operational efficiency (Zhou et al., 2023; Xu et al., 2024a; Xie et al., 2024; Bonatti et al., 2024); (3) insufficient

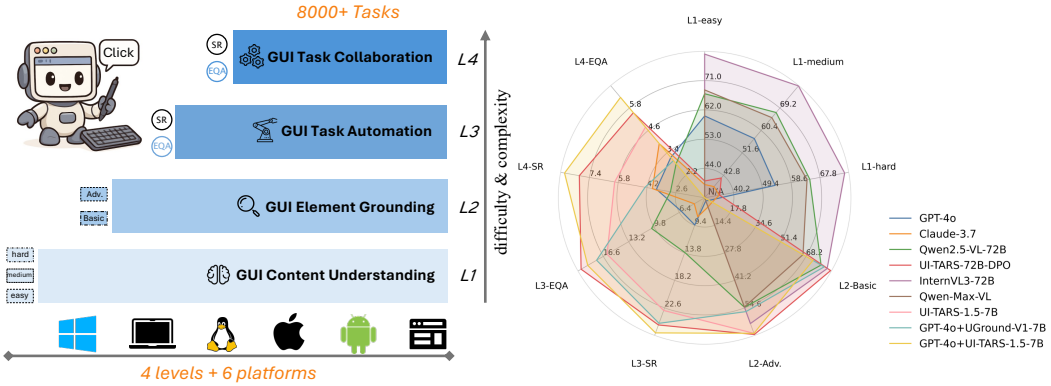


Figure 1: **MMBench-GUI**: a hierarchical benchmark spanning four levels of increasing difficulty, covering over 8,000 tasks across six commonly used platforms. From L1 to L4, task complexity increases progressively, placing growing demands on the agent's generalization and reasoning abilities. Based on this benchmark, we visualize the performance of various models in the right figure, clearly illustrating their respective strengths as well as areas with substantial room for improvement.

coverage of evaluation scenarios fails to fully represent widely used GUI systems in real-world applications (He et al., 2024; Xie et al., 2024; Rawles et al., 2024; Sun et al., 2025).

To address these issues, we propose MMBench-GUI, a hierarchical, multi-platform benchmark framework designed for the systematic evaluation of GUI agents. As shown in Figure 1, this framework comprises four progressive evaluation levels: (1) GUI Content Understanding, (2) GUI Element Grounding, (3) GUI Task Automation, and (4) GUI Task Collaboration. Each level addresses critical capabilities from basic interface understanding to complex cross-application task execution, ensuring comprehensive and systematic evaluation. Additionally, we introduce the Efficiency-Quality-Aware (EQA) metric, which evaluates both task accuracy and operational efficiency, encouraging agents to complete tasks with minimal interaction steps. Furthermore, to ensure practical relevance, we have constructed a multi-platform dataset covering Windows, macOS, Linux, iOS, Android, and Web platforms, effectively reflecting diverse real-world scenarios.

Leveraging extensive evaluations and analysis with MMBench-GUI, we identify key limitations and reveal the current state of GUI agents. Our findings indicate that: (1) although general language models excel at high-level planning and reasoning tasks, they significantly lag in precise visual interaction capabilities. Precise visual grounding is identified as a core determinant of task success, underscoring the critical need for enhanced localization accuracy; (2) efficiency now eclipses raw success rate as the next hurdle. Our EQA metric exposes the considerable surplus steps incurred by contemporary agents during task execution–redundancies that stem from localization inaccuracies, incomplete action spaces, and short-sighted or inadequate planning; (3) agent performance notably degrades when faced with complex, ambiguous, and cross-application tasks, exposing weaknesses in memory management, state tracking, and adaptive reasoning mechanisms. Addressing these shortcomings is crucial for future advancements in GUI agents.

In summary, our primary contributions are as follows:

- We propose a cross-platform, hierarchical benchmark designed to comprehensively evaluate GUI agents across multiple task types and difficulty levels. Inspired by a human-centered perspective, the benchmark is structured in a progressive manner, covering four essential capabilities. For static tasks (L1 and L2), we introduce fine-grained difficulty stratification to enable stepwise assessment. For dynamic tasks (L3 and L4), we provide both cleaned data splits and novel task constructions to better reflect real-world variability.
- We develop the first evaluation benchmark that spans all widely used operating systems, including Windows, Linux, macOS, Android, iOS, and the Web. To the best of our knowledge, this is the first work to enable consistent multi-platform evaluation of GUI agents under a unified protocol. This broad coverage allows for more realistic and application-aligned performance assessment.

Notably, our benchmark is also the first to include online task scenarios for macOS, filling a long-standing gap in GUI agent evaluation.

- We introduce a novel metric, Efficiency-Quality-Aware (EQA), to jointly assess both the success and efficiency of agent behavior in online tasks. While most prior works focus solely on success rate (SR), EQA additionally considers when the task is completed within the step budget, providing a measure of action redundancy. This metric offers deeper insights into agent behavior and promotes the development of agents that are not only capable but also efficient—an often overlooked dimension in prior benchmarks.

2 Related works

GUI Agents. GUI agents have attracted growing interest, driven by advances like Anthropic's Computer-Use Agent[1] and OpenAI's Operator[2]. Currently, GUI Agents mainly fall into two paradigms: a) Modular agent schemes (Cheng et al., 2024; Gou et al., 2024; Yang et al., 2024; Zhang et al., 2025; Wu et al., 2025; Xie et al., 2025; Wang et al., 2025), which typically employs general-purpose VLMs (*i.e.*, GPT-4o) as planners, integrated with a specially trained GUI grounding model for focused UI element localization; b) Native agent schemes (Xu et al., 2024b; Wu et al., 2024b; Lin et al., 2024; Sun et al., 2024; Qin et al., 2025; Yang et al., 2024), where planning and grounding are trained in an end-to-end manner. Modular approaches benefit from state-of-the-art components but face challenges in system-level alignment Cheng et al. (2024); Gou et al. (2024). In contrast, the native agent paradigm aligns capabilities more naturally during training (Wu et al., 2024b; Xu et al., 2024b; Qin et al., 2025). Both paradigms can use screenshots (Niu et al., 2024; Liu et al., 2024a), accessibility trees (A11y Trees) (Gao et al., 2023), and HTML pages (Furuta et al., 2023; Deng et al., 2023b) as input. However, A11y Trees and HTML codes vary across platforms, are prone to noise, and may cause excessive token length (Zheng et al., 2024; Hong et al., 2024; Cheng et al., 2024). Generally, in this work, we focus exclusively on the screenshot-only setting and propose a hierarchical, multi-platform benchmark to evaluate these vision-only native agents.

GUI Benchmarks. Effectively GUIs requires a sophisticated grasp of intertwined visual and textual cues, yet this complex domain remains largely outside the scope of general-purpose multimodal QA benchmarks (Liu et al., 2024c; Yue et al., 2024; Masry et al., 2022). While ScreenQA (Hsiao et al., 2022) and WebSRC (Chen et al., 2021) provide large-scale QA datasets based on Android screenshots and web pages respectively, and GUI-World introduces cross-platform GUI QA via video data, these efforts offer limited support for interactive GUI agents. To evaluate visual grounding in GUI contexts, several benchmarks have emerged. ScreenSpot (Cheng et al., 2024) and its improved versions (Wu et al., 2024b; Li et al., 2025) support cross-platform UI grounding with progressively enhanced realism and annotation quality. UI-I2E-Bench (Liu et al., 2025) and UI-Vision (Nayak et al., 2025) further expand this by aligning natural language instructions with GUI elements of varying scale and type. For reasoning and planning, offline benchmarks like (Rawles et al., 2023; Chen et al., 2024a; Li et al., 2024; Deng et al., 2023a; Kapoor et al., 2024; Lu et al., 2024) assess action prediction from fixed trajectories, while online benchmarks (Zhou et al., 2023; Xie et al., 2024; Bonatti et al., 2024; Rawles et al., 2024; Xu et al., 2024a; Liu et al., 2024b) enable interactive evaluation across platforms. However, macOS remains underexplored. Our MMBench-GUI benchmark addresses this gap by enabling online evaluation on macOS and emphasizing cross-platform robustness, providing a realistic and comprehensive evaluation for GUI agents.

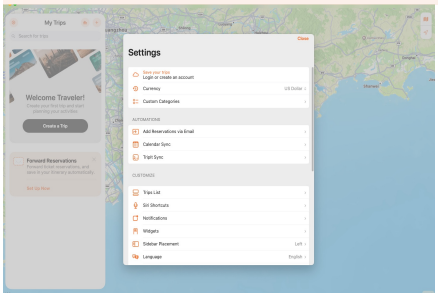
3 MMBench-GUI

In this paper, we introduce MMBench-GUI, a benchmark designed to comprehensively evaluate the capabilities of AI agents in operating graphical user interface (GUI) across a broad spectrum of platforms, including Windows, Linux, macOS, Web, Android, and iOS. Informed by a cognitive analysis of essential human abilities for GUI tasks, MMBench-GUI's evaluation process is organized into a multi-level hierarchy. The underlying principles of this hierarchical framework are discussed in Section 3.1. Sections 3.2–3.5 elaborate each level of MMBench-GUI, including task formulation,

[1] https://www.anthropic.com/news/3-5-models-and-computer-use

[2] https://openai.com/index/computer-using-agent

L1 - GUI Content Understanding



Question:
Which of the following features would allow the user to have their travel details automatically added to calendar application?

Options:
A.Forward Reservations B.Calendar Sync C.Trips List D.Custom Categories

Answer: B

Explanation:
The 'Calendar Sync' option is listed under the AUTOMATIONS section in the setting panel, which would enable synchronization between trip data and a calendar application. The other options serve different functions: Forward Reservations is for ticket management, Trips List is for customizing trip displays, Custom Categories is for organizing trips.

Difficulty: Medium

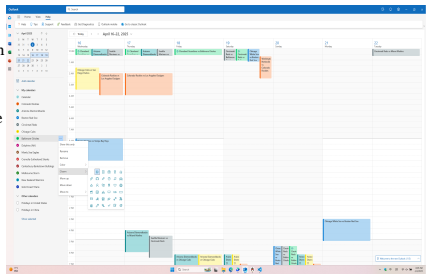
Question:
What action is available in the context menu that appears when right-clicking on a calendar in the My calendars list?

Options:
A.Show this only B.Print calendar C.Delete event D.Create new event E.Change time zone

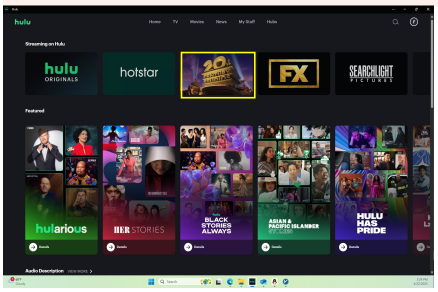
Answer: A

Explanation:
The context menu shows the 'Show this only' option, but does not include print, delete event, create new event, or change time zone.

Difficulty: Hard



L2 - GUI Element Grounding



Basic instruction:
The iconic golden 20th Century Studios logo which features a monument-like structure with searchlights against a purple-blue sky background.

