

Navigating the Digital World as Humans Do: UNIVERSAL VISUAL GROUNDING FOR GUI AGENTS

Anonymous authors

Paper under double-blind review

ABSTRACT

Multimodal large language models (MLLMs) are transforming the capabilities of graphical user interface (GUI) agents, facilitating their transition from controlled simulations to complex, real-world applications across various platforms. However, the effectiveness of these agents hinges on the robustness of their grounding capability. Current GUI agents predominantly utilize text-based representations such as HTML or accessibility trees, which, despite their utility, often introduce noise, incompleteness, and increased computational overhead. In this paper, we advocate a human-like embodiment for GUI agents that perceive the environment entirely visually and directly take pixel-level operations on the GUI. The key is visual grounding models that can accurately map diverse referring expressions of GUI elements to their coordinates on the GUI across different platforms. We show that a simple recipe, which includes web-based synthetic data and slight adaptation of the LLaVA architecture, is surprisingly effective for training such visual grounding models. We collect the largest dataset for GUI visual grounding so far, containing 19M GUI elements and their referring expressions over 1.3M screenshots, and use it to train UGround, a strong universal visual grounding model for GUI agents. Empirical results on six benchmarks spanning three categories (grounding, offline agent, and online agent) show that 1) UGround substantially outperforms existing visual grounding models for GUI agents, by up to 20% absolute, and 2) agents with UGround outperform state-of-the-art agents, despite the fact that existing agents use additional text-based input while ours only uses visual perception. These results provide strong support for the feasibility and promises of GUI agents that navigate the digital world as humans do.

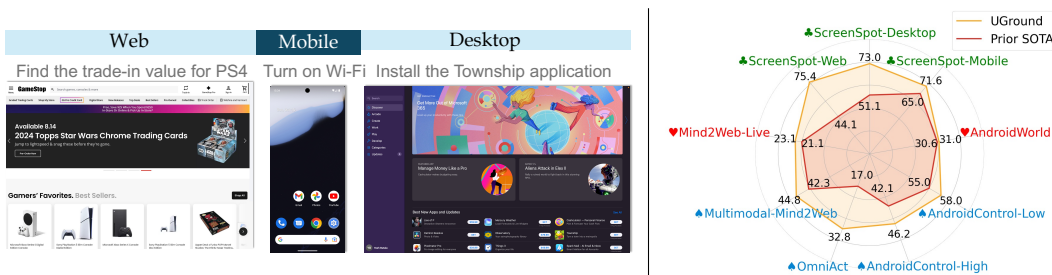


Figure 1: Examples of agent tasks across platforms and performance on GUI grounding (♣: ScreenSpot), offline agent (♠: Multimodal-Mind2Web, AndroidControl, and OmniAct), and online agent benchmarks (♥: Mind2Web-Live and AndroidWorld) when using GPT-4 as the planner.

1 INTRODUCTION

GUI (graphical user interface) agents, which are autonomous agents acting in the digital world via operating on GUIs, have been rapidly co-evolving with large language models (LLMs). On the one hand, the general multimedia understanding and generation capability of (multimodal) LLMs empower GUI agents to generalize beyond simple simulated settings (Shi et al., 2017; Humphreys et al., 2022) to diverse and complex real-world environments, including the web (Deng et al., 2023;

Zhou et al., 2024; Yao et al., 2022), desktop (Xie et al., 2024; Wu et al., 2024) and mobile operating systems (Rawles et al., 2023; Yan et al., 2023; Rawles et al., 2024). On the other hand, GUI agents have become an important testbed for LLMs, providing both the necessary breadth and depth for driving continued development as well as a pathway to many commercially viable automation applications.

Most humans perceive the digital world visually and act via keyboards, mice, or touchscreens. In principle, the embodiment of a GUI agent should already be *complete* if it can 1) visually perceive the GUI renderings, and 2) have effectors equivalent to a keyboard for typing and equivalent to a mouse or touchscreen for pixel-level operations like clicking and hovering.¹ However, current GUI agents assume more than that. For perception, most current agents rely on reading the underlying text-based representations such as HTML or accessibility (a11y) trees (Deng et al., 2023; Gur et al., 2024; Zhou et al., 2024).² Only with the recent advances in multimodal LLMs (MLLMs) does visual perception become broadly viable, but text-based representations are still used jointly (Zheng et al., 2024; Koh et al., 2024; Zhang et al., 2024a). For effectors, most current agents act via selecting from a list of options, e.g., HTML elements (Deng et al., 2023; Zheng et al., 2024) or labeled bounding boxes (He et al., 2024; Zhang et al., 2024a), instead of pixel-level operations directly on the GUI. Obtaining those options in turn often requires access to text-based representations and/or separate models for detecting objects and text (Wang et al., 2024a; Kapoor et al., 2024).

