Do GUI Grounders Truly Understand UI Elements?

Surgan Jandial[*†1], **Yinheng Li**[2], **Justin Wagle**[2], **Kazuhito Koishida**[2]
[1]Carnegie Mellon University [2]Microsoft

Abstract

Graphical User Interface (GUI) grounding is critical for effective GUI agents. Despite recent progress, key challenges remain: 1) existing grounding models and benchmarks are skewed toward web and mobile environments, neglecting desktop interfaces (especially windows); and 2) grounding capability is assessed using accuracy on a single "best" instruction per UI element. However, users can refer to a UI element in diverse valid ways – via visual attributes, spatial relations, etc, and a capable grounding model should produce consistent outputs across such variations. Focusing on desktop environments, we introduce **GUI Grounding Sensitivity Benchmark**, which investigates the model sensitivity to multiple descriptions of the same UI element. We design an automatic pipeline to generate multiple valid instructions per UI element, and develop nuanced data validation methods, as frontier models even hallucinate to produce a single instruction. Evaluation of **12** models reveals they are reasonably sensitive and their performance on existing benchmarks does not reflect their true ability. Building on the insight that a given grounding model struggles more with certain instructions or relations, we introduce the **GUI Grounding Diagnosis Agent**, which generates challenging instructions using model feedback and iterative refinement. Our agent reports high success rate **(upto 84%)** in generating instructions that fail the state-of-the-art GUI grounding models.

1 Introduction

Graphical User Interface (GUI) automation (Nguyen et al., 2024) is an emerging application of Large Language Models (LLMs) (Qwen et al., 2025; Grattafiori et al., 2024) and Vision Language Models (VLMs) (Bai et al., 2025; OpenAI, 2025a). Given a UI and a user instruction, the goal is to perform the appropriate action(s) to fulfill the instruction. A key challenge in this setting is *GUI grounding*—predicting the exact coordinates or bounding box where an action (e.g., click) should occur. This task is particularly difficult because: (1) it demands high precision, as small coordinate errors can trigger incorrect interactions and lead to unrecoverable states; and (2) the variability in UI layouts and instructions across platforms (web, mobile, desktop) exacerbates the problem. To tackle these challenges, recent works generally train large-scale GUI grounding models (Qin et al., 2025; Gou et al., 2025; Wu et al., 2024), and develop diverse grounding benchmarks (Cheng et al., 2024; Li et al., 2025).

Despite this, prior work remains heavily skewed toward web and mobile environments (Rawles et al., 2025; Koh et al., 2024; Deng et al., 2023), limiting the development and analysis of desktop agents. This imbalance largely stems from the ready availability of HTML and XML, which allows easy extraction of UI element–coordinate pairs. In contrast, desktop environments like Windows typically provide only UI screenshots, making dataset creation dependent on manual annotation or proprietary VLMs—both hard to scale and prone to hallucinations. Consequently, desktop-specific GUI grounding benchmarks and datasets are scarce, with only a few recent works (Hui et al., 2025; Bonatti et al., 2024; Xie et al., 2024) explicitly targeting this domain.

Beyond these limitations, a crucial yet often overlooked aspect of GUI grounding research is understanding the model's ability to handle diverse user instructions. Most prior works use either short (e.g., "close button") (Cheng et al., 2024) or simplified instructions (e.g., "Click the Pictures icon") (Li et al., 2025), which fail to capture the complexity of real-world interactions. Although recent efforts (Liu et al., 2025a; Nayak et al., 2025; Xie et al., 2025)

[*]Work completed during an internship at Microsoft
[†]Corresponding Author: sjandial@cs.cmu.edu

Findings of the Association for Computational Linguistics: EACL 2026, pages 2772–2785
March 24-29, 2026 ©2026 Association for Computational Linguistics

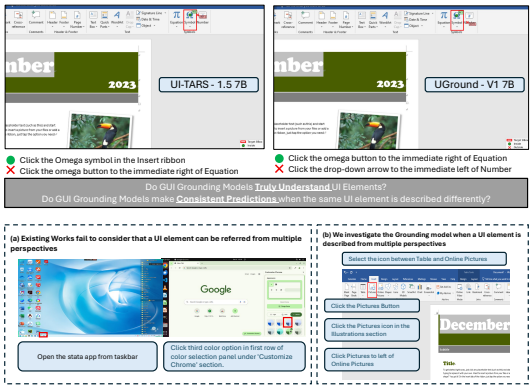


Figure 1: GUI grounding models struggle when UI elements are described from different perspectives (**Row 1**). Existing evaluations overlook this phenomenon by measuring accuracy on a single "best"/"common" reference (**Row 2**). Instead, we estimate the sensitivity (or consistency) of model predictions across different but valid references in our **GUI Grounding Sensitivity Benchmark**.

incorporate more implicit expressions (e.g., "click the left icon"), they do not assess whether models truly comprehend the multifaceted ways users might describe an element. This is partly due to their focus on generating a single "best" instruction per element, which often results in the most common or obvious phrasing. In practice, a UI element can be referenced in multiple valid ways—by visual traits, relative position, or surrounding context. As such, judging grounding models by their ability to ground a UI element for a single reference yields an incomplete view. Instead, to assess whether GUI grounding models genuinely grasp element identification, we must ask: **Are grounding model predictions consistent across diverse descriptions of the same UI element?**

We focus on desktop environments – a challenging yet underrepresented setting – and propose the first work to investigate the sensitivity of grounding models to multiple ways of referring to the same UI element. Since manually annotating multiple diverse instructions per UI element is impractical, we design an automatic pipeline for data generation and validation. This pipeline produces high-quality image-instruction sets, which are subsequently human-verified to construct our benchmark: **GUI Grounding Sensitivity Benchmark**.

We evaluate **12** GUI grounding models and report reasonable sensitivity in their predictions. Importantly, we find that although many recent models Qwen-2.5 (Bai et al., 2025), Jedi (Xie et al., 2025) outperform on popular GUI benchmarks they lag behind earlier models like UGround-V1 (Gou et al., 2025), OS-Atlas (Wu et al., 2024)) on our proposed benchmark. The aforementioned variation in sensitivity across models suggests that for a given GUI grounding model, certain instructions or contextual relationships are more challenging than others. This leads us to our next research question: developing a diagnostic pipeline that automatically generates diverse instructions, uses model feedback, and iteratively refines them to expose failure cases of the model. We call this system – **GUI Grounding Diagnosis Agent**. In summary:

- We propose the first framework for evaluating the **sensitivity** of GUI grounding models to diverse references of the same UI element.
- We focus on the underrepresented **desktop** setting and propose an automatic data generation pipeline, followed by human verification, to develop the **GUI Grounding Sensitivity Benchmark**.
- We present a **GUI Grounding Diagnosis Agent** to automatically extract failure cases of grounding models.