Advanced instruction :
Explore content from the production company responsible for films like Avatar and The Simpsons.

Data type: icon

Platform: windows

App name: Hulu

Bbox: [0.4109, 0.1656, 0.5801, 0.3206]

Size: 2560x1600

Basic instruction:
An unchecked checkbox labeled 'Search subcollections' in the Advanced Search dialog box.

Advanced instruction :
Enable the option to include subcollections in your Zotero library search.

Data type: text

Platform: linux

App name: Zotero

Bbox: [0.3406, 0.4231, 0.4182, 0.4454]

Size: 1920x1080

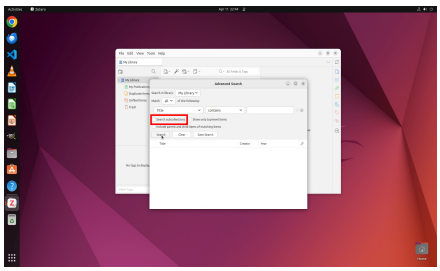


Figure 2: **Examples for L1&L2.** Both of them are offline tasks. We provide examples from different platforms for each level. For clarity, some less critical fields are not shown here and full examples are available for download in our public repository.

data sources, and evaluation protocols. Finally, we offer a statistical analysis of the proposed MMBench-GUI in Section 3.6.

3.1 Hierarchical Evaluation

Existing GUI agents typically complete assigned tasks by emulating human operations, such as mouse clicks and keyboard input. This requires them to understand the graphical user interface and possess long-horizon planning capabilities. However, existing benchmarks tend to focus on isolated aspects. For instance, Screenspot (Cheng et al., 2024) evaluates spatial localization, while OSWorld (Xie et al., 2024) emphasizes end-task success—without directly assessing the full range of underlying competencies. As a result, the relationships among different abilities remain unclear and it

is difficult to determine which specific factors contribute to an agent's success or failure. Taking into account these limitations and motivated by the use of leveled definitions in the domain of autonomous driving, we developed a hierarchical evaluation framework, MMBench-GUI, to systematically and comprehensively assess the capabilities of GUI Agents, as shown in Figure 1. Specifically, we organize the evaluation framework into four ascending levels: ❶ L1-GUI Content Understanding, ❷ L2-GUI Element Grounding, ❸ L3-GUI Task Automation, ❹ L4-GUI Task Collaboration.

Each level is associated with a set of tasks of increasing complexity, designed to test the Agent's proficiency in progressively more demanding scenarios. The complete benchmark includes over 8,000 tasks that span diverse platforms, with detailed statistics provided in Section 3.6.

3.2 L1-GUI Content Understanding

To accurately complete an automated task based on the provided instructions, GUI agents need to integrate their domain knowledge with visual observations, enabling them to interpret the layout, functionalities, and informational content embedded in the interface. To achieve this goal, agents are capable of handling many challenges, such as substantial variability in UI design paradigms across different platforms (e.g., desktop **vs.** mobile), discrepancies in interface conventions among various applications even within the same platform, and fragmented background knowledge for some domain-specific software tools. These complexities underline the critical importance of advanced perception and understanding mechanisms. However, due to a lack of comprehensive and well-defined benchmarks, the understanding capabilities of GUI agents have not been effectively and explicitly evaluated.

Therefore, we propose the first level of task: *L1-GUI Content Understanding*. This task is placed at the beginning because we believe that understanding the GUI is a fundamental prerequisite to successfully complete any subsequent tasks.

Task Definition. At this level of task, our objective is specifically to assess the ability of an agent to extract, comprehend, and reason about information present in GUI screenshots, without requiring explicit attention to precise element localization or specific operational actions. Therefore, we formalize this assessment as a Multiple-Choice Question-and-Answer (MCQA) task based on visual observations (GUI screenshots), enabling quantifiable and standardized output that simplifies the evaluation process. Formally, the task can be defined as

$$o^* = \text{Agent}(\mathbf{V}, q, \mathcal{O}) \tag{1}$$

where $\mathbf{V}$ denotes the visual observation (GUI screenshot) presented to the agent, q represents the question about the observation to evaluate comprehension, and $\mathcal{O} = \{o_1, o_2, \ldots, o_k\}$ represents the set of k candidate options for question q, among which only one o can correctly answer q.

The agent's goal is to analyze the question q, identify relevant information within $\mathbf{V}$, conduct reasoning, and finally select an option o^* from $\mathcal{O}$ as the predicted answer. The key to achieving this goal is the proper construction of the pair $(q, \mathcal{O})$, as the effectiveness of the evaluation depends on both the quality of the question and the relevance of the options. Therefore, we argue that a diverse and sufficiently large set of well-constructed $(q, \mathcal{O})$ pairs about GUI elements and operations, with varying levels of difficulty, is essential to effectively evaluate the GUI understanding capabilities of the agent.

Data Collection and Annotation. We manually collected screenshots from widely used applications and websites across all supported platforms, selected for their high usage frequency and representative user scenarios. In addition, we supplemented our data with a small number of screenshots sourced from publicly available datasets (Cheng et al., 2024; Li et al., 2025). To ensure diversity, we include screenshots of varying sizes, ranging from single-window to full-screen views, and accompanied each image with metadata; filenames were anonymized using an MD5-based encoding scheme constructed from a combination of platform, application name, and original file path, to avoid path conflicts and information leakage.

Then, we followed a four-step strategy to construct high-quality Question-Options-Answer pairs:

Prompt 1:

You are an expert GUI analyst for {os_name} and item-writer.

Input:

1. One screenshot of a GUI application.
2. The application's name ({app_name} or "Not available") — optional and for background only; do **not** mention it in any question text.

Task:

Create *exactly one* multiple-choice question about the screenshot at **each** of three difficulty levels (easy, medium, hard). For **every** question you generate:

- Write the stem in clear English that can be answered *only* by understanding the screenshot. Avoid trivial facts (e.g., "What color is the button?") unless color is functionally meaningful.
- Focus on tasks, labels, hierarchy, states, or affordances shown in the UI.
- Provide **4–6** answer options labeled "A", "B", "C", ... in a JSON sub-object called `"options"`.
- Ensure **one and only one** option is strictly correct; the others must be clearly incorrect but plausible. Give the answer key (the letter of the correct option).
- Double-check yourself that the correct answer is indeed unique and unambiguous.
- Do **not** include the {app_name} or any other identifying text of the app in the stem or options.
- Give a concise `"explanation"` stating *why* the correct option is right and the others are not in 1–3 sentences.
- The hard question should require the answerer to think more about the screenshot, the question, and the options (you can also make options be easy to confuse).

Output format:

Return a single valid JSON array containing three objects (one per difficulty), in *English*, structured exactly like this schema:

```
[
  {
    "difficulty": "easy",
    "question": "<stem>",
    "options": {
      "A": "<option text>",
      "B": "<option text>",
      "C": "<option text>",
      "D": "<option text>"   // add "E","F" only if needed
    },
    "answer": "A",
    "explanation": "<brief rationale>"
  },
  ...
]
```

Important Constraints:

1. Produce only the JSON text—no markdown, headings, or commentary.
2. Validate that the JSON is syntactically correct before outputting.
3. After generation, internally review each Q&A for accuracy and compliance.

- Step 1: Claude 3.7 (Anthropic, 2025) was used to generate three questions for each image, corresponding to three levels of difficulty: easy, medium, and hard. Each question includes 4 to 6 answer options, with exactly one correct choice. In addition, Claude 3.7 was instructed to provide an explanation for each question, detailing the reasoning process that leads to the correct answer. In designing the questions for each image, we guided Claude 3.7 to focus on various aspects of the GUI, including the functionality of UI elements, structural relationships within the interface, content states, hierarchical layout, and executable tasks. The detailed prompt for this process is shown in Prompt 1.

- Step 2: We then used GPT-o4-mini (OpenAI, 2025) to verify the validity of each question, set of options, and answer, jointly considering the UI interface and the generated explanation. The errors were corrected with justification and revised explanations.
- Step 3: Then, GPT-o3 (OpenAI, 2025) was used to further review and refine the revised items following the same Prompt 2 as in Step 2.
- Step 4: Finally, manual sampling was performed to ensure overall quality and consistency.

By incorporating three different strong models across the pipeline, we reduced the risk of model-specific hallucinations and stylistic bias. We provide some example in Figure 2, which contains screenshots, metadata, and generated annotations.

Prompt 2:

You are a meticulous GUI-QA evaluator.

Input:

1. One screenshot (`image`) of a GUI application running on a {os_name}.
2. The application's name (`app_name`) – optional and strictly for background; *never* mention it in your output.
3. A JSON-like array (`qa_items`) containing three single-choice questions about the screenshot (intended levels: `easy`, `medium`, `hard`). Each object is expected to have the keys question, options, answer, difficulty, and optionally explanation.

* Ignore cosmetic or syntactic issues in the supplied JSON (e.g., extra backticks, missing quotes, inconsistent key order, markdown fences).
* Focus **only** on the content of `question`, `options`, and `answer` when deciding validity.

Task:

For each question, decide whether it is content-valid for use in a test. A question is valid only if all the following hold:

- The stem can be answered solely by inspecting the screenshot (no outside knowledge).
- Exactly one option is correct and that option is the one listed in `answer`.
- Incorrect options are clearly wrong yet still plausible.
- Neither stem nor options reveal the `app_name`.
- The difficulty label is reasonable (honor system; do not reject only for minor mislabelling).

The hard level should allow the answerer to think more deeply about the screenshot, the question, and the options. You may make the options easy to confuse.
* Do **not penalise** minor formatting faults that do not affect the five substantive criteria.

Output format:

Return a JSON array of three objects in the original order, each with:

```
{
  "difficulty": "<same as input>",
  "valid": "yes" | "no",
  "comment": "<if valid: empty string; if not valid: brief reason
      why>",
  "fix": <if valid: null; if not valid: a *fully corrected* object
      that replaces the faulty one (same schema as above, with all
      issues fixed)>
}
```

Notes:

1. Provide an empty string (" ") for `comment` and "null" for `fix` when `valid` is "yes".
2. When `valid` is "no", supply both an actionable `comment` and a complete `fix` object that meets all criteria.
3. Do not wrap the result in markdown or add explanations outside the JSON.
4. Verify that the final JSON is syntactically correct before sending it.

Evaluation Metrics. For each question, we adopt accuracy as the evaluation metric, consistent with common QA tasks. Formally, the accuracy for an evaluation set comprising N Question-Options-Answer pairs can be defined as:

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N}\Theta(o_i^* = o_i), \tag{2}$$

where $\Theta(o_i^* = o_i)$ is an indicator function that equals 1 if the predicted answer o_i^* for the i-th pair matches the ground-truth answer o_i and 0 otherwise.

