# ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use

**Kaixin Li**
National University of Singapore
likaixin@u.nus.edu

**Ziyang Meng**
East China Normal University

**Hongzhan Lin**
Hong Kong Baptist University

**Ziyang Luo**
Salesforce Research

**Yuchen Tian**
Hong Kong Baptist University

**Jing Ma**
Hong Kong Baptist University

**Zhiyong Huang**
National University of Singapore

**Tat-Seng Chua**
National University of Singapore

## Abstract

Recent advancements in Multi-modal Large Language Models (MLLMs) have led to significant progress in developing GUI agents for general tasks such as web browsing and mobile phone use. However, their application in professional domains remains under-explored. These specialized workflows introduce unique challenges for GUI perception models, including high-resolution displays and complex environments which lead to smaller target sizes. In this paper, we introduce **ScreenSpot-Pro**, a new benchmark designed to rigorously evaluate the grounding capabilities of MLLMs in high-resolution professional settings. The benchmark comprises authentic high-resolution images from a variety of professional domains with expert annotations. It spans 23 applications across five industries and three operating systems. Existing GUI grounding models perform poorly on this dataset, with the best model achieving only 18.9%. Our experiments reveal that strategically reducing the search area enhances accuracy. Based on this insight, we propose **ScreenSeekeR**, a visual search method that utilizes the GUI knowledge of a strong planner to guide a cascaded search, achieving state-of-the-art performance with 48.1% without any additional training. We hope that our benchmark and findings will advance the development of GUI agents for professional settings. The code, data and benchmark are available at `https://gui-agent.github.io/grounding-leaderboard/`.

## 1 Introduction

Imagine a future where the everyday burdens of repetitive computer tasks are lifted, unleashing people's full productivity and creativity. A GUI agent capable of taking over the mundane operations of complex professional applications like Visual Studio Code, AutoCAD, Photoshop, could greatly enable computer users to focus exclusively on the work that truly matters. Recent advancements in Multi-modal Large Language Models (MLLMs) [1, 2, 3, 4] have significantly invigorated this pursuit, driving intensive research efforts in creating pure-vision based GUI agent models that can directly interact with electronic devices that are integral to modern life [5, 6].

Preprint. Under review.

However, many existing studies primarily address general and easy tasks, such as general computer control [7, 8], web browsing [9, 10, 11], lifestyle and utility apps [12, 13]. In contrast, professional applications remain largely unexplored, with only a few works featuring specialized tasks such as coding in VSCode [14]. These software are designed to provide a comprehensive suite of advanced features, catering to specialized tasks and workflows, and are thus fundamental in productivity and creative industries. Developing GUI agent systems could not only reduce the manual burden of repetitive actions but also enhance productivity and lower the barrier to entry for non-expert users.

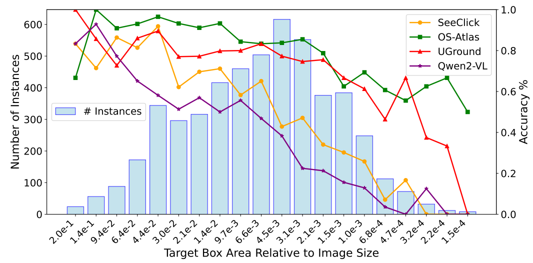
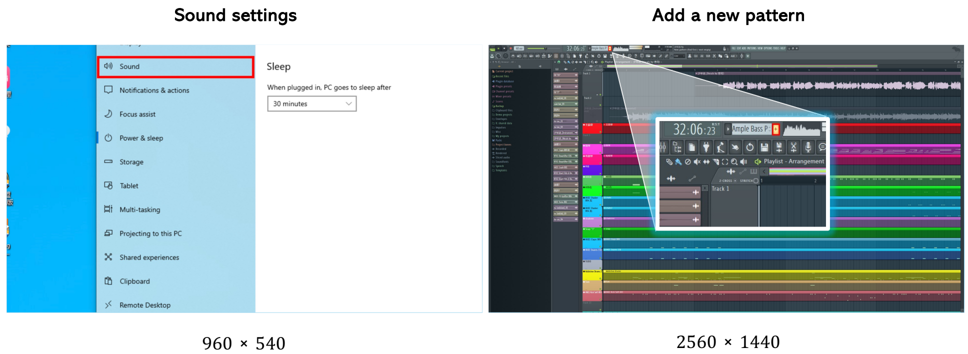



Figure 1: Performance of the expert GUI grounding models SeeClick [8], OS-Atlas-7B [15], UGround [16], and the generalist MLLM Qwen2-VL-7B [2] on the ScreenSpot-v2 GUI grounding benchmark [15]. The elements on the x-axis are arranged in logarithmically *decreasing* order, representing their relative size in the entire image. There is a universal decrease in accuracy as the target box size becomes smaller.

To advance toward this vision, we focus on a previously underexplored challenge: GUI grounding in professional, high-resolution software environments. Given a natural language instruction and a screenshot, the goal is to ground the instruction to the precise location of the target UI element. The primary challenges in applying GUI grounding models to these professional applications are threefold: (1) the significantly greater complexity of professional applications compared to general-use software, which often requires higher resolutions that may be difficult for existing MLLMs to handle; (2) as user interfaces are typically designed with fixed pixel sizes and users need to display more content on the screen, the increased resolution results in smaller relative target sizes within the screenshot. This often leads to poorer performance of GUI grounding models, as we demonstrate in Figure 1; (3) professional users frequently rely on supplementary documents and external tools to assist their workflows, further complicating the screen and introducing additional challenges for GUI understanding. Consequently, even if the MLLMs are able to comprehend the user instructions, it is difficult for them to ground the instructions into precise locations in such complex screenshots.

To fill the notable gap in research on GUI operations in professional environments, we introduce ScreenSpot-Pro, a novel GUI grounding benchmark that includes 1,581 expert-annotated instructions in their authentic workflows, each in a unique screenshot. They are sourced from 23 applications in five types of industries, as well as common usages in 3 operating systems. ScreenSpot-Pro differentiates itself from previous grounding benchmarks [8, 15, 17] in that: (1) *Data Diversity:* ScreenSpot-Pro includes authentic high-resolution images and tasks from a wide range of professional applications and domains, beyond simple web browsing or mobile use, reflecting the complexity and variety of real-world professional scenarios; (2) *Applicability:* ScreenSpot-Pro provides full screenshots, avoiding the unrealistic evaluation of GUI grounding in cropped local regions; (3) *Quality:* ScreenSpot-Pro is annotated by professional users, ensuring rigorous quality control to maintain the validity of test samples, thereby guaranteeing reliable and meaningful evaluation results. A visual comparison of ScreenSpot-Pro and ScreenSpot [8], a widely-used GUI grounding benchmark, is presented in Figure 2.

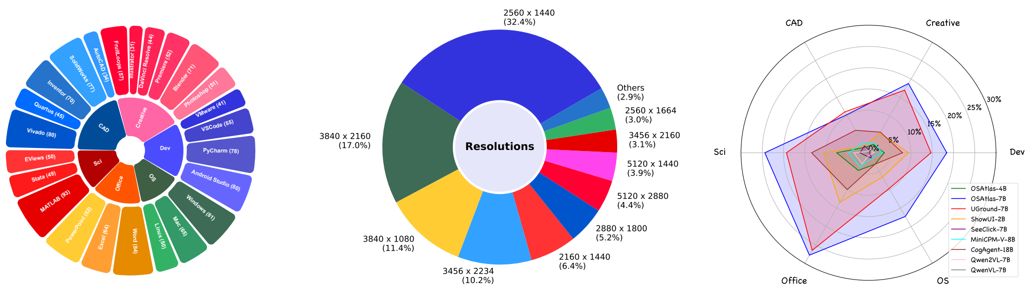
Through extensive experiments, we found that strategically narrowing the search area within an image leads to significant performance improvements. Building on this insight, we propose several baseline methods for the task, including ScreenSeekeR, an agentic framework designed as a baseline approach for GUI grounding in high-resolution environments. It leverages the inherent hierarchical structures in GUI screenshots and the rich GUI-related knowledge within the MLLM planner to guide the search process. Instead of directly identifying the target UI element, it systematically reasons over user instructions to predict the most probable regions. These regions are progressively cropped to remove irrelevant distractions, allowing the grounding model to operate on a simplified subarea of the image. With this approach, ScreenSeekeR boosts the OS-Atlas-7B [15] model's performance from 18.9% to 48.1%, achieving a 254% relative improvement. This finding, along with ScreenSpot-Pro, offers essential insights that could guide the development of future advanced GUI grounding models.




(a) ScreenSpot

(b) ScreenSpot-Pro (Ours)

Figure 2: ScreenSpot [8] (left) vs ScreenSpot-Pro (right). ScreenSpot-Pro features screenshots of the entire screen, while ScreenSpot contains unrealistic screenshots cropped to local areas. Targets are highlighted in red boxes.




(a) Software categories and the number of tasks.

(b) Resolution distribution.

(c) Results of representative GUI Grounding models across 6 categories of applications.

Figure 3: Task distribution and benchmark results of ScreenSpot-Pro.