However, there is no free lunch, and those additional requirements come with their limitations. On the one hand, *text-based representations are noisy and incomplete*. Full HTMLs contain a considerable amount of irrelevant information. A11y trees are more compact and mainly contain semantic information, but similar to other semantic annotations that rely on voluntary participation, they widely suffer from incomplete and incorrect annotations.³ In contrast, visual renderings, by design, are information-complete and only contain information relevant to users. On the other hand, *the additional input increases latency and inference costs*. Zheng et al. (2024) found that HTML can consume up to 10 times more tokens to encode than the corresponding visual. Meanwhile, obtaining the a11y tree can be time-consuming in itself, especially in desktop or mobile environments. The added latency and cost at every step are further compounded in the long-horizon agent tasks, compromising user experience and practicality.

In this work, we are interested in *how far GUI agents with a human-like embodiment, i.e., only visual observation of environments and pixel-level operations, can go*. There have been a few attempts (Shaw et al., 2023; Hong et al., 2024; Cheng et al., 2024), but they are rarely adopted in state-of-the-art solutions. We find that a major bottleneck is *grounding*, i.e., mapping textual plans generated by an (M)LLM to the precise locations on the GUI. There are three desiderata for a GUI agent grounding model: 1) *High accuracy*. A single grounding error can get an agent stuck and fail the whole task. 2) *Strong generalization*. It should work on different GUIs: desktop (Windows, Linux, macOS), mobile (Android, iOS), different websites, etc. 3) *Flexibility*. It should plug and play in different MLLMs instead of being tightly coupled with a certain model. Existing visual grounding methods for GUI agents (Shaw et al., 2023; Hong et al., 2024; Cheng et al., 2024) fail to meet these desiderata, hindering the advances towards GUI agents with human-like embodiment.

The main contributions of this work are three-fold:

1. We make careful arguments and a strong case for GUI agents with human-like embodiment that perceive the digital world entirely visually and take pixel-level operations on GUIs, and propose a generic framework, **SeeAct-V**, for building such agents by adapting from the popular SeeAct framework (Zheng et al., 2024).
2. We show that a simple recipe, which includes web-based synthetic data and slight adaptation of the LLaVA architecture (Liu et al., 2024c), is surprisingly effective for GUI visual grounding. Using this recipe, we construct and release the **largest GUI visual grounding dataset** to date, covering

¹Except for auditory perception, which is out of scope of this study.

²The a11y tree is a compact yet informative representation intended for assistive technologies to facilitate people with disabilities, e.g., visual impairment.

³A 2024 survey over the top one million websites found that 95.9% of the home pages had accessibility conformance errors such as missing alternative text for images or missing form input labels, with an average of 56.8 errors per page (WebAIM, 2024).

19M GUI elements and their referring expressions over 1.3M GUI screenshots. We also train and release a universal visual grounding model, **UGround**, on the dataset.

3. We conduct the most comprehensive evaluation for GUI agents to date, covering six benchmarks spanning three categories (Figure 1): **grounding** (desktop, mobile, and web), **offline agent evaluation** (desktop, mobile, and web), and **online agent evaluation** (mobile and web). The results demonstrate: 1) UGround substantially outperforms existing visual grounding models for GUI agents across the board, by up to 20% absolute. 2) SeeAct-V agents with UGround can achieve at least comparable and often much better performance than state-of-the-art agents that use additional text-based input. These results provide strong support for the feasibility and promises of GUI agents that navigate the digital world as humans do.

2 METHOD

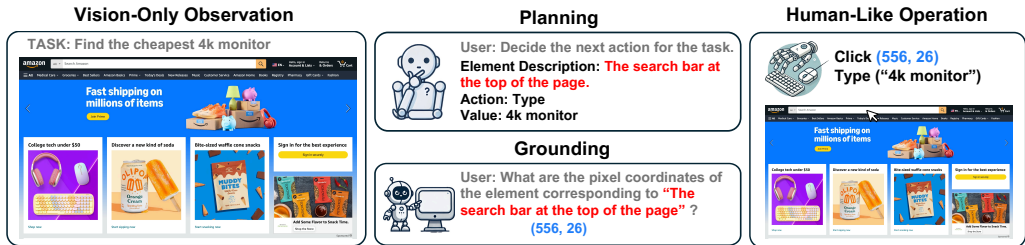


Figure 2: **SeeAct-V**, which uses screenshots as the only environmental observation (task instructions are input as text), without relying on HTML or a11y trees. It includes an MLLM that generates textual plans and a visual grounding model to map textual plans into coordinates on the screenshot. *Note: “Click” is always automatically inserted before “Type”.*

2.1 OVERVIEW

We adapt the popular SeeAct framework (Zheng et al., 2024) to one in which agents only take visual observation of the environment and directly conduct pixel-level operations, denoted as *SeeAct-V* (Figure 2). The original SeeAct has two stages, planning and grounding. An MLLM is used for both planning and grounding. At each step, an MLLM first generates a textual plan, and grounding is then done by asking the MLLM to select from a short list of grounding candidates. The grounding candidates are either filtered HTML elements or labels of Set-of-Mark (SoM; Yang et al. (2023)) annotations on the screenshot, both of which require HTMLs or a11y trees as additional input. In contrast, SeeAct-V only uses screenshots for environmental observation. For grounding, SeeAct-V uses a separate model specialized for visual grounding that directly produces the coordinates on the current screen where the agent should act.