2 Related Work

Desktop GUI Agents: Vision Language Model (VLM) agents show strong potential in automating GUIs using natural language user instructions (e.g., "install apps", "close settings") and environment representations – text, images, or both (Koh et al., 2024; Rawles et al., 2023, 2025; Zheng et al., 2024; He et al., 2024; Deng et al., 2023; Jandial et al., 2025). While HTML, XML, or other text inputs are helpful, it is critical to improve GUI agents for image-only settings without access to such information (e.g desktop). Our work focuses on visual GUI grounding for desktop agents, which demands robust perception and GUI understanding. Prior works (Gou et al., 2025; Cheng et al., 2024; Qin et al., 2025) tackle this by collecting large-scale image-position pair data, however their focus is primarily mobile and web. UGround (Gou et al., 2025) collects 1.3M web-only GUI screenshot data, while GUICourse does 82K samples spanning web and mobile interfaces. OS-Atlas further curates 2.23M cross-platform screenshots (web, mobile, desktop), though their dataset remains web-heavy, with only 54K desktop-specific samples. Other approaches like

UI-Tars (Qin et al., 2025), combine closed-source and open-source data; however, they still report the desktop samples contribute the least to the open-source split. Although a few works (Xie et al., 2025; Hui et al., 2025; Xu et al., 2025) introduce desktop-focused datasets covering common usage scenarios, there is still a pressing need for broader attention to this domain. Thus, steering our focus towards the limitations and sensitivities of Desktop Grounding models.

GUI Grounding Evaluation: While recent models show improvements on existing benchmarks (Cheng et al., 2024; Li et al., 2025), the lack of rigorous desktop-specific testing in those evaluations limits our ability to accurately gauge their effectiveness in real desktop environments. In addition, these evaluations face two challenges: 1) they either test short or straight-forward explicit/implicit instructions that reflect the most common ways to refer to elements(Cheng et al., 2024; Li et al., 2025), 2) they only test the effect of image variations on GUI grounding (Zhao et al., 2025; Li et al., 2025; Zhang et al., 2025). Their fundamental limitation is that none consider the fact that there are often many ways to refer to a single UI element within a GUI screenshot. More precisely, for every GUI screenshot, a single UI element has multiple unique identifiers – visual, position, and so on. In other words, while GUI grounding models understand a few instructions (or common ways) to refer to a UI element, there is no guarantee that it will understand other ways to refer to the same element. Our analysis (Fig. 1) supports this observation, showing that current models frequently fail to identify a UI element when it is described from varied perspectives. To address this, we present the first effort to understand the sensitivity of grounding models to diverse referring expressions for the same UI element. Based on these insights, we introduce a novel agentic diagnostic pipeline that automatically uncovers instruction formulations where grounding models fail, enabling robust evaluation.

3 GUI Grounding Sensitivity

3.1 Background and Motivation

Given an image I and a UI element e, we can extract multiple visual and position identifiers:

$$\begin{aligned} e_v &= \{v_1, v_2, v_3, \ldots, v_n\} \\ e_p &= \{p_1, p_2, p_3, \ldots, p_m\} \end{aligned} \tag{1}$$

where e_v is the set of visual identifiers v_i and e_p is the set of position identifiers p_i. We define a unique identifier for e as any individual identifier (visual or position) or combination of v_i and p_i that matches only e—i.e., no other element can be described using it. Consequently, multiple identifiers can uniquely identify e:

$$\begin{aligned} e = \{ {v_1}^u,\ {v_2}^u,\ \cdots,\ {p_1}^u,\ {p_2}^u,\ \cdots, \\ (v_i,\ p_k)^u,\ \cdots,\ (v_j,\ p_k)^u,\ \cdots \} \end{aligned} \tag{2}$$

where v_i^u and p_i^u are individual unique identifiers, while $(v_i, p_k)^u$ denotes a combination unique identifier. Given that we aim to investigate the sensitivity of GUI grounding models against various ways of referring to e, our goal is to develop a data generation pipeline that reliably produces K distinct unique identifiers.

3.2 Data Generation

Image Selection: We select the desktop screenshots (particularly windows) from ScreenSpot (Cheng et al., 2024) and OmniAct (Kapoor et al., 2024), yielding 173 images. Then, we filter out screenshots that were too easy to understand. Particularly, we prompt VLM[1] to skip images with too few UI elements, a static, or minimally interactive screen requiring no meaningful GUI reasoning.

UI Element Extraction: For each image I, we first extract the bounding box (b_i) and OCR text (t_i) of all elements E using OmniParser (Lu et al., 2024):

$$E = \{e_i\}_{i=1}^N = (b_i, t_i)\}_{i=1}^N = \text{OmniParser}(I) \tag{3}$$

To allow for sensitivity in model predictions, we focus on UI elements that can be challenging to ground. We employ some heuristics to guide this selection - 1) visual similarity to other elements, 2) positional ambiguity – grouped closely with similar elements, overlapping, and 3) functional overlap – elements appear to serve similar purposes. We then ask VLM to sample relatively challenging UI elements from E:

$$H = VLM(E) = \{h_i\}_{i=1}^N \tag{4}$$

where h_i are the confusion risk scores for each element. We then select the top-D elements $E_D \subseteq E$ with the highest scores.

[1] all our pipelines use GPT-4.1 as the VLM

Further, it is also crucial that our pipeline generates K accurate instructions for a UI element e within I. However, selecting e based solely on confusion risk scores can be unreliable, as VLMs often struggle to ground elements that are small or poorly visible (see Fig 10). To address this, we incorporate two cues: the element's visibility score (v_i), which reflects how clearly it is visible on the screen (obtained by prompting the VLM), and its bounding box area (area(b_i)). We select the target element e from E_D as the one maximizing $v_i \times \text{area}(b_i)$.

Before proceeding, we manually inspect a random subset of images and observe that sometimes E_D is problematic. Specifically, several bounding boxes generated by OmniParser are incorrect—either empty, containing partial or multiple UI elements in a single box, or capturing non-interactive regions (like plain text). We prompt the VLM to filter out such cases, resulting in a set of **100** high-quality images.