Our contribution is summarized as follows:

- We present ScreenSpot-Pro, a novel benchmark for GUI grounding with authentic tasks collected from 26 high-resolution professional desktop environments.
- We identify key challenges in GUI grounding and introduce baseline methods performing visual search to tackle the difficulties posed by the high resolution and small relative target sizes.
- We propose ScreenSeekeR, an agentic framework for adapting existing GUI grounding models to perform visual search in high-resolution screenshots in a training-free fashion, achieving state-of-the-art performance on ScreenSpot-Pro.

## 2 Related Works

### 2.1 GUI Grounding

The aspiration to build autonomous agents that assist humans in daily tasks has long captivated researchers. Recently, Multimodal Large Language Models [18, 1, 2] have demonstrated remarkable progress in image understanding and reasoning. These advancements have greatly inspired applications in GUI agents to process both visual and linguistic inputs, allowing them to handle a wider range of tasks [19, 12, 20, 21, 22]. A fundamental aspect of GUI agents is grounding, which translates high-level plans into executable actions located on the screen. Leveraging the capabilities of MLLMs, GUI grounding models [8, 16, 15, 23] are fine-tuned on large-scale text-position pairs extracted from screenshots. This process significantly enhances their ability to align language commands with visual elements, improving the accuracy and effectiveness of GUI agents in real-world applications.

To evaluate GUI grounding abilities, previous benchmarks have primarily focused on simple tasks such as web browsing and mobile interactions. However, these benchmarks oversimplify the problem. For instance, ScreenSpot [8] facilitates artificial screenshots by cropping regions from full-screen images. ScreenSpot-v2 [15] fixes annotation errors in ScreenSpot. VisualWebBench [17] reformulates location prediction as multiple-choice questions by providing candidate targets. Furthermore, these benchmarks overlook the importance and challenge of productivity tools in professional settings. To address these limitations, we introduce ScreenSpot-Pro, a benchmark designed to provide a more rigorous evaluation of GUI grounding in high-resolution professional environments.

### 2.2 Processing High Resolution Images

Though several approaches have been proposed to tackle the challenge of processing high-resolution images in MLLMs, including resolution scaling [3] and simple cropping [24, 6], these methods struggle to perform effectively at ultra-high resolutions due to inherent model limitations, such as short context lengths and low-resolution training data. For instance, UGround [16] supports resolutions up to $1344 \times 1344$, while QwenVL [25] operates at $448 \times 448$. Further increasing input resolutions necessitates innovative model architectures and significant computational resources for retraining. An alternative approach involves utilizing visual search techniques [26, 27]. However, these methods depend on predefined splitting strategies, which constrain search flexibility and may result in missing contextual information in GUI environments. Moreover, V* [26] requires training the MLLM with an additional segmentation module to generate guidance maps, which makes it impractical for GUI tasks due to the diversity of UI functionalities, compounded by the lack of large available datasets.

## 3 ScreenSpot-Pro: Benchmarking GUI Grounding for Professional High-Resolution Computer Use

In this section, we introduce the data collection range, criteria, processing procedure, quality control measures, and provide a statistical overview of ScreenSpot-Pro.

### 3.1 Scope of Data Collection

ScreenSpot-Pro includes six distinct application genres, with a primary focus on four types of professional applications. Additionally, it features office productivity software and common operating system tasks. A detailed list of the collection can be found in Table 1. These categories include:

**Development and Programming.** Development and programming software supports the entire lifecycle of software development, from writing code to debugging and testing applications. These tools provide integrated environments that enhance productivity and collaboration, offering features like syntax highlighting, version control integration, and debugging tools. The applications in this category include **VSCode** (code editor), **PyCharm** (Python IDE), **Android Studio** (Android app development), and **Quartus** (FPGA programming). Additionally, virtualization is critical for creating scalable computing solutions and managing virtual environments, so we also include **VMware Fusion** (virtual machine management).

**Creative Software.** Creative software includes applications designed for the creation and editing of visual, audio, and video content. These tools are essential in industries such as graphic design, video production, and music composition, enabling professionals to produce high-quality media for various platforms. The tools in this category include **Photoshop** (image editing), **Premiere** (video editing), **Illustrator** (vector graphic design), **FruitLoops Studio** (music production), **DaVinci Resolve** (color grading and video editing), **Unreal Engine** (game engine and 3D simulation), and **Blender** (3D modeling and animation).

**Computer-Aided Design (CAD) and Engineering.** CAD and engineering software are used to design and model physical objects and systems. These applications are vital in fields such as engineering, architecture, and product manufacturing, where precision design and simulation are required. They enable professionals to create detailed 2D drawings, 3D models, and simulate the behavior of mechanical structures. The tools in this category include **AutoCAD** (2D/3D design), **SolidWorks** (3D CAD and simulation), **Inventor** (mechanical design), and **Vivado** (circuit design and FPGA programming).

**Scientific and Analytical.** Scientific and analytical software is designed for data analysis, numerical computation, and mathematical modeling. These applications are indispensable in fields like research, engineering, and data science, providing robust environments for analyzing large datasets, solving complex mathematical problems, and running simulations. The software in this category includes **MATLAB** (numerical computation and algorithm development), **Origin** (data analysis and scientific visualization), **Stata** (statistical analysis), and **EViews** (econometric modeling).

**Office Software.** Office software includes applications designed to facilitate productivity in tasks such as document creation, data analysis, communication, and presentation. These tools are widely used across various industries to manage workflows and support collaborative environments. Key applications in this category include **Word** (word processing), **Excel** (spreadsheets and data analysis), **PowerPoint** (presentation design).

**Operation System Commons.** Apart from professional software, ScreenSpot-Pro also includes basic operating system operations to evaluate models in high-res environments. These samples are referred to as Operating System Commons, encompassing the general use and interaction with an OS. These include file management, system utilities, etc., that are fundamental to day-to-day tasks on any OS. For this category, we include **Windows**, **macOS**, and **Linux**.

### 3.2 Collection Method and Criteria

ScreenSpot-Pro captures realistic tasks in real-world challenges across various platforms and applications. Experts with at least five years of experience using the relevant applications were invited to record the data. They were instructed to perform their regular work routine to ensure the authenticity of the tasks whenever possible. To minimize disruptions to their workflow, we developed a silently running screen capture tool, accessible through a shortcut key. When activated, this tool takes a screenshot and overlays it on the screen, allowing experts to label the bounding boxes and provide instructions directly. This method enhances the consistency and quality of the annotations, as experts can label tasks in real-time without the need to recall the purposes and context of their actions in hindsight.

To obtain authentic high-resolution images, we prioritized screens with a resolution greater than 1080p ($1920 \times 1080$), a configuration commonly found among annotators. Monitor scaling was disabled. In dual-monitor setups, images were captured to span both displays.

Following SeeClick [8], we also specify the type of the target element, categorizing it as either *text* or *icon*. We refined the classification criteria to better discriminate ambiguous cases where icons are accompanied by text labels, which is common in AutoCAD and Office suites. Specifically, a target is classified as *icon* only when no text hints are present. If text labels are present, the target is labeled as *text*, even if an icon is included.

### 3.3 Quality Control

ScreenSpot-Pro has undergone strict quality control to ensure its high-quality in three notable aspects.

**Task Validity.** Each instance in the dataset is reviewed by at least two annotators to ensure its correctness. Specifically, we removed instructions that caused ambiguity: each instruction must refer to, and only to, a single area in the image. It is also guaranteed that all instructions can be executed directly on the screenshot without requiring further actions, such as switching to other windows, opening menus, or right-clicking.

**Target Box Precision.** To ensure precise and reliable annotations, the annotations are required to tightly encompass all parts of interactable regions. For instance, the bounding box for a menu

Table 1: List of software collected in ScreenSpot-Pro.