To account for variations in the number of answer choices, we introduce a simple dynamic adjustment factor α to rescale the original accuracy of each question. Taking Windows platform which has N_{win} questions as an example, the accuracy of L1 is computed as:

$$\text{Acc}_{win} = \frac{1}{N_{win}}\sum_{i=1}^{N_{win}}\alpha\cdot\Theta(o_i^* = o_i), \quad \alpha = \frac{m_i - 1}{m_i} \tag{3}$$

where m_i is the number of options for question i. Accordingly, for any given difficulty level, the agent's understanding ability (i.e., accuracy) can be computed as:

$$\text{Score} = \sum_{j\in\mathcal{O}}\frac{N_j}{N}\cdot\text{Acc}_j \tag{4}$$

where $\mathcal{O} = \{\texttt{win}, \texttt{linux}, \texttt{mac}, \texttt{ios}, \texttt{android}, \texttt{web}\}$ denotes the set of operation platforms, N_j is the number of questions for platform j, $N = \sum_{j\in\mathcal{O}} N_j$ is the total number of questions across all platforms.

3.3 Level 2: GUI Element Grounding

The precise grounding of interactive UI elements is a fundamental prerequisite for effective execution of GUI-based tasks. This capability requires agents to accurately localize the spatial positions of target elements within the GUI, conditioned on the current task objective and corresponding observation (e.g., a screenshot). Despite significant progress in this direction, several inherent challenges remain: (1) visual ambiguity caused by highly similar elements, such as identical buttons or icons with only subtle differences; (2) dynamic UI disruptions, including pop-up windows or transient notifications that obscure intended targets; and (3) the difficulty of distinguishing inactive or grayed-out regions from their active counterparts. Addressing these issues is crucial, as grounding directly influences an agent's reliability and effectiveness in performing GUI-based tasks.

However, existing benchmarks such as ScreenSpot (Cheng et al., 2024; Wu et al., 2024b) have become nearly saturated, and ScreenSpot Pro (Li et al., 2025) is curated within limited application domains. More critically, the instructions employed by current benchmarks are often overly simplistic and direct (e.g. "submit the paper"), which fails to reflect the nuanced ways in which GUI agents refer to and reason about UI elements during real-world task execution. This mismatch results in a gap between the benchmark tasks and the genuine challenges faced by agents in practical scenarios. To address these limitations, we draw inspiration from the strengths of prior benchmarks while introducing a new dataset encompassing a broader range of application domains and more diverse and realistic instructions. Specifically, we systematically categorize instructions by their descriptive types, aiming to more accurately assess model weaknesses and bridge the gap between benchmark evaluation and real-world agent reasoning.

Task Definition. Accurate perception and understanding by an agent typically require validation through concrete actions, analogous to human interactions with GUI elements, to execute subsequent task steps. Building upon the comprehension capabilities assessed in L1, we propose *L2-GUI Element Grounding* to further measure the agent's spatial localization ability, specifically, the accurate identification of actionable GUI elements. This ability aligns precisely with the requirements of a grounding task, formally defined as:

$$p = \text{Agent}(\texttt{ins}, \mathbf{V}) \tag{5}$$

Prompt 3:

You are a GUI agent currently operating on a {os_name}.

Input:

1. The first image is a screenshot from the {application} {app_or_web}, in which a selected element is highlighted with a distinctive red box and a red arrow.
2. The second image is the cropped region containing the selected element and corresponding box and arrow.
3. A simple and coarse description of the selected element.

Task:

Your task is to understand the possible role, function, and related global contextual information of the selected element on the current page from the first image. Then, from the second image, you can combine the global information from the first image to further analyze the relationship between the selected element and its surrounding information. The simple and coarse description can be regarded as a prior for the selected element. Finally, you are required to conclude two types of instructions for the selected element:

* *Basic Instruction*: Informative description that summarizes key information.
* *Advanced Instruction*: An indirect yet specific instruction that refers to the selected element.

Guidelines for Generating Descriptions:

Basic Instruction:

- Concise summary including appearance and position.
- Avoid referencing the red box or arrow.
- Examples:
  - "A circular icon with a white background and a magnifying glass symbol in black."
  - "Located in the top-right corner, to the right of the profile avatar icon."

Advanced Instruction:

- Focus on function and reasoning.
- Avoid visual/positional terms.
- Examples:
  - "Search some latest posts"
  - "Type in text to discover related content"

Output format:

Return a dictionary with:

```
{
   "basic_instruction": ["xxxx", "xxx", "xxx"],
   "advanced_instruction": ["xxxxx", "xxx", "xxx"]
}
```

Notes:

1. Ensure instructions are clear, unambiguous, and concise.
2. Do not mention the red box and arrow.
3. Coarse descriptions are only priors.

where `ins` represents an instruction for the GUI element to be localized, which can be derived from a direct user task or the agent's internal reasoning process. The output p denotes the resulting location of the target element, typically represented by the coordinates (x, y) that indicate the activation point of the interactive element. The definition of `ins` constitutes the core component of this level. In the context of GUI tasks, the description of an element can encompass various attributes, including appearance, approximate spatial position, and functionality.

Data Collection and Annotation. We reuse the data from L1 to annotate additional agent capabilities, enabling multidimensional analysis on a consistent data foundation. This design facilitates exploration of inter-task correlations and addresses earlier research questions. We also manually labeled the

positions of interactive elements, i.e. user-operable components such as buttons or icons, using bounding boxes, and categorized them as either Text or Icon, following the classification scheme used in ScreenSpot (Cheng et al., 2024).

We adopt a three-step procedure to generate grounding instructions for annotated interactive elements:

- Step 1: Claude 3.7 was prompted to produce two types of instruction per element: **Basic**, which describes visual features and approximate location to test perception-based grounding, and **Advanced**, which targets functional understanding through implicit cues. To increase diversity, three stylistic variants were generated for each type. The detailed prompt for this step is shown in Prompt 3.
- Step 2: We developed an annotation tool to manually review and refine these instructions, ensuring that each uniquely maps to a specific element.
- Step 3: A validated instruction per type was selected to form the final evaluation set.

Examples of annotated data can be found in Figure 2, and we attach two types of instructions for each element of a screenshot.

Evaluation Metrics. Following the evaluation protocol of ScreenSpot (Cheng et al., 2024), we computed accuracy separately for the Basic and Advanced instruction types. For each interactive element, a prediction was considered successful if the agent's predicted point of interaction–represented as a coordinate (x, y)–fell within the annotated bounding box. Otherwise, it was marked as a failure. The final accuracy was calculated as the proportion of successful predictions over the total number of evaluated elements.

3.4 Level 3: GUI Task Automation

To successfully accomplish user-specified tasks within a single application environment, agents must integrate their comprehension of the interface content and precise localization of relevant elements with advanced planning and dynamic reasoning. The typical workflow begins with interpreting the task instruction, perceiving the content, and grounding the target UI components. The agent then decomposes the high-level task into a sequence of executable actions, such as clicking, typing, or selecting, iteratively interacting with the environment and adapting its strategy based on real-time feedback. This tightly coupled cycle of perception, decision-making, and interaction constitutes the essence of robust GUI task completion, especially for complex, multi-step scenarios.

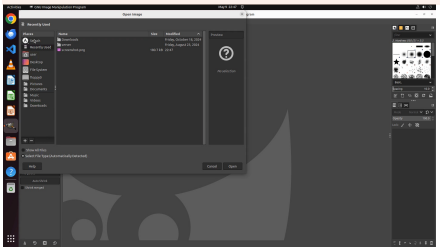
Key challenges at this iterative cycle include resolving ambiguous or under-specified instructions, navigating dynamic UI states (e.g., pop-ups, context changes), and efficiently planning multi-step operations to achieve the desired outcome. Despite their prevalence in real-world automation scenarios, such capabilities within the single application are rarely evaluated in a systematic manner across multi-platforms. Therefore, we propose *L3-GUI Task Automation* as the third level in our benchmark, focusing on the agent's ability to perform end-to-end automation within a single, potentially complex, application environment. This level serves as a crucial bridge between low-level perception/understanding and higher-level, generalizable task-solving skills.

Task Definition. Building upon the challenges outlined above, we formally define the *L3-GUI Task Automation* as follows: The agent is required to complete a multi-step task within a single application by generating a sequence of actions that directly manipulate the user interface to fulfill a specified objective. At each time step t, the agent receives a visual observation $\mathbf{V}_t$ of the current UI state and generates an action A_t with corresponding parameters P_t, based on the task instruction $\texttt{ins}$, the history $\mathcal{H}_t$, and involved applications $\mathcal{S}$ (with $\mathcal{S} \in \{\texttt{App}_1, \texttt{App}_2, \ldots, \texttt{App}_n\}$ for single-app scenarios). The process is formally described as:

$$\begin{aligned} A_t, P_t &= \text{Agent}(\texttt{ins}, \mathbf{V}_t, \mathcal{H}_t, \mathcal{S}_s) \\ \mathbf{V}_{t+1} &= \text{Env}(A_t, P_t) \\ \mathcal{H}_{t+1} &= \{\mathcal{H}_t, (V_t, A_t, P_t)\} \end{aligned} \tag{6}$$

Here, $\mathcal{H}_t$ denotes the contextual history, which normally consists of previous observations and action sequences. In practice, the implementation of history typically follows two styles. The

L3 - GUI Task Automation



Task instruction :
Please adjust the brightness of the image that named as 'Panda' on the desktop as 40.

Evaluation function:
Check_Gimp_Status(content='image', items=['brightness', 40])

Platform : linux

Type: Single

App name: [Gimp]

Max steps: 50

Task instruction :
Please tell me the number of commits kilian made on 05 Mar, 2023 for the A11Y project on Gitlab.com.

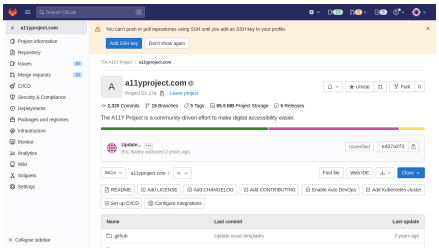
Evaluation function:
String_Match(must_include, ["1"])

Platform : Web

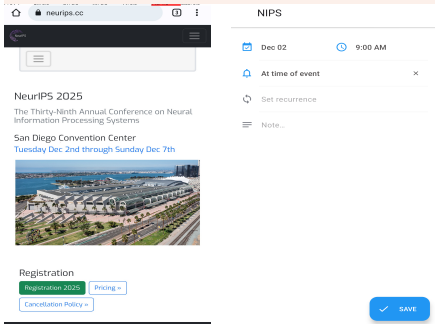
Type: Single

App name: [Gitlab.com]

Max steps: 15



L4 - GUI Task Collaboration



Task instruction :
Use Google to search for the conference start date of NIPS 2025 (UTC-0), and add an event titled "NIPS" on the date in Calendar.