Candidate Instruction Generation: Now, we ask the VLM to generate a diverse set of referring instructions for every (e, I) pair by incorporating multiple visual and positional identifiers (Eqn. 1). Particularly for positional cues, the VLM is instructed to describe e based on its immediate neighbors, broader local context, or the section it belongs to. While this approach yields many candidate instructions, we observe a notable number of hallucinations, consistent with findings in (Gou et al., 2025). Thus, unlike prior work, which assumes a single instruction per element, our setting necessitates a robust verification pipeline, described next.

3.3 Data Validation

Overall Correctness Verification: Our first validation step is a simple check: we highlight the element e using a bounding box and ask the VLM to check whether the instruction correctly refers to e. Particularly, the VLM is instructed to assess whether: 1) the elements mentioned in the instruction are visible, 2) the described visual details are accurate, and, 3) the stated semantic relationships with surrounding elements are correct.

Position Correctness Verification: Manual inspection reveals that while the previous step filtered some positional errors, it struggled with others. This is partly because VLMs tend to overgeneralize spatial terms like "next to" or "between," or lack precise interpretations for relations such as "left" or "above"—none of which were explicitly enforced earlier. To address this, we define strict spatial semantics (e.g., "left" refers to horizontal left; "next" means immediate adjacency). For instructions that pass the earlier check, we now perform explicit position verification by guiding the VLM with clear definitions of terms like "above," "below," "left of," and "right of," and filter out any spatially imprecise instructions.

Uniqueness Verification: In this step, we verify if the remaining instructions uniquely refer to the UI element e. That is, given an instruction i that refers to the UI element e, we ask the VLM to find (or count) if there exist other UI elements in image that can reasonably/substantially match the instruction.

3.4 Data Transformation

After manual inspection, we found that most generated instructions were acceptable, with the exception of a small subset that failed to uniquely identify e. Instructions containing visual cues (e.g., calendar icon) or specific positional references (e.g., first list item) were typically reliable, whereas those relying solely on relative position (e.g., icon between list and sidebar) often lacked uniqueness (see Fig 11) . Rather than repeatedly attempting to fix such cases, we hypothesize that making such instructions more specific (by adding visual details) is a more effective strategy for ensuring correctness.

We begin by isolating the set of instructions that refer to e purely through its relative position to nearby elements. Next, we evaluate whether these positional relationships are sufficient to uniquely identify e. To do this, we crop the image around e to include its local context—that is, neighboring elements $\mathcal{N}(e)$ located within a distance threshold d_{th}:

$$\mathcal{N}(e) = \left\{ j \;\middle|\; \|\mathbf{c}_j - \mathbf{c}_e\|_2 \leq d_{\text{th}} \right\}$$

Here, $\mathbf{c}_j$ denotes the center of the element j. We use $\mathcal{N}(e)$ to compute the local crop I_{local}:

$$I_{\text{local}} = I\Big[\min_{j\in\mathcal{N}(e)} x_{1,j},\ \min_{j\in\mathcal{N}(e)} y_{1,j},\ \max_{j\in\mathcal{N}(e)} x_{2,j},\ \max_{j\in\mathcal{N}(e)} y_{2,j}\Big] \tag{5}$$

We then use I_{local} to prompt the VLM to assess whether the instruction uniquely identifies e (similar to Sec. 3.3). This yields a set of successful and failing instructions (r_i^s, r_i^f). We transform r_i^f into r_{tr} by augmenting it with visual identifiers ($\{v_1, v_2, v_3, \ldots, v_n\}$), resulting in the final output

instructions r_o:

$$r_{tr} = \{ (v_1, r_1^f),\ (v_2, r_2^f),\ \ldots,\ (v_n, r_n^f) \}$$
$$r_o = \{r_s, r_{tr}\} \tag{6}$$

Finally, we perform a manual review of the generated instructions, resulting in **1,712** validated instructions across all images — approximately 17 instructions per UI element. Refer to Fig 4 for the visual illustration of the entire data generation pipeline.

3.5 Evaluation Metrics

In this section, we describe our evaluation measures to assess the proposed sensitivity of grounding models M.

Mean Distance (s_{mean}): Given image I, UI element e, and K different perspective instructions ($\{r_k\}_{k=1}^K$), M predicts K output coordinates $\{c_k\}_{k=1}^K$. We compute their centroid (c_{centroid}), and then define the sensitivity to understand e as the mean distance:

$$d_{mean} = \frac{1}{K} \sum_{k=1}^{K} \frac{|c_k - c_{\text{centroid}}|}{diag(b)} \tag{7}$$

where $diag(b)$ is the diagonal of the bounding box b enclosing e. Finally, we average d_{mean} across input images ($I \in \mathcal{I}$) to obtain s_{mean}:

$$s_{mean} = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} d_{mean}^I \tag{8}$$

Worst Distance (s_{worst}): We define the worst-case sensitivity as the maximum distance of any predicted coordinate c_k from c_{centroid}:

$$d_{worst} = \max_k \frac{|c_k - c_{\text{centroid}}|}{diag(b)} \tag{9}$$

Then, we average d_{worst} across input images ($I \in \mathcal{I}$) to obtain s_{worst}:

$$s_{worst} = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} d_{worst}^I \tag{10}$$

Instruction Out-of-Box Rate (s_{oob}): This metric measures the number of instructions where the output coordinate c does not lie inside the bounding box b containing e:

$$s_{oob} = \frac{1}{\sum_{I \in \mathcal{I}} K_I} \sum_{I \in \mathcal{I}} \sum_{k=1}^{K_I} 1\{c_k \notin b_k\} \tag{11}$$

3.6 Experiments

Models Used: We evaluate **12** models: OpenAI Operator (OpenAI, 2025b), UI-Tars-1.5-7B (Qin et al., 2025), UGround-V1-2B/7B (Gou et al., 2025), OS-Atlas-7B (Wu et al., 2024), ShowUI-2B (Lin et al., 2024), Jedi-3B/7B (Xie et al., 2025), Qwen-2.5-VL-7B/32B (Bai et al., 2025), InfiGUI-R1-3B (Liu et al., 2025b), UI-R1-E-3B (Lu et al., 2025).

Performance on GUI Grounding Sensitivity Benchmark: We report our results in Table 1 (Base Column), which also includes model performance on ScreenSpot(SS)-Pro (Li et al., 2025), a widely used GUI grounding benchmark. Our findings indicate that the Operator is the least sensitive model, while the Jedi family has the highest sensitivity.

Among open-source models, UI Tars 1.5 7B and UGround-V1-7B perform best, likely due to large-scale training and curated datasets. We observe that sensitivity generally improves with model size – UGround-V1-7B and Qwen-2.5-VL 32B outperform their smaller versions.