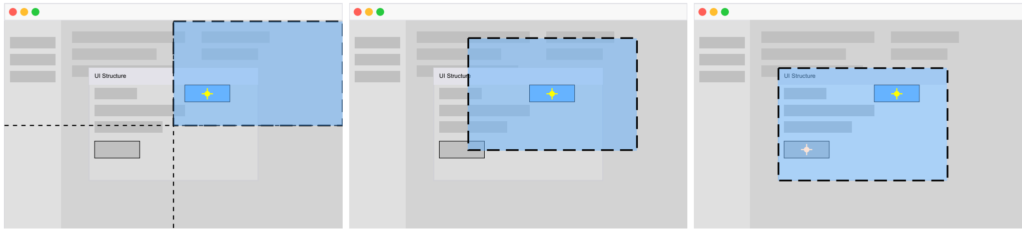
| Icon | Abbr. | Application | Edition & Version | OS | Icons | Texts |
|---|---|---|---|---|---|---|
| | | | **Development and Programming** | | | |
| | **VSC** | Visual Studio Code | 1.95 | macOS | 22 | 33 |
| | **PyC** | PyCharm | 2023.3 | macOS | 38 | 40 |
| | **AS** | Android Studio | 2022.2 | macOS | 44 | 36 |
| | **Qrs** | Quartus | II 13.0 SP1 | Windows | 32 | 13 |
| | **VM** | VMware | Fusion 13.6.1 | macOS | 9 | 32 |
| | | | **Creative** | | | |
| | **PS** | Photoshop | 2020 | Windows | 25 | 26 |
| | **PR** | Premiere | 2025 | Windows | 24 | 28 |
| | **AI** | Adobe Illustrator | 2025 | Windows | 19 | 12 |
| | **Bl** | Blender | 4.0.2 | Windows | 15 | 56 |
| | **FL** | FruitLoops Studio | 20.8.3 | Windows | 31 | 26 |
| | **UE** | Unreal Engine | 5.4.4 | Windows | 6 | 29 |
| | **DR** | DaVinci Resolve | 19.0.3 | macOS | 23 | 21 |
| | | | **CAD and Engineering** | | | |
| | **CAD** | AutoCAD | Mechanical 2019 | Windows | 7 | 27 |
| | **SW** | SolidWorks | Premium 2018 x64 | Windows | 14 | 63 |
| | **Inv** | Inventor | Professional 2019 | Windows | 11 | 59 |
| | **Vvd** | Vivado | 2018.3 | Windows | 32 | 48 |
| | | | **Scientific and Analytical** | | | |
| | **MAT** | MATLAB | R2022b | Windows | 19 | 74 |
| | **Org** | Origin | 2018 | Windows | 43 | 19 |
| | **Stt** | Stata | SE 16 | Windows | 41 | 8 |
| | **Evw** | EViews | 10 | Windows | 7 | 43 |
| | | | **Office Suite** | | | |
| | **Wrd** | Word | Office 365 (16.90) | macOS | 15 | 69 |
| | **PPT** | PowerPoint | Home and Student 2019 | Windows | 25 | 57 |
| | **Exc** | Excel | Office 365 (16.82) | macOS | 13 | 51 |
| | | | **Operating System Commons** | | | |
| | **Win** | Windows | 11 Professional | - | 47 | 34 |
| | **mac** | macOS | Sonoma 14.5 | - | 23 | 42 |
| | **Lnx** | Linux | Ubuntu 24.04 | - | 19 | 31 |

item should not only include the visible text but also extend to cover its full clickable area. This approach minimizes ambiguity in the bounding boxes, providing a more accurate representation of the elements for rigorous evaluation.

**Icon/Text Classification.** During annotation, we observed some icons accompanied by text. To standardize the criteria, we classify an element as text if there are hint labels on or around the area, even if the target is graphical.

### 3.4 ScreenSpot-Pro Statistics

Figure 3 summarizes the collected GUI data, encompassing many applications and resolutions, offering a level of diversity unmatched by previous benchmarks. The text constitute 62.6% of the



(a) Splitting (b) Narrowing (c) Structural

Figure 4: Comparison of visual search strategies used by methods.

**Algorithm 1** ScreenSeekeR

**Input:** Instruction $T$, Image $I_{\text{img}}$, Max Depth $D_{\text{max}}$, Min Size $S_{\text{min}}$
**Output:** Target Bounding Box $b$
**function** $\text{VISUALSEARCH}(T, I, D_{\text{max}}, S_{\text{min}}, d)$
  $d \leftarrow 0$, $viewport \leftarrow (0, 0, 1, 1)$
  **if** $depth \geq D_{\text{max}}$ or $I_{\text{img}}$ too small **then**
    **return** $\text{DIRECTGROUNDING}(I, viewport)$
  **end if**
  $candidates \leftarrow \text{POSITIONINFERENCE}(Y, I)$
  $patches \leftarrow \text{GROUND}(candidates)$
  $dilated\_patches \leftarrow \text{DILATE}(patches, S_{\text{min}}, R_{\text{max}})$
  $scores \leftarrow \text{SCOREPATCHES}(nms\_patches)$
  $nms\_patches \leftarrow \text{NMS}(dilated\_patches)$
  $sorted\_patches \leftarrow \text{SORT}(nms\_patches, scores)$
  **for** each $patch \in sorted\_patches$ **do**
    $sub\_image \leftarrow \text{CROPIMAGE}(I, patch)$
    $b \leftarrow \text{VISUALSEARCH}(T, sub\_image, d + 1)$
    **if** $b$ is not None **then**
      **return** $b$
    **end if**
  **end for**
  **return** `None`
**end function**

elements, with the remainder being icons. Notably, targets in ScreenSpot-Pro occupy 0.07% of the screenshot area on average, a significant reduction compared to 2.01% of ScreenSpot [8].

## 4 Baseline Methods

In this section, we present five baseline methods evaluated on the ScreenSpot-Pro benchmark. We begin with a straightforward approach that utilizes the powerful the GPT-4o [1] model to refine the instructions. Recognizing that the primary challenge stems from the high resolution of the screenshots and the small size of UI targets, we then propose planner-free visual search strategies that employ multi-round grounding to progressively reduce the search area. Lastly, we introduce ScreenSeekeR, a method that leverages guidance from a MLLM planner to further improve the search process.

### 4.1 GPT Instruction

In line with prior work such as UGround [16] and OS-Atlas [15], we leverage GPT-4o to examine the screenshot and generate a rewritten detailed instruction optimized for the grounding model. The prompt used are listed in the supplementary materials.

### 4.2 Planner-Free Visual Search Methods

**Iterative Splitting (Figure 4a).** Inspired by V*'s iterative approach [26], Iterative Splitting first performs grounding directly on the whole screenshot, and splits the screenshot into smaller patches.

At each step, it chooses the patch the prediction falls into to continue searching within. We always use a 2 row $\times$ 2 column splitting strategy.

**Iterative Narrowing (Figure 4b).** This baseline operates in the same ground-and-zoom procedure as Iterative Splitting, but the patches are cropped to center the prediction. The patch size is set to half the width and height of the image at each step. This approach closely aligns with a concurrent work [28].

**ReGround.** We assess a simple baseline that crops the region surrounding the initial prediction to re-ground and make a final determination. Comparing to Iterative Narrowing, the size of the crop is fixed and can be manually configured based on the optimal input size of the models.

### 4.3 ScreenSeekeR: An Agentic Grounding Framework

Unlike natural images, the UI of applications typically follows a well-defined hierarchy. For example, menus, tools, and properties are often organized within sub-panels or child windows, providing potential cues on where to search for a UI target (Figure 4c). Based on the observation, we propose ScreenSeekeR, adopting the idea of visual search to address the problem of GUI grounding in professional high-resolution computer screens.

The core idea behind ScreenSeekeR is to utilize the GUI knowledge of a strong planner (GPT-4o) to generate possible areas to guide the search. Given a text instruction $T$ and an image $I$, the algorithm begins the search over the entire image and progressively narrows the search area based on inferred positions. First, the planner proposes the most possible areas to search within based on the screenshot. The candidate areas are filtered and scored using the predictions of the grounder model. Then, the planner continues to search recursively or terminate if it thinks the target is found. The algorithm is summarized in Algorithm 1 and an example is visualized in Figure 5.

**Position Inference** The core of the algorithm lies in Position Inference, where GPT-4o analyzes the instruction $T$ to predict the potential locations of the target. Initially, it identifies the approximate location of the target UI and predicts a series of *areas* that likely enclose the target. It then leverages common knowledge to infer possible *neighboring* UI elements in proximity to the target. For example, a "new" button typically appears near the "delete" button. This allows the model to generate a set of candidate regions in the image that are likely to contain the target. The prompts can be found in Appendix B.

**Candidate Area Scoring** The grounded bounding boxes are often noisy, so we apply box dilation to expand smaller ones into larger candidate areas, reducing the risk of missing the target. Next, candidates are ranked based on the sum of their scores across all grounded boxes to determine the search order. Each candidate's score from a given box is computed using a predefined function that considers the distance between their center points:

$$s = \begin{cases} \exp\left(-\frac{(x'-0.5)^2+(y'-0.5)^2}{2\sigma^2}\right), & \text{if point inside} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$x' = \frac{x - x_1}{x_2 - x_1}, \quad y' = \frac{y - y_1}{y_2 - y_1} \tag{2}$$

where $(x, y)$ is the center of a voting box, and $(x_1, y_1, x_2, y_2)$ represent the coordinates of the candidate area. $\sigma$ is set to 0.3 in all experiments. Candidates with more voting boxes closer to their center receive higher scores, while those further away are assigned progressively lower scores. This centrality-based approach emulates human visual attention, and mitigates the scoring bias towards large areas, which would otherwise slow down the search process.

The candidates are then subjected to non-maximum suppression (NMS) to decrease overlapping regions. When two boxes overlap greatly, the one with a higher score is kept.