A strong visual grounding model therefore becomes the key for making SeeAct-V a compelling framework. Ideally, it should generalize across platforms (e.g., web, desktop, and mobile) and handle diverse ways of referring to GUI elements. Considering the rapid evolution of MLLMs, this grounding model should be easily pluggable into different MLLMs to help ground their plans into different GUI environments. Finally, GUI screenshots can vary drastically in resolution and orientation, therefore the grounding model should handle a wide range of input resolutions. *The main technical contribution of this work is a surprisingly simple recipe (incl. data and modeling) for training such universal visual grounding models.* We introduce our simple data synthesis strategy in §2.2, followed by modeling considerations in §2.3. With this simple recipe, we construct the largest training data for GUI grounding to date and train *UGround*, a strong universal visual grounding model for GUI agents.

2.2 DATA CONSTRUCTION

We synthesize a large, high-quality, and diverse set of $\langle \text{screenshot}, \text{referring expression}, \text{coordinates} \rangle$ triplets as training data for visual grounding, where we use the center point coordinates of an element

as the expected output. Our data synthesis will be based on webpages. Webpages are ideal for grounding data synthesis because their *dual* representation—we can easily get the full HTML, the visual rendering, and fine-grained correspondences between the two (e.g., HTML elements to precise bounding boxes). HTML elements also contain rich metadata such as CSS or accessibility attributes, opening numerous opportunities for synthesizing diverse referring expressions (REs). Finally, since GUI designs share many similarities across platforms, we hypothesize that visual grounding models trained only on web data may still generalize to other platforms like desktop and mobile UIs.

Common RE Types for GUIs. People use diverse ways to refer to GUI elements (Figure 3). Previous visual grounding works (Hong et al., 2024; Cheng et al., 2024) have not sufficiently considered this dimension of diversity. We categorize common REs for GUI elements into three types: 1) **Visual REs**, i.e., salient visual features like text or image content, element types (e.g., buttons or input fields), shapes, colors, etc. 2) **Positional REs**, including both absolute (e.g., “at the top left of the page”) and relative positions (e.g., “to the right of element X”) to other elements. Besides straightforward positional information, *contextual references* (e.g., “for Item A,” “under the section X”) are more challenging for grounding because they require understanding both positional relationships and semantic relationships between elements (e.g., a like button is associated with a product). 3) **Functional REs**, i.e., referring to elements by their main functions (e.g., “Navigate to Home,” “Go to My Cart”). Composite types that combine two or more of these types are also common, especially when stronger disambiguation is needed, e.g., “click the heart button under the Pokémon shirt to add to favorite.”

Hybrid RE Synthesis from Web. We propose a novel hybrid synthesis pipeline, orchestrating both carefully curated rules as well as LLMs to generate diverse REs for HTML elements: 1) **Primary Descriptors:** We extract abundant visual and functional information that are embedded in the attributes of HTML elements. For example, HTML attributes like `inner-text` and `alt` provide visual clues (including text content), while accessibility attributes like `aria-label` reveal more functional aspects of an HTML element. However, HTML attributes are often incomplete. To harvest visual and functional signals beyond HTML attributes, we use an open MLLM, LLaVA-NeXT-13B (Liu et al., 2024b). We input the visual rendering of an HTML element along with its available attributes to the MLLM and prompt it to generate diverse REs. This process often yields composite REs that combine some HTML attributes with visual features (e.g., “hollow heart”) or new knowledge from the MLLM (e.g., a blue bird icon represents Twitter). Similar to Lai et al. (2023), we also employ an LLM (Llama-3-8B-Instruct; AI@Meta (2024)) to make these generated REs more concise. We randomly select one of the following as the *primary descriptor* of an element: a visual HTML attribute, a functional HTML attribute, or the synthesized description by LLMs. 2) **Positional Expressions:** We curate rules to generate positional REs according to the absolute position of an element in the screenshot as well as its spatial relationship to neighboring elements (e.g., “at the top of the page,” “between element A and B”). We also create multiple rules to generate contextual references. For example, we identify elements of certain types in the screenshot (e.g., radio buttons, checkboxes, input fields) as *landmarks*, and generate REs for them based on their spatial and structural relationship (e.g., hierarchical structure of the DOM tree) to others (e.g., “the input field labeled *Birthday*”).

We collect screenshots (mix of portrait and landscape views in various resolutions) and metadata of web elements (salient HTML attributes, bounding box coordinates) from Common Crawl,⁴ and then apply data synthesis pipeline to get our main training dataset (**Web-Hybrid**). We leave more details to Appendix C.1.

Supplementary Data. There have been multiple prior efforts on constructing grounding data for Android, so we incorporate the existing datasets as well. We also use GPT-4o to directly synthesize

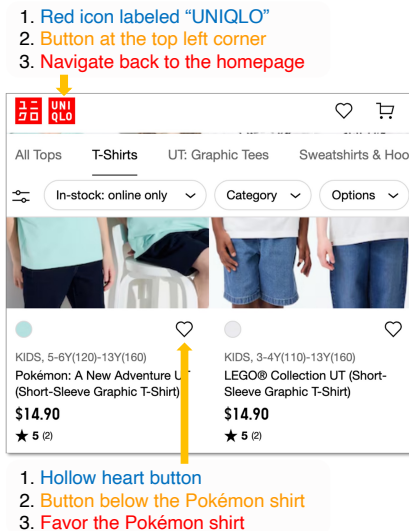


Figure 3: Examples of visual, positional, and functional REs.