Interestingly, reinforcement learning-trained models (Shao et al., 2024; Ahmadian et al., 2024) like InfiGUI-R1-3B and UI-R1-E 3B are competitive with 7B-scale models and sometimes surpass OS-Atlas-7B, Qwen-2.5-VL 7B.

Finally, consistent with our hypothesis, we find that strong performance on existing GUI benchmarks does not imply that models better understand UI elements from multiple perspectives. For example, although UI Tars 1.5 7B outperforms Operator on SS-Pro, it lags behind in our results. Similar trends are noted for other pairs like UGround-V1-7B vs. Qwen-2.5-VL 32B and OS-Atlas-7B vs. Qwen-2.5-VL 7B, and so on.

Performance over different Image Inputs: Here, we analyze the sensitivity of grounding models by varying common ways in which UI elements appear within GUI screenshots. For the first setup, **Local Crop**, we use Eq.5 to zoom-in and generate a crop around the UI element. In the second setup, **Window-in-Background**, we simulate realistic scenarios where applications appear in smaller windows over desktop backgrounds by overlaying the original screenshot onto a background image (see Fig. 7). Results in Table 1 show no clear benefit of local cropping on sensitivity. This is because, although cropping may improve visibility of target element, it does not necessarily help the model understand the semantic relationship between ele-

Model	Base			Local Crop			Window-in-Background			Language			SS-Pro
	$s_{\text{mean}} \downarrow$	$s_{\text{worst}} \downarrow$	$s_{\text{oob}} \downarrow$	$s_{\text{mean}} \downarrow$	$s_{\text{worst}} \downarrow$	$s_{\text{oob}} \downarrow$	$s_{\text{mean}} \downarrow$	$s_{\text{worst}} \downarrow$	$s_{\text{oob}} \downarrow$	$s_{\text{mean}} \downarrow$	$s_{\text{worst}} \downarrow$	$s_{\text{oob}} \downarrow$	Acc. ↑
Operator	0.3217	1.3342	**13.52**	**0.2364**	0.8817	**14.02**	1.0200	3.5524	**33.06**	0.1632	0.5346	2.96	36.6
UI-TARS-1.5-7B	**0.3021**	1.2723	20.61	0.2406	**0.8683**	33.24	**0.9063**	**2.5977**	38.94	0.1017	0.4773	2.50	**61.6**
Qwen2.5-VL-32B	0.5059	1.6707	28.44	0.3790	1.1481	36.36	1.4885	3.8425	54.82	0.1316	0.3727	2.73	47.6
Qwen2.5-VL-7B	0.4548	1.5358	41.20	0.3613	1.0161	46.76	1.7426	4.3578	73.12	0.1101	0.3324	4.82	27.6
UI-R1-E-3B	0.4207	1.4895	26.75	0.2503	0.7685	42.76	1.0909	3.4827	46.10	0.1043	0.2837	2.28	33.5
InfiGUI-R1-3B	0.3854	1.3938	26.52	0.2536	0.8966	41.53	1.2301	3.6199	44.00	0.1023	0.4426	2.42	35.7
UGround-V1-7B	0.3176	**1.1512**	22.55	0.2392	0.8616	24.82	1.0558	3.9273	39.92	**0.0718**	0.3212	**1.28**	31.1
UGround-V1-2B	0.6218	1.9334	39.54	0.4074	1.1863	38.38	1.4080	4.5210	54.90	0.0813	0.3618	3.07	25.0
OS-Atlas-7B	0.4641	1.6698	29.96	0.2747	0.8819	27.57	1.3227	3.8103	54.09	0.0967	**0.2029**	1.32	18.9
ShowUI-2B	0.6423	2.1535	47.20	0.3630	1.0758	45.79	1.1918	3.7318	71.15	0.2156	0.8340	6.56	7.7
Jedi-7B-1080p	1.0426	2.3546	77.39	0.4955	1.2417	79.21	1.2737	2.8370	87.94	0.7152	2.0161	7.58	39.5
Jedi-3B-1080p	0.9158	2.0822	76.20	0.5062	1.0818	74.47	1.1964	2.6510	81.67	0.7365	1.9309	13.97	36.1

Table 1: **Main Results**. (a). Sensitivity scores ($s_{mean}, s_{worst}, s_{oob}$) of **12** grounding models (**Base** Column) reveal that strong performance on existing benchmarks (**SS-Pro** Column) does not imply models truly understand a UI element. (b) Enhancing target element visibility via zoom-in (**Local Crop**) does not improve sensitivity, while increasing image complexity (**Windows in Background**) worsens it. (c) Models perform well under linguistic perturbations (**Language**), suggesting that sensitivity mainly stems from inherent limitations in understanding UI elements. Results for SS-Pro are taken from existing papers.

ments, disambiguate similar-looking components, or overcome inherent biases and limitations. Finally, we observe that increased visual complexity of the Window-in-Background worsens sensitivity.

Performance over language variations: Here, we explore grounding sensitivity to linguistic variations, including spelling errors, synonym replacements, and paraphrasing. To ensure that performance changes are attributed solely to these variations, we apply perturbations to instructions that most clearly and directly describe the UI element – cases where the model is expected to perform confidently. Results in Tab. 1 (Language) show that most models, with the exception of the Jedi family, exhibit robustness to linguistic variations. Moreover, models exhibit substantially lower sensitivity compared to results in Table 1 (Base). For instance, Uground-V1-7B has $s_{mean} = 0.3176$ in Tab. 1 (Base) vs. 0.07 in Language; similarly, OS-Atlas-7B has 0.46 vs. 0.09, and Qwen2.5-VL-32B has 0.50 vs. 0.13.

4 Automated Grounding Diagnosis

Motivation: Results from the previous section highlight that state-of-the-art GUI grounding models exhibit sensitivity when a UI element is described from multiple perspectives. That is, some instructions or spatial relations are more difficult for the model to interpret than others. This naturally motivates our exploration towards an agentic pipeline that can iteratively explore instructions, understand model's behavior on those instructions, and automatically identify cases that lead to grounding failures. In this section, we present our pipeline, called **Automatic Grounding Diagnosis Agent**, which comprises three key components.

Step I - Seed Instruction Generation: To initiate our agentic loop, we first obtain K seed instructions $(R_1, R_2, \ldots, R_K)$ for a UI element e using Sections 2.2–2.3. We then predict output coordinates o_c^k for each R_k using the grounding model M, and record whether o_c^k lies within the bounding box b containing e.