Table 2: Model Performance by Software. The abbreviations used in the table are defined in Table 1.

| Model | Development | | | | | Creative | | | | | | CAD | | | | | Scientific | | | | Office | | | OS | | | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AS | PyC | VSC | VM | UE | PS | Bl | PR | DR | AI | FL | CAD | SW | Inv | Qrs | Vvd | MAT | Org | Evw | Stt | PPT | Exc | Wrd | Lnx | mac | Win | |
| OS-Atlas-7B | 8.8 | 15.4 | 25.5 | 34.1 | 22.9 | 17.6 | 22.5 | 17.3 | 27.3 | 3.2 | 10.5 | 2.9 | 3.9 | 2.9 | 13.3 | 26.3 | 23.7 | 11.3 | 54.0 | 12.2 | 22.0 | 12.5 | 44.0 | 20.0 | 20.0 | 12.3 | 18.9 |
| UGround (7B) | 7.5 | 7.7 | 21.8 | 31.7 | 20.0 | 21.6 | 25.4 | 17.3 | 11.4 | 0.0 | 14.0 | 2.9 | 0.0 | 7.1 | 15.6 | 28.7 | 23.7 | 6.5 | 46.0 | 0.0 | 25.6 | 15.6 | 36.9 | 18.0 | 12.3 | 2.5 | 16.5 |
| AriaUI (3.9/25.3B MoE) | 0.0 | 3.8 | 21.8 | 2.4 | 0.0 | 27.5 | 26.8 | 17.3 | 2.3 | 0.0 | 12.3 | 0.0 | 1.3 | 1.4 | 20.0 | 17.5 | 21.5 | 1.6 | 44.0 | 6.1 | 6.1 | 1.6 | 36.9 | 2.0 | 3.1 | 2.5 | 11.3 |
| ShowUI (2B) | 3.8 | 7.7 | 5.5 | 22.0 | 11.4 | 5.9 | 7.0 | 5.8 | 0.0 | 3.2 | 3.5 | 0.0 | 0.0 | 1.4 | 15.6 | 5.0 | 8.6 | 12.9 | 16.0 | 6.1 | 9.8 | 6.3 | 22.6 | 4.0 | 10.8 | 4.9 | 7.7 |
| CogAgent (18B) | 2.5 | 5.1 | 16.4 | 9.8 | 2.9 | 11.8 | 7.0 | 7.7 | 0.0 | 0.0 | 5.3 | 0.0 | 1.3 | 0.0 | 11.1 | 18.8 | 16.1 | 1.6 | 34.0 | 2.0 | 6.1 | 0.0 | 21.4 | 2.0 | 4.6 | 2.5 | 7.7 |
| OS-Atlas-4B | 1.3 | 1.3 | 12.7 | 2.4 | 0.0 | 0.0 | 2.8 | 1.9 | 2.3 | 3.2 | 5.3 | 0.0 | 0.0 | 1.4 | 2.2 | 3.8 | 7.5 | 3.2 | 20.0 | 0.0 | 4.9 | 0.0 | 8.3 | 6.0 | 0.0 | 3.7 | 3.7 |
| MiniCPM-V (7B) | 0.0 | 2.6 | 9.1 | 2.4 | 0.0 | 3.9 | 0.0 | 3.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 6.7 | 11.3 | 2.2 | 1.6 | 18.0 | 0.0 | 4.9 | 0.0 | 3.6 | 0.0 | 3.1 | 3.7 | 3.0 |
| Qwen2-VL-7B | 0.0 | 0.0 | 5.5 | 0.0 | 2.9 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.8 | 0.0 | 0.0 | 0.0 | 2.2 | 1.3 | 2.2 | 0.0 | 12.0 | 2.0 | 2.4 | 0.0 | 6.0 | 2.0 | 0.0 | 0.0 | 1.6 |
| SeeClick (7B) | 0.0 | 0.0 | 0.0 | 2.4 | 0.0 | 0.0 | 1.4 | 1.9 | 0.0 | 0.0 | 0.0 | 2.9 | 0.0 | 5.7 | 0.0 | 0.0 | 0.0 | 0.0 | 8.0 | 2.0 | 0.0 | 0.0 | 2.4 | 2.0 | 1.5 | 1.2 | 1.1 |
| Qwen2-VL-72B | 0.0 | 1.3 | 1.8 | 0.0 | 0.0 | 2.0 | 1.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.3 | 1.4 | 2.2 | 0.0 | 0.0 | 0.0 | 8.0 | 4.1 | 0.0 | 0.0 | 2.4 | 0.0 | 0.0 | 1.2 | 1.0 |
| GPT-4o | 0.0 | 1.3 | 0.0 | 2.4 | 2.9 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.3 | 2.9 | 0.0 | 1.3 | 2.2 | 0.0 | 2.0 | 0.0 | 0.0 | 1.6 | 1.2 | 0.0 | 0.0 | 0.0 | 0.8 |
| QwenVL-7B | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |

Table 3: Performance breakdown of various models across application categories on ScreenSpot-Pro.

| Model | Development | | | Creative | | | CAD | | | Scientific | | | Office | | | OS | | | Avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Text | Icon | Avg | Text | Icon | Avg | Text | Icon | Avg | Text | Icon | Avg | Text | Icon | Avg | Text | Icon | Avg | Text | Icon | Avg |
| OSAtlas-7B | 33.1 | 1.4 | 17.7 | 28.8 | 2.8 | 17.9 | 12.2 | 4.7 | 10.3 | 37.5 | 7.3 | 24.4 | 33.9 | 5.7 | 27.4 | 27.1 | 4.5 | 16.8 | 28.1 | 4.0 | 18.9 |
| UGround (7B) | 26.6 | 2.1 | 14.7 | 27.3 | 2.8 | 17.0 | 14.2 | 1.6 | 11.1 | 31.9 | 2.7 | 19.3 | 31.6 | 11.3 | 27.0 | 17.8 | 0.0 | 9.7 | 25.0 | 2.8 | 16.5 |
| AriaUI (3.9/25.3B MoE) | 16.2 | 0.0 | 8.4 | 23.7 | 2.1 | 14.7 | 7.6 | 1.6 | 6.1 | 27.1 | 6.4 | 18.1 | 20.3 | 1.9 | 16.1 | 4.7 | 0.0 | 2.6 | 17.1 | 2.0 | 11.3 |
| CogAgent (18B) | 14.9 | 0.7 | 8.0 | 9.6 | 0.0 | 5.6 | 7.1 | 3.1 | 6.1 | 22.2 | 1.8 | 13.4 | 13.0 | 0.0 | 10.0 | 5.6 | 0.0 | 3.1 | 12.0 | 0.8 | 7.7 |
| ShowUI (2B) | 16.9 | 1.4 | 9.4 | 9.1 | 0.0 | 5.3 | 2.5 | 0.0 | 1.9 | 13.2 | 7.3 | 10.6 | 15.3 | 7.5 | 13.5 | 10.3 | 2.2 | 6.6 | 10.8 | 2.6 | 7.7 |
| OSAtlas-4B | 7.1 | 0.0 | 3.7 | 3.0 | 1.4 | 2.3 | 2.0 | 0.0 | 1.5 | 9.0 | 5.5 | 7.5 | 5.1 | 3.8 | 4.8 | 5.6 | 0.0 | 3.1 | 5.0 | 1.7 | 3.7 |
| MiniCPM-V (7B) | 7.1 | 0.0 | 3.7 | 2.0 | 0.0 | 1.2 | 4.1 | 1.6 | 3.4 | 8.3 | 0.0 | 4.7 | 2.8 | 3.8 | 3.0 | 3.7 | 1.1 | 2.6 | 4.5 | 0.7 | 3.0 |
| Qwen2-VL-7B | 2.6 | 0.0 | 1.3 | 1.5 | 0.0 | 0.9 | 0.5 | 0.0 | 0.4 | 6.3 | 0.0 | 3.5 | 3.4 | 1.9 | 3.0 | 0.9 | 0.0 | 0.5 | 2.5 | 0.2 | 1.6 |
| SeeClick (7B) | 0.6 | 0.0 | 0.3 | 1.0 | 0.0 | 0.6 | 2.5 | 0.0 | 1.9 | 3.5 | 0.0 | 2.0 | 1.1 | 0.0 | 0.9 | 2.8 | 0.0 | 1.5 | 1.8 | 0.0 | 1.1 |
| GPT-4o | 1.3 | 0.0 | 0.7 | 1.0 | 0.0 | 0.6 | 2.0 | 0.0 | 1.5 | 2.1 | 0.0 | 1.2 | 1.1 | 0.0 | 0.9 | 0.0 | 0.0 | 0.0 | 1.3 | 0.0 | 0.8 |
| QwenVL-7B | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.7 | 0.0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.1 |

**Recursive Search** The algorithm recursively searches each candidate area by cropping out a sub-image, which is passed into the recursive search function, $VisualSearch(I, sub\_image, d{+}1)$. The grounder model is invoked if the patch size is sufficiently small (a hyperparameter set to 1280 pixels), and the planner verifies the correctness of the bounding box. This recursive process continues until the planner determines that the target has been found or until the maximum search depth is reached.

## 5 Experiments

With ScreenSpot-Pro, we rigorously evaluate the correctness whether the model's predictions fall into the annotated ground truth boxes. For models inferencing boxes, we consider the center point of the generated box as the prediction.

**End-to-end Models.** We conduct the experiments on several MLLMs that support GUI Grounding: QwenVL-7B [25], Qwen2VL-7B [2], MiniCPM-V-2.6 (8B) [29], CogAgent (18B) [1] [6], SeeClick (7B) [8], UGround (7B) [16], OSAtlas-4B, OSAtlas-7B [15], ShowUI (2B) [30] and Aria-UI (Mixture of Experts, 3.9B active) [23]. We handle the varying formats of the location outputs to ensure a fair comparison across models.