Evaluation function :
Check_Calendar_Status(content='Events', items=['NIPS', 'Dec 2nd'])

Platform : android

Type : Multi

App name : [Chrome, Calendar]

Max steps : 15

Task instruction :
Calculate how many years, months, weeks and days are between 10/08/1980 (MM/DD/YYYY) and 8/2/2024 using the calculator app, and save the result in a file called 'Differences.txt' on the Desktop (e.g. X years, Y months, Z weeks, W days)

Evaluation function :
Exact_Match(type='is_file_saved_desktop', filename='Differences.txt', textcontent='43 years, 9 months, 3 weeks, 4 days')

Platform : windows

Type : Multi

App name : [Calculator, Notepad]

Max steps : 50

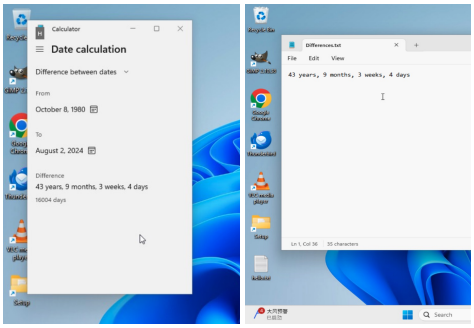


Figure 3: **Examples for L3&L4.** Tasks of these levels are evaluated in the virtual environment with an online manner. In L4, we provide two images belonging to different applications as examples to demonstrate that collaboration is the core aspect for this level.

first style encapsulates the entire interaction process within multi-turn dialogues, while the second one condenses history into natural language and injects it into the prompt. The agent-environment interaction proceeds iteratively until a maximum number of steps ($t = T_{\max}$) is reached or a terminal action ($A_t \in$ `[FINISH, FAIL]`) is predicted.

Task Collection and Curation. To ensure broad coverage and real-world relevance, our GUI task automation benchmark encompasses tasks across multiple major platforms, including Windows, Linux, macOS, web, and Android. Due to the inherent restrictions of the iOS ecosystem, iOS tasks are not currently included.

The majority of tasks are sourced from established public benchmarks, each of which leverages virtualization technology to provide robust and reproducible GUI environments. Specifically, tasks for the Linux platform are drawn from OSWorld (Xie et al., 2024), Android from AndroidWorld (Rawles et al., 2024), web from WebArena (Zhou et al., 2023), and Windows from WindowsAgentArena (Bonatti et al., 2024). These resources have been extensively validated in prior research and collectively provide a diverse set of task scenarios. Importantly, our use of these benchmarks is not a simple replication. Each task underwent a rigorous manual review process, during which we excluded any instances likely to result in agent failure due to non-agent factors such as unstable network conditions, required account authentication, or platform-specific anomalies. This curation ensures that the performance evaluations reflect true capabilities of the agent, rather than artifacts of the benchmarking environment.

To address the lack of existing online evaluation resources for the macOS platform, we introduce *MMBench-GUI-macOS*, a novel set of 70 curated tasks spanning 9 widely used macOS applications. Of these, 35 tasks are categorized as L3 tasks and the remaining 35 as L4 tasks. Task design for macOS follows the same principles as for other platforms, utilizing paired natural language instructions and screenshots to simulate virtual environments, thereby ensuring consistency and comparability across all platforms. This multi-platform, carefully curated task set provides a comprehensive and fair foundation for benchmarking GUI agents in realistic and heterogeneous settings. We provide two illustrative examples in the upper part of Figure 3 to demonstrate the details of L3 tasks.

Evaluation Metrics. From a user-centric standpoint, an ideal agent should be both accurate and efficient. However, existing benchmarks typically rely solely on Success Rate (SR), neglecting how quickly tasks are completed. To address this limitation, we propose the **Efficiency–Quality Area (EQA)**, a unified metric inspired by the AP computation protocol in COCO (Lin et al., 2014). EQA jointly considers task success and completion speed, rewarding agents that solve more tasks using fewer steps.

Specifically, we define EQA as a continuous-time recall metric over cumulative agent effort. Consider an ordered set of N tasks. For each task $i \in \{1, 2, \ldots, N\}$, let:

- $s_i = 1$ if the agent successfully completes task i, and $s_i = 0$ otherwise,
- $t_i > 0$ be the number of steps the agent takes to complete task i.

We define the cumulative cost and cumulative success after the first k tasks as:

$$T_k = \sum_{j=1}^{k} t_j, \qquad S_k = \sum_{j=1}^{k} s_j. \tag{7}$$

Let the global budget be $T_{\text{max}} = N \cdot t_{\text{max}}$, where t_{max} is the maximum step limit per task. We normalize the cumulative effort as:

$$u_k = \frac{T_k}{T_{\text{max}}} \in [0, 1]. \tag{8}$$

The instantaneous recall at normalized time u is defined as:

$$R(u) = \max_{k:\, u_k \le u} \frac{S_k}{N}, \qquad u \in [0, 1]. \tag{9}$$

Finally, EQA is computed as the area under the step-wise non-decreasing recall curve:

$$\text{EQA} = \int_0^1 R(u)\, du \;\approx\; \frac{1}{M} \sum_{m=0}^{M-1} R\left(\frac{m}{M-1}\right), \tag{10}$$

where $M = 101$ denotes the number of uniformly spaced evaluation points. This metric encourages agents to complete more tasks in fewer steps, offering a holistic measure of task performance.

Table 4: **Statistics of the evaluation data in MMBench-GUI.** Owing to the inherent restrictions of the iOS ecosystem, we were unable to include online tasks for iOS in L3&L4. All other platforms are covered in full.

	Windows	MacOS	Linux	iOS	Android	Web	Overall
L1				L1 - Easy			
	271	84	196	115	307	221	1194
				L1 - Medium			
	271	84	196	115	307	221	1194
				L1 - Hard			
	271	84	196	115	307	221	1194
L2				L2 - Basic			
	271	345	191	314	356	310	1787
				L2 - Advanced			
	272	346	196	330	335	308	1787
L3	145	35	268	-	116	155	719
L4	35	35	101	-	30	47	248
Total	**1536**	**1013**	**1344**	**989**	**1758**	**1483**	**8123**

3.5 Level 4: GUI Task Collaboration

Real-world task automation frequently requires agents to coordinate actions across multiple applications or environments, orchestrating complex workflows that involve heterogeneous interfaces and interdependent subtasks. To address such scenarios, task completion should extend beyond localized planning, requiring agents to develop a global perspective—tracking dependencies among applications, sequencing operations coherently, and managing cross-app information flow. In this situation, agents must not only exhibit sophisticated long-horizon reasoning and planning abilities, but also handle practical challenges such as recovering from execution errors, coping with unexpected interface changes, and adapting to runtime variability in application responses. These factors collectively pose significant hurdles for contemporary GUI agents, making it a rigorous and realistic testbed for general-purpose automation intelligence.

Despite the critical role of collaboration and global reasoning in real-world workflows, existing benchmarks rarely address these aspects in a comprehensive and principled fashion. Accordingly, we introduce *L4-GUI Task Collaboration* as the fourth level of our benchmark, designed to systematically assess an agent's ability to use reasoning, collaboration, and adaptive automation across applications.

Task Definition. Extending the formulation above, L4 evaluates the agent's ability to coordinate complex workflows involving multiple applications. The agent must generate and execute a sequence of actions that may interact with any application in the set $\mathcal{S}_m$, where $\mathcal{S}_m$ represents a subset of k applications selected from the available pool, i.e., $\mathcal{S}_m \subseteq \{\texttt{App}_1, \texttt{App}_2, \ldots, \texttt{App}_N\}$ with $|\mathcal{S}_m| = k$, to accomplish a collaborative high-level task. Formally, the agent-environment loop in Equation 6 changes as follows:

$$\begin{aligned} A_t, P_t &= \text{Agent}(\texttt{ins}, \mathbf{V}_t, \mathcal{H}_t, \mathcal{S}_m) \\ \mathbf{V}_{t+1} &= \text{Env}(A_t, P_t) \\ \mathcal{H}_{t+1} &= \{\mathcal{H}_t, (V_t, A_t, P_t)\} \end{aligned} \tag{11}$$

Meanwhile, $\mathcal{H}_t$ now aggregates the interaction history across all relevant app environments. The process terminates when either the step limit is reached or a terminal action is predicted.

Task Collection and Design. L4 tasks are designed as an extension of the single-app automation tasks in L3, with a primary focus on multi-application collaboration and information transfer across heterogeneous interfaces. For tasks in existing benchmarks that inherently involve multiple applications, we included them in our evaluation after a careful review of their availability and robustness. In addition, for those benchmarks lacking native multi-app workflows, we manually designed new

tasks that explicitly require inter-app coordination. We also supplemented original multi-app tasks to further enrich the variety and complexity of cross-application scenarios.

A key design principle in constructing L4 tasks is to ensure that actions in one application provide necessary context or information for subsequent operations in another application. For example, in a representative macOS task, the agent is required to search online for the time and location of CVPR 2023 and then create a corresponding event in the Calendar app on the same date and month, but in the year 2090. To avoid issues related to time-sensitive information or changing event details, we decoupled the evaluation criteria from the actual event date, ensuring that the correctness of task completion is independent of the assessment time.

This systematic approach to task collection and design enables comprehensive evaluation of an agent's ability to reason globally, manage inter-app dependencies, and execute complex workflows that mirror real-world user demands in multi-application environments. In the lower part of Figure 3, we provide examples to illustrate how collaborative tasks involving two applications can be constructed.

Evaluation Metrics. We adopt the same evaluation metrics as in L3, i.e., SR and EQA. For both levels, the completion result is determined by verifying the final state and counting the number of steps taken, without the need to consider the individual states of multiple applications in $\mathcal{S}_m$.

3.6 Benchmark Statistics

Table 4 enumerates the complete task inventory, 8123 distinct instances, broken down by operating platform, level, and difficulty band. Our benchmark has the following characteristics:

- L1-GUI Content Understanding (3 × QA splits). Each of the six platforms contributes an identical triplet of 271/84/196/115/307/221 items (Windows → Web), yielding 1194 examples per difficulty (Easy, Medium, Hard) and 3582 in total. This symmetry ensures that any performance gap across the three difficulty tiers cannot be attributed to data imbalance.
- L2-GUI Element Grounding (Basic vs. Advanced). The grounding set is roughly 50% larger than Level 1, with 1787 examples per split (Basic=Advanced). Note the deliberate platform skew: mobile platforms (iOS + Android = 686 or 38%) receive more queries than desktop platforms, reflecting the higher UI diversity and screen density of mobile apps.
- L3-GUI Task Automation (single application). A compact but varied set of 719 trajectories focuses on long-horizon planning within one application. Linux dominates (268 tasks) to capture the complexity of desktop productivity apps, while mobile splits are omitted for this level to avoid conflating OS diversity with task length.
- L4-GUI Task Collaboration (multiple applications). The hardest tier comprises 248 cross-application workflows. Although smaller, it intentionally spans all three desktop platforms and major mobile browsers (47 Web tasks, 30 Android tasks) to stress test memory hand-off and state persistence.
- Aggregate balance. Across the whole benchmark Windows (1536) and Android (1758) provide the two largest pools, but no single platform exceeds 22% of the corpus, guarding against model over-specialisation. The progressive shrinkage, from 3582 (L1) to 248 (L4), mirrors the increasing cost and difficulty of annotation, while still offering enough samples (about 250) for a statistically meaningful evaluation in the top tier.

Overall, the benchmark delivers (1) platform diversity, (2) controlled difficulty gradation, and (3) a realistic taper in task count that matches real-world annotation effort, thereby enabling fine-grained diagnosis of GUI agent capabilities at every competence level.