Step II - Diagnosis Plan Generation: Next, we use the instructions where M succeeds (i.e. ${o_c}^k$ is inside b) to devise a diagnosis plan that could fail M. We begin by asking the VLM to analyze previous successful instructions, and come up with reasons r that made those instructions easy to ground:

> ...
> - Reason why instruction did not introduce ambiguity.
> - Highlight specific cues/phrasing that made disambiguation easy.
>
> ...

We then use r and ask VLM to generate a plan P which results in instructions that can potentially fail M. To ensure VLM generates P that is grounded in realistic challenges faced by M, we provide a list of heuristics to consider in the generated strategy:

> Your instruction Generation Strategy may include:
> - **Spatial phrasing traps** with nearby elements
> - **Visual lookalike confusion** based on color, size, iconography
> - **Functional overlaps** with elements that do similar things
> - **OCR proximity issues** for ambiguity with nearby text

Note that even though the aforementioned step involves two logical parts, we find that incorporating them in a single prompt is sufficient to achieve the goal.

Step III - Diagnostic Instruction Generation: We now require A to generate diagnostic instructions $(R_1^d, R_2^d, \cdots, R_N^d)$ that are different and appropriately incorporate P.

```
{
 "improvement_strategy": "how you used the
Plan to craft harder instructions",

 "generated_instructions": [
   {
     "instruction": "user instruction #1",
     "reason": "brief explanation"
   },
   ...
 ]
}
```

Since every R_k^d may not fail the model (i.e. ${o_c}^k$ is outside b), we filter those that achieve the intended goal. Then, we loop through Steps II, III for a maximum of $T = 10$ iterations or until $S = 5$ desired instructions are obtained.

Step IV - Instruction Selection: We find that some of the S generated instructions may contain incorrect details. However, our goal is not to collect S failing cases, but to identify the most accurate and effective one. Furthermore, since each instruction may include a different detail of the UI element, a detail missed or misrepresented in one may be correctly captured in another. This motivates a joint evaluation: instead of assessing correctness in isolation, we verify every instruction in the context of other instructions. Specifically, we input the image and complete instruction set into the VLM, allowing it to consider details from all instructions to inform its decisions. In cases where no instruction is deemed correct, we record no failing instruction for that image.

Next, multiple instructions may pass the previous step. In the final step, we select the best one—defined as the instruction with the fewest identifiers that could be confused with other elements. To do this, we ask the VLM to describe all UI elements that loosely match each instruction and choose the one with the fewest matches. For example, given "click the photos icon in the top left," a photo icon in the bottom right would still be counted as a loose match.

```
{
 "number": [number],
 "elements": [
   {
     "description": "element description",
     "clarity_rating": "(1-5), how clearly does
     instruction refer to the element"
   },
   ...
 ]
}
```

4.1 Experiments

In this section, we apply our Diagnostic Agent on state-of-the-art GUI grounding models. Due to OpenAI costs, we select 50 images and two models—UGround-V1-7B and ShowUI-2B.

Performance of Diagnostic Agent: Given an input image, the diagnostic agent aims to produce a single instruction that causes the grounding model to fail. There are two possible outcomes: (1) the agent fails to find such an instruction—either by hitting the iteration limit or being filtered out during Instruction Selection; or (2) it succeeds in generating an instruction that triggers a grounding failure, however, instruction may be correct (i.e., it accurately refers to a UI element) or incorrect. Importantly, we do not expect the agent to succeed for every image. The goal is to uncover potential failure cases of the model, with an emphasis on the accuracy of these failures. Specifically, we define **Success Rate (SR)** as the percentage of correct instructions among all failure-inducing instructions. To ensure evaluation reliability, we manually annotate the generated instructions when computing SR, avoiding potential inaccuracies from VLM-based evaluation. Our results in 2 (a) show that our Diagnostic Agent achieves a high success rate (upto 84%) in generating valid instructions that fail the grounding model. We include example failure cases generated by the agent in Fig. 3.

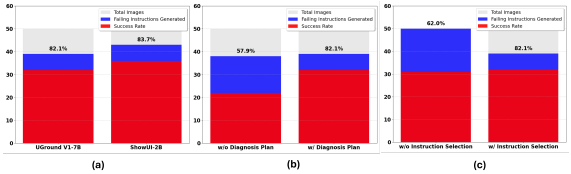
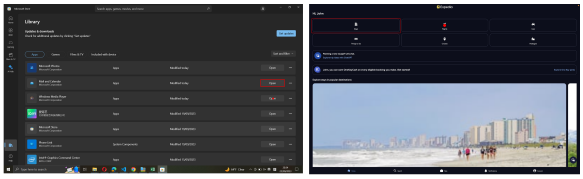


Figure 2: **Evaluation** of our Diagnostic agent. (a) **High Success Rate** of diagnostic instructions that fail UGround-V1-7B and ShowUI-2B, (b). **Diagnosis Plan Generation** improves the precision of generated instructions (c). **Instruction Selection** effectively filters out incorrect failing instructions.

Transfer to other GUI models: In this experiment, we test if the correct failing instructions for previ-



(a). Use the Open button directly above Windows Media Player

(b). Pick the option in the upper row, furthest from Cars.

Figure 3: **Failing Instructions** generated for UGround-V1-7B. ▭ : Target UI element, ● : Incorrect model prediction. Zoom in for clarity.

ous models potentially fail other GUI grounding models. Our metric here is Instruction failure rate, i.e, how many failed a given model. The results in Tab 2 demonstrate that failing instructions for one model can also lead to failure for other models.

Model	UGround-V1-7B	ShowUI-2B
Operator	**12.50%**	**16.67%**
UI-TARS-1.5-7B	34.38%	25.71%
UGround-V1-7B	100.00%	44.44%
UGround-V1-2B	62.50%	69.44%
Qwen2.5-VL-32B-Instruct	43.75%	38.89%
Qwen2.5-VL-7B-Instruct	62.50%	66.67%
InfiGUI-R1-3B	34.38%	50.00%
UI-R1-E	37.50%	55.56%
OS-Atlas-Base-7B	43.75%	40.00%
ShowUI-2B	62.50%	100.00%
Jedi-3B-1080p	59.38%	77.78%
Jedi-7B-1080p	75.00%	83.33%

Table 2: **Instruction failure rate**(↓) of GUI grounding models on failing instructions generated for UGround-V1-7B and ShowUI-2B.

Impact of different components: In this section, we ablate our core components – Diagnosis Plan Generation and Instruction Selection component. We use UGround-V1-7B as the reference model, and manually annotated Success Rate (SR) for each study.

