

# 000 GUI KNOWLEDGE BENCH: REVEALING THE KNOWL- 001 EDGE GAP BEHIND VLM FAILURES IN GUI TASKS 002

003 **Anonymous authors**  
004

005 Paper under double-blind review  
006

## 007 ABSTRACT 008

009 Large vision–language models (VLMs) have advanced graphical user interface  
010 (GUI) task automation but still lag behind humans. We hypothesize this gap stems  
011 from missing core GUI knowledge, which existing training schemes (such as su-  
012 pervised fine-tuning and reinforcement learning) alone cannot fully address. By  
013 analyzing common failure patterns in GUI task execution, we distill GUI knowl-  
014 edge into three dimensions: (1) interface perception, knowledge about recognizing  
015 widgets and system states; (2) interaction prediction, knowledge about GUI inter-  
016 action conventions; and (3) instruction understanding, knowledge about proce-  
017 dural knowledge of GUI operations. We further introduce GUI Knowledge Bench, a  
018 benchmark with multiple choice and yes/no questions across six platforms (Web,  
019 Android, MacOS, Windows, Linux, IOS) and 292 applications. Our evaluation  
020 shows that current VLMs identify widget functions but struggle with perceiving  
021 system states, predicting actions, and interpreting task goals. Experiments on real  
022 world GUI tasks further validate the close link between GUI knowledge and task  
023 success. By providing a structured framework for assessing GUI knowledge, our  
024 work supports the selection of VLMs with greater potential prior to downstream  
025 training and provides insights for building more capable GUI agents.  
026

## 027 1 INTRODUCTION 028

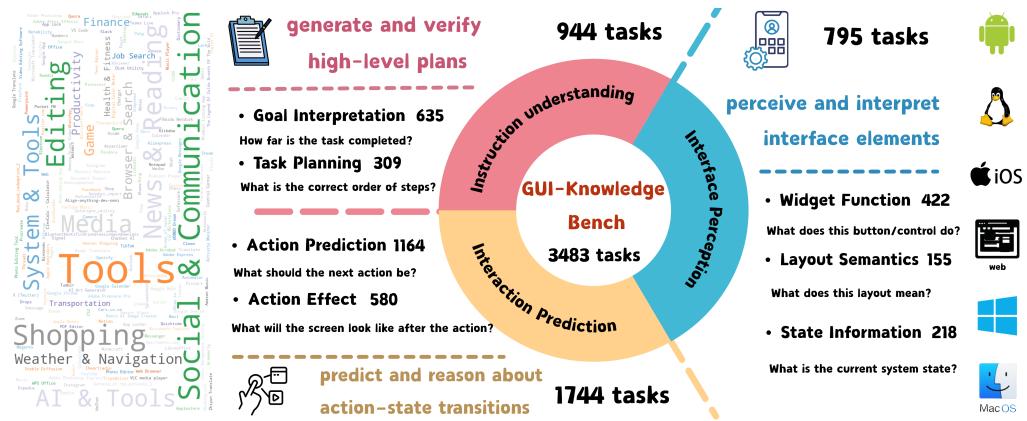


Figure 1: GUI Knowledge Bench: A benchmark evaluating VLMs on GUI knowledge across six platforms (Web, Android, MacOS, Windows, Linux, IOS). It measures three types of knowledge: Interface Perception, which evaluates understanding of GUI components, layout, and system state; Interaction Prediction, which assesses the knowledge of GUI interaction conventions; and Instruction Understanding, which tests whether a model knows the procedural knowledge of completing a GUI task.

Graphical User Interface (GUI) task automation, such as booking a flight, editing a presentation, or configuring system settings, poses unique challenges for AI agents (Wu et al., 2024a; Hong et al., 2024; Xu et al., 2024a; He et al., 2024). Recent approaches have leveraged large vision–language

054 models (VLMs) with techniques such as prompt engineering (Agashe et al., 2025; Xie et al., 2025a),  
 055 supervised fine-tuning (SFT) (Wu et al., 2024b; Hong et al., 2024; Lin et al., 2025; Liu et al., 2025;  
 056 Xu et al., 2024b), and reinforcement learning (RL) (Lian et al., 2025; Luo et al., 2025), achieving  
 057 strong task performance in many applications. However, GUI agents still fail in many real-world  
 058 scenarios (Xie et al., 2025c). For example, agents may misinterpret widget functions in unfamiliar  
 059 applications, fail to predict correct action parameters, or struggle with multi-step planning and error  
 060 recovery in long-horizon GUI tasks. Our analysis suggests that a primary reason for these failures  
 061 is that the used VLMs lack the necessary GUI knowledge. While prompt engineering, SFT, and RL  
 062 can improve reasoning, grounding, and planning abilities, they contribute little to injecting new GUI  
 063 knowledge (Ovadia et al., 2024), which also plays important roles in solving GUI tasks.

064 Different from most existing benchmarks that primarily evaluate task success, which mainly focus on  
 065 the grounding (Li et al., 2025; Cheng et al., 2024; Jurmu et al., 2008), reasoning, and planning (Lin  
 066 et al., 2024) capabilities of GUI agents, our work targets the missing dimension of knowledge eval-  
 067 uation. To systematically examine these knowledge gaps, we introduce GUI-Knowledge Bench, a  
 068 benchmark designed to assess the extent of GUI knowledge encoded in VLMs prior to downstream  
 069 tasks, while also serving as a diagnostic tool to guide the design of VLM-based agent systems. The  
 070 benchmark is constructed from over 40,000 screenshots and 400 execution trajectories spanning 292  
 071 applications across six platforms (Web, Android, MacOS, Windows, Linux, IOS)). Through a com-  
 072 bination of automated generation and manual annotation, we derive a set of 3483 knowledge-centric  
 073 questions that systematically test VLMs’ knowledge in GUI.

074 We categorize the GUI knowledge into three complementary aspects derived from common agent  
 075 failure modes: (1) interface perception, which involves recognizing widget functions, layout se-  
 076 mantics, and perceiving state information (e.g., enabled/disabled, selected/focused); (2) interaction  
 077 prediction, which involves assessing knowledge of GUI interaction conventions (e.g., what changes  
 078 after toggling a switch or submitting a form, and which parameters are required); and (3) instruction  
 079 understanding, which focuses on grounding natural-language instructions into executable, multi-  
 080 step operation sequences with coherent plans. This categorization enables a systematic examination  
 081 of which components of GUI knowledge are already present in current models and which remain  
 082 underdeveloped.

083 Our evaluation reveals that current VLMs are still short of enough knowledge in these three cat-  
 084 egories for completing real world GUI tasks. First, although VLMs perform well at discerning  
 085 different widget functions and layout semantics but struggle to accurately perceive system states.  
 086 Second, VLMs underperform in interaction prediction, showing difficulties in anticipating correct  
 087 action outcomes and required parameters. They frequently confuse click actions with other types  
 088 of actions, a behavior commonly observed in many models. Third, VLMs struggle with judging  
 089 task completion states and understanding human instructions. Some tasks are easy to complete, yet  
 090 they still fail because the models do not understand the goals of the tasks. These findings highlight  
 091 critical gaps in the internal GUI knowledge of current VLMs, revealing that while they can per-  
 092 ceive interface elements, their understanding about system states and interaction outcomes remains  
 093 limited. Our contributions are as followed:

- 094 • We introduce GUI-Knowledge bench, designed to evaluate GUI knowledge in both both  
 095 general and GUI-tuned VLMs. Experiments on real world GUI environment further vali-  
 096 dates the close link between GUI knowledge and task success.
- 097 • Our evaluation identifies key gaps in perceiving system states, understanding the effect of  
 098 common GUI interactions, and judging task completion, providing guidance for selecting  
 099 or training VLMs prior to downstream GUI tasks.

## 100 2 RELATED WORK

### 101 2.1 GUI AGENT

102 Progress in GUI task automation has largely relied on pretrained vision-language models (VLMs),  
 103 with improvements driven by supervised fine-tuning (SFT), reinforcement learning (RL), and syn-  
 104 thetic data generation. SFT-based methods train VLMs on large-scale GUI datasets to enhance  
 105 element grounding and action prediction, as seen in OS-Atlas (Wu et al., 2024b), CogAgent (Hong  
 106 et al., 2024), and ShowUI (Lin et al., 2025), while multi-stage pipelines such as InfiGUIAgent (Liu  
 107 et al., 2025),

108

109  
110  
111  
Table 1: Comparison of existing GUI benchmarks and our proposed benchmark across evaluation  
scope, operating system coverage, application diversity, and data scale. Our benchmark systemati-  
cally spans multiple OS and applications with a comprehensive scope of GUI knowledge evaluation.

112

Benchmark	Scope	OS	Apps	Task Num.
ScreenSpot-Pro (Li et al., 2025)	Action	3	23	1581
SeeClick (Cheng et al., 2024)	Action	5	20+	1272
VideoGUI (Lin et al., 2024)	Task	1	11	463
OSWorld (Xie et al., 2025c)	Task	1	9	369
MacOSworld (Yang et al., 2025)	Task	1	30	202
AndroidWorld (Rawles et al., 2024)	Task	1	20	116
MMBench-GUI (Wang et al., 2025)	Knowledge	6	-	8000+
Web-CogBench (Guo et al., 2025)	Knowledge	1	14	876
<b>GUI-Knowledge-Bench</b>	Knowledge	<b>6</b>	<b>292</b>	<b>3483</b>

122

123

124

125  
et al., 2025) and Aguvis (Xu et al., 2024b) further inject reasoning and planning abilities with syn-  
126  
thetic data. RL approaches, including UI-AGILE (Lian et al., 2025) and GUI-R1 (Luo et al., 2025),  
127  
refine action selection through long-horizon rewards or policy optimization, sometimes achieving  
128  
superior performance with less training data. To address data scarcity, OS-Genesis () and UI-  
129  
Genie (Sun et al., 2024) generate high-quality synthetic trajectories, while multi-agent systems such  
130  
as GUI-OWL and Mobile-Agent-v3 (Wanyan et al., 2025) decompose perception, reasoning, and  
131  
planning across modules to improve robustness in long-horizon tasks.

132

133  
Despite these advances, most approaches primarily optimize execution strategies—whether through  
134  
imitation of expert trajectories, reward shaping, or modular design—without fundamentally enrich-  
135  
ing the model’s internal GUI knowledge. The trained models still fall short in interacting with  
136  
unfamiliar applications or understanding complex system states. To address this gap, our work sys-  
137  
tematically evaluates these foundational knowledge deficiencies and introduces a benchmark that  
138  
identifies missing GUI knowledge in VLMs prior to downstream training, providing insights into  
139  
how future approaches may extend beyond standard fine-tuning paradigms.