⁴<https://commoncrawl.org/>

Table 1: Overview of training datasets used for UGround.

Dataset	Annotation	# of Elements	# of Screenshots	Platform
Web-Hybrid (Ours)	Rule + LLM	18M	773K	Web
Web-Direct (Ours)	GPT	408K	408K	Web
GUIAct (Chen et al., 2024)	GPT + Human	140K	13K	Web
AndroidControl (Li et al., 2024b)	Human	47K	47K	Android
Widget Caption (Li et al., 2020b)	Human	41K	15K	Android
UIBert (Bai et al., 2021)	Human	16K	5K	Android
AITZ (Zhang et al., 2024b)	GPT + Human	8K	8K	Android
Total		19M	1.3M	Web + Android

a small set of REs for web elements, with a focus on more open-ended REs (no constraints on the type) and functional REs (**Web-Direct**). These additions help provide more diverse REs and cover elements in Android, especially those not commonly found on the web (e.g., toggle buttons).

In total, we compile a dataset totaling 19M UI elements, with the majority (95%) from our hybrid synthesis pipeline (Table 1). Elements on the same screenshot are batched to accelerate training.

2.3 MODEL DESIGN

We adopt a widely used open-source model architecture, 7B LLaVA-NeXT (Liu et al., 2024b), as our backbone model for visual grounding. We make a few adaptations to tailor it for GUI grounding.

Input-Output Formulation. We always instruct the model to answer “*In the screenshot, what are the pixel element coordinates corresponding to {Description}?*” Following recent work in visual grounding (Cheng et al., 2024), we represent the answer in natural language so we can directly use autoregressive decoding. Specifically, we opt for coordinates in the numerical form (e.g., “(1344, 1344)”) to precisely point to an element without any normalization.

Image Resolution. GUI screenshots are much larger than typical natural images, often requiring a resolution above 1,000px for legibility. LLaVA (Liu et al., 2024c;a) was initially built for 336px images, and was later scaled up to at most 772px via the AnyRes technique (Cheng et al., 2023; Gao et al., 2024; Liu et al., 2024b; Xu et al., 2024; Dong et al., 2024). It resizes and splits a large image into small slices, encodes each slice independently with the vision encoder, and adds a special token at the end of each row to help the language model keep track of the image shape. AnyRes allows easy scaling up of input resolution. However, it is always a trade-off between the diversity of supported resolutions and the speed of training and inference. To strike a balance and avoid meaningless excessive resolutions, we enlarge the allowed input sizes to 36 ViT (Dosovitskiy et al., 2020) slices, and use CLIP@224px (Radford et al., 2021) as the image encoder for more flexible splitting, pushing the maximum supported resolution to $1,344 \times 1,344$ (landscape) and $896 \times 2,016$ (portrait). Additionally, we use Vicuna-1.5-7b-16k (Zheng et al., 2023) with 16K context length to handle long visual contexts. Finally, there is a low-resolution image fusion module commonly used in AnyRes. However, we find it ineffective for GUI grounding, as 224px is too small to provide informative global context, so we leave it out from our model. More details are in Appendix D.

3 EXPERIMENTS

Most existing studies on GUI agents typically evaluate on one or two benchmarks. In contrast, we conduct a much more comprehensive evaluation on GUI agents to show the universality of our method. Our evaluation employs six benchmarks that span all three major platforms (i.e., web, desktop, and mobile) and cover three settings: *visual grounding* (§3.1), *offline agent evaluation* on cached environment states (§3.2), and *online agent evaluation* in live environments (§3.3). The visual grounding setting focuses on the grounding performance of UGround, while the agent settings test the end-to-end effectiveness of the SeeAct-V framework with UGround integrated. On the agent benchmarks, we compare the vision-only SeeAct-V framework with prior SOTA methods that usually require additional text-based representations (HTML or a11y tree) as input. Within SeeAct-V, we also compare UGround with existing visual grounding models whenever possible.

Table 2: Grounding accuracy on ScreenSpot (Standard Setting). Results for GPT-4, CogAgent, and SeeClick are from Cheng et al. (2024).

Grounding Model	Mobile		Desktop		Web		Average
	Text	Icon/Widget	Text	Icon/Widget	Text	Icon/Widget	
GPT-4	22.6	24.5	20.2	11.8	9.2	8.8	16.2
GPT-4o	20.2	24.9	21.1	23.6	12.2	7.8	18.3
CogAgent (Hong et al., 2024)	67.0	24.0	74.2	20.0	70.4	28.6	47.4
SeeClick (Cheng et al., 2024)	78.0	52.0	72.2	30.0	55.7	32.5	53.4
UGround (Ours)	82.8	60.3	82.5	63.6	80.4	70.4	73.3

Table 3: Grounding accuracy on ScreenSpot (Agent Setting) with planner-generated REs.