**Baseline Methods.** We compare the five baseline methods introduced in Section 4 with OS-Atlas-7B [15] as the grounding model, and GPT-4o [1] as the planner if applicable. The number of iterations in Iterative Splitting and Iterative Narrowing are both set to 3 following **(author?)** [28] for a fair comparison.

### 5.1 Results of End-to-End Models

**Models struggle on ScreenSpot-Pro, even the specialist models** The full results of the GUI grounding models are presented in Table 2. OS-Atlas-7B leads the performance with an accuracy of 18.9%, closely followed by UGround and AriaUI. None of the other models achieved an accuracy above 10%. Notably, GPT-4o scored only 0.8%, highlighting its limitations for the GUI grounding task despite its strong understanding capability.

**Icons targets are more difficult to ground than texts** Table 3 demonstrates that the benchmarked models struggle significantly in identifying and grounding icon elements in the GUI, a consistent

[1] THUDM/cogagent-chat-hf

finding with **(author?)** [8]. The challenge is exacerbated by the professional applications, as they may feature an extensive number of icons as a result of the complex functionality, e.g. Origin's toolbar (see Figure 3 in the Appendix). Moreover, the icons carry unique meanings within professional contexts that are rarely encountered in the web data, on which many models are primarily trained.

## 5.2 Results of Baseline Methods

Table 4: Comparison of methods on ScreenSpot-Pro with OS-Atlas-7B.

| Model | Dev | Creative | CAD | Scientific | Office | OS | Overall Text | Overall Icon | Overall Avg |
|---|---|---|---|---|---|---|---|---|---|
| OS-Atlas-7B | 17.7 | 17.9 | 10.3 | 24.4 | 27.4 | 16.8 | 28.1 | 4.0 | 18.9 |
| GPT-4o Instruction | 19.7 | 19.6 | 10.7 | 32.3 | 27.4 | 15.3 | 30.2 | 5.6 | 20.8 |
| Iterative Splitting | 33.1 | 27.3 | 23.8 | 25.2 | 43.9 | 36.2 | 43.5 | 10.8 | 31.0 |
| Iterative Narrowing | 34.4 | 27.3 | 20.3 | 29.5 | 40.9 | **43.9** | 43.5 | 13.1 | 31.9 |
| ReGround | **37.5** | **38.1** | **33.3** | **37.8** | **59.1** | 37.8 | **55.7** | **15.1** | **40.2** |
| w/o Recursive Search | 40.8 | 35.5 | 33.3 | 44.5 | 58.7 | 43.4 | 51.8 | 16.2 | 41.9 |
| w/o Neighbor Inference | 46.8 | 41.6 | 33.3 | 44.9 | 63.0 | **53.6** | 62.4 | 20.4 | 46.4 |
| w/o Patch Scoring | 48.5 | **42.8** | 34.1 | **47.6** | 61.3 | 50.0 | 63.3 | 20.2 | 46.8 |
| ScreenSeekeR | **49.8** +32.1 | 41.9 +24.0 | **37.9** +27.6 | 47.2 +22.8 | **64.3** +36.9 | 52.0 +35.2 | **64.1** +36.0 | **22.4** +18.4 | **48.1** +29.2 |

**ReGround achieves the best result among planner-free methods.** The results of baseline models are listed in Figure 4. Interestingly, the simplest baseline ReGround achieved the highest performance with OS-Atlas-7B, reaching 40.2%. Iterative Narrowing slightly outperformed Iterative Focusing, likely due to its superior image-splitting strategy handling targets near the center of the screenshot without cutting them off.

Table 5: Performance comparison of different planner models and grounding models in the ScreenSeekeR algorithm on ScreenSpot-Pro.

| Planner | Grounder | Text | Icon | Avg |
|---|---|---|---|---|
| Qwen2-VL-72B | UGround (7B) | 33.7 | 9.6 | 24.5 |
| | OS-Atlas-7B | 35.7 | 8.3 | 26.0 |
| GPT-4o | UGround (7B) | 56.2 | 15.2 | 40.5 |
| | OS-Atlas-7B | **64.1** | **22.4** | **48.1** |

**ScreenSeekeR achieves SOTA on ScreenSpot-Pro** Table 5 demonstrates the superior performance of ScreenSeekeR on ScreenSpot-Pro. While the base model, OS-Atlas-7B, achieves only 18.9% accuracy, and explaining instructions with GPT-4o results in only a 1.9% improvement, our method significantly boosts its accuracy to an impressive 48.1% without any additional training. These results highlight that the primary bottleneck lies in the grounding model. With a proper design, models with strong screenshot understanding capabilities, even if not specifically optimized for grounding, can still be leveraged to significantly improve grounding performance.

**ScreenSeekeR generates intuitive and explainable search traces** As shown in the case study in Figure 1, given the task of "delete file or folder", the plain model completely fails and ReGround was misled into grounding the file tab in the background VSCode window, as its initial grounding attempt was too far away from the ground truth. In contrast, ScreenSeekeR not only successfully grounds the target UI but also generates a natural search trajectory. It first focuses on the open Explorer window, then searches the top action bar before identifying the target, closely aligning with a human user's thought process. This feature is not only effective but also makes it possible to interpret the model, as it provides a clear and understandable explanation of the search process.

## 5.3 Ablation studies

**Ablations on the crop size of ReGround** Table 6 examines the impact of crop size in ReGround on the two top-performing models, OS-Atlas-7B and UGround (7B). Both models exhibit peak performance within specific resolution ranges, with performance declining as image sizes deviate. OS-Atlas-7B achieves its best

Table 6: Ablation of crop size in ReGround.

| Crop Size | $512 \times 512$ | $768 \times 768$ | $1024 \times 1024$ | $1280 \times 1280$ |
|---|---|---|---|---|
| OS-Atlas-7B | 25.1 | 34.2 | **40.2** | 40.1 |
| UGround (7B) | 27.0 | **28.8** | 28.2 | 26.3 |

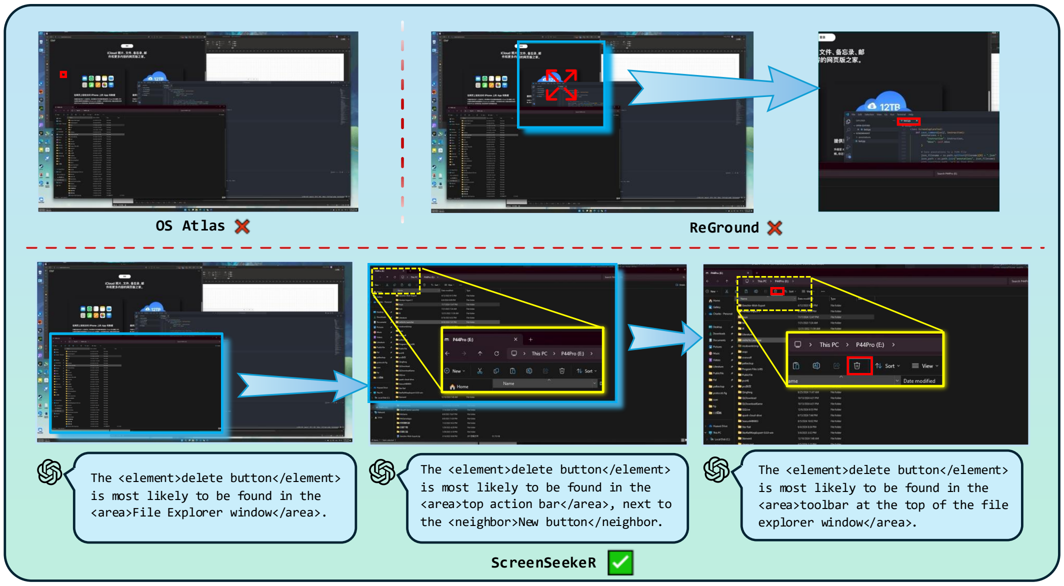



Figure 5: A case study comparing ScreenSeekeR (bottom) with the plain model prediction (top left) and ReGround (top right). The task is "delete file or folder". Grounding results are marked in the same color as the text references under the screenshots. Final results are drawn in red boxes.

score with 1024×1024 crops, while UGround performs optimally with 768×768 crops. This behavior is expected: when images are too small, crucial context is lost [28], whereas images that are too large exceed the model's processing capacity.

**Ablation on key designs of ScreenSeekeR** To evaluate the impact of each key component, we conducted ablation studies on ScreenSeekeR. In the bottom part of Table 4, we show that removing subsequent searches and retaining only the first planner decision led to the most significant performance drop, reducing accuracy to 41.9%. When neighbor inference is ablated, limiting the planner to only identifying the target's location, performance decreased slightly by 1.7%. Additionally, substituting the patch scoring method with a simple majority vote strategy resulted in a performance drop to 46.8%. These results underscore the crucial role each design element plays in the effectiveness of ScreenSeekeR.