4 Benchmarking GUI Agent Baselines

In this section, we evaluate a representative spectrum of contemporary VLM and LLM models, including both open-source and closed models, on the MMBench-GUI benchmark to provide a comprehensive portrait of current GUI-agent performance. MMBench-GUI supplies each method solely with screenshots and task descriptions, deliberately omitting auxiliary artifacts such as accessibility (A11y) trees and Set-of-Marks (SoM) data, thereby more closely mirroring real-world deployment

scenarios. Since different models possess varying capabilities, the set of models evaluated is not entirely consistent across different levels of tasks. The details are as follows:

L1&L2: Proprietary models: GPT-4o (Hurst et al., 2024), Claude-3.7 (Anthropic, 2025), Qwen-Max-VL (Bai et al., 2023). Open-sourced models: Qwen2.5 series (Bai et al., 2025), UI-TARS series (Qin et al., 2025), InternVL series (Zhu et al., 2025), Aguvis (Xu et al., 2024b), ShowUI (Lin et al., 2024), UGround (Gou et al., 2024), OS-Atlas (Wu et al., 2024b).

L3&L4: Proprietary models: GPT-4o (Hurst et al., 2024), Claude-3.7 (Anthropic, 2025). Open-sourced models: UI-TARS series (Qin et al., 2025), Qwen2.5-VL-72B (Bai et al., 2025), Aguvis (Xu et al., 2024b), GPT-4o+UGround-V1-7B (Gou et al., 2024), GPT-4o+UI-TARS-1.5-7B (Qin et al., 2025).

4.1 Benchmarking details

To ensure fairness, we evaluated all candidate models through a unified interface compatible with the OpenAI API protocol. Specifically, each model was deployed as an API-style service, and outputs were obtained by sending POST requests to the service endpoint along with the conversation input. For each model, we crafted both system and user prompts strictly based on official documentation or released code. For proprietary models, we designed detailed and effective prompts to elicit high-quality responses as faithfully as possible. Apart from model-specific settings, all other parameters, such as temperature and top-p, were kept consistent across evaluations.

During evaluation, the input and output processing pipeline was tailored to the requirements of each task level. For L1-GUI Content Understanding and L2-GUI Element Grounding, the input to the model comprised the GUI screenshot paired with either the relevant instruction or the question-options set. Model outputs were assessed using `exact-match` evaluation protocol, analogous to standard practices in grounding and QA tasks. However, given the variability in instruction-following abilities across different models, for example, the QA tasks in L1, we observed that some model outputs could not be reliably parsed. To address this, we implemented a hybrid parsing mechanism based on multiple regular expressions to robustly extract valid answers. In our codebase, we expose a customizable `parse_function` for each method, enabling tailored post-processing strategies to accommodate the unique output formats of various models.

For L3-GUI Task Automation and L4-GUI Task Collaboration, evaluation focused solely on whether the agent successfully achieved the desired end state, without the need to interpret intermediate natural language outputs. Therefore, parsing functions were not required for these levels; instead, we compared the final state directly against predefined success criteria to determine task completion.

4.2 Benchmark results on L1-GUI Content Understanding

Table 5 summarizes the performance of all evaluated models on the GUI Understanding task (L1) across three difficulty levels (Easy, Medium, Hard) and six platforms (Windows, MacOS, Linux, iOS, Android, Web), as well as the overall average. Across all settings, InternVL3-72B consistently achieves the highest scores, outperforming all other models on every platform and difficulty tier. Qwen2.5-VL-72B and Qwen-Max-VL generally rank just below InternVL3-72B. GPT-4o exhibits moderate performance, while the Claude variants (3.5 and 3.7) and UI-TARS-72B-DPO perform less favorably across all settings.

Several consistent trends emerge from the results:

- Difficulty effect: Model performance decreases as task difficulty increases, with scores on the Easy level always exceeding those on Medium and Hard levels.
- Cross-platform variability: For most models, macOS and Linux yield slightly higher scores, whereas Android and Web present greater variability and, in some cases, lower accuracy, indicating additional platform-specific challenges.
- Model ranking and robustness: InternVL3-72B maintains its leading position across all difficulty tiers (overall: 79.2%, 77.9%, and 75.7% on Easy, Medium, and Hard, respectively) and shows the smallest decline in performance as difficulty increases. Qwen2.5-VL-72B consistently ranks second, while GPT-4o experiences a sharper drop on harder items. The Claude variants and UI-TARS-72B-DPO show both lower accuracy and limited robustness across difficulty levels.

Table 5: **Performance on L1-GUI Content Understanding.** 'Overall' represents the aggregated score across all platforms, calculated as a weighted sum of individual platform scores. Here, the score of each platform is computed following Equation 4, where α adjusts credit based on the number of candidate options for a question.

Model	Windows	MacOS	Linux	iOS	Android	Web	Overall
Easy Level							
GPT-4o (2024)	62.47	67.89	62.38	58.52	56.41	58.51	60.16
Claude-3.5 (2024)	41.34	50.04	41.61	42.03	38.96	41.79	41.54
Claude-3.7 (2025)	34.66	49.05	39.37	42.76	37.45	40.80	39.08
Qwen-Max-VL (2023)	69.05	72.51	69.91	70.82	63.09	69.46	68.15
Qwen2.5-VL-72B (2025)	65.86	75.23	73.02	67.24	58.09	72.08	66.98
UI-TARS-72B-DPO (2025)	41.59	28.52	35.16	31.08	52.25	35.33	40.18
InternVL3-72B (2025)	**74.67**	**78.72**	**79.16**	**83.57**	**80.10**	**81.18**	**79.15**
Medium Level							
GPT-4o (2024)	56.33	63.13	59.70	54.06	57.69	54.98	57.24
Claude-3.5 (2024)	39.28	47.63	45.97	44.57	42.03	34.33	41.26
Claude-3.7 (2025)	39.34	39.23	42.28	39.45	36.05	36.17	38.39
Qwen-Max-VL (2023)	63.40	73.85	66.90	68.02	63.66	64.59	65.44
Qwen2.5-VL-72B (2025)	66.29	72.73	72.63	59.27	66.24	68.24	67.45
UI-TARS-72B-DPO (2025)	38.83	41.60	37.14	41.72	54.74	31.55	41.77
InternVL3-72B (2025)	**71.46**	**78.58**	**79.88**	**78.43**	**81.36**	**78.67**	**77.89**
Hard Level							
GPT-4o (2024)	60.69	60.38	52.42	45.27	50.93	50.83	53.49
Claude-3.5 (2024)	37.40	42.70	34.07	40.86	36.96	38.11	37.55
Claude-3.7 (2025)	32.99	34.48	31.97	39.20	36.99	38.92	35.65
Qwen-Max-VL (2023)	66.64	67.59	65.80	60.23	58.78	65.34	63.69
Qwen2.5-VL-72B (2025)	70.68	68.91	70.98	57.59	53.94	68.10	64.56
UI-TARS-72B-DPO (2025)	31.48	35.87	24.19	36.33	58.13	19.94	35.78
InternVL3-72B (2025)	**75.08**	**77.44**	**76.19**	**70.37**	**75.73**	**78.11**	**75.70**

Overall, these results demonstrate clear differences in model capabilities on GUI content understanding tasks, providing a solid quantitative basis for the in-depth analysis presented in the next section.

4.3 Benchmark results on L2-GUI Element Grounding

Table 6 reports the results of all evaluated models on the L2 task, including both Basic and Advanced instructions, across six platforms. We can summarize the following:

- Significant variation is observed among models. GPT-4o and Claude-3.7 exhibit extremely limited grounding ability, with scores consistently near zero across all platforms and instruction types. In contrast, open-source models such as UI-TARS-72B-DPO, InternVL3-72B, UGround-V1-7B, and Qwen2.5-VL-72B achieve substantially higher scores.
- The best-performing models (UI-TARS-72B-DPO and InternVL3-72B) demonstrate both high overall averages (74.25% and 72.20%, respectively) and strong cross-platform consistency. For example, UI-TARS-72B-DPO achieves average scores above 80% for MacOS, Android, and Web in the Basic setting, while maintaining robust performance on iOS (62.72%) and Linux (68.59%). InternVL3-72B similarly shows strong results across all platforms.
- A clear platform-dependent pattern emerges. For most high-performing models, grounding accuracy is generally higher on mobile (iOS, Android) and web platforms, with somewhat lower scores on desktop environments (Windows, MacOS, Linux). For instance, UI-TARS-72B-DPO achieves 93.54% on Android (Basic), 88.71% on Web (Basic), but comparatively lower scores on Windows (78.60%, Basic) and Linux (68.59%, Basic).

Table 6: **Performance on the L2-GUI Element Grounding.** "Adv." stands for advanced, while "Avg." refers to the weighted average of all results in a row, where the weights correspond to the proportion of tasks for each platform and mode relative to the total number of tasks.

Model	Windows		MacOS		Linux		iOS		Android		Web		Avg
	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	
GPT-4o (2024)	1.48	1.10	8.69	4.34	1.05	1.02	5.10	3.33	2.53	1.41	3.23	2.92	2.87
Claude-3.7 (2025)	1.48	0.74	12.46	7.51	1.05	0.00	13.69	10.61	1.40	1.40	3.23	2.27	4.66
Qwen-Max-VL (2023)	43.91	36.76	58.84	56.07	53.93	30.10	77.39	59.09	79.49	70.14	74.84	58.77	58.03
Aguvis-7B-720P (2024b)	37.27	21.69	48.12	33.27	33.51	25.00	67.52	65.15	60.96	50.99	61.61	45.45	45.66
ShowUI-2B (2024b)	9.23	4.41	24.06	10.40	25.13	11.73	28.98	19.70	17.42	8.73	22.90	12.66	15.96
OS-Atlas-Base-7B (2024b)	36.90	18.75	44.35	21.68	31.41	13.27	74.84	48.79	69.6	46.76	61.29	35.39	41.42
UGround-V1-7B (2024)	66.79	38.97	71.30	48.55	56.54	31.12	92.68	70.91	**93.54**	70.99	88.71	64.61	65.68
InternVL3-72B (2025)	70.11	42.64	75.65	52.31	59.16	41.33	**93.63**	80.61	92.70	78.59	**90.65**	65.91	72.20
Qwen2.5-VL-72B (2025)	55.72	33.82	49.86	30.06	40.31	20.92	56.05	28.18	55.62	25.35	68.39	45.78	41.83
Qwen2.5-VL-7B (2025)	31.37	16.54	31.30	21.97	21.47	12.24	66.56	55.15	35.11	35.21	40.32	32.47	33.85
UI-TARS-1.5-7B (2025)	68.27	38.97	68.99	44.51	64.40	37.76	88.54	69.39	90.45	69.29	80.97	56.49	64.32
UI-TARS-72B-DPO (2025)	**78.60**	**51.84**	**80.29**	**62.72**	**68.59**	**51.53**	90.76	**81.21**	92.98	**80.00**	88.06	**68.51**	**74.25**

- Model performance generally drops from Basic to Advanced instruction types. While top models maintain a high level on both, their scores under Advanced instructions are consistently lower than under Basic instructions, suggesting increased difficulty with more abstract or functional cues.

In summary, these results indicate wide gaps in GUI element grounding capabilities among current models, as well as persistent platform and instruction-type differences. We will make deeper analysis in the following sections.