Planner	Grounding	Mobile		Desktop		Web		Avg.
		Text	Icon/Widget	Text	Icon/Widget	Text	Icon/Widget	
GPT-4	SeeClick	76.6	55.5	68.0	28.6	40.9	23.3	48.8
	UGround	90.1	70.3	87.1	55.7	85.7	64.6	75.6
GPT-4o	SeeClick	81.0	59.8	69.6	33.6	43.9	26.2	52.3
	UGround	93.4	76.9	92.8	67.9	88.7	68.9	81.4

3.1 GUI VISUAL GROUNDING

We first evaluate UGround on the ScreenSpot benchmark (Cheng et al., 2024), which is specifically designed for visual grounding on GUIs. The benchmark consists of 1,272 single-step instructions and the corresponding bounding box of the target elements across mobile (e.g., iOS and Android), desktop (e.g., macOS and Windows), and web environments. These elements vary between text-based elements, icons (e.g., the trash can icon) and widgets (e.g., to-do lists), representing diverse GUI element types.

We evaluate under two settings: 1) **Standard Setting**. In the standard setting of ScreenSpot, the instructions are written by human annotators with a primary focus on *functional description* of the target elements, e.g., simply “close” to refer to the ‘X’ button that closes a window or “set an alarm for 7:40” when the input image shows the iPhone clock app with a list of inactive alarms. 2) **Agent Setting**. For GUI agents, a grounding model needs to work with a planning model (e.g., an MLLM) and ground the REs it generates, which includes not only functional REs but also visual and positional REs (see §2.2). To provide a more comprehensive evaluation on visual grounding for GUI agents, we input each ScreenSpot example to an MLLM, which acts as a planning model, and asks it to generate diverse REs for the target element. This setting is therefore more representative of the grounding challenges in GUI agents. We mainly compare UGround with SeeClick (Cheng et al., 2024), the state-of-the-art visual grounding model on ScreenSpot, and another visual grounding model CogAgent Hong et al. (2024). To show the challenge of visual grounding for general-purpose models, we also compare with GPT-4 and GPT-4o.

Results. As shown in Table 2 and Table 3, UGround outperforms all existing models across all the settings and platforms by a substantial margin, about an absolute improvement of **20%** on average under the standard setting and **29%** under the agent setting. Interestingly, UGround performs remarkably well on desktop UIs, despite the fact that it is never trained on desktop screenshots (Table 1). Compared with existing models, UGround performs especially well on icons and widgets, which are generally more challenging for grounding because that requires deeper understanding of the contextual (e.g., positional) and semantic (e.g., functional) information. Overall, the strong results on ScreenSpot clearly demonstrates UGround’s universal grounding capability across platforms and planners as well as the remarkable effectiveness of our simple data synthesis and modeling recipe.

3.2 OFFLINE AGENT EVALUATION

We discuss the experimental setup for three offline agent evaluation benchmarks followed by result discussion. Concrete examples from each benchmark are given in Appendix B.

Web: Multimodal-Mind2Web. We use Multimodal-Mind2Web (Zheng et al., 2024), the multimodal extension of Mind2Web (Deng et al., 2023), for our evaluation on realistic web tasks. The test

Table 4: Element accuracy on Multimodal-Mind2Web. Results by Choice and SoM are from Zheng et al. (2024). The SoM results are on subsets of 30 tasks for each split.

Input	Planner	Grounding	Cross-Task	Cross-Website	Cross-Domain	Avg.
Image + Text	GPT-4	Choice	46.4	38.0	42.4	42.3
		SoM	29.6	20.1	27.0	25.6
Image (SeeAct-V)	GPT-4	SeeClick	29.7	28.5	30.7	29.6
		UGround	45.1	44.7	44.6	44.8
	GPT-4o	SeeClick	32.1	33.1	33.5	32.9
		UGround	47.7	46.0	46.6	46.8

split consists of 1,013 tasks spanning over 100 different websites. Each task contains a high-level task instruction and a sequence of actions, with a screenshot of the webpage before each action, as the golden trajectory. All the webpages along the golden trajectory are cached to support offline evaluation. The tasks are crowdsourced with a focus on ensuring real-world meaningfulness (i.e., what real users would need on those websites).

Zheng et al. (2024) have clearly demonstrated the necessity of visual perception for web agents, so we mainly compare with zero-shot methods that use MLLMs as planners and omit text-only LLMs. Zheng et al. (2024) have also identified grounding as the main challenge and proposed several grounding strategies, including 1) **Choice**, where the planner is asked to choose from a short list of filtered HTML elements, and 2) **SoM**, where the input screenshot is superposed with set-of-mark (Yang et al., 2023) labels and the planner is asked to select from the labels. Both strategies require additional text-based representations (i.e., HTML) to obtain the candidates and/or locate the elements in the screenshot to label. We report *element accuracy*, i.e., accuracy of selecting the correct element, and omit operation scores because they are orthogonal to grounding comparisons.