**Ablation of planner and grounder of ScreenSeekeR** We study the impact of different planner and grounder models in ScreenSeekR in Table 5. Given the absence of planner models specifically trained for GUI visual search tasks, we include Qwen2-VL-72B [2] as a representative comparison. Our analysis reveals that Qwen2-VL-72B struggles with interpreting GUI screenshots, often producing ambiguous references such as "other tools" and "icons," which lack actionable specificity. Despite this limitation, it still outperforms the two standalone grounder models by 8.0% and 7.1%, respectively. Incorporating GPT-4o as the planner significantly enhances performance, with OS-Atlas demonstrating a larger margin over UGround in this configuration.

### 5.4 Error Analysis

We randomly sampled 78 examples (3 per application) with three baseline methods, and manually grouped the errors into four categories: *Misled by Icon* and *Misled by Text*, where the model selects other elements instead of the actual target; *Near Miss*, where the prediction is close (within 3× the size of the ground-truth box); *Random Guess*, where the output lacks clear relevance; and *Not Found* where the model fails to find the target.

The results, shown in Figure 6, underscore the need for more precise grounder models, as all methods exhibited frequent near misses. Enhancing GUI understanding and instruction-following capabilities is also critical. Many errors stem from misinterpreting visually or semantically similar elements. For example, the model selects the word "tool" instead of the specific tool mentioned in the instruction.

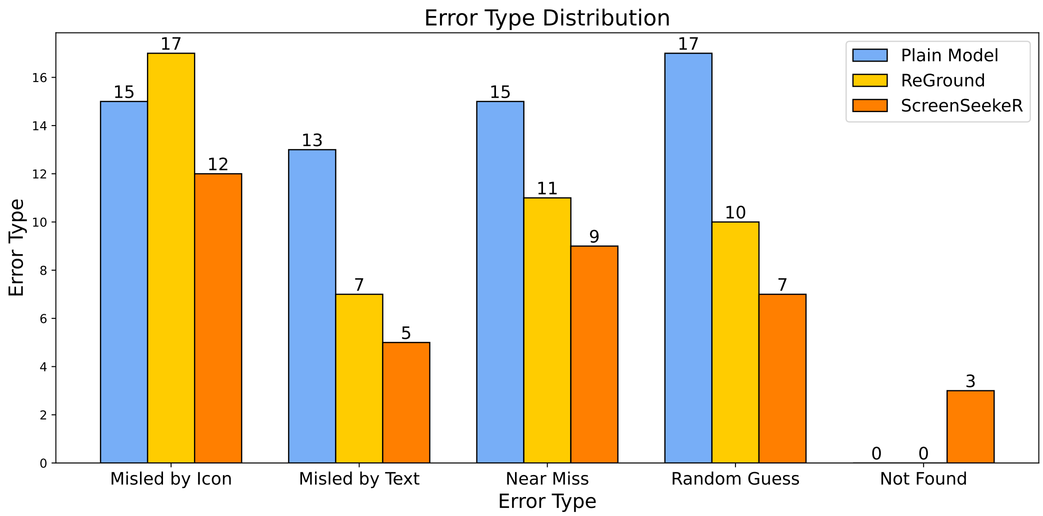



Figure 6: Error distribution with OS-Atlas-7B as the grounder.

A significant portion of mistakes also arose from incorrect icon recognition, suggesting limited domain knowledge in current MLLMs. Among the methods, ScreenSeekR achieved the lowest error rates across all categories except for "Not Found", where the planner fails to locate the target and terminates the search. In contrast, the other two methods always produce a prediction, but this often results in a higher number of "Random Guess" errors.

## 6 Conclusion

The growing capabilities of MLLMs present new opportunities for GUI grounding, yet existing models struggle with the unique challenges of high-resolution interfaces. We introduced ScreenSpot-Pro, a benchmark that rigorously evaluates GUI grounding in complex professional environments. Our evaluation showed that current models perform poorly, highlighting the need for better strategies. Inspired by our findings, we proposed several visual search baseline models, including ScreenSeekeR, an agentic framework that enhances accuracy by refining the search space, achieving a substantial performance boost without additional training. ScreenSpot-Pro has the potential to shift the research focus from grounding simplistic tasks to more realistic scenarios where agents must interpret and interact with an entire screen, powering development of practical GUI agent tools.

## Acknowledgement

We would like to thank the annotators for their valuable contributions to this work. The names are listed in alphabetical order by last name: Du He, Xiufeng Huang, Han Li, Xiangguo Li, Zehao Li, Ziyuan Luo, Yuling Ma, Xiangrui Yan, and Molei Zhou.

## References

[1] OpenAI. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`, 2024. Accessed: 2024-11-01.

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

[3] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023.

[4] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

[5] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms, 2024.

[6] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023.

[7] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, pages 9466–9482. PMLR, 2022.

[8] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents, 2024.

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

[10] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.

[11] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

[12] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.

[13] Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024.

[14] 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. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

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

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

[17] Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding? *arXiv preprint arXiv:2404.05955*, 2024.

[18] OpenAI. Gpt-4v(ision) system card. `https://cdn.openai.com/papers/GPTV_System_Card.pdf`, 2023. Accessed: 2024-02-03.

[19] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*, 2024.

[20] Meng Ziyang, Yu Dai, Zezheng Gong, Shaoxiong Guo, Minglong Tang, and Tongquan Wei. VGA: Vision GUI assistant - minimizing hallucinations through image-centric fine-tuning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1261–1279, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[21] Izzeddin Gur, Hiroki Furuta, Austin V. Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. In *ICLR*, 2024.

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

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

[24] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024.

[25] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 1(2):3, 2023.

[26] Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. *arXiv preprint arXiv:2312.14135*, 2023.

[27] Wenbin Wang, Liang Ding, Minyan Zeng, Xiabin Zhou, Li Shen, Yong Luo, and Dacheng Tao. Divide, conquer and combine: A training-free framework for high-resolution image perception in multimodal large language models. *arXiv preprint arXiv:2408.15556*, 2024.

[28] Anthony Nguyen. Improved gui grounding via iterative narrowing. *arXiv preprint arXiv:2411.13591*, 2024.

[29] Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*, 2024.

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

[31] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Table 7: Performance of GUI grounding models with Chinese instructions. The abbreviations used in the table are defined in Table 1 in the main text.

| Model | Development | | | | | Creative | | | | | | CAD | | | | | Scientific | | | | Office | | | OS | | | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AS | PyC | VSC | VM | UE | PS | Bl | PR | DR | AI | FL | CAD | SW | Inv | Qrs | Vvd | MAT | Org | Evw | Stt | PPT | Exc | Wrd | Lnx | mac | Win | |
| OS-Atlas-7B | 11.3 | 15.4 | 21.8 | 34.1 | 22.9 | 11.8 | 23.9 | 21.2 | 11.4 | 6.5 | 14.0 | 5.9 | 3.9 | 2.9 | 8.9 | 23.8 | 14.0 | 11.3 | 44.0 | 12.2 | 17.1 | 10.9 | 36.9 | 16.0 | 16.9 | 14.8 | 16.8 |
| AriaUI (MOE, 3.9B/25.3B) | 0.0 | 3.8 | 18.2 | 2.4 | 0.0 | 23.5 | 12.7 | 11.5 | 0.0 | 0.0 | 10.5 | 0.0 | 0.0 | 0.0 | 13.3 | 18.8 | 19.4 | 1.6 | 52.0 | 6.1 | 2.4 | 0.0 | 20.2 | 2.0 | 6.2 | 2.5 | 9.0 |
| UGround (7B) | 3.8 | 2.6 | 10.9 | 14.6 | 8.6 | 9.8 | 11.3 | 3.8 | 9.1 | 3.2 | 7.0 | 0.0 | 0.0 | 4.3 | 6.7 | 12.5 | 10.8 | 4.8 | 30.0 | 2.0 | 12.2 | 4.7 | 6.0 | 12.0 | 7.7 | 3.7 | 7.7 |
| ShowUI (2B) | 3.8 | 6.4 | 5.5 | 22.0 | 5.7 | 7.8 | 4.2 | 3.8 | 0.0 | 0.0 | 3.5 | 5.9 | 2.6 | 1.4 | 15.6 | 7.5 | 9.7 | 11.3 | 18.0 | 10.2 | 9.8 | 1.6 | 8.3 | 4.0 | 10.8 | 6.2 | 7.0 |
| CogAgent (18B) | 0.0 | 5.1 | 10.9 | 4.9 | 0.0 | 5.9 | 5.6 | 5.8 | 0.0 | 3.2 | 3.5 | 0.0 | 1.3 | 0.0 | 6.7 | 5.0 | 7.5 | 1.6 | 14.0 | 2.0 | 1.2 | 0.0 | 2.4 | 4.0 | 3.1 | 2.5 | 3.7 |
| OS-Atlas-4B | 0.0 | 1.3 | 7.3 | 0.0 | 0.0 | 2.0 | 2.8 | 0.0 | 4.5 | 0.0 | 7.0 | 5.9 | 0.0 | 1.4 | 0.0 | 3.8 | 5.4 | 4.8 | 12.0 | 0.0 | 4.9 | 1.6 | 2.4 | 4.0 | 0.0 | 2.5 | 2.8 |
| MiniCPM-V (7B) | 1.3 | 2.6 | 3.6 | 0.0 | 0.0 | 0.0 | 0.0 | 1.9 | 0.0 | 0.0 | 3.5 | 0.0 | 1.3 | 0.0 | 4.4 | 8.8 | 0.0 | 0.0 | 28.0 | 0.0 | 3.7 | 3.1 | 0.0 | 0.0 | 1.5 | 2.5 | 2.5 |
| Qwen2-VL-7B | 0.0 | 0.0 | 3.6 | 0.0 | 0.0 | 2.0 | 1.4 | 3.8 | 0.0 | 0.0 | 1.8 | 0.0 | 0.0 | 0.0 | 4.4 | 1.3 | 2.2 | 1.6 | 22.0 | 6.1 | 2.4 | 0.0 | 2.4 | 0.0 | 1.5 | 0.0 | 2.0 |
| GPT-4o | 2.5 | 0.0 | 0.0 | 0.0 | 2.9 | 2.0 | 1.4 | 3.8 | 0.0 | 0.0 | 0.0 | 0.0 | 2.6 | 1.4 | 0.0 | 0.0 | 2.2 | 0.0 | 2.0 | 0.0 | 1.2 | 0.0 | 0.0 | 0.0 | 1.5 | 0.0 | 0.9 |
| SeeClick (7B) | 0.0 | 2.6 | 0.0 | 0.0 | 0.0 | 0.0 | 2.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.3 | 0.0 | 0.0 | 1.1 | 0.0 | 8.0 | 0.0 | 1.2 | 0.0 | 1.2 | 0.0 | 1.5 | 0.0 | 0.9 |
| QwenVL-7B | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.5 | 0.0 | 0.2 |