140

141

## 2.2 GUI BENCHMARK

142

143  
Evaluating GUI agents is essential for advancing their capabilities, and existing benchmarks gen-  
144  
erally fall into three categories. Action-level benchmarks focus on the precision of low-level op-  
145  
erations such as mouse and keyboard inputs and accurate element grounding. Examples include  
146  
ScreenSpot-Pro (Li et al., 2025) highlights grounding challenges in professional high-resolution in-  
147  
terfaces, SeeClick (Cheng et al., 2024) and ScreenSpot (Jurm et al., 2008) for cross-environment  
148  
grounding. In contrast, we intentionally decouple grounding from the evaluation (by providing  
149  
visual marks on screenshots), so that we can isolate and measure the knowledge deficits of cur-  
150  
rent VLMs in GUI interactions, which other grounding-based benchmarks cannot reveal. Plan-  
151  
level evaluations extend beyond single actions to hierarchical execution. VideoGUI (Lin et al.,  
152  
2024), for instance, evaluates GUI agents with high-level and mid-level planning. Task-level bench-  
153  
marks emphasize end-to-end task success in simulated environments, such as OSWorld (Xie et al.,  
154  
2025c), OSWorld-Verified (Xie et al., 2025b), MacOSworld (Yang et al., 2025), and AndroidWorld  
155  
(Rawles et al., 2024). Beyond execution, a few recent efforts assess GUI knowledge, such as  
156  
MMBench-GUI (Wang et al., 2025), which tests content understanding and widget semantics, and  
157  
Web-CogBench (Guo et al., 2025), which probes cognitive reasoning in web navigation. However,  
158  
these benchmarks remain narrow in application scopes and domain knowledge coverage.

159

160

161

Our benchmark carefully categorizes the GUI knowledge into three complementary aspects derived  
from common agent failure modes, interface perception, interaction prediction and goal interpreta-  
tion. Our benchmark offers a systematic and comprehensive evaluation of GUI knowledge, spanning  
multiple platforms and applications, thereby providing a more complete evaluation of base model’s  
GUI knowledge.

162 

### 3 GUI KNOWLEDGE BENCH

164 

#### 3.1 BENCHMARK OVERVIEW

166 We introduce GUI Knowledge Bench, a benchmark for systematically evaluating the knowledge  
 167 VLMs need to complete GUI tasks. Based on common failure patterns in GUI task execution,  
 168 we identify three complementary dimensions: interface perception, which covers recognizing GUI  
 169 elements, their states, and layout semantics; interaction prediction, which tests whether models  
 170 understand the effect and conventions of common GUI interactions; and instruction understanding,  
 171 which examines whether models can interpret task goals and know the procedural knowledge of  
 172 completing a GUI task. Together, these dimensions capture the core knowledge required for reliable  
 173 GUI task completion and form the foundation of our benchmark.

174 

#### 3.2 DATA SOURCES AND COLLECTION PIPELINE

176 To build GUI Knowledge Bench, we aggregate data from multiple sources to ensure both trajectory-  
 177 level interaction coverage and diverse standalone screenshots.

178 We leverage existing benchmarks such as GUI-Odyssey (Lu et al., 2024) and VideoGUI (Lin et al.,  
 179 2024), which provide screenshots paired with tasks and action annotations. In addition, we collect  
 180 new trajectories by running UI-Tars-7B agents in environments including OSWorld and MacOS-  
 181 World, capturing realistic interaction sequences across both mobile and desktop platforms.

182 To further increase visual diversity and cover a wider range of application interfaces and operating  
 183 systems, we further gather standalone GUI screenshots. Specifically, we sample from ScreenSpot v2  
 184 and extract representative key frames from YouTube tutorials, ensuring coverage of real-world ap-  
 185 plications, operating systems, and interface layouts. For less common actions, we manually perform  
 186 operations on MacOS, Linux, and Windows, recording screenshots and corresponding actions.

187 Together, these sources yield a heterogeneous pool of GUI images and trajectories. From this pool,  
 188 we construct task-specific question-answer pairs for each evaluation dimension, ensuring sufficient  
 189 diversity and coverage while minimizing redundancy. Please refer the appendix for detailed statistics  
 190 of our benchmark.

192 

#### 3.3 INTERFACE PERCEPTION

194 A fundamental requirement for completing GUI tasks is the ability to accurately perceive and inter-  
 195 pret interactive elements in GUI. We aim to evaluate whether VLMs possess sufficient knowledge  
 196 about graphical interfaces.

197 Specifically, this dimension encloses three aspects: (i) widget function understanding, i.e., recog-  
 198 nizing the roles of common interface elements (e.g., three vertical dots for settings, speech bubbles  
 199 for messaging apps); (ii) state information understanding, such as detecting whether a button is en-  
 200 abled/disabled, selected/focused, or toggled on/off; and (iii) layout semantics understanding, where  
 201 spatial arrangement encodes critical information (e.g., distinguishing departure and arrival cities by  
 202 their relative positions, identifying senders and receivers in an email, or inferring file hierarchy from  
 203 indentation). Correctly perceiving these cues is essential for grounding subsequent reasoning and  
 204 action.

205 **Task Definition.** We formalize the evaluation as a unified multiple-choice question-answering task.  
 206 Given a question  $q$ , a set of candidate options  $O$ , and a screenshot  $S$ , the model is required to select  
 207 the correct answer  $o^*$  and provide its reasoning in thought  $t$ :  $VLM : (S, q, O) \mapsto (t, o^*)$ .

208 Our questions include two types: (1) multiple-choice with four candidates, and (2) judgment with  
 209 Yes/No/Unknown. To reduce the burden of visual grounding, the relevant regions in the screenshot  
 210  $S$  are highlighted using red dots or bounding boxes. This design ensures the evaluation focuses on  
 211 whether the model possesses the required GUI knowledge rather than its grounding ability.

213 **Task Collection and Curation.** To construct the evaluation set, we first have human annotators de-  
 214 sign an initial set of seed questions based on the collected GUI screenshots. We then leverage GPT-5  
 215 to expand this pool with additional candidate questions, increasing diversity while maintaining relevance.  
 Questions that can be answered based solely on the text, without viewing the screenshot, are

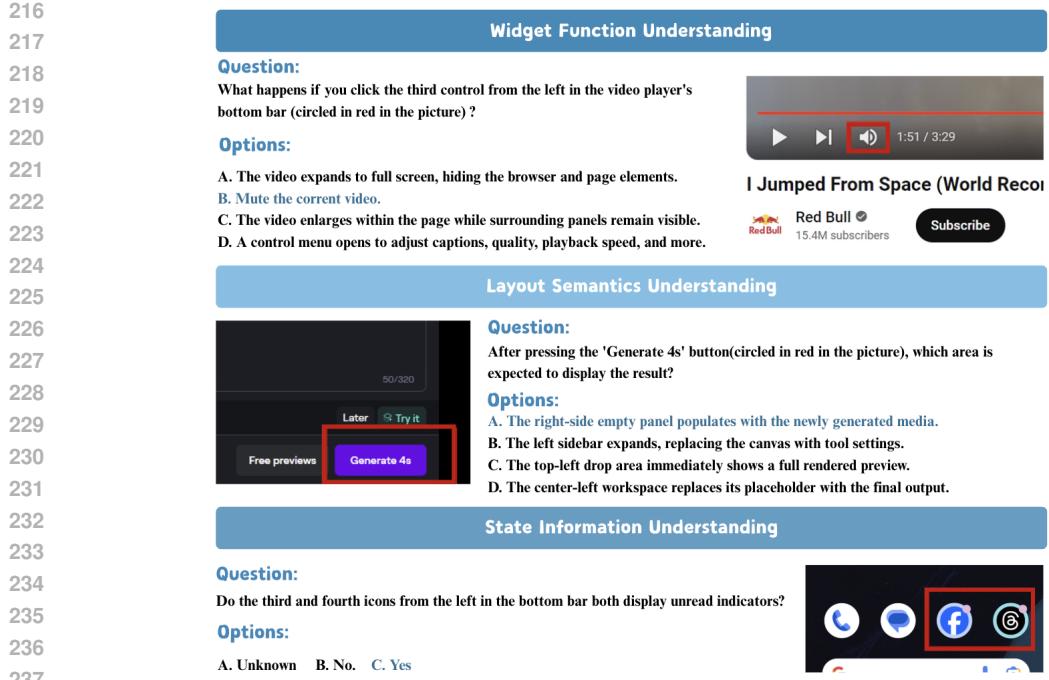


Figure 2: Example questions for Interface Perception. red bounding box

241 removed using Qwen-2.5-VL-7B to ensure visual understanding is necessary. Finally, the remaining  
242 questions are manually verified for correctness, and relevant regions in the screenshots are annotated  
243 to support precise visual grounding. This pipeline ensures that the evaluation focuses on interface  
244 perception knowledge rather than being confounded by grounding or annotation errors.

### 246 3.4 INTERACTION PREDICTION

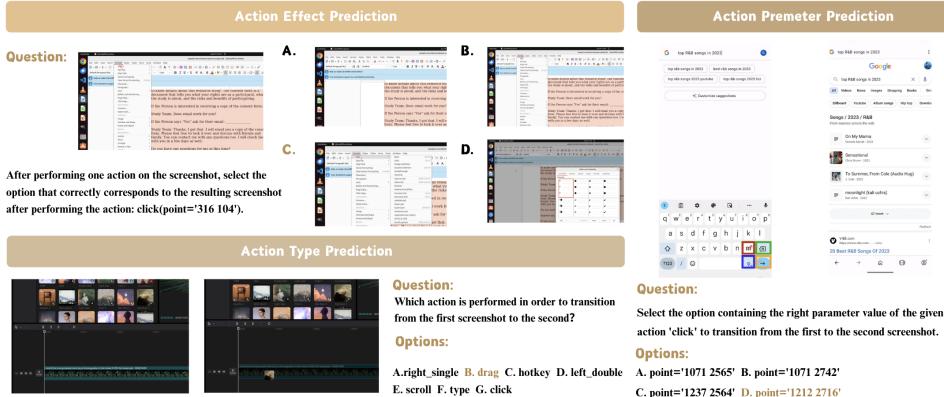


Figure 3: Example questions for Interaction Prediction.

265 A core requirement for solving GUI tasks is to know the interaction conventions and the effect of  
266 GUI operations. Unlike physical environments, GUI interactions follow symbolic and platform-  
267 specific rules (e.g., toggling a switch, typing text, dragging windows), which are often subtle and  
268 context-dependent. Without a proper understanding of these interaction conventions, models cannot  
269 reliably predict the consequences of actions or predict right action types/parameters to complete a  
GUI task. This motivates our evaluation of whether VLMs know GUI interaction conventions.