Mobile: AndroidControl. We use AndroidControl (Li et al., 2024b), a large-scale Android dataset comprising 15K unique tasks over 833 Apps. Screenshots, action sequences, and a11y trees are cached from human demonstrations as golden trajectories for training and evaluation purposes. Each action is also labeled by a corresponding low-level instruction (e.g., “set the hours to 6”). Following Li et al. (2024b), we use 500 random steps from the test set. We compare with the SOTA zero-shot method, the text-only version of M3A (Rawles et al., 2024), which instructs GPT-4 to generate textual actions as well as select elements from the a11y tree (**Choice**). We adopt the two task settings in Li et al. (2024b): high-level tasks, where only the high-level intent is provided, and low-level tasks, where both the high-level intent and the corresponding low-level instruction for each step are available. We use the standard metric, *step-wise accuracy*, where a step is considered successful only if all the predicted actions, elements, and arguments (if applicable) are correct.

Desktop: OmniACT. We use OmniACT (Kapoor et al., 2024) to evaluate the accuracy of UGround on desktop tasks. The dataset consists of 9,802 tasks covering 38 desktop applications and 27 websites across different desktop platforms (macOS, Windows, and Linux). Each task requires the generation of a PyAutoGUI script, which is a sequence of actions to complete the task on a single screenshot. The SOTA method, DetACT (Kapoor et al., 2024), extracts UI elements and their coordinates through a combination of OCR (optical character recognition), icon matching, and color detection modules. These elements are filtered by task relevance and then passed to LLMs or MLLMs to generate the PyAutoGUI script with the appropriate coordinates for interaction.

For SeeAct-V, we replace the input of the DetACT pipeline with only screenshots and instruct MLLMs to generate element descriptions rather than directly generate coordinates. We then employ UGround to obtain the coordinates of the elements, which are subsequently integrated into the PyAutoGUI scripts. To ensure a fair comparison, we strictly follow the approach in Kapoor et al. (2024), including the same prompt and retrieval strategy that selects five in-context examples from the training set based on task similarity. We report the *action score*, which measures the accuracy of the action sequences while penalizing errors in generated arguments.

Results. As shown in Table 4, Table 5, and Table 6, SeeAct-V with UGround outperforms all the baselines across the board, despite only using raw screenshots as input while baselines use additional input. UGround also consistently outperforms a strong GUI grounding model, SeeClick. These results provide solid support for human-like vision-only embodiment for GUI agents, a position this work aims to make a case for. The results also further validate UGround’s efficacy as a universal grounding model for GUI agents.

Table 5: Step accuracy on AndroidControl over 500 random actions from the test split. Baseline results are from Li et al. (2024b).

Input	Planner	Grounding	Step Accuracy	
			High	Low
Text	GPT-4	Choice	42.1	55.0
Image (SeeAct-V)	GPT-4	SeeClick	39.4	47.2
		UGround	46.2	58.0
	GPT-4o	SeeClick	41.8	52.8
		UGround	48.4	62.4

Table 6: Action scores (AS) on OmniACT. Baseline results are from Kapoor et al. (2024).

Inputs	Planner	Grounding	AS
Text	GPT-4	DetACT	11.6
Image + Text		DetACT	17.0
Image (SeeAct-V)	GPT-4	SeeClick	28.9
		UGround	31.1
	GPT-4o	SeeClick	29.6
		UGround	32.8

Table 7: Completion rate (CR) and task success rate (SR) on Mind2Web-Live. Baseline results are from Pan et al. (2024).

Inputs	Planner	Grounding	CR	SR
Text	GPT-4	Choice	44.3	21.1
	GPT-4o		47.6	22.1
Image (SeeAct-V)	GPT-4	UGround	50.7	23.1
	GPT-4o		50.8	19.2

Table 8: Task success rate (SR) on AndroidWorld. Baseline results are from Rawles et al. (2024)

Input	Planner	Grounding	SR
Text	GPT-4	Choice	30.6
Image + Text		SoM	25.4
Image (SeeAct-V)	GPT-4	UGround	31.0
	GPT-4o		32.8

3.3 ONLINE AGENT EVALUATION

We further evaluate our approach in an end-to-end manner on two online agent benchmarks that closely resemble the offline web and Android benchmarks in §3.2, but involve interactions with live websites and mobile applications. Due to the high cost of online evaluation, we only use UGround for grounding.

Web: Mind2Web-Live. We use the test set from Mind2Web-Live (Pan et al., 2024). The benchmark is built on Mind2Web (Deng et al., 2023) by adding functional evaluation to the tasks that makes automated evaluation possible on live websites. Specifically, it defines and annotates *key nodes* for each task, which are critical steps that must be completed for a task to be considered successful, regardless of which trajectory an agent takes. The baseline agent from Pan et al. (2024) is text-only, perceives and interacts with webpages by hundreds of HTML elements at a time. For SeeAct-V, we change the observation to be screenshots only, and make necessary changes to the original action space to fully eliminate the dependency on HTML during planning, grounding, and execution (details in Appendix E.5). We use standard metrics: *micro completion rate*, which measures the proportion of completed key nodes across all the tasks, and *task success rate*, which measures the proportion of fully completed tasks.