From Fig. 2b, we observe that total number of failure-inducing instructions is comparable with and without the Diagnosis Plan Generation step. However, the accuracy of the generated instructions is significantly higher when this step is included, indicating the benefit of structured planning in instruction quality. Fig. 2c further shows that omitting the Instruction Selection step increases the number of failing instructions, but many are incorrect, resulting in a sharp drop in precision. With Instruction Selection, the agent achieves a higher accuracy—82.1% vs. 62.0%—demonstrating its effectiveness to filter out noisy instructions.

5 Conclusion

In this work, we focus on GUI grounding and highlight key limitations of existing approaches and benchmarks. Current methods largely overlook desktop environments, focus mainly on image variation, and evaluate grounding models using only a single "best" instruction per UI element. However, users may refer to UI elements through diverse visual and positional cues. Therefore, grounding models should produce consistent outputs across such variations, as single-instruction accuracy provides an incomplete view of model performance.

To address this, we design an automated data synthesis pipeline that generates multiple instructions per UI element and release the GUI Grounding Sensitivity Benchmark to assess sensitivity to diverse descriptions. Our findings show that current models are indeed sensitive to instruction variation, and strong performance on existing benchmarks does not imply true UI understanding. We also observe robustness to language variation, while increasing image complexity worsens sensitivity.

Finally, recognizing that grounding models lack a comprehensive understanding of UI elements, we propose extracting the relations or descriptions they struggle with. We introduce the GUI Grounding Diagnosis Agent that automatically generates diverse instructions, incorporates model feedback, and iteratively refines them to expose failure cases. This diagnostic tool provides a new direction for evaluating and improving grounding models.

6 Limitations

Despite our contributions, our work has several limitations. First, due to constraints of GPT APIs and manual verification, our sensitivity analysis is confined to Windows desktop environments. We chose this setting deliberately, as windows remains the most underexplored environment compared to the others. Second, our dataset construction relies heavily on existing resources such as ScreenSpot and OmniAct, which naturally constrain the diversity of software and interaction scenarios included in our benchmark. Consequently, there remain many unexplored domains where sensitivity to varied instructions could present unique challenges. Finally, while our analysis sheds light on the importance of evaluating GUI grounding from multiple perspectives, we did not conduct experiments to explicitly establish how such multiple-description datasets could improve model generalization. We

hope that our benchmark and diagnostic pipeline inspire future research to address these limitations and build models that better adapt to diverse user instructions.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *Preprint*, arXiv:2402.14740.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

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

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *Preprint*, arXiv:2401.10935.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

Zheng Hui, Yinheng Li, Dan zhao, Tianyi Chen, Colby Banbury, and Kazuhito Koishida. 2025. Winclick: Gui grounding with multimodal large language models. *Preprint*, arXiv:2503.04730.

Surgan Jandial, Yinong Oliver Wang, Andrea Bajcsy, and Fernando De la Torre. 2025. On the fine-grained planning abilities of VLM web agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 25347–25380, Suzhou, China. Association for Computational Linguistics.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *Preprint*, arXiv:2402.17553.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *Preprint*, arXiv:2401.13649.

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

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

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

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025b. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *Preprint*, arXiv:2504.14239.

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

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *Preprint*, arXiv:2503.21620.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A. Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M. Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. 2025. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *Preprint*, arXiv:2503.15661.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim,

Ruiyi Zhang, Tong Yu, Mehrab Tanjim, and 10 others. 2024. Gui agents: A survey. *Preprint*, arXiv:2412.13501.

OpenAI. 2025a. Gpt-4.1: Improved coding, instruction-following, and long-context abilities. API release blog post, OpenAI.

OpenAI. 2025b. Operator: A computer-using ai agent. arXiv preprint arXiv:2501.10114. Research preview of the "Operator" AI agent; released January 23, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. *Preprint*, arXiv:2505.13227.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Preprint*, arXiv:2404.07972.

Yibin Xu, Liang Yang, Hao Chen, Hua Wang, Zhi Chen, and Yaohua Tang. 2025. Deskvision: Large scale desktop region captioning for advanced gui agents. *Preprint*, arXiv:2503.11170.

Yanzhe Zhang, Tao Yu, and Diyi Yang. 2025. Attacking vision-language computer agents via pop-ups. *Preprint*, arXiv:2411.02391.

Haoren Zhao, Tianyi Chen, and Zhen Wang. 2025. On the robustness of gui grounding models against image attacks. *Preprint*, arXiv:2504.04716.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. *Preprint*, arXiv:2401.01614.

A Additional Visualizations

In this section, we present additional visual illustrations that were omitted from the main paper due to space constraints: 1) Fig 4 is our data generation pipeline in Sec 3, 2) Fig 5 includes dataset examples for our GUI Grounding Sensitivity Benchmark, 3) Fig 6 includes qualitative outputs of state-of-the-art GUI grounding models on our benchmark, 4) Fig 7 is the visualization for Local Crop View and Windows-In-Background View.

B GUI Grounding Performance over granularity of instructions

Results in Sec 3.6 demonstrate that current models struggle in consistently predicting outputs over multiple ways to describe UI elements. In this section, we understand the performance as a variation of instruction specificity. More particularly, we consider three levels of specificity: 1) High – most direct and unambiguous combination of visual and position indicators, 2) Medium – relatively less specific visual indicators and position indicators, 3) Low – no visual indicator and only relies on position indicator. Note that "no position indicator and only visual indicator" is not plausible because images often have more than one UI element of same visual appearance, and it would not necessarily produce a correct instruction. We report the Instruction Out-of-Box rate (s_{oob}) in Fig. 9 and observe a clear trend: model performance degrades (higher s_{oob}) as instruction specificity decreases, highlighting their sensitivity and supporting our hypothesis.