Interaction prediction is evaluated through two complementary tasks: (i) Action effect prediction, where the model is provided with a current screenshot  $S$  and an action  $a$ , and must select the resulting screenshot  $S'$  from a set of candidate options; (ii) Action prediction, where the model is given two consecutive screenshots  $(s, s')$  and must infer the action  $a$  that caused the transition. Action prediction is further divided into action type prediction, which identifies the correct action category, and action parameter prediction, which selects the appropriate arguments such as click coordinates, typed content, or drag vectors. We do not require the model to generate precise parameters of these coordinates. Exact coordinates in options are annotated in the image as shown in Figure 3. Therefore, the grounding demand is minimal.

**Task Definition.** We formalize GUI interaction dynamics as a state–action transition  $S + a \rightarrow S'$ , where  $S$  represents the current screenshot,  $S'$  the consequent screenshots and  $a$  the action  $a = (a_{\text{type}}, a_{\text{param}})$ . (i) Action Effect Prediction. The model is given  $S$  and  $a$ , and is required to select the resulting screenshots from a set of candidate screenshot options  $O$ :  $\text{VLM} : (S, a, O) \mapsto S'$ . (ii) Action Prediction. The model is given two consecutive screenshots  $(S, S')$  and a set of candidate action types  $O_{\text{type}}$ , and must select the correct action type  $a_{\text{type}}$ :  $\text{VLM} : (S, S', O_{\text{type}}) \mapsto a_{\text{type}}$ . Given the correct action type  $a_{\text{type}}$  and the same state pair  $(S, S')$ , the model selects the correct action parameters from a candidate set  $O_{\text{param}}$ :  $\text{VLM} : (S, S', a_{\text{type}}, O_{\text{param}}) \mapsto a_{\text{param}}$ .

**Task Collection and Curation.** To construct challenging distractor options, we design task-specific strategies for different prediction settings. For action effect prediction, candidate screenshots include the preceding screenshots, the true next screenshot, and visually similar but different screenshots sampled from current trajectories. For action type prediction, the model must choose from the full set of possible actions defined for the platform, with a unified action space across different platforms. (seven actions for desktop platforms and four actions for mobile platforms) For action parameter prediction, For clicks, bounding boxes of candidate elements are identified using OmniParser, and nearby but incorrect coordinates are sampled; for drags, distractors include reversed directions, shortened distances, or swapped start and end points; for scrolls, distractors vary in direction (up, down, left, right); for typing, inputs are perturbed with case changes, partial deletions, or common typos; and for hotkeys, distractors are drawn from a predefined set of common shortcuts. This design ensures that solving the tasks requires precise reasoning about GUI action dynamics rather than relying on superficial visual or layout cues.

### 3.5 INSTRUCTION UNDERSTANDING

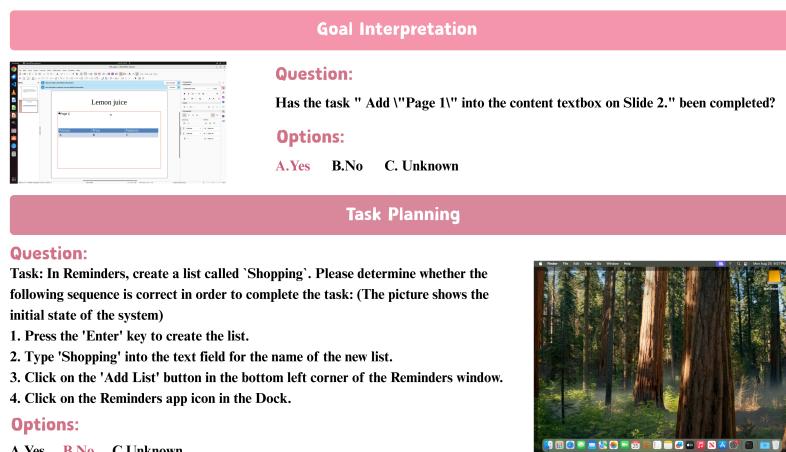


Figure 4: Example questions for Instruction Understanding.

Instruction understanding evaluates whether a model can interpret natural-language tasks and map them to a sequence of GUI operations. This ability is critical because many failures in GUI automation stem from misunderstanding task goals or misinterpreting user intent. Accurately understanding instructions and ordering a feasible sequence of steps is essential for performing high-level tasks in GUI environments. The questions do not ask the model to construct a plan or generate sub-steps.

324 Instead, the complete operation sequence is already provided in these questions to minimize the  
 325 reasoning load.

326 We assess two complementary abilities: (i) goal interpretation, which evaluates whether a model  
 327 can determine if a task has been successfully completed based on history screenshots; and (ii) task  
 328 planning, which evaluates whether a model can reorder a set of candidate option steps into the  
 329 correct sequence required to achieve the task goal. Together, these tasks test the model’s ability to  
 330 both verify and choose high-level plans. Once the underlying procedural knowledge is known, the  
 331 answer becomes immediately obvious and requires minimal reasoning.

332 **Task Definition.** For goal interpretation, the model receives a natural-language task description  $t$   
 333 and history screenshots, and must select the correct option  $o^* \in \{\text{Yes, No, Unknown}\}$  indicating  
 334 whether the task is completed:  $\text{VLM} : (S_{1:T}, t, O) \mapsto o^*$ . For task planning, the model is given  
 335 a natural-language task description  $t$ , current screenshot  $S$  and a set of candidate orderings  $O =$   
 336  $\{\pi_1, \pi_2, \dots, \pi_m\}$ , where each  $\pi_i$  is a possible permutation of the operation steps. The model must  
 337 select the correct ordering  $\pi^*$  from  $O$ :  $\text{VLM} : (t, S, O) \mapsto \pi^*$ .

338 **Task Collection and Curation.** For goal interpretation, human annotators label each trajectory as  
 339 successful or unsuccessful based on last one to five screenshots of the trajectory, and some success-  
 340 ful trajectories are augmented by removing the final screenshot to create unsuccessful ones. For task  
 341 planning, operation plans are first generated by Chat-GPT-5 and then verified by annotators. The  
 342 annotated steps are automatically shuffled to form multiple-choice ordering questions, with longer  
 343 sequences retaining initial steps and only permuting later steps. For shorter sequences, additional  
 344 question formats are created by converting the shuffled sequence into Yes/No/Unknown questions,  
 345 or into operation-level fill-in-the-blank questions with distractor steps. Tasks solvable without ob-  
 346 serving screenshots are filtered out using Qwen-VL-2.5-7B ensuring the difficulty of the questions.

## 348 4 BENCHMARKING VLMS

### 351 4.1 SETTINGS

353 We evaluate a diverse set of both open- and closed-source models on the GUI Knowledge Bench.  
 354 The closed-source set includes Claude-Sonnet-4-5 (Anthropic, 2025a), Claude-Sonnet-4 (Anthropic,  
 355 2025b), Doubaov-Pro (Doubaov-1.5-Thinking-Vision-Pro-250428) (Team, 2025b), Gemini-2.5-  
 356 Pro (Gemini Team, 2025), GPT-5-chat (OpenAI, 2025a), O3(OpenAI, 2025b), and GLM-4.5 (Team,  
 357 2025c). The open-source set covers Qwen2.5-72B (Qwen2.5-VL-72B-Instruct), Qwen2.5-7B  
 358 (Qwen2.5-VL-7B-Instruct) (Team, 2025d), Qwen3-vl-8b-thinking Team (2025e). Besides we also  
 359 include GUI finetune models such as UITARS-1.5-7B (Team, 2025a) and GUI-OWL-7b (Ye et al.,  
 360 2025). Apart from necessary model-specific settings, all other parameters (e.g., temperature, top-p)  
 361 were kept consistent across evaluations. Please refer the appendix for the detailed message template  
 362 for each knowledge categories.

### 363 4.2 BENCHMARKING RESULTS

365 Table 2 summarizes the performance of all evaluated models on three knowledge categories. Experi-  
 366 mental results highlight the following key observations.

368 **First**, o3 achieves strong performance across multiple metrics, consistent with its high success rate  
 369 in real GUI tasks; notably, in the OSWorld benchmark under the Agent framework category, four of  
 370 the top five agents leverage o3 (e.g., Agent-S2.5 w/ O3 50-step version and 100-step version, Jedi-  
 371 7B w/ O3 w/ 50-step version and 100-step version). (Agashe et al., 2025; Xie et al., 2025a) This is  
 372 likely because o3 effectively replaces the auxiliary modules that were removed or made optional.

373 **Second**, UITARS-1.5-7B, trained on Qwen2.5VL-7B, shows improvements in instruction under-  
 374 standing and goal reasoning but a decline in interface perception. Upon examining model outputs,  
 375 we note the following recurring error types of UITARS-1.5-7B, thought-action mismatch where the  
 376 internal reasoning conflicts with the executed code or selection; weakened instruction following that  
 377 overlooks visual prompts and answers from prior knowledge; and degraded general perception, with  
 frequent mistakes on basic attributes such as quantity, color, and length.

378 **Third**, smaller models retain only limited knowledge, suggesting that retrieval-augmented generation or knowledge-base integration may be a viable approach to enhance GUI agent performance.  
 379  
 380  
 381

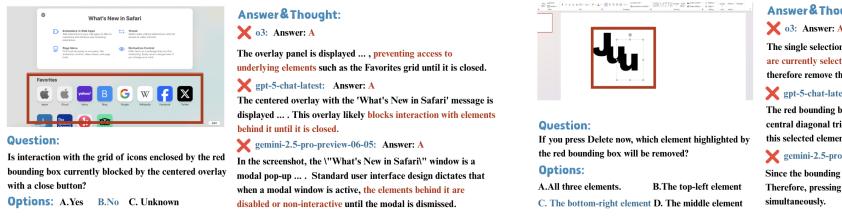
382 Table 2: Performance on GUI Knowledge Bench across three dimensions. Bold numbers indicate  
 383 the best results in each sub-task.