Mobile: AndroidWorld. We use AndroidWorld (Rawles et al., 2024), an online mobile agent benchmark running in Android emulators. It includes 116 tasks across 20 Apps, with evaluation based on the final states of the device. We compare with the SOTA agent M3A and its text-only variant from Rawles et al. (2024). They receives both raw and SoM images, together with textual UI elements, or only the textual UI elements as the observation respectively. Both variants employ a ReAct-style reasoning process (Yao et al., 2023) to select the next target element from a list of UI elements. Additionally, they integrate self-reflection (Shinn et al., 2024) for the agent to summarize its current action and improve decision-making in subsequent steps. We report *task success rate*, which measure the percentage of fully completed tasks.

Results. SeeAct-V with UGround gets comparable or higher performance in online agent evaluation, as shown in Table 7 and Table 8. Particularly, it achieves a much higher success rate compared with the SoM variant of M3A, even though Android environments have less dense UI layouts and are generally more suitable for SoM (i.e., less obstruction by the SoM labels). These results again provide solid support for the feasibility and promises of human-like vision-only embodiment for GUI agents and the effectiveness of UGround.

432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485

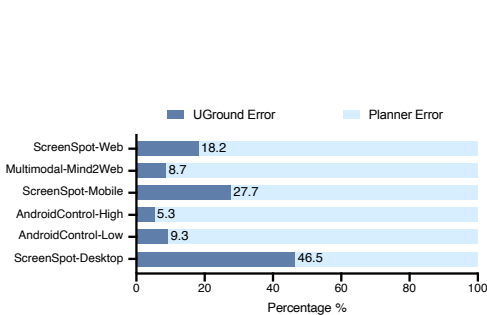


Figure 4: Error distribution from manual analysis.

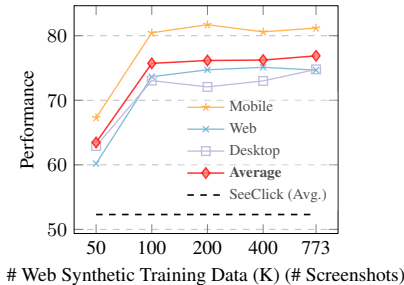


Figure 5: Scaling curve of UGround on ScreenSpot w.r.t. Web-Hybrid data size.

3.4 ERROR ANALYSIS

We conduct a manual error analysis of the best performing method, SeeAct-V with UGround, to understand the bottleneck for further improvement. We randomly sample 60 failure cases from each split of ScreenSpot (agent setting with GPT-4o), AndroidControl, and Multimodal-Mind2Web. Except for data annotation errors, errors from the models can be categorized into *planning errors*, i.e., generating plans with incorrect element descriptions, and *grounding errors*, i.e., predicting incorrect coordinates for a correct element description from the planner.

As shown in Figure 4, planning errors are the dominant cause of failures across all benchmarks, further confirming the strong grounding capability of UGround. The most frequent error is that the planner generates (otherwise correct) description of an incorrect element on the screen, indicating a lack of correct understanding of either the task and/or the elements. Other common planning errors include hallucinating non-existent elements or producing overly generic descriptions that are too vague to uniquely locate the target element, even for human evaluators.

On the other hand, on ScreenSpot-Mobile and ScreenSpot-Desktop, a considerable portion of the failures do stem from grounding errors. Both desktop and mobile UIs feature a pervasive use of icons with idiosyncratic meaning. For example, a stylized dollar sign represents the Zelle App, or an icon with two cartoon people represents one’s contact list in Microsoft Outlook. We find that pretrained MLLMs and our web-centric grounding training are effective in capturing the semantics of popular icons (e.g., icons representing Google) or commonsense meaning (e.g., clock icons usually represent time-related functions like alarms). However, it is challenging to capture the idiosyncratic semantics of icons in the long tail, which arguably requires either additional documentation or more targeted exploration to learn. This is a major cause of the grounding errors. Interestingly, when tested on more realistic agent tasks, e.g., in AndroidControl, AndroidWorld, and OmniACT, UGround still proves to be relatively robust. This is because most of the agent tasks concern things in the head of the distribution; things in the long tail are naturally rare (though still important). This explains the strong performance of UGround on mobile and desktop agent benchmarks. Nonetheless, how to capture idiosyncratic semantics in the long tail is still an open challenge for grounding.

3.5 TRAINING DATA ANALYSIS: SCALING AND ABLATIONS

We conduct scaling analysis and ablation studies on our training data to better understand the contribution of different data for UGround’s strong performance. We use the agent setting of ScreenSpot with GPT-4o as the planner.

Scaling Curve on Web-Hybrid. We investigate the scaling of our primary synthetic dataset, Web-Hybrid, which consists of 18M data instances over 773K web screenshots in total. The scaling results in Figure 5 show that the average performance consistently improves as the data scales up, though the return starts diminishing after 100K screenshots. Notably, with just 50K screenshots (about 1M elements) as training data, UGround surpasses SeeClick by more than 10%, which is trained on about 3M web and Android elements from about 400K screenshots. The results clearly show the high data quality and the effectiveness for grounding training of our data synthesis pipeline. Upon manual inspection, we observe that additional data after 100K screenshots primarily enhances understanding of less frequent elements such as radio buttons, checkboxes, or very small text elements. As data

Table 9: Training data ablations for UGround on ScreenSpot.