4.4 Benchmark results on L3-GUI Task Automation and L4-GUI Task Collaboration

Tables 7 and 8 report the results for all models on single-app (L3) and multi-app (L4) GUI automation tasks under 15 and 50 steps. For all models, we employed a unified evaluation pipeline, using standardized prompts and action spaces for general-purpose models, and official configurations for GUI-specific agents.

For L3 tasks, overall performance is limited across all models and platforms. The best-performing method, GPT-4o + UI-TARS-1.5-7B, achieves an average SR of 26.60% , while most other models remain below 20%. The EQA scores follow a similar trend. Among GUI-specific models, UI-TARS-72B-DPO shows the best overall SR and EQA, particularly outperforming other agents on Linux and Android. Notably, language-centric models, that is, GPT-4o and Claude-3.7, perform less favorably across platforms and metrics. However, combining general-purpose models with GUI-specific grounders, such as UGround or UI-TARS, consistently boosts performance; for instance, GPT-4o alone achieves 6.13% SR, but this rises to over 17% with planner+grounder variants.

For L4, model success rates are considerably lower. The top method (GPT-4o + UI-TARS-1.5-7B) achieves only 8.78% average SR, and most models fall below 6%. This substantial drop compared to L3 underscores the increased difficulty in execution of cross-application tasks.

Increasing the maximum allowed steps from 15 to 50 improves SR and EQA values for all models and settings, but the overall task completion rates remain low, indicating that simply allowing longer action sequences does not fully address the challenges. This suggests that even with greater execution flexibility, many agents still struggle with effective long-horizon planning and multi-step task execution.

Platform-wise, Android and Web tend to yield higher SR and EQA for top-performing models (e.g., GPT-4o + UI-TARS-1.5-7B achieves SR/EQA of 33.10%/25.81% on Android and 20.72%/20.72%

Table 7: **Evaluation result of L3-GUI Task Automation.** Values in **bold** indicate the highest score within each group; underlined values indicate the second highest.

Model	Windows		MacOS		Linux		Android		Web		Avg	
	SR	EQA	SR	EQA	SR	EQA	SR	EQA	SR	EQA	SR	EQA
Max Step=15												
GPT-4o (2024)	5.56	3.27	0.00	0.00	6.83	4.35	18.97	8.93	1.94	1.53	7.14	4.05
Claude-3.7 (2025)	7.09	4.28	8.57	2.76	7.43	4.20	11.21	3.49	1.94	1.46	6.84	3.44
Aguvis-72B (2024b)	4.14	2.02	0.00	0.00	3.09	1.63	18.10	10.78	9.03	3.75	6.85	3.56
UI-TARS-1.5-7B (2025)	11.08	5.98	**11.43**	6.58	26.51	18.65	30.17	17.93	12.26	6.98	20.18	12.87
UI-TARS-72B-DPO (2025)	11.08	5.44	**11.43**	**7.79**	**30.31**	**18.91**	**43.10**	**26.62**	10.32	6.94	**23.27**	**14.31**
Qwen2.5-VL-72B (2025)	11.77	7.18	2.86	2.01	9.80	5.37	16.37	9.77	15.58	9.92	12.17	7.26
GPT-4o + UGround-V1-7B (2024)	13.10	**8.11**	2.86	1.00	16.13	8.69	34.48	21.14	**23.23**	**16.69**	19.36	11.93
GPT-4o + UI-TARS-1.5-7B (2025)	**14.52**	6.76	2.86	0.91	20.23	11.12	33.62	15.17	22.58	14.65	20.90	11.16
Max Step=50												
GPT-4o (2024)	3.49	2.26	2.86	1.65	11.64	9.05	21.55	10.81	3.23	2.24	9.35	6.13
Claude-3.7 (2025)	6.40	4.03	**11.43**	4.23	10.28	6.25	11.21	3.62	2.58	2.11	8.04	4.39
Aguvis-72B (2024b)	3.49	1.63	0.00	0.00	4.21	2.04	19.83	14.67	8.39	3.10	7.28	4.12
UI-TARS-1.5-7B (2025)	15.86	11.29	**11.43**	7.03	29.82	21.26	31.58	22.15	14.19	9.22	23.02	16.10
UI-TARS-72B-DPO (2025)	17.93	11.84	**11.43**	**8.38**	**31.38**	**25.44**	45.69	35.22	9.68	7.53	25.33	**19.58**
Qwen2.5-VL-72B (2025)	9.66	6.86	5.71	3.96	10.63	7.85	27.59	21.80	14.38	9.74	13.74	10.12
GPT-4o + UGround-V1-7B (2024)	20.73	11.89	5.71	3.18	19.48	10.91	**47.41**	**37.19**	**26.45**	**22.50**	25.07	17.50
GPT-4o + UI-TARS-1.5-7B (2024)	**26.21**	**17.28**	8.57	5.01	22.85	13.82	42.24	33.10	25.81	20.72	**26.60**	18.69

Table 8: **Evaluation result of L4-GUI Task Collaboration.** "-" represents that these model's action space can't handle browser tab switch situation, so we don't test them.

Model	Windows		MacOS		Linux		Android		Web		Avg	
	SR	EQA	SR	EQA	SR	EQA	SR	EQA	SR	EQA	SR	EQA
Max Step=15												
GPT-4o (2024)	7.49	5.90	0.00	0.00	3.50	2.41	0.00	0.00	2.13	0.10	2.85	1.80
Claude-3.7 (2025)	3.57	1.61	2.86	**2.01**	7.32	4.76	0.00	0.00	2.13	0.03	4.30	2.46
Aguvis-72B (2024b)	3.21	3.01	0.00	0.00	1.62	0.37	3.33	3.15	-	-	1.50	0.94
UI-TARS-1.5-7B (2025)	3.21	3.01	2.86	0.82	4.95	3.98	6.67	6.58	-	-	3.68	2.96
UI-TARS-72B-DPO (2025)	3.21	3.05	**5.71**	1.47	**7.46**	**5.87**	10.00	9.64	-	-	5.53	4.20
Qwen2.5-VL-72B (2025)	6.24	4.20	0.00	0.00	2.53	1.65	6.67	6.08	-	-	3.35	2.47
GPT-4o + UGround-V1-7B (2024)	9.27	5.12	0.00	0.00	3.60	2.51	3.33	3.22	**4.26**	0.61	3.94	2.23
GPT-4o + UI-TARS-1.5-7B (2025)	**12.30**	**6.36**	0.00	0.00	5.58	3.82	**23.33**	**21.12**	**4.26**	**0.68**	**7.6**	**5.13**
Max Step=50												
GPT-4o (2024)	6.24	4.98	0.00	0.00	5.94	5.35	0.00	0.00	2.13	1.52	3.68	3.16
Claude-3.7 (2025)	6.24	4.34	2.86	2.03	**9.30**	**7.35**	0.00	0.00	2.13	0.04	5.47	3.90
Aguvis-72B (2024b)	3.21	3.14	0.00	0.00	1.62	0.37	6.67	6.42	-	-	1.91	1.36
UI-TARS-1.5-7B (2025)	6.24	6.00	2.86	0.89	7.63	5.50	13.33	13.07	-	-	6.00	4.78
UI-TARS-72B-DPO (2025)	9.27	6.22	**5.71**	**2.27**	8.45	7.19	20.00	11.78	-	-	7.96	5.55
Qwen2.5-VL-72B (2025)	6.24	5.23	0.00	0.00	1.62	1.31	6.67	6.51	-	-	2.90	2.82
GPT-4o + UGround-V1-7B (2024)	9.27	5.03	0.00	0.00	5.48	3.75	6.67	6.40	0.00	0.00	4.31	2.99
GPT-4o + UI-TARS-1.5-7B (2024)	**12.30**	**6.84**	2.86	0.95	7.46	5.59	**23.33**	**21.65**	**4.26**	**2.02**	**8.78**	**6.37**

on Web, compared to 26.21%/17.28% on Windows and 8.57%/5.01% on macOS), while desktop environments, especially macOS, generally show lower results.

The results from Tables 7 and 8 highlight both the effectiveness of combining planning and grounding for L3 tasks and the substantial gap in agent performance when moving to L4 scenarios, especially under long-horizon and multi-step conditions.

5 Analysis and Discussion

In this section, we conduct an in-depth analysis to delve into the underlying causes and implications reflected in our benchmark results. Our investigation is structured around three primary dimensions: platform, task, and model, and adheres to a single-variable control principle to ensure the validity of our comparisons. Through systematic examination and post-processing of the empirical results along these axes, we distill a series of actionable findings that reveal the fundamental bottlenecks currently constraining agent performance. These findings not only elucidate the essential challenges facing contemporary GUI agents, but also offer valuable guidance for future research and development in this domain.

Finding 1: General-purpose language models excel at task decomposition, planning, and self-reflection but struggle with fine-grained visual interactions.

Across different model categories, general-purpose language models, exemplified by GPT-4o and Claude, demonstrate pronounced limitations in fine-grained GUI tasks. As shown in Table 6 and the right part of Figure 4, their average scores in L2 are merely 2.87 for GPT-4o and 4.66 for Claude-3.7, in contrast to the specialized visual grounding model UGround-V1-7B, which achieves a score of 65.68%. This discrepancy underscores a key limitation: general-purpose models inherently lack the capacity for accurate perception and localization of UI components. A similar trend emerges in L3 tasks. For instance, GPT-4o alone achieves success rates (SR) of only 4.05%/6.13% in single-app automation scenarios (Max Step = 15/50, see Table 7). However, when paired with domain-specific grounding modules such as UGround-V1-7B or UI-TARS-1.5-7B, the SR of GPT-4o rises substantially to 11.93%/17.50%. This phenomenon suggests that specialized perception modules can effectively compensate for the perceptual shortcomings of general-purpose LLMs.

Beyond the two direct strategies, namely, incorporating auxiliary localization modules during training and increasing the amount of fine-grained perceptual data, a more fundamental and forward-looking direction lies in embracing a modular architecture. This approach enables the model to dynamically interface with external modules based on its own capability gaps (e.g., visual grounding), effectively allowing for targeted augmentation through specialized "external agents". This architecture not only compensates for inherent deficiencies but also promotes a flexible, cooperative paradigm in which general-purpose models can be extended and adapted to complex GUI automation tasks.

Finding 2: Accurate visual grounding significantly determines the success rate of GUI task execution.

The full decision-making pipeline of a GUI agent can be abstracted into three stages: perceive accurately $\Rightarrow$ reason properly $\Rightarrow$ act precisely. If the first step (element localization) fails, subsequent planning and reasoning, no matter how advanced, are unlikely to compensate. To examine the critical role of localization, we systematically assessed its impact on downstream automation tasks (L3 and L4), and conversely, investigated whether enhanced planning alone could offset poor visual grounding.

We designed two complementary experimental setups as shown in the left part of Figure 4: (1) fixing the planner while incrementally improving the grounder, and (2) fixing the grounder while varying the planner. Correlation analyses revealed a clear pattern: with the same planner, improving localization alone led to a $2.8\times$ ($\Delta = 17.25$) increase in SR. In contrast, when localization performance remained roughly constant, replacing the planner with a stronger VLM yielded marginal returns ($1.15\times$, $\Delta = 3.58$). These results lead to a clear conclusion: visual grounding is the primary performance bottleneck. Gains from improved localization are nearly linear, whereas once the agent "sees well enough," the marginal utility of enhancing its reasoning diminishes.