384	385	Model	386 Interface Perception			387 Interaction Prediction			388 Instruction Understanding		389 Overall
			390 state	391 widget	392 layout	393 effect	394 type	395 parameter	396 goal	397 plan	
398	399	O3 (OpenAI, 2025b)	<b>83.03%</b>	84.12%	<b>88.39%</b>	<b>74.83%</b>	75.98%	45.75%	69.45%	<b>95.47%</b>	<b>73.30%</b>
400	401	Gemini-2.5-Pro (Gemini Team, 2025)	81.19%	84.36%	87.10%	71.03%	73.25%	<b>46.97%</b>	67.72%	92.56%	71.69%
402	403	GPT-5-Chat (OpenAI, 2025a)	78.90%	84.12%	<b>88.39%</b>	71.55%	71.55%	43.85%	68.98%	91.26%	70.97%
404	405	Qwen3-vl-8b-thinking (Team, 2025e)	68.81%	76.30%	83.23%	67.07%	70.36%	40.73%	64.09%	91.26%	66.81%
406	407	Claude-Sonnet-4.5 (Anthropic, 2025a)	74.77%	81.52%	82.58%	49.83%	70.19%	43.33%	<b>70.30%</b>	91.56%	66.53%
408	409	Qwen2.5VL-72B (Team, 2025d)	69.27%	77.49%	80.00%	61.72%	64.91%	38.99%	62.20%	85.44%	63.88%
410	411	Doubaao-V-Pro (Team, 2025b)	72.48%	83.65%	81.29%	67.24%	75.64%	41.07%	33.07%	94.17%	63.42%
412	413	Claude-Sonnet-4 (Anthropic, 2025b)	70.18%	78.44%	78.06%	41.90%	62.52%	42.11%	65.20%	94.82%	62.16%
414	415	Qwen2.5VL-7B (Team, 2025d)	53.21%	67.77%	60.00%	51.72%	50.60%	39.34%	16.22%	48.87%	45.16%
416	417	UITARS-1.5-7B (Team, 2025a)	49.54%	59.48%	59.35%	22.24%	59.11%	34.32%	38.74%	55.34%	44.27%
418	419	GUI-OWL-7b (Ye et al., 2025)	60.09%	64.93%	63.23%	21.55%	55.37%	36.05%	21.26%	39.81%	40.74%
420	421	GLM-4.5 (Team, 2025c)	49.54%	48.10%	53.55%	27.07%	17.55%	35.53%	28.98%	91.91%	38.10%

### 394 4.3 ERROR ANALYSIS AND DISCUSSION

#### 395 4.3.1 INTERFACE PERCEPTION.

396 Most models handle widget functions and layout semantics well but struggle with system state perception.  
 397 As shown in Figure 5, in Safari, an update notification is often mistaken for a blocking  
 398 pop-up, leading to incorrect predictions, while in PowerPoint, models can understand the effect of  
 399 delete action but not which element is selected. Our benchmark reveals that current models under-  
 400 perform in system state perception, despite its crucial role in GUI tasks.  
 401



411 Figure 5: Example failure cases of interface perception questions.

#### 412 4.3.2 INTERACTION PREDICTION.

413 On desktop, models often confuse click, double-click, and right-click. This is partly because different  
 414 operating systems and applications treat these interactions differently: in some contexts, single,  
 415 double, or right clicks can substitute for each other, while in others the distinction is strict. Humans  
 416 often try multiple actions to achieve a goal, but models predict a single action based on learned  
 417 patterns. As a result, less frequent actions like double-click or right-click are more prone to mispre-  
 418 diction, especially for smaller models. Please refer to appendix to see the confusion matrix of action  
 419 type prediction.

#### 420 4.3.3 INSTRUCTION UNDERSTANDING.

421 Our benchmark highlights a failure in goal understanding. In Figure 6, three models (o3, gpt-5-  
 422 chat-latest, and gemini-2.5-pro) all respond “Yes,” claiming that the Freeform icon is gone from the  
 423 Dock, even though a different app was removed. This shows that the models cannot reliably judge  
 424 whether the requested task is actually completed.

### 425 4.4 RESULTS ON REAL-WORLD GUI TASKS

426 This section examines the role of three types of knowledge in enabling successful real-world GUI  
 427 task execution.

**Question:**

According to the screenshots above, has the task "Demonstrate to me how to remove Freeform from the Dock" been completed?

**Options:** A.Yes B.No C.Unknown

**Answer & Thought:**

✗ o3: Answer: A

The screenshots show the user right-clicking the Freeform icon in the Dock, ... This clearly demonstrates the procedure for removing Freeform from the Dock, fulfilling the requested task.

✗ gpt-5-chat-latest: Answer: A

The screenshots show the process of right-clicking on the Freeform icon in the Dock... In the final screenshot, Freeform is no longer visible in the Dock, indicating that it has been successfully removed.

✗ gemini-2.5-pro-preview-06-05: Answer: A

... The fourth screenshot confirms that Freeform is no longer present in the Dock. This clearly demonstrates the complete process of removing Freeform from the Dock.

Figure 6: Example failure cases of instruction understanding questions.

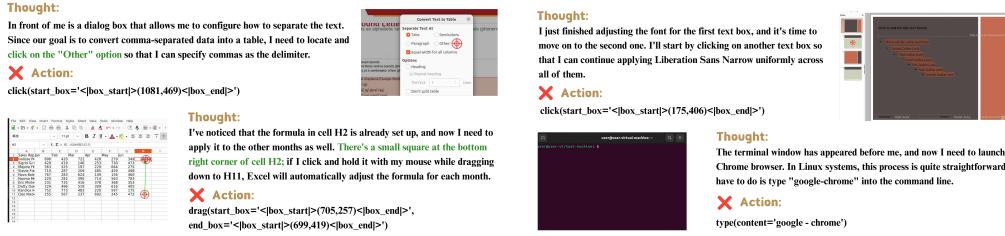


Figure 7: Example failure cases due to lack of interaction prediction knowledge.

#### 4.4.1 QUALITATIVE ANALYSIS: INTERFACE PERCEPTION AND INTERACTION PREDICTION

**Interface Perception Knowledge.** In our evaluation, we identified several tasks that the model consistently failed to solve even under the pass@32 setting. We attribute some of them to lack of knowledge of the interface. Two example failure cases are shown in Figure 8. When asked to add a note, the model repeatedly attempted to insert comments or text boxes, incorrectly treating these actions as equivalent to adding a note. In reality, adding a note requires first enabling the Notes pane through the View menu and then placing the note at the bottom of the slide. Similarly, in converting comma-separated text into a table, the model repeatedly failed because it did not specify the delimiter, a necessary step for correct execution. These cases suggest that the failures stem from missing application-specific knowledge rather than inherent reasoning limitations. Importantly, once the required knowledge was provided, the model was able to complete these tasks successfully.

Table 3: Effect of appending operation plans on UITARS-1.5-7B.

Metric	UITARS-1.5-7B (base)	+ GPT-4o plan	+ OSWorld-human plan	+ o3 plan
Pass@1 (%)	24.81	27.59	28.20	<b>30.79</b>

**Interaction Prediction Knowledge.** As shown in Figure 7, many errors of models occur because it lacks knowledge for localizing interface elements correctly. Prior work has improved localization using masks, accessibility trees, or APIs. Another promising approach is to leverage actions themselves for self-verification, using visual prompts to check if the executed action was correct.

#### 4.4.2 QUANTITATIVE ANALYSIS: THE IMPACT OF PLAN INJECTION

**Instruction Understanding Knowledge.** We generated knowledge about operation plans from GPT-4o and o3 conditioned on task instructions, and used human-authored operation plans from OSWorld-human. Each plan was appended to the original task description as an additional knowledge for UITARS-1.5-7B. Results are summarized in Table 3. These results show that providing knowledge about operation plans improves task performance, highlighting the importance of instruction understanding for task completion. Notably, o3-generated plans achieve the largest gain, surpassing human-authored plans and aligning with o3's top performance across our benchmark evaluations.

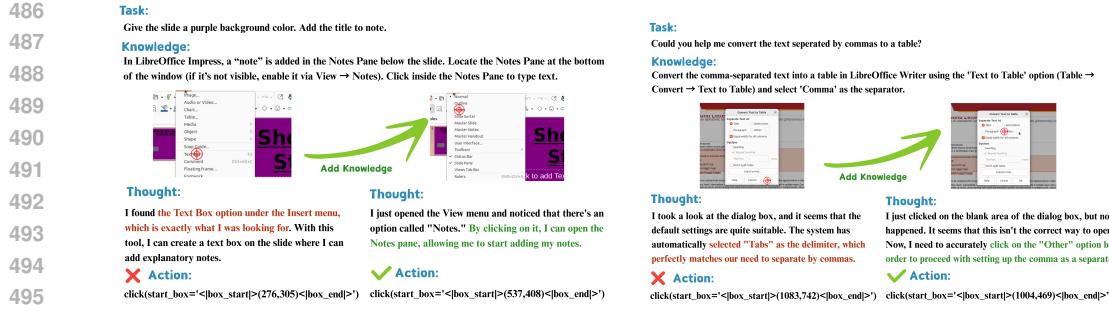


Figure 8: Example failure cases due to lack of interface perception knowledge.

#### 4.4.3 VALIDATION STUDY: KNOWLEDGE AS A NECESSARY CONDITION

While the previous experiment demonstrates the benefit of injecting Instruction knowledge, Interface and Interaction knowledge are intrinsic and difficult to inject externally. To address this, we conduct a validation study that mirrors the the 'Instruction Understanding' analysis for 'Interface Perception' and 'Interaction Prediction'. Concretely, we transform 39 questions in our benchmark into practical GUI tasks, where the knowledge tested in questions is key to completing the GUI tasks. Two examples are as follows. From **Knowledge Question A**: "Are the search results limited to a specific region? (Yes/No/Unknown)" to **Transformed Task A**: "Set the search results restricted to a specific region: Japan." (Ground Truth: Click the toggle switch). From **Knowledge Question B**: "Is the list currently arranged from older to newer items?" to **Transformed Task B**: "Sort the items from oldest to newest according to the time added." (Ground Truth: Click the column header).

We define  $S_1$  as answering the knowledge question correctly, and  $S_2$  as successfully completing the corresponding GUI tasks. Results are shown in Table 4. Prompts for completing GUI tasks are provided in the Appendix.

Table 4: Correlation between GUI Knowledge ( $S_1$ ) and Task Completion ( $S_2$ ).

Model	$P(S_2 \checkmark   S_1 \checkmark)$	$P(S_2 \times   S_1 \checkmark)$	$P(S_2 \times   S_1 \times)$	$P(S_2 \checkmark   S_1 \times)$
claude-sonnet-4 (Anthropic, 2025b)	20.00%	80.00%	100.00%	0.00%
claude-sonnet-4-5 (Anthropic, 2025a)	8.33%	91.67%	100.00%	0.00%
Doubao-V-Pro (Team, 2025b)	0.00%	100.00%	100.00%	0.00%
Gemini-2.5-Pro (Gemini Team, 2025)	0.00%	100.00%	100.00%	0.00%
GPT-5-Chat (OpenAI, 2025a)	5.56%	94.44%	100.00%	0.00%

The results establish GUI knowledge as a necessary but not sufficient condition for task completion: Lacking knowledge guarantees execution failure (e.g., 100% for Gemini-1.5), validating that knowledge is the strict lower bound for control. Even with correct understanding, execution often fails due to grounding precision. This confirms that our benchmark measures a foundational capability. While having knowledge doesn't guarantee success (due to downstream grounding issues), lacking knowledge guarantees failure.