# A ScreenSpot-Pro-CN

In many real-world settings, especially in regions where English is not the primary language, users often switch between their native language and English while performing digital tasks. This creates a need for GUI agents that can understand and operate across multiple languages without losing context or functionality. To reflect this multilingual reality, each task in our benchmark includes instructions in both English and Chinese. The Chinese instructions were initially generated using GPT-4 [31], then carefully reviewed and refined by the authors to ensure the meaning, tone, and intent matched the English version. This bilingual setup allows us to test how well agents handle language-switching scenarios, ensuring they remain useful and accurate across diverse linguistic environments.

We observed that Chinese instructions consistently present greater challenges for current models. As detailed in Table 7, the majority of models experienced a notable decline in performance when evaluated with Chinese instructions compared to their English counterparts. For example, the state-of-the-art OS-Atlas-7B [15] achieved only 16.8% accuracy under Chinese instructions, 2.1% lower than its performance on English instructions. Among all models, UGround-7B [16] exhibited the most dramatic decrease, falling from 16.4% to just 7.7%, which underscores its limited capacity to generalize across languages. Interestingly, GPT-4o [1] and QwenVL-7B [25] were exceptions to this trend, showing slight improvements in their Chinese performance. However, these gains are marginal and do not meaningfully impact their overall low accuracy, suggesting that effective multilingual understanding remains an open challenge for most current models.

# B Prompts

The prompts used in this work are listed in Table 8, 9 and 10.

# C ScreenSpot-Pro vs. ScreenSpot-v2

In this section, a further comparison of our proposed ScreenSpot-Pro against the ScreenSpot [15] benchmark is provided. We select its v2 version because it fixed many incorrect annotations from the original version. Figure 7 compares grounder model performance on these two benchmarks: ScreenSpot-v2[15] on the x axis and the our ScreenSpot-Pro on the y axis. Each point represents a model, with its size reflecting the model's parameter count. While some models like OSAtlas-7B[15] and UGround (7B) [16] perform well on both benchmarks, others such as ShowUI (2B)[30] and OSAtlas-4B [15] show significant discrepancies in performance, excelling in ScreenSpot-v2 but underperforming in ScreenSpot-Pro. Specifically, SeeClick is a fine-tuned version of Qwen2-VL-7B [2]. While its performance on ScreenSpot-v2 increases by around 15%, the accuracy on ScreenSpot-Pro drops. Interestingly, Qwen2-VL-72B [2] performs worse than Qwen2-VL-7B despite its 10 times larger size. These variations underscore the distinct nature of the tasks represented in ScreenSpot-Pro.

# D Data Examples

We showcase some examples from the ScreenSpot-Pro benchmark in Figure 8 and 9.

Table 8: Position Inference Prompt

**Position Inference Prompt**

I want to identify a UI element that best matches my instruction. Please help me determine which region(s) of the screenshot to focus on and list the UI elements that might appear next to the target.
If the target does not exist in the screenshot, please output ”No target”.
Output Requirements:
1. List the possible regions in descending order of probability.
2. Always make specific, clear and unique references to avoid ambiguity. References such as ”Other icons” and ”window” are NOT allowed.
3. Use the following XML tags to describe items in the screenshot:
- ¡element¿: Wrap a specific UI element.
- ¡area¿: Describe an area of the UI containing multiple elements.
- ¡neighbor¿: Describe a UI element that may appear around the target.
Example Output:
The ¡element¿shortcut link¡/element¿ is most likely to be found in the ¡area¿Settings window¡/area¿, in the ¡area¿tools panel¡/area¿, next to the ¡neighbor¿Search button¡/neighbor¿.
Important Notes:
- The target UI element is guaranteed to be present in the screenshot.
Do not speculate about operations that could change the screenshot.
Instruction:
{instruction}

Table 9: Result checking Prompt

**Result Checking Prompt**

You are given a cropped screenshot. Your task is to evaluate whether the marked element in the red box matches the target described in my instruction.
Please follow these steps:
1. Analyze the screenshot by describing its visible content and functionalities.
2. Determine which of the following applies:
- ’is_target’: The marked element is the target.
- ’target_elsewhere’: The marked element is not the target, but it exists elsewhere.
- ’target_not_found’: The marked element is not the target, and it does not exist.
3. If the target exists, rewrite the instruction to make it clearer.
After your analysis, provide the result in JSON format:
- ”result”: (str) One of ’is_target’, ’target_elsewhere’, or ’target_not_found’.
- ”new_instruction”: (str, default null) A clearer version of the instruction.
Here is my instruction:
{instruction}

Table 10: GPT Instruction Refinement Prompt

Element Description Prompt

**Task Description**
You are an excellent agent for mobile, web, and desktop navigation tasks.
Describe the target element for this task based on the provided screenshot:
**Task:** {task}
—
**Element Description Requirements**
- Provide a concise description of the element you want to operate.
- Ensure your description is both concise and complete, covering all the necessary information in less than 30 words, and organized into one sentence.
- If you find identical elements, specify their location and details to differentiate them from others.
—
**Output Format**
Your output should only include the element description itself and follow the requirements.
Do **not** start with “the target element” or “the element”.

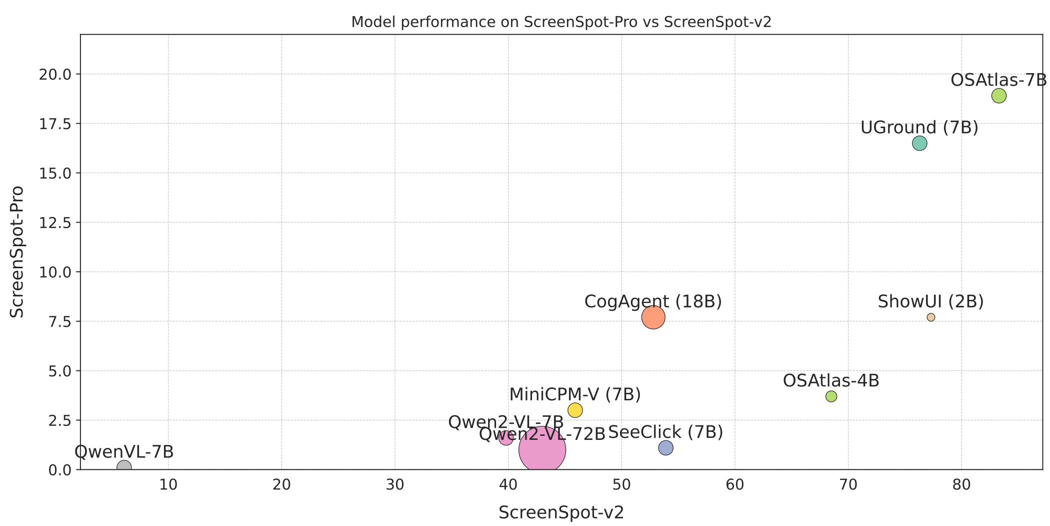
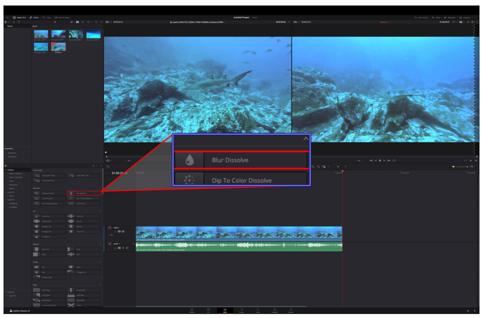
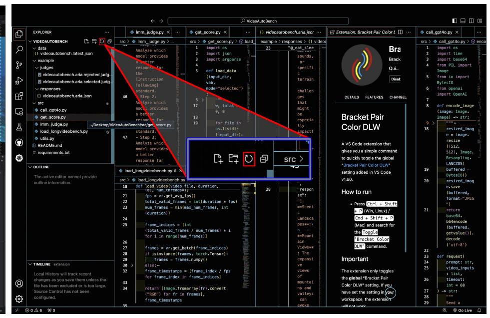



Figure 7: Model performance on ScreenSpot-Pro vs. ScreenSpot-v2. The circle sizes represent the size of the models.