This finding underscores that, at the current stage, the most leverage-efficient breakthrough for improving GUI task automation lies in advancing high-precision, cross-platform visual localization

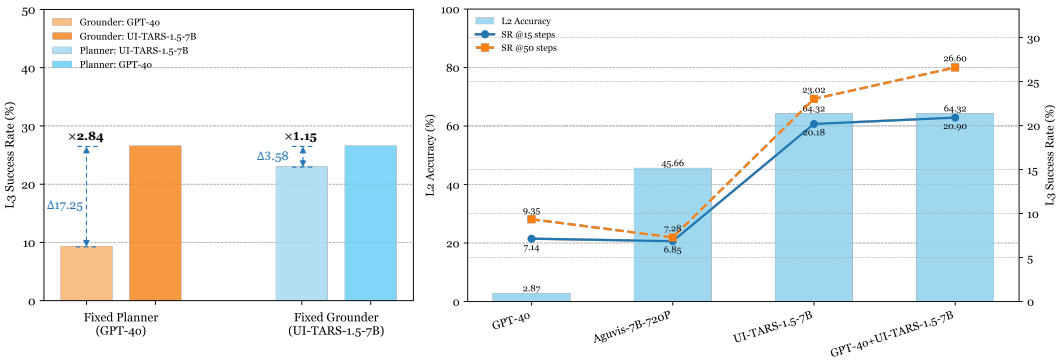


Figure 4: **Left:** Demonstrates the relative contribution of visual grounding versus planning in driving performance gains under current conditions. We consider two experimental conditions—fixing the planner while varying the grounder, and vice versa—and examine how different combinations affect task success rate. Similar color hues denote groups with the same fixed planner or grounder. **Right:** Task success grows roughly linearly with visual-grounding accuracy. General-purpose language models are virtually "blind" at the L2 grounding stage, which drives their L3 automation success rate (SR) sharply down. Plugging in a dedicated visual grounder restores precise perception and, in turn, lifts SR dramatically—highlighting fine-grained grounding as the principal bottleneck.

capabilities. As also suggested by Finding 1, within a modular architecture, the visual grounder should be treated as the first and most critical plug-in component. Ensuring its reliability provides a solid foundation upon which LLM-based planning, long-range memory, and reflection mechanisms can be effectively layered.

Finding 3: Efficiency, including step minimization and early stopping, is a critical yet underexplored dimension of GUI agent performance.

The introduction of the EQA metric enables us to move beyond evaluating whether an agent simply completes a task, by shifting attention to how efficiently the task is accomplished. This novel perspective facilitates deeper insights through a more fine-grained analysis of agent behavior.

We additionally compute two derived metrics, $\frac{EQA}{SR}$ and $SR - EQA$, to facilitate a more comprehensive analysis. Based on the definition of the EQA and SR, we further reformulate them as:

$$EQA = \frac{1}{N}\sum_{i\in\mathcal{C}}(1-u_i), \quad SR = \frac{|\mathcal{C}|}{N}, \tag{12}$$

where $\mathcal{C}$ denotes the set of all successfully completed tasks, and $u_i = \frac{T_i}{T_{max}} \in (0, 1]$ represents the normalized completion step of task i within the global step budget. From this, $\frac{EQA}{SR}$ and $SR - EQA$ can be derived as:

$$\frac{EQA}{SR} = \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}(1-u_i) = 1 - \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}u_i, \tag{13}$$

$$SR - EQA = \frac{1}{N}\sum_{i\in\mathcal{C}}u_i = SR \times (\frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}u_i), \tag{14}$$

where $\frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}u_i$ denotes the average steps in which a task is completed.

Therefore, $\frac{EQA}{SR}$ has an intuitive physical interpretation: it reflects the average remaining steps per successful task. Its upper bound is 1, which corresponds to the idealized case where all successful tasks are completed almost immediately (i.e., at the first step). Conversely, its lower bound is 0, indicating that all successful completions occur only at the very end of the allowed budget. $\frac{EQA}{SR}$ quantifies how many steps, on average, are consumed before successful completion. Meanwhile, $SR - EQA$ also has an intuitive physical interpretation: it is approximately proportional to the total normalized time consumed across all successful tasks, and can be interpreted as a "redundant step bill". A larger difference between EQA and SR implies a greater average normalized completion time

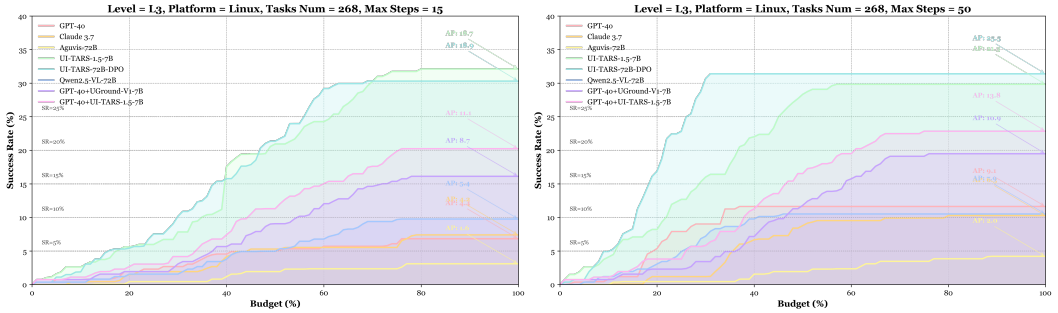


Figure 5: **EQA visualization across different models under L3 for different allowed steps.** As discussed in Section 3.4, EQA reflects a combination of task completion and efficiency (i.e., the number of steps used upon completion). In practice, we compute it by interpolating both the step budget and the success rate (SR) 100 times. The area under the curve formed by these interpolated SR values yields the final EQA score.

Model	ΔSR	ΔEQA	EQ^1_{15}	EQ^1_{50}	EQ^2_{15}	EQ^2_{50}	ΔEQ^1	ΔEQ^2
GPT-4o	2.21	2.08	0.567	0.656	3.09	3.22	0.088	0.13
Claude-3.7	1.2	0.95	0.503	0.546	3.40	3.65	0.043	0.25
Aguvis-72B	0.43	0.56	0.520	0.566	3.29	3.16	0.046	-0.13
UI-TARS-1.5-7B	2.84	3.23	0.638	0.699	7.31	6.92	0.062	-0.39
UI-TARS-72B-DPO	2.06	5.27	0.615	0.773	8.96	5.75	0.158	-3.21
Qwen2.5-VL-72B	1.57	2.86	0.597	0.737	4.91	3.62	0.140	-1.29
GPT-4o+UGround-V1-7B	5.71	5.57	0.616	0.698	7.43	7.57	0.082	0.14
GPT-4o+UI-TARS-1.5-7B	5.7	7.53	0.534	0.703	9.74	7.91	0.169	-1.83
Avg.	2.72	3.51	0.574	0.672	6.02	5.23	0.098	-0.79

Table 9: Additional metrics derived by SR and EQA. Here, EQ^1_{15} and EQ^2_{15} denotes for $\frac{EQA}{SR}$ and SR − EQA, respectively, when the maximal step is 15. $\Delta EQ^1 = EQ^1_{50} - EQ^1_{15}$ and so is the ΔEQ^2. Similarly, $\Delta SR = SR_{50} - SR_{15}$ and $\Delta EQA = EQA_{50} - EQA_{15}$

u_i for the successful set, meaning that tasks tend to be completed closer to the end of the budget—i.e., with more redundant steps. Conversely, a smaller difference (approaching zero) indicates that most successful tasks are completed early, near the beginning of the budget, suggesting minimal or no redundancy. Thus, the magnitude of the gap between EQA and SR effectively captures how "wasteful" the agent is, even among the tasks it completes.

We re-organize the $\frac{EQA}{SR}$ and $SR - EQA$ using the average results in Table 7 as EQ^1 and EQ^2, and present the aggregated findings in Table 9. Combining with Figure 5, we can disclose four complementary patterns. First, the modular pairing of a powerful planner with a specialized grounder, exemplified by GPT-4o + UGround-V1-7B and GPT-4o + UI-TARS-1.5-7B, elevates the success rate under a 50-step budget by roughly 5.7%, yet still incurs a substantial redundant step cost (EQ^2 = 7-8), signaling that cross-module coordination and early termination heuristics remain inadequate. Second, the large-scale DPO-aligned UI-TARS-72B-DPO achieves the strongest efficiency profile, increasing EQ^1 to 0.773 while compressing EQ^2 from 8.96 to 5.75 (ΔEQ^2 = -3.21); this demonstrates that aligning to human preferences that explicitly reward rapid task completion can translate directly into tangible efficiency gains. Third, general-purpose agents such as GPT-4o and Claude-3.7 extract minimal benefit from a longer budget (ΔSR<2.5%) and even exhibit higher redundant step costs (EQ^2 increases from 3.09 to 3.22 and 3.40 to 3.65, respectively), underscoring that simply extending the interaction horizon cannot compensate for their limited visual granularity and action precision, therefore, integrating specialized perception or actuation modules is becoming indispensable. Lastly, none of the curves in Figure 5 attains the ideal "hug-the-top-left-corner" profile, underscoring a pervasive lack of effective early-stopping heuristics and cost-aware search strategies.

To mitigate the efficiency bottlenecks aforementioned, we identify three possible research avenues. (1) Confidence- or value-based early-termination policies: equip agents with stopping rules that immediately end an episode when the marginal utility of further actions falls below a threshold,

rather than passively consuming the entire step budget. (2) Cost-sensitive fine-tuning: during reinforcement-learning (or DPO-style) alignment, impose explicit penalties for every superfluous action so that optimization shifts from maximizing success rate (SR) alone to jointly maximizing the success-conditioned efficiency score EQA. (3) Progress-aware self-reflection: require the planner to periodically estimate the set of remaining sub-goals and, upon detecting that all objectives are satisfied, issue an immediate FINISH action. Together, these interventions target the twin goals of cutting redundant steps and encouraging agents to "know when to stop", thereby narrowing the gap between current GUI agents and human-level operational efficiency.

Finding 4: The limitation of action space restricts the agent's ability to execute planned actions, especially in GUI task collaboration scenarios.

In Table 8, a notable fraction of models fail to complete the task on the web platform. The underlying cause is that, during web-interaction execution, the models lack the ability to trigger action `switch_tab` to enable 'press Tab to switch tabs'. In headless-browser settings, this omission blocks seamless navigation across multiple tabs, preventing cross-window information from being transferred from one context to another and ultimately derailing task completion.

On the other hand, due to the inherent heterogeneity of interactions across desktop, mobile, and web platforms, the current prompt-based definition of action functions struggles to comprehensively capture the full spectrum of platform-specific operations. Moreover, during inference, models may confuse actions across platforms, producing incorrect or incompatible output actions. Such issues can directly lead to task failure, even in single-platform, multi-app scenarios, and become particularly pronounced in multi-platform, multi-app settings, for example, when copying text from a web page and pasting it into a desktop application like Word for further formatting.