## 5 CONCLUSION

We introduce GUI Knowledge Bench, a novel benchmark designed to evaluate the GUI knowledge encoded in vision-language models (VLMs) before downstream training. By analyzing common failure patterns in GUI task execution, the benchmark categorizes GUI knowledge into three dimensions: interface perception, interaction prediction, and instruction understanding. The evaluation reveals significant gaps in current VLMs' understanding of system states, action outcomes, and task completion verification. These findings highlight the necessity of enriching VLMs with domain-specific GUI knowledge to enhance their performance in real-world GUI tasks and provide insights to guide the development of more capable GUI agents.

540 REFERENCES  
541

542 Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent  
543 s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint*  
544 *arXiv:2504.00906*, 2025.

545 Anthropic. Claude sonnet 4.5. <https://docs.claude.com/docs/about-claude/models/whats-new-claude-4-5>, 2025a. Accessed: 2025-11-22.

546 Anthropic. System card: Claude opus 4 & claude sonnet 4. Technical report, Anthropic PBC, May  
547 2025b. URL <https://www.anthropic.com/claude-4-system-card>. Accessed:  
548 2025-09-25.

549 Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong  
550 Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the*  
551 *62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,  
552 pp. 9313–9332, 2024.

553 Google Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality,  
554 long context, and next generation agentic capabilities, 2025. URL <https://arxiv.org/abs/2507.06261>.

555 Yuhang Guo, Cong Guo, Aiwen Sun, Hongliang He, Xinyu Yang, Yue Lu, Yingji Zhang, Xuntao Guo,  
556 Dong Zhang, Jianzhuang Liu, et al. Web-cogreasioner: Towards knowledge-induced cognitive  
557 reasoning for web agents. *arXiv preprint arXiv:2508.01858*, 2025.

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

561 Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan  
562 Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents.  
563 In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp.  
564 14281–14290, 2024.

565 Marko Jurmu, Sebastian Boring, and Jukka Riekki. Screenspot: Multidimensional resource discov-  
566 ery for distributed applications in smart spaces. In *Proceedings of the 5th Annual International*  
567 *Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 1–9,  
568 2008.

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

572 Shuquan Lian, Yuhang Wu, Jia Ma, Zihan Song, Bingqi Chen, Xiawu Zheng, and Hui Li. Ui-agile:  
573 Advancing gui agents with effective reinforcement learning and precise inference-time grounding.  
574 *arXiv preprint arXiv:2507.22025*, 2025.

575 Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Wu, Mingyi Yan, Zhengyuan Yang, Lijuan  
576 Wang, and Mike Zheng Shou. Videogui: A benchmark for gui automation from instructional  
577 videos. *arXiv preprint arXiv:2406.10227*, 4, 2024.

578 Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian  
579 Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui  
580 visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp.  
581 19498–19508, 2025.

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

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

594 Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language  
 595 action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

596

597 OpenAI. Gpt-5 system card, Aug 2025a. URL <https://cdn.openai.com/gpt-5-system-card.pdf>. Accessed: 2025-09-25.

598

599 OpenAI. Openai o3 and o4-mini system card. Technical report, OpenAI, April 2025b. URL  
 600 <https://openai.com/index/o3-o4-mini-system-card/>. Accessed: 2025-09-25.

601

602 Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. Fine-tuning or retrieval? com-  
 603 paring knowledge injection in llms, 2024. URL <https://arxiv.org/abs/2312.05934>.

604

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

608

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

612

613 ByteDance Team. Ui-tars: Pioneering automated gui interaction with native agents, 2025a. URL  
<https://arxiv.org/abs/2501.12326>.

614

615 ByteDance Seed Team. Seed1.5-vl technical report, 2025b. URL <https://arxiv.org/abs/2505.07062>.

616

617 GLM Team. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025c. URL  
 618 <https://arxiv.org/abs/2508.06471>.

619

620 Qwen Team. Qwen2.5-vl technical report, 2025d. URL <https://arxiv.org/abs/2502.13923>.

621

622 Qwen Team. Qwen3 technical report, 2025e. URL <https://arxiv.org/abs/2505.09388>.

623

624 Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu,  
 625 Qingyun Li, Xuan Dong, Zhe Chen, et al. Mmbench-gui: Hierarchical multi-platform evaluation  
 626 framework for gui agents. *arXiv preprint arXiv:2507.19478*, 2025.

627

628 Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming  
 629 Yan, Fei Huang, Xiaoshan Yang, et al. Look before you leap: A gui-critic-r1 model for pre-  
 630 operative error diagnosis in gui automation. *arXiv preprint arXiv:2506.04614*, 2025.

631

632 Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao  
 633 Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement,  
 2024a. URL <https://arxiv.org/abs/2402.07456>.

634

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

638

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

642

643 Tianbao Xie, Mengqi Yuan, Danyang Zhang, Xinzhuang Xiong, Zhennan Shen, Zilong Zhou,  
 644 Xinyuan Wang, Yanxu Chen, Jiaqi Deng, Junda Chen, Bowen Wang, Haoyuan Wu, Jixuan Chen,  
 645 Junli Wang, Dunjie Lu, Hao Hu, and Tao Yu. Introducing osworld-verified. *xlang.ai*, July 2025b.  
 646 URL <https://xlang.ai/blog/osworld-verified>.

647

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua  
 Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal  
 648 agents for open-ended tasks in real computer environments. *Advances in Neural Information  
 649 Processing Systems*, 37:52040–52094, 2025c.

648 Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang,  
 649 Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android au-  
 650 tonomous agents, 2024a. URL <https://arxiv.org/abs/2410.24024>.

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

654 Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for  
 655 gui agents. *arXiv preprint arXiv:2506.04135*, 2025.

656 Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu  
 657 Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan.  
 658 Mobile-agent-v3: Fundamental agents for gui automation, 2025. URL <https://arxiv.org/abs/2508.15144>.

## 661 662 A APPENDIX

### 663 A.1 QUESTION GENERATION PROMPT TEMPLATE FOR INTERFACE PERCEPTION

#### 664 665 Prompt for widget function understanding.

##### 666 667 Widget Function Prompt

##### 668 669 System Prompt:

##### 670 671 [Role]

672 You will be provided with a single screenshot of a system interface (desktop app, web UI, or  
 673 mobile app). Generate exactly one challenging GUI reasoning question about that screenshot that  
 674 requires inspecting the image to answer.

##### 675 676 [Knowledge Scope of the question]

677 Ask about the intended function of a specific UI widget (button, toggle, slider, icon, etc.) inferred  
 678 from the widget’s iconography and surrounding context. Avoid universally trivial icons unless  
 679 combined with contextual clues.

##### 680 681 [Generation Guidelines]

1. Question length: one concise sentence only. No hints, no steps, no extra context.
2. Position-only references: Do NOT use any visible text, icon names, or labels from the  
 682 screenshot. Refer ONLY by position or coordinates (examples: “top-right corner”, “third  
 683 from left in the top toolbar”, “second row, third column”, “left sidebar, bottom icon”, or  
 684 “ $<x, y>$  with origin top-left). The question must be unsolvable without the screenshot.
3. Question types and options:
  - If `multiple_choice`: produce exactly 4 options. The first option MUST be the  
 685 correct answer.
  - If `yes_or_no`: produce exactly 3 options: {“yes”, “no”, “unknown”} and the  
 686 correct one must be first.
  - If the correct answer is genuinely not deducible from the screenshot or you cannot  
 687 answer the correct answer, then use:
    - `multiple_choice`: first option = “none of the other options are correct.”
    - `yes_or_no`: first option = “unknown”
4. Option style: Options must describe actions or effects (not icon shapes). Keep options  
 688 parallel in length and style ( $\approx$  6–16 words).
5. Distractors: The 3 incorrect options must be plausible and similar to the correct one.
6. Contextual reasoning: Prefer questions requiring reasoning across UI elements (e.g.,  
 689 highlighted rows, active tab, enabled/disabled states, adjacent panels).
7. Based on the provided screenshot, identify which application is currently being used and  
 690 include this information in your output JSON under the field `app_type`.

```

702
703 [Output JSON schema — return exactly this JSON object (no extra text)]
704 {
705   "question_type": "multiple_choice" or "yes_or_no",
706   "question_text": "<one concise sentence using only positions>",
707   "option_text": ["<first option correct>", "<distractor 1>",
708   ↳ "<distractor 2>", "<distractor 3>"],
709   "app_type": "<application type of the current screenshot>",
710   "os_type": "Linux" | "Windows" | "Android" | "MacOS" | "IOS" | "Web"
711 }
712 [Example Output]
713 {
714   "question_type": "yes_or_no",
715   "question_text": "While cell B5 in the 'First Name' column shows
716   ↳ 'Walter' in the formula bar and the checkmark and 'X' icons are
717   ↳ visible beside it, will clicking the 'X' icon clear formatting in
718   ↳ the selected cell",
719   "option_text": ["yes", "no", "unknown"],
720   "app_type": "Excel",
721   "os_type": "Linux"
722 }
723 {
724   "question_type": "multiple_choice",
725   "question_text": "Which of the following statement is correct
726   ↳ according to the screenshots?",
727   "option_text": [
728     "The camera is not currently connected to WiFi",
729     "The camera can not be controlled remotely from the phone",
730     "Pressing the 'phone' mode icon in the top bar can lead to turning
731     ↳ on the phone's airplane mode",
732     "Pressing the 'clone' mode icon in the top bar can lead to signing
733     ↳ out of the cloud gallery"
734   ],
735   "app_type": "Excel",
736   "os_type": "Linux"
737 }

```

#### Prompt for layout semantics understanding.

##### Layout Semantics Prompt

###### System Prompt:

###### [Role]

You will be provided with a single screenshot of a system interface (desktop app, web UI, or mobile app). Generate exactly one challenging GUI reasoning question about that screenshot that requires inspecting the image to answer.

###### [Knowledge Scope of the question]

The questions should assess whether the model understands positional and grouping relationships between UI elements, inferring their roles from placement and hierarchy.

###### [Generation Guidelines]

1. Question length: one concise sentence only. No hints, no steps, no extra context.
2. Position-only references: Do NOT use any visible text, icon names, or labels from the screenshot. Refer ONLY by position or coordinates (examples: "top-right corner", "third from left in the top toolbar", "second row, third column", "left sidebar, bottom icon", or "<x, y>" with origin top-left). The question must be unsolvable without the screenshot.
3. Question types and options:
  - If `multiple_choice`: produce exactly 4 options. The first option MUST be the correct answer.