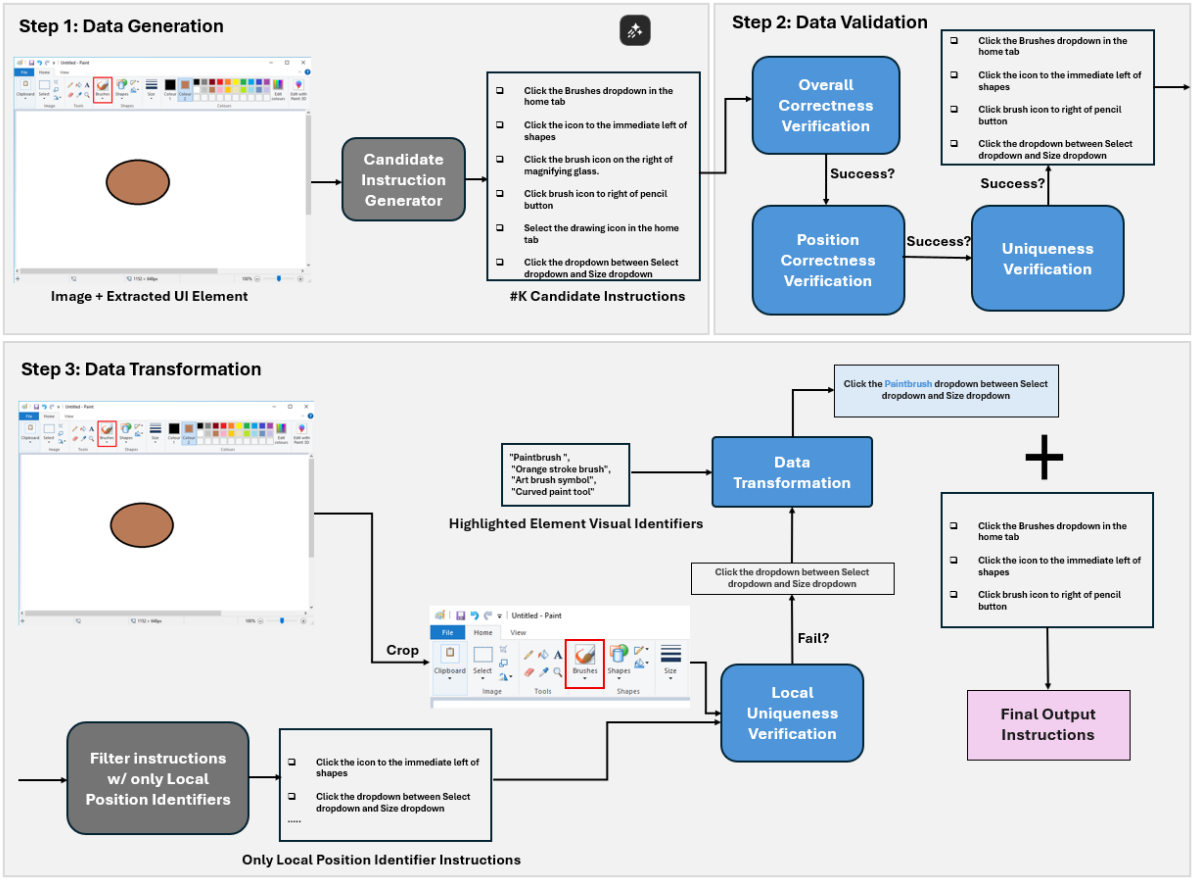


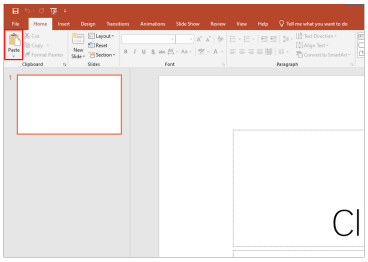
Figure 4: Data synthesis pipeline for generating multiple instructions per UI element. For more details, refer to Sec 3.2 for Step 1, Sec 3.3 for Step 2, Sec 3.4 for Step 3.

C Unreliability to generate descriptions of small UI elements

As discussed in Section 3.2, considering both the visibility score and the bounding box of UI elements is crucial to ensure that proprietary VLMs (such as GPT-4.1) can accurately interpret the elements and generate multiple valid, high-quality instructions. In Fig 10, we present empirical results supporting our observation that instruction generation is often unreliable for small UI elements.

D Limitations of Data Validation

As discussed in Section 3.4, even after the data validation step, some instructions may still ambiguously refer to multiple UI elements within the image, i.e, failing to uniquely identify the target element e. We now present qualitative examples in Fig. 11 to better understand on what kind of samples does the validation process fails.



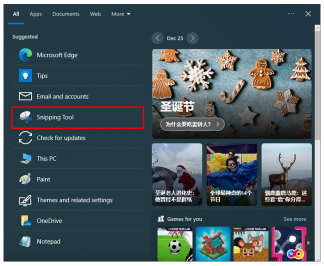
"Click the Paste icon in the Clipboard section“

"Use the Paste option in the upper left corner",

"Hit the downward arrow under the clipboard image in Home"

"Click the clipboard icon in the Home tab section"

"Select the large button above the word Paste in the ribbon"



"Select the option in the Suggested section just after Email and accounts"

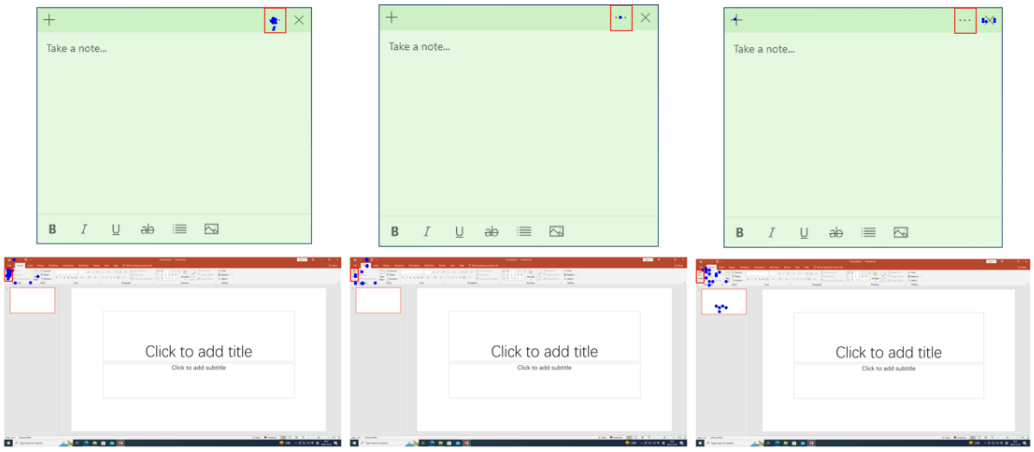
"Open the white oval feature between the envelope and the circular arrow"

"Launch the fourth item in the Suggested list“

"Click the scissors icon in the Suggested section"

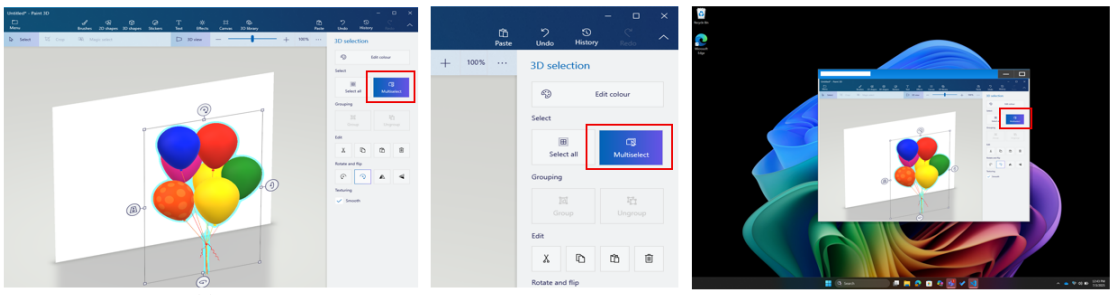
"Select the tool between Email and accounts and Check for updates",

Figure 5: Dataset examples from the GUI Grounding Sensitivity Benchmark. For illustration, we display only 5 instructions per UI elements.