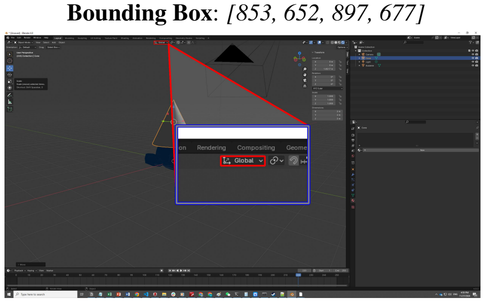

**Instruction**: *Blur Dissolve.*
**Application**: *davinci*
**Type**: *icon*
**Bounding Box**: *[460, 1344, 709, 1375]*



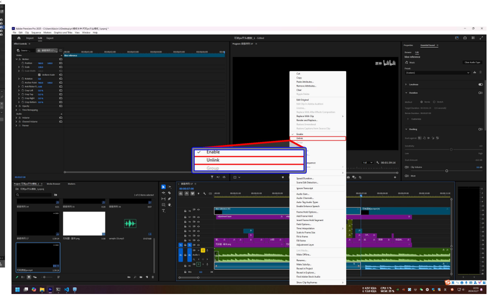

**Instruction**: *Refresh the file explorer.*
**Application**: *vscode*
**Type**: *icon*
**Bounding Box**: *[473, 183, 503, 219]*

**Instruction**: *choose chord type for 1.*
**Application**: *fruitloops*
**Type**: *text*
**Bounding Box**: *[853, 652, 897, 677]*


**Instruction**: *Execute Python scripts.*
**Application**: *unreal engine*
**Type**: *text*
**Bounding Box**: *[246, 2035, 377, 2054]*

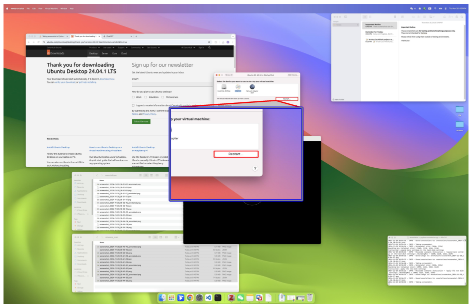



**Instruction**: *Change the coordinate mode of the object.*
**Application**: *blender*
**Type**: *icon*
**Bounding Box**: *[803, 54, 882, 71]*

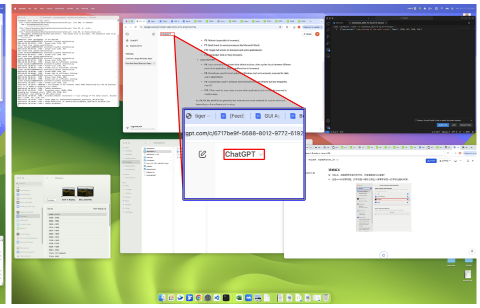



**Instruction**: *unlink audio and video.*
**Application**: *premiere*
**Type**: *text*
**Bounding Box**: *[1499, 592, 1801, 613]*

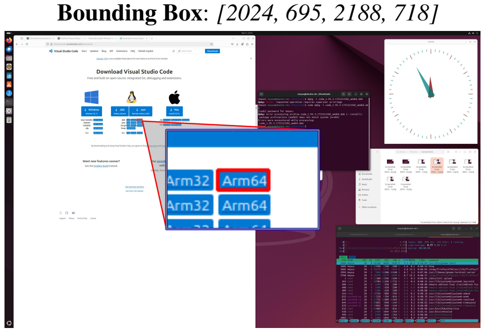
Figure 8: Examples of tasks in ScreenSpot-Pro.



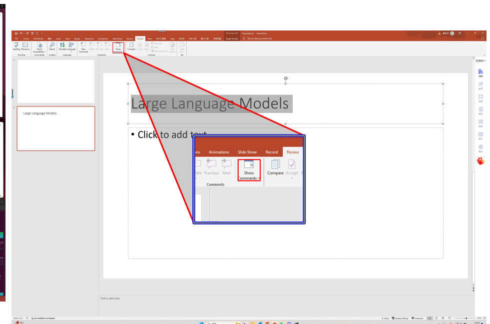
**Instruction**: *restart from CD.*
**Application**: *VMWare*
**Type**: *text*
**Bounding Box**: *[2024, 695, 2188, 718]*



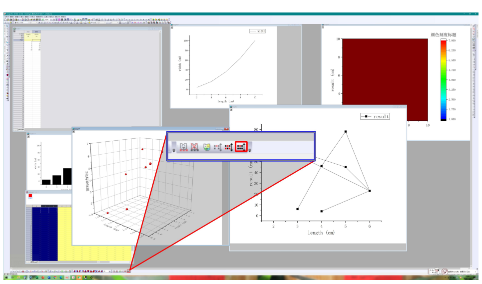

**Instruction**: *Change model.*
**Application**: *macOS*
**Type**: *text*
**Bounding Box**: *[1109, 211, 1209, 236]*



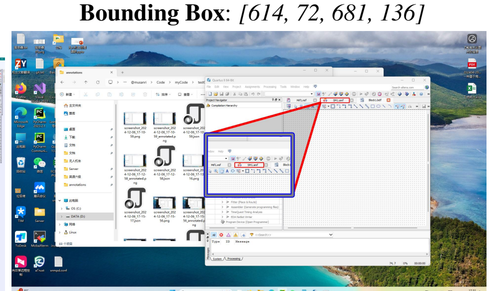

**Instruction**: *select the correct deb package to download according to the error message in the terminal.*
**Application**: *linux common*
**Type**: *text*
**Bounding Box**: *[960, 639, 1001, 655]*




**Instruction**: *Show comments.*
**Application**: *powerpoint*
**Type**: *text*
**Bounding Box**: *[614, 72, 681, 136]*



**Instruction**: *disable masking.*
**Application**: *origin*
**Type**: *icon*
**Bounding Box**: *[998, 2078, 1021, 2097]*



**Instruction**: *select the SM1.smf file in Quartus window.*
**Application**: *quartus*
**Type**: *text*
**Bounding Box**: *[1248, 270, 1365, 289]*

Figure 9: More examples of tasks in ScreenSpot-Pro.

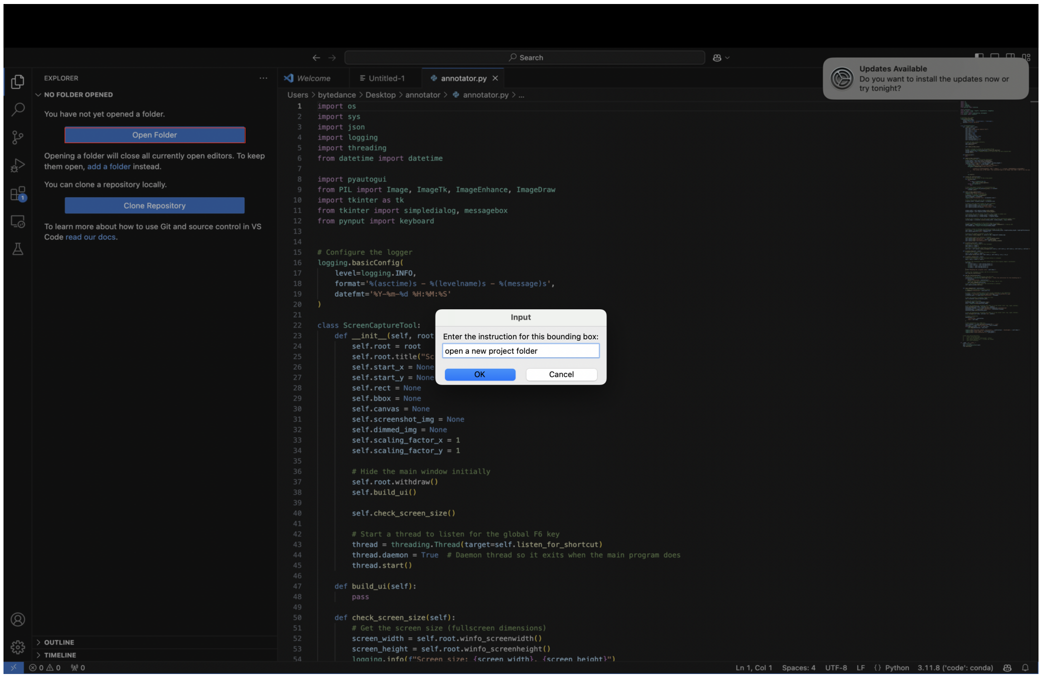



Figure 10: An example of the annotation tool. When activated, the tool captures a screenshot and overlays it on the screen, allowing experts to drag to label the bounding box (the red box around "Open Folder") and input the instruction in the popup dialog directly.