756

- 757 • If `yes_or_no`: produce exactly 3 options: {"yes", "no", "unknown"} and the correct one must be first.

758

- 759 • If the correct answer is genuinely not deducible from the screenshot or you cannot answer the correct answer, then use:
  - 760 – `multiple_choice`: first option = "none of the other options are correct."
  - 761 – `yes_or_no`: first option = "unknown"

762

- 763 4. Option style: Options must describe actions or effects (not icon shapes). Keep options parallel in length and style ( $\approx$  6–16 words).

764

- 765 5. Distractors: The 3 incorrect options must be plausible and similar to the correct one.

766

- 767 6. Contextual reasoning: Prefer questions requiring reasoning across UI elements (e.g., highlighted rows, active tab, enabled/disabled states, adjacent panels).

768

- 769 7. Based on the provided screenshot, identify which application is currently being used and include this information in your output JSON under the field `app_type`.

770

771 **[Output JSON schema — return exactly this JSON object (no extra text)]**

```
772 {
  773   "question_type": "multiple_choice" or "yes_or_no",
  774   "question_text": "<one concise sentence using only positions>",
  775   "option_text": ["<first option correct>", "<distractor 1>",
  776   ↳ "<distractor 2>", "<distractor 3>"],
  777   "app_type": "<application type of the current screenshot>",
  778   "os_type": "Linux" | "Windows" | "Android" | "MacOS" | "IOS" | "Web"
  779 }
```

780 **[Example Output]**

```
781 {
  782   "question_type": "multiple_choice",
  783   "question_text": "What is likely to be the departure city?",
  784   "option_text": ["Beijing", "Shanghai", "Guangzhou", "None of the
  785   ↳ other options."],
  786   "app_type": "website",
  787   "os_type": "Windows"
  788 }
  789 {
  790   "question_type": "yes_or_no",
  791   "question_text": "Is the folder in the second row under the
  792   ↳ 'Documents' folder?",
  793   "option_text": ["yes", "no", "unknown"],
  794   "app_type": "Thunderbird",
  795   "os_type": "Windows"
  796 }
  797 {
  798   "question_type": "multiple_choice",
  799   "question_text": "Who sends this email. Please answer the email
  800   ↳ address.",
  801   "option_text": ["li@gmail.com", "zhang@gmail.com", "wang@gmail.com",
  802   ↳ "None of the other options."],
  803   "app_type": "Email",
  804   "os_type": "Windows"
  805 }
```

806 **Prompt for state information understanding.**

807

808 State Information Prompt

809

System Prompt:

810

**[Role]**

811 You will be provided with a single screenshot of a system interface (desktop app, web UI, or  
 812 mobile app). Generate exactly one challenging GUI reasoning question about that screenshot that  
 813 requires inspecting the image to answer.

**[Knowledge Scope of the question]**

814 Ask about the current state information of the system, such as whether a control is enabled/disabled,  
 815 a process is in-progress/completed, a request is pending, or the system is online/offline.  
 816 Prefer reasoning that requires subtle visual cues or multi-element context.

**[Generation Guidelines]**

817

1. Question length: one concise sentence only. No hints, no steps, no extra context.
2. Position-only references: Do NOT use any visible text, icon names, or labels from the screenshot. Refer ONLY by position or coordinates (examples: “top-right corner”, “third from left in the top toolbar”, “second row, third column”, “left sidebar, bottom icon”, or “ $<x, y>$  with origin top-left). The question must be unsolvable without the screenshot.
3. Question types and options:
  - If `multiple_choice`: produce exactly 4 options. The first option MUST be the correct answer.
  - If `yes_or_no`: produce exactly 3 options: {“yes”, “no”, “unknown”} and the correct one must be first.
  - If the correct answer is genuinely not deducible from the screenshot or you cannot answer the correct answer, then use:
    - `multiple_choice`: first option = “none of the other options are correct.”
    - `yes_or_no`: first option = “unknown”
4. Option style: Options must describe actions or effects (not icon shapes). Keep options parallel in length and style ( $\approx$  6–16 words).
5. Distractors: The 3 incorrect options must be plausible and similar to the correct one.
6. Contextual reasoning: Prefer questions requiring reasoning across UI elements (e.g., highlighted rows, active tab, enabled/disabled states, adjacent panels).
7. Based on the provided screenshot, identify which application is currently being used and include this information in your output JSON under the field `app_type`.

**[Output JSON schema — return exactly this JSON object (no extra text)]**

818

```
{
  "question_type": "multiple_choice" or "yes_or_no",
  "question_text": "<one concise sentence using only positions>",
  "option_text": ["<first option correct>", "<distractor 1>",
    "→ <distractor 2>", "<distractor 3>"],
  "app_type": "<application type of the current screenshot>",
  "os_type": "Linux" | "Windows" | "Android" | "MacOS" | "IOS" | "Web"
}
```

**[Example Output]**

819

```
{
  "question_type": "multiple_choice",
  "question_text": "The button in the lower toolbar is active, but the
    → button next to it is greyed out. Which condition is most likely
    → not met yet?",
  "option_text": [
    "All required fields are filled",
    "Network connection is active",
    "File format is supported",
    "None of the other options"
  ],
  "app_type": "Form Editor",
  "os_type": "Web"
}
```

```

864
865  {
866      "question_type": "multiple_choice",
867      "question_text": "How can the user enable more controls over the
868      ↳ alignment of objects?",
869      "option_text": [
870          "Select more than one object",
871          "Double click the alignment button",
872          "None of the other options",
873          "User is logged in"
874      ],
875      "app_type": "Graphics Editor",
876      "os_type": "Windows"
877  }
878
879  {
880      "question_type": "yes_or_no",
881      "question_text": "Will the option in the toolbar become available
882      ↳ immediately after selecting a file?",
883      "option_text": ["yes", "no", "unknown"],
884      "app_type": "Document Editor",
885      "os_type": "MacOS"
886  }
887
888  {
889      "question_type": "yes_or_no",
890      "question_text": "Is the movie export function currently available?",
891      "option_text": ["no", "yes", "unknown"],
892      "app_type": "Video Editor",
893      "os_type": "Linux"
894  }

```

## A.2 PLAN GENERATION PROMPT TEMPLATE FOR OSWORLD TASKS.

### User Instruction Prompt

#### User Prompt:

Analyze the given GUI task and break it down into essential, actionable steps. You will receive:  
 - a task instruction: {task\_instruction} - the app where the task occurs: {app\_name} -  
 the initial screenshot image

Your goal is to output a Python list of clear, concise steps in logical order to complete the task  
 within the app. Each step should represent a key state, action, or milestone. Use simple, direct  
 language. Avoid ambiguity or unnecessary complexity.

#### Output format:

- A valid Python list of strings, e.g.:
 

```
["First step.", "Second step.", "Third step."]
```
- Each string must use double quotes ("), and the output must be directly parsable using eval() or ast.literal\_eval().
- Output only the list. No explanation, no extra text.

#### Constraints:

- Ensure each step is actionable and unambiguous,
- Ensure each step is necessary for task completion,
- Ensure each step is easy to follow by a user.

918 A.3 EVALUATION MESSAGE PROMPT TEMPLATE  
919920 A.3.1 INTERFACE PERCEPTION.  
921922 All evaluation questions in this knowledge category use the same prompt template as shown below.  
923924 GUI Agent Inference Prompt  
925**System**

You are a Graphical User Interface (GUI) agent. You will be given a screenshot, a question, and corresponding options. You need to choose one option as your answer.

**User**

```
{question_images}  
{question_texts}  
{question_options}
```

**Response Rules****If question\_type == 'yes\_or\_no':**

Think step by step. You must respond strictly in JSON format following this schema:

```
{  
  "thought": "<your reasoning>",  
  "answer": "<yes/no/unknown>"  
}
```

**If question\_type == 'multiple\_choice':**

Think step by step. You must respond strictly in JSON format following this schema:

```
{  
  "thought": "<your reasoning>",  
  "answer": "<A/B/C/D>"  
}
```

947

**Interaction Prediction.**

948

949 GUI Agent Task-Solving Prompt

**System**

You are a Graphical User Interface (GUI) agent. You will be given a task instruction, a screenshot, several GUI operations, and four options. Your goal is to select the best option that could solve the task.

```
{question_images}
```

**User**

```
{question_text}
```

Which of the above options are correct according to the screenshots? Think step by step. You must respond strictly in JSON format following this schema.

**Response Schema**

```
{  
  "thought": "<your reasoning>",  
  "answer": "<A/B/C/D>"  
}
```

967

968 A.3.2 INTERACTION PREDICTION  
969970 **ActionEffect**  
971

972 GUI Agent Next-State Selection Prompt  
 973  
 974 **System**  
 975 You are a Graphical User Interface (GUI) agent. You will be given a screenshot, action de-  
 976 scriptions, and multiple options, each containing an image. After performing one action on the  
 977 screenshot, your goal is to select the option that correctly corresponds to the resulting screenshot  
 978 after performing the action. Below is a short description of the action space:  
 979  
 980 if platform == Desktop:  
 981 Action Space  
 982 - click(point='x1 y1'): left click a position on the screen.  
 983 - left\_double(point='x1 y1'): left double click a position on  
 984 ↵ the screen.  
 985 - right\_single(point='x1 y1'): right single click a position on  
 986 ↵ the screen.  
 987 - drag(start\_point='x1 y1', end\_point='x2 y2'): drag the mouse  
 988 ↵ from one position to another.  
 989 - hotkey(key='ctrl c'): keyboard shortcut, split keys with  
 990 ↵ spaces  
 991 - type(content='xxx'): type an answer, use escape characters  
 992 ↵ (', ", \n) when needed. Add \n at the end if it is the  
 993 ↵ final submission.  
 994 - scroll(point='x1 y1', direction='down or up or right or  
 995 ↵ left'): scroll to see more content  
 996  
 997 if platform == Mobile:  
 998 Action Space  
 999 - click(point='x1 y1')  
 1000 - long\_press(point='x1 y1')  
 1001 - type(content='') #If you want to submit your input, use "\\n"  
 1002 ↵ at the end of `content`.  
 1003 - scroll(point='x1 y1', direction='down or up or right or  
 1004 ↵ left'): scroll to see more content  
 1005  
 1006 The size of the image is {w}x{h}. \n  
 1007  
 1008 **User**  
 1009 {question\_image}  
 1010 Above is the current screenshot.  
 1011 After I perform the described action 'action\_type(action\_parameter)' (as drawn  
 1012 in the initial screenshot), which of the following options correctly corresponds to the resulting  
 1013 screenshot?  
 1014 A. {option\_image\_A}  
 1015 B. {option\_image\_B}  
 1016 C. {option\_image\_C}  
 1017 D. {option\_image\_D}  
 1018  
 1019 **Response Schema**  
 1020 Think step by step. You must respond strictly in JSON format following this schema:  
 1021  
 1022 {  
 1023 "thought": "<your reasoning>",<br>  
 1024 "answer": "<A/B/C/D>"  
 1025 }  
 1026  
 1027  
 1028 **ActionPrediction - Parameter**