Building on these observations, we argue that a more generalizable, extensible, and potentially platform-agnostic definition of the action space is worth pursuing. One intuitive and straightforward direction is to construct a unified API abstraction layer that comprehensively covers multi-platform operations. Under this design, the agent interacts with the environment by invoking platform-independent APIs, while the backend of the API is responsible for platform-specific adaptations. An alternative route focuses on operation atomization. Unlike current action spaces that rely on fixed, platform-tied commands, an ideal action space would emphasize a set of primitive operations, decoupled from any particular environment. Agent-issued instructions are then mapped to these primitives via a many-to-many translation schema, where each high-level intent may correspond to a combination of atomic steps. These atomic units can then be recompiled into platform-specific execution commands, enabling robust and consistent interaction across environments. Beyond these two approaches, we believe that the research community should continue to explore better formulations of the action space, those characterized by strong generality, high extensibility, and minimal platform dependence.

Finding 5: Although many GUI agents excel in simple cases, their effectiveness diminishes significantly as task complexity rises, revealing limited generalization capabilities.

As shown in Figure 6, although many systems perform impressively on easy scenarios, their accuracy/success rate deteriorates sharply as soon as either (i) the local difficulty within a level increases (easy $\rightarrow$ medium $\rightarrow$ hard; basic $\rightarrow$ advanced) or (ii) the global task complexity rises from L1 to L4. These steep drops - especially pronounced for general-purpose LLMs - indicate that today's agents still lack robust generalization to harder, less stereotyped GUI situations. For example, the GUI understanding score of GPT-4o drops from 60.2% (easy) to 53.5% (hard), a -11% decrease, while even the highly tuned InternVL3-72B loses 4% (Table 5). In element grounding, switching from 'Basic' to semantically implicit 'Advanced' queries slashes GPT-4o's mean accuracy by nearly 40% and still costs the specialist UI-TARS-72B-DPO 16% (Table 6). The effect compounds across levels: the strongest agent (GPT-4o + UI-TARS-1.5-7B) succeeds in 26.6% of tasks at L3 but only 8.8% once multi-app collaboration is required in L4, a 67% collapse that is mirrored by other models (Tables 7–8). Concomitant declines in EQA confirm that agents not only fail more often but also waste proportionally more steps before failing.

These sharp drops expose three intertwined bottlenecks: (1) ill-posed perceptual clues (small widgets, non-salient text), (2) longer credit-assignment chains, and (3) noisy action spaces inflate the search space exponentially. Current models, trained largely on static screenshots, lack the robust abstract representations and error-driven exploration strategies needed to cope.

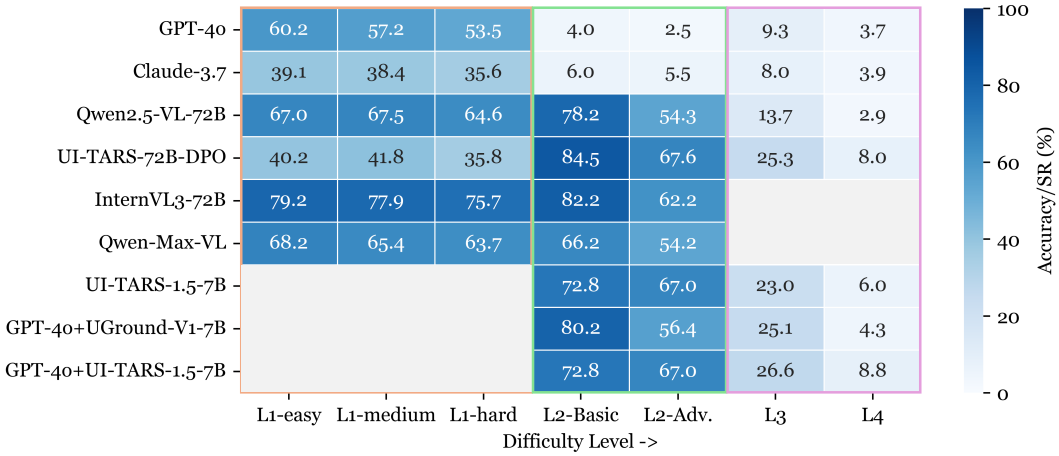


Figure 6: **Difficulty-Gradient Heatmap.** Models' scores across difficulty levels are encoded with a single-hue palette whose saturation fades from high (dark) to low (light). Colored rectangles outline comparable model groups. Within and across these groups, the color consistently fades from L1, L2 to L3 and L4, indicating that higher task complexity amplifies each model's weaknesses and causes a steep performance drop-off.

Possible targeted remedies include: (1) Curriculum & hard-negative mining. Intentionally up-sample adversarial layouts (occlusion, theme changes, deceptive affordances) during instruction tuning to inoculate perceptual modules against distribution shift. (2) Dynamic skill routing. Teach planners to self-diagnose uncertainty and automatically invoke auxiliary skills (OCR, vision transformers, memory retrieval) as difficulty rises. (3) Hierarchical planners with macro-actions. Introduce option-level abstractions (e.g., `open-browser-tab`) so that sparse EQA-style rewards can flow to high-level decisions instead of individual clicks. (4) Unified state schema for all applications. Store "App → Page → Element" graphs in an external memory that survives context switches, allowing the planner to reason over shared entities rather than raw pixel buffers.

We believe that by attacking these verified failure modes, the community can turn today's hardest cases, from implicitly described buttons to multi-window workflows, into stepping-stones toward truly general-purpose GUI agents.

Finding 6: The failures in multi-application environments primarily stem from limited cross-context memory and action space, rather than issues with perception or planning.

Success drops that cannot be explained by harder screenshots or longer action chains alone appear as soon as the agent must pass information between applications. The strongest single-app system, GPT-4o+UI-TARS-1.5-7B, falls from **26.6%** SR on L3 to just **8.8%** on L4 (Tables 7–8); UI-TARS-72B-DPO shows an almost identical collapse (25.3% to 8.0%). Failures concentrate at window or tab boundaries: five models are labeled '-' on the Web platform simply because they cannot express the primitive `switch_tab`. At the same time, EQA shrinks far more than the accompanying $SR - EQA$ penalty (e.g., 18.7% → 6.4% for GPT-4o + UI-TARS), signaling that agents waste many steps rediscovering the context they have just lost. These phenomena point to a deficit in working memory and action-space coverage, rather than in perception or generic planning.

Addressing these failures may require agents to focus on memory-centric research avenues, including: (1) External episodic buffer. Log every UI observation and write-back (*copy*, *navigate*, *paste* ...) to an append-only timeline that the language planner can query with natural language—much like retrieval-augmented generation, but for GUI states. (2) Semantic anchors. Tag entities (e.g., "flight-price $514") with stable IDs when first seen; subsequent references use the anchor, so the planner no longer depends on window focus to recall an object. (3) Cross-context consistency checks. Inject lightweight assertions, for example, "clipboard should now contain X" and "target window title equals Y". Violations trigger immediate self-repair instead of long, fruitless trial-and-error loops, cutting the redundant steps that dominate L4 failures.

6 CONCLUSION

In this work, we presented MMBench-GUI, a novel hierarchical multi-platform evaluation framework that comprehensively assesses the capabilities and limitations of GUI automation agents. Through rigorous evaluations across multiple operating systems and diverse tasks, we uncovered critical insights into key performance bottlenecks, particularly highlighting the importance of accurate visual grounding, sophisticated planning, and robust cross-platform generalization. Our findings demonstrate that modular architectures integrating specialized grounding modules significantly improve performance, addressing inherent limitations of general-purpose language models. Additionally, our analysis underscores the importance of improving long-horizon reasoning, adaptive error recovery, and effective memory and state management to address complex and ambiguous GUI scenarios. MMBench-GUI thus provides a foundational benchmarking resource and actionable guidance for future research efforts, advancing the development of robust, reliable, and practically applicable GUI automation agents.

FUTURE WORK

We will strengthen our study along three aspects: (1) Broader model coverage. We will evaluate a wider spectrum of models—including open-source, proprietary, and the latest RL-based systems—so that each model is tested across all difficulty levels. (2) Deeper analysis. With a richer experimental pool, we will perform fine-grained analyses to produce more robust and generalizable findings. (3) Task expansion & error attribution. We plan to add more online tasks to cover a broader set of applications, validate their correctness step by step, and log sufficient runtime details to pinpoint the exact causes of failure.

REFERENCES

Sonnet Anthropic. Model card addendum: Claude 3.5 haiku and upgraded claude 3.5 sonnet. 2024. URL https://api.semanticscholar.org/CorpusID:273639283.

Sonnet Anthropic. Claude 3.7 sonnet system card. 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

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

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, et al. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*, 2024a.

Xingyu Chen, Zihan Zhao, Lu Chen, Danyang Zhang, Jiabao Ji, Ao Luo, Yuxuan Xiong, and Kai Yu. Websrc: a dataset for web-based structural reading comprehension. *arXiv preprint arXiv:2101.09465*, 2021.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024b.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023a.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023b. URL https://openreview.net/forum?id=kiYqbO3wqw.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, et al. Assistgui: Task-oriented desktop graphical user interface automation. *arXiv preprint arXiv:2312.13108*, 2023.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Yu-Chung Hsiao, Fedir Zubach, Gilles Baechler, Victor Carbune, Jason Lin, Maria Wang, Srinivas Sunkara, Yun Zhu, and Jindong Chen. Screenqa: Large-scale question-answer pairs over mobile app screenshots. *arXiv preprint arXiv:2209.08199*, 2022.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178. Springer, 2024.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent, 2024. URL https://arxiv.org/abs/2411.17465.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*, 2024a.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024b.

Xinyi Liu, Xiaoyi Zhang, Ziyun Zhang, and Yan Lu. Ui-e2i-synth: Advancing gui grounding with large-scale instruction synthesis. *arXiv preprint arXiv:2504.11257*, 2025.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pp. 216–233. Springer, 2024c.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*, 2024.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*, 2022.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, et al. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*, 2025.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. 2024.

OpenAI. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, et al. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*, 2025.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*, 2025.

Bowen Wang, Xinyuan Wang, Jiaqi Deng, Tianbao Xie, Ryan Li, Yanzhe Zhang, Gavin Li, Toh Jing Hua, Yu Su, Diyi Yang, Yi Zhang, Zhiguo Wang, Victor Zhong, and Tao Yu. Computer agent arena: Compare & test computer use agents on crowdsourced real-world tasks, 2025. URL https://arena.xlang.ai.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024a.

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

LLM-Core-Team Xiaomi. Mimo-vl technical report, 2025. URL https://arxiv.org/abs/2506.03569.

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

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025.

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

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024b.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*, 2024.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9556–9567, 2024.

Junlei Zhang, Zichen Ding, Chang Ma, Zijie Chen, Qiushi Sun, Zhenzhong Lan, and Junxian He. Breaking the data barrier–building gui agents through task generalization. *arXiv preprint arXiv:2504.10127*, 2025.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=piecKJ2DlB.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Yuchen Duan, Hao Tian, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.