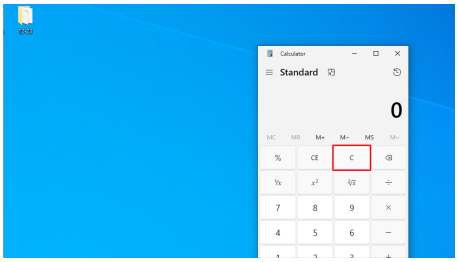
(a). Operator **(b).** UGround-V1-7B **(c)** ShowUI-2B

Figure 6: Qualitative visualizations of grounding model outputs on diverse instructions referring to the same element show that the predicted coordinates often vary within a region, appear in separate areas, or fall outside the target bounding box.

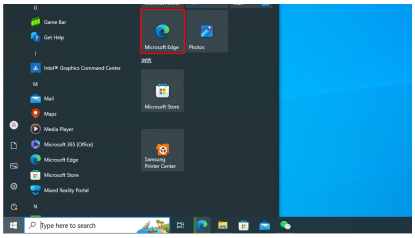


(a) Original Image **(b) Local Crop** **(c) Window-in-Background**

Figure 7: Diverse ways in which UI elements can appear



Reason: 'C' and 'CE' (ID 14) are very similar both visually and functionally. References to 'clear' are highly ambiguous.



Reason: There is also a Microsoft Edge entry in the left app list (ID 11) and a Microsoft Edge shortcut in the taskbar (ID 20). All share the same icon and label, and a generic instruction ('open Edge') could refer to any. The tile and list versions are especially similar.

Figure 8: Qualitative visualization of the VLM's inferred reasoning for why a UI element is confusing to ground (See Eqn 4 in the main paper).

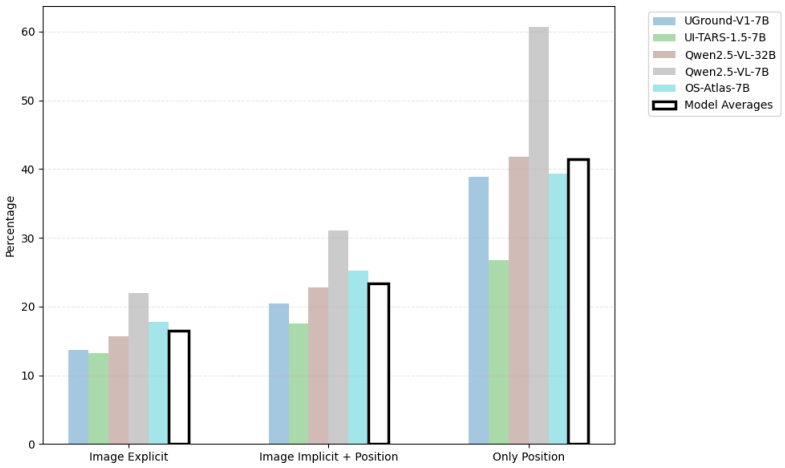


Figure 9: Grounding Performance over granularity of Instructions

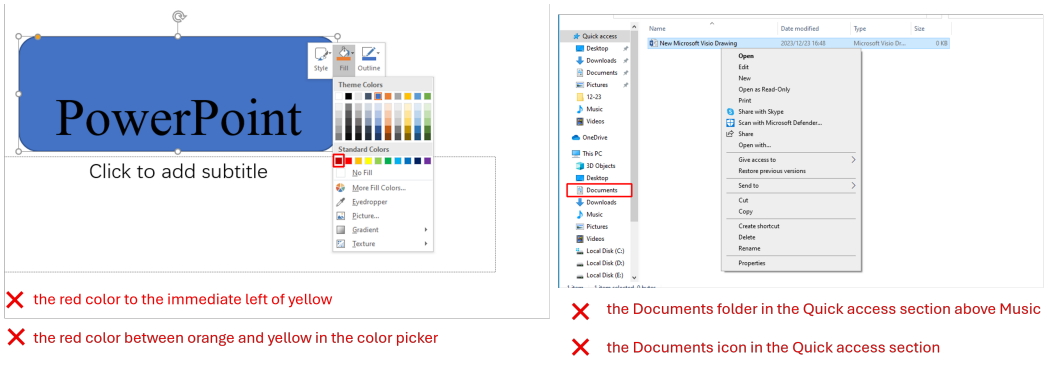
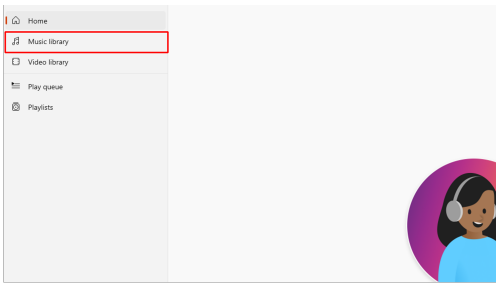
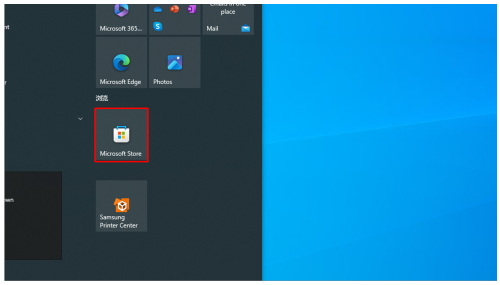


Figure 10: Proprietary VLMs like GPT-4.1 generate incorrect referring instructions for small elements (shown in red bounding box).



Choose the library entry in the sidebar below Home

Reason: There are two library icons – Music and Video and both are below Home



Open the app tile lies between Photos and Samsung Printer Center

Reason: There are two apps – Microsoft Edge and Microsoft Store

Figure 11: Examples where Data Validation step (Sec 3.3) fails to filter out instructions that do not uniquely identify the target element.