1029 GUI Agent Action-Parameter Selection Prompt  
 1030  
 1031 **System**  
 1032 You are a Graphical User Interface (GUI) agent. You will be given two consecutive screenshots  
 1033 of the GUI, action descriptions, and multiple options. Your goal is to select which action was  
 1034 performed to transition from the first screenshot to the second. If the description specifies an  
 1035 action type, select the correct parameter value for the given action.

```

1026
1027     if platform == Desktop:
1028         Action Space
1029         - click(point='x1 y1'): left click a position on the screen.
1030         - left_double(point='x1 y1'): left double click a position on
1031             ↵ the screen.
1032         - right_single(point='x1 y1'): right single click a position on
1033             ↵ the screen.
1034         - drag(start_point='x1 y1', end_point='x2 y2'): drag the mouse
1035             ↵ from one position to another.
1036         - hotkey(key='ctrl c'): keyboard shortcut, split keys with
1037             ↵ spaces
1038         - type(content='xxx'): type an answer, use escape characters
1039             ↵ (', ", \n) when needed. Add \n at the end if it is the
1040             ↵ final submission.
1041         - scroll(point='x1 y1', direction='down or up or right or
1042             ↵ left'): scroll to see more content
1043
1044     if platform == Mobile:
1045         Action Space
1046         - click(point='x1 y1')
1047         - long_press(point='x1 y1')
1048         - type(content=''): #If you want to submit your input, use "\\n"
1049             ↵ at the end of `content`.
1050         - scroll(point='x1 y1', direction='down or up or right or
1051             ↵ left'): scroll to see more content
1052
1053 The size of the image is {w}x{h}. \n
1054 {question_images}

```

### User

Above are two consecutive screenshots. Your task is to select the option containing the right parameter value of the given action ' {action\_type} ' to transition from the first to the second screenshot.

As is drawn in the first screenshot. Which of the above options are correct according to the screenshots?

- A. {option\_text}
- B. {option\_text}
- C. {option\_text}
- D. {option\_text}

### Response Schema

Think step by step. You must respond strictly in JSON format following this schema:

```

1062
1063 {
1064     "thought": "<your reasoning>",
1065     "answer": "<A/B/C/D>"
1066
1067
1068
1069 GUI Agent Action Identification Prompt
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

```

### ActionPrediction - Type

#### GUI Agent Action Identification Prompt

### System

You are a Graphical User Interface (GUI) agent. You will be given two consecutive screenshots of the GUI, action descriptions, and multiple options. Your goal is to select which action was performed to transition from the first screenshot to the second. If the description specifies an action type, select the correct parameter value for the given action.

```

1076
1077     if platform == Desktop:
1078         Action Space
1079         - click(point='x1 y1'): left click a position on the screen.
1080         - left_double(point='x1 y1'): left double click a position on
1081             ↵ the screen.

```

```

1080
1081     - right_single(point='x1 y1'): right single click a position on
1082     ↪ the screen.
1083     - drag(start_point='x1 y1', end_point='x2 y2'): drag the mouse
1084     ↪ from one position to another.
1085     - hotkey(key='ctrl c'): keyboard shortcut, split keys with
1086     ↪ spaces
1087     - type(content='xxx'): type an answer, use escape characters
1088     ↪ (' ', "\n) when needed. Add \n at the end if it is the
1089     ↪ final submission.
1090     - scroll(point='x1 y1', direction='down or up or right or
1091     ↪ left'): scroll to see more content
1092
1093     if platform == Mobile:
1094         Action Space
1095         - click(point='x1 y1')
1096         - long_press(point='x1 y1')
1097         - type(content='') #If you want to submit your input, use "\n"
1098         ↪ at the end of `content`.
1099         - scroll(point='x1 y1', direction='down or up or right or
1100         ↪ left'): scroll to see more content
1101
1102     The size of the image is {w}x{h}. \n
1103     {question_images}

1104 User
1105 Above are two consecutive screenshots. Your task is to select which action is performed in order
1106 to transition from the first screenshot to the second.
1107
1108     if platform == Desktop:
1109         {seven action types}
1110         Which of the above options are correct according to the
1111         ↪ screenshots?
1112         Think step by step. You must respond strictly in JSON format
1113         ↪ following this schema:
1114         {"thought": "<your reasoning>", "answer": "<A/B/C/D/E/F/G>" }

1115     if platform == Mobile:
1116         {four action types}
1117         Which of the above options are correct according to the
1118         ↪ screenshots?
1119         Think step by step. You must respond strictly in JSON format
1120         ↪ following this schema:
1121         {"thought": "<your reasoning>", "answer": "<A/B/C/D>" }

1122 Response Schema (Desktop)
1123
1124     {
1125         "thought": "<your reasoning>",
1126         "answer": "<A/B/C/D/E/F/G>"
1127     }
1128
1129 Response Schema (Mobile)
1130
1131     {
1132         "thought": "<your reasoning>",
1133         "answer": "<A/B/C/D>"
1134     }

```

### A.3.3 INSTRUCTION UNDERSTANDING

#### Goal Interpretation

1134 Task Completion Verification Prompt  
 1135  
**1136 System**  
 1137 You are a Graphical User Interface (GUI) agent. You will be given a sequence of screenshots, a  
 1138 task instruction, and three possible answer options: yes, no, unknown. Your goal is to select  
 1139 the best option that indicates whether the task is completed.  
 1140 • **yes** — The task is clearly completed.  
 1141 • **no** — The task is not completed.  
 1142 • **unknown** — The screenshots do not provide enough evidence to determine completion.  
**1144 User**  
 1145 According to the screenshots below, has the task "{task}" been completed?  
 1146 {question\_images}  
**1147 Response Schema**  
 1148 Think step by step. You must respond strictly in JSON format following this schema:  
 1149 {  
 1150 "thought": "<your reasoning>,"  
 1151 "answer": "<yes/no/unknown>"  
 1152 }  
 1153  
 1154  
 1155 **TaskPlanning**  
 1156  
 1157  
 1158 GUI Agent Conditional QA Prompt  
 1159  
**1160 System**  
 1161  
**1162 If question\_type == 'yes\_or\_no':**  
 1163 You are a Graphical User Interface (GUI) agent. You will be given a screenshot, a question, and  
 1164 corresponding options. You need to choose one option as your answer.  
**1165 If question\_type == 'multiple\_choice':**  
 1166 You are a Graphical User Interface (GUI) agent. You will be given a task instruction, a screenshot,  
 1167 several GUI operations, and four options. Your goal is to select the best option that could solve  
 1168 the task.  
 1169 {question\_images}  
 1170  
**1171 User**  
 1172 {question\_text}  
 1173 {option\_texts}  
 1174 Which of the above options are correct according to the screenshot?  
**1175 Response Rules**  
 1176  
**1177 If question\_type == 'yes\_or\_no':**  
 1178 Think step by step. You must respond strictly in JSON format following this schema:  
 1179 {  
 1180 "thought": "<your reasoning>,"  
 1181 "answer": "<yes/no/unknown>"  
 1182 }  
**1183 If question\_type == 'multiple\_choice':**  
 1184 Think step by step. You must respond strictly in JSON format following this schema:  
 1185 {  
 1186 "thought": "<your reasoning>,"  
 1187 "answer": "<A/B/C/D>"  
 1188 }

1155 **TaskPlanning**  
 1156  
 1157  
 1158 GUI Agent Conditional QA Prompt  
 1159  
**1160 System**  
 1161  
**1162 If question\_type == 'yes\_or\_no':**  
 1163 You are a Graphical User Interface (GUI) agent. You will be given a screenshot, a question, and  
 1164 corresponding options. You need to choose one option as your answer.  
**1165 If question\_type == 'multiple\_choice':**  
 1166 You are a Graphical User Interface (GUI) agent. You will be given a task instruction, a screenshot,  
 1167 several GUI operations, and four options. Your goal is to select the best option that could solve  
 1168 the task.  
 1169 {question\_images}  
 1170  
**1171 User**  
 1172 {question\_text}  
 1173 {option\_texts}  
 1174 Which of the above options are correct according to the screenshot?  
**1175 Response Rules**  
 1176  
**1177 If question\_type == 'yes\_or\_no':**  
 1178 Think step by step. You must respond strictly in JSON format following this schema:  
 1179 {  
 1180 "thought": "<your reasoning>,"  
 1181 "answer": "<yes/no/unknown>"  
 1182 }  
**1183 If question\_type == 'multiple\_choice':**  
 1184 Think step by step. You must respond strictly in JSON format following this schema:  
 1185 {  
 1186 "thought": "<your reasoning>,"  
 1187 "answer": "<A/B/C/D>"  
 1188 }

1188  
1189

## A.4 PROMPTS FOR COMPLETING GUI TASKS

1190  
1191

## GUI Agent Conditional QA Prompt

1192

## System

1193

You are a helpful assistant.

1194

## User

1195

1196

You are a GUI agent. You are given a task and your action history, with screenshots. You need to perform the next action to complete the task.

1197

1198

## Output Format

1199

“ Thought: ... Action: ... “

1200

## Action Space

1201

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>',
end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key="")
type(content="") If you want to submit your input, use "
n" at the end of 'content'.
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', direction='down or up or right or left')
done() If you think the instruction is finished, parameters none
```

1211

1212

## User Instruction

1213

Task Instruction

1214

## Image Info

1215

Image size (pixels): width=image\_size[0], height=image\_size[1]. Output absolute pixel coordinates.

1216

1217

## A.5 DATASET STATISTICS OVERVIEW

1218

We show more detailed statistics of our benchmark in Figure 9.

1219

1220

## A.6 FULL APPLICATION LIST

1221

1222

Here we include the full list of applications involved in our benchmark.

1223

1224

## List of Applications

1225

1226

**Office (30):** Apple Notes, Apple Reminders, Calendar, Docs, Document Viewer, Evince, Gedit, Google Calendar, Google Docs, Google Keep, Keynote, Lark, Libreoffice, Notability, Note-taking App, Notepad, Notes, Notion, Numbers, Office, Overleaf, Pages, Powerpoint, Spreadsheet, Text Editor, VS Code, WPS Office, Microsoft Word, Xcode, Freeform.

1227

1228

**Media (18):** Amazon Music, Amazon Prime Video, Iheartradio, Likee, Music, Music Player, Pandora, Pocket FM, Podcast Player, Quicktime, Roku, Sofascore, Spotify, TikTok, Tubi, VLC media player, YouTube, YouTube Music.

1229

1230

**Game (12):** Arena\_of\_valor, CS2, Chess, Defense\_of\_the\_ancients\_2, Dream, Genshin\_impact, Minecraft, Nintendo, Pubg, Red\_dead\_redemption\_2, Steam, The Legend Of Zelda Breath Of The Wild.

1231

1232

**Editing (20):** 3dviewer, Adobe Acrobat, Adobe After Effects, Adobe Express, Adobe Photoshop, Adobe Photoshop Express, Adobe Premiere Pro, CapCut, Davinci Resolve, Draw.io, Gimp, Paint, PDF Editor, Photo Editing Tool, Photo Editor, Picsart, Procreate, Runway, Snapseed, Video Editing Software.

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

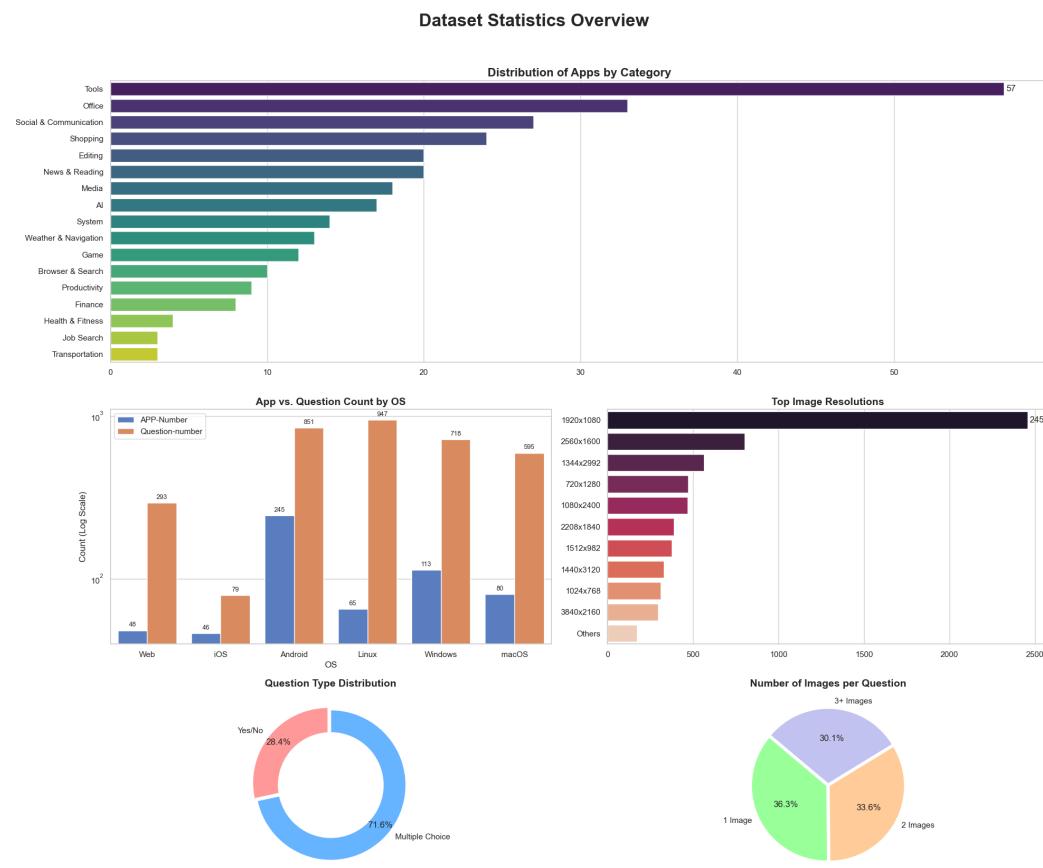


Figure 9: Dataset Statistics Overview

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296 **Social & Communication (28):** Discord, Facebook, Flickr, Gmail, Google Meet, Google Mes-  
 1297 sages, Imessage, Instagram, LinkedIn, Mail, Messenger, Outlook, Phone, Pinterest, Quora, Red-  
 1298 dit, Signal, Slack, Teams Live, Telegram, Threads, Thunderbird, Tumblr, WeChat, Weibo, What-  
 1299 sApp, X (Twitter), Zoom.  
 1300

1301 **Shopping (25):** 12306, Alibaba, Aliexpress, Amazon Shopping, Apartments.com, Applestore,  
 1302 Autoscout24, Autouncle, Booking.com, Car Marketplace, Cars.co.za, Ebay, Edmunds, Expedia,  
 1303 Magento, Offerup, Onestopmarket, Product Listing App, Realtor.com, Redfin, Shop, Taobao,  
 1304 Tripadvisor, Walmart, Wish.

1305 **AI & Tools (17):** AI Art Generator, Align-anything-dev-omni, Amazon Alexa, Chatbot AI, Chat-  
 1306 gpt, Chaton AI, DeepL Translate, Google Translate, Grammarly, Microsoft Copilot, Microsoft  
 1307 Translator, Remix AI Image Creator, Stable Diffusion, Translate, WOMBO Dream, Yandex  
 1308 Translate, Zhiyun Translate.

1309 **Browser & Search (10):** Bing, DuckDuckGo, Firefox, Google App, Google Chrome, Google  
 1310 Search, Opera, Safari, Web Browser, Web.

1311 **Tools (60):** Accerciser, Activities, Activity Monitor, App Lock, App Locker, Applock Pro, Au-  
 1312 tomator, Baidu Netdisk, Bluetoothnotificationareaiconwindowclass, Calculator, Camera, Clean,  
 1313 ClevCalc - Calculator, Color Management Utility, Colorsync\_utility, Contacts, Control Center,  
 1314 Cursor, Desktop, Dictionary, Digital Color Meter, Disk Utility, Drops, Electron, Email Client,  
 1315 File, File Explorer, File Manager, Files, Filezilla, Finder, Font Book, GPS, Image Viewer, Iphone-  
 1316 lockscreen, Kid3, Launcher, Mi Mover, Microsoft Store, Preview, Recorder, Rosetta Stone, Sci-  
 1317 entific Calculator Plus 991, Script\_editor, Search, Shortcuts, Spotlight, Stickies, System Infor-  
 1318 mation, System Search, System Settings, Task Manager, Terminal, Totem, ToDesk, Trash, Vim,  
 1319 Voicememos, Vottak, Wallpaper Picker.

1320 **Productivity (9):** Any.do, Drive, Dropbox Paper, Google Drive, Onedrive, Paperflux, Things,  
 1321 TickTick, Todoist.

1322 **News & Reading (22):** AP News, BBC News, BBC Sport, Bloomberg, Crimereads, Espn,  
 1323 Forbes, Goodreads, Google News, Google Play Books, Google Scholar, Kindle, Kobo Books,  
 1324 Metacritic, Microsoft News, Newsbreak, Wikidata, Wikipedia, Yahoo Sports, Apple News, Travel  
 1325 Guide App, Travel Review App.

1326 **Weather & Navigation (12):** Accuweather, Apple Maps, Citymapper, Google Maps, Mapillary,  
 1327 Miuiweather, Msnweather, Navigation App, Openstreetmap, Waze, Weather, Windy.

1328 **Finance (8):** Alipay, Budgeting App, Investing.com, Paymore, Stocks, Wallet For Your Business,  
 1329 Wallet: Budget Money Manager, Yahoo Finance.

1330 **Health & Fitness (4):** Fitbit, Fiton, Mideaair, Mifitness.

1331 **Job Search (3):** Indeed, Job Search By Ziprecruiter, Ziprecruiter.

1332 **Transportation (3):** Didi, Ryanair, Uber.

1333 **System & Tools (15):** Android, Android Home Screen, Android Launcher, Android Settings,  
 1334 Android Share Sheet, App Store, Apple, Applibrary, Gnome, Mobile Home Launcher, Mobile  
 1335 Launcher, Mobile Web Browser, OS, Ubuntu, Ubuntu Desktop.

1336 **A.7 ACTION TYPE PREDICTION CONFUSION MATRIX**

1337 Figure 10 and Figure 11 show the confusion matrix of tested models on desktop and mobile. All of  
 1338 these models have a tendency for predicting click instead of the right actions.

1339  
 1340  
 1341  
 1342  
 1343  
 1344  
 1345  
 1346  
 1347  
 1348  
 1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

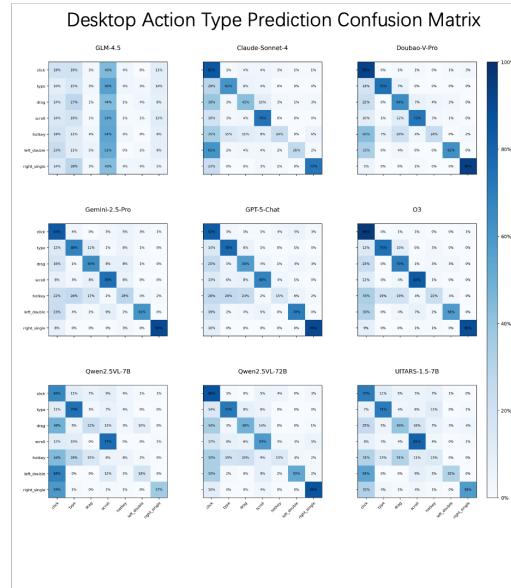


Figure 10: Confusion matrix of action type prediction in desktop.

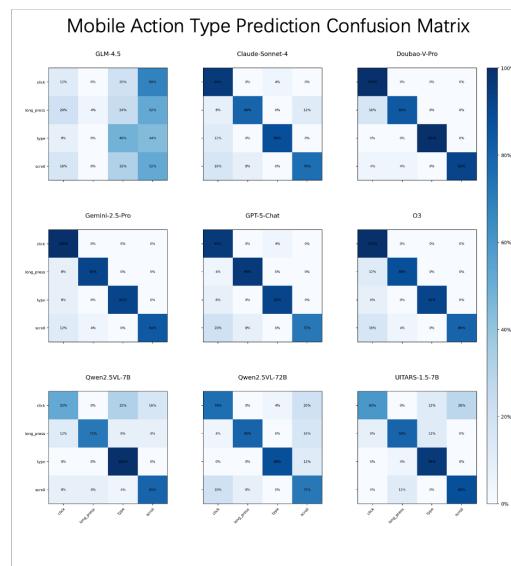


Figure 11: Confusion matrix of action type prediction in mobile.