Training Data	Mobile		Desktop		Web		Average
	Text	Icon/Widget	Text	Icon/Widget	Text	Icon/Widget	
Web-Hybrid	89.0	73.4	88.1	61.4	84.8	64.6	76.9
Others	92.3	71.2	84.5	46.4	87.0	59.2	73.4
All	93.4	76.9	92.8	67.9	88.7	68.9	81.4

increases, the model can point to the center of element bounding boxes more accurately and better handle tiny hyperlinks.

Training Data Ablations. To further investigate the impact of training data sources, we compare the performance of UGround trained on only Web-Hybrid, only the supplementary data, or both (see Table 1). Results in Table 9 further validate the necessity of Web-Hybrid. Training on other data without Web-Hybrid often underperforms training on Web-Hybrid alone. This is most evident on icons and widgets, which require understanding more diverse aspects, such as visual features and functions, than text-based elements. Finally, these two data sources are complementary and their combination yield the best performance across the board.

4 CONCLUSIONS AND LIMITATIONS

We introduce UGround, a universal GUI visual grounding model developed with large-scale web-based synthetic data. UGround shows strong cross-platform generalization and significantly outperforms the prior SOTA model SeeClick on ScreenSpot. We propose a vision-only framework SeeAct-V that allows pixel-level interactions based solely on visual input. Our evaluations on both offline and online benchmarks demonstrate that SeeAct-V agents with UGround can achieve comparable and often better performance than prior SOTA agents that rely on additional textual inputs like HTML or a11y trees for observation or grounding.

Nevertheless, there are still some limitations that can be addressed in future work to advance visual grounding in GUI and visually grounded GUI agents. First, UGround is trained on very large-scale synthetic data. Considering the similarity and repetition of elements between web pages, there is much room to improve data efficiency during training, for example by better data grouping and deduplication. On the other hand, despite the cross-platform generalization shown in our experiment results, the issue of long-tail elements remains unaddressed in this work. Mobile UIs and desktop UIs often feature specific icons, where it can be impractical to account for every long-tail element in the training set. Additionally, no desktop UI data is incorporated in the training of this work, which limits the performance on desktop UIs. Given the scarcity of training datasets for desktop UIs, we anticipate the development of more comprehensive datasets in this domain. Lastly, UGround depends on an external planner, and without training on downstream tasks, it cannot function independently as a GUI agent. Nonetheless, we hope that our datasets, model, and framework can contribute to future studies of vision-only agents, as well as contribute to advancing the grounding capabilities of future end-to-end models.

REFERENCES

- AI@Meta. Llama 3 model card, 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.
- Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. Uibert: Learning generic multimodal representations for ui understanding. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 1705–1712, 2021.
- 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*, 2023.
- Pratyay Banerjee, Shweti Mahajan, Kushal Arora, Chitta Baral, and Oriana Riva. Lexi: Self-supervised learning of the ui language. *arXiv preprint arXiv:2301.10165*, 2023.

- 540 Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang
541 Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, et al. Spider2-v: How far are multimodal
542 agents from automating data science and engineering workflows? *arXiv preprint arXiv:2407.10956*,
543 2024.
- 544 Jun Chen, Deyao Zhu, Xiaoqian Shen, Xiang Li, Zechun Liu, Pengchuan Zhang, Raghuraman
545 Krishnamoorthi, Vikas Chandra, Yunyang Xiong, and Mohamed Elhoseiny. Minigpt-v2: large
546 language model as a unified interface for vision-language multi-task learning. *arXiv preprint*
547 *arXiv:2310.09478*, 2023a.
- 548 Keqin Chen, Zhao Zhang, Weili Zeng, Richong Zhang, Feng Zhu, and Rui Zhao. Shikra: Unleashing
549 multimodal llm’s referential dialogue magic. *arXiv preprint arXiv:2306.15195*, 2023b.
- 550 Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu,
551 Guirong Chen, Yupeng Huo, et al. Guicourse: From general vision language models to versatile
552 gui agents. *arXiv preprint arXiv:2406.11317*, 2024.
- 553 Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiy-
554 ong Wu. Seeclck: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint*
555 *arXiv:2401.10935*, 2024.
- 556 Siyuan Cheng, Bozhong Tian, Qingbin Liu, Xi Chen, Yongheng Wang, Huajun Chen, and Ningyu
557 Zhang. Can we edit multimodal large language models? In *Proceedings of the 2023 Conference*
558 *on Empirical Methods in Natural Language Processing*, pp. 13877–13888, 2023.
- 559 Biplob Deka, Zifeng Huang, Chad Franzen, Joshua Hibschan, Daniel Afegan, Yang Li, Jeffrey
560 Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design
561 applications. In *Proceedings of the 30th annual ACM symposium on user interface software and*
562 *technology*, pp. 845–854, 2017.
- 563 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su.
564 Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing*
565 *Systems*, volume 36, pp. 28091–28114, 2023.
- 566 Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Songyang Zhang,
567 Haodong Duan, Wenwei Zhang, Yining Li, et al. Internlm-xcomposer2-4khd: A pioneering
568 large vision-language model handling resolutions from 336 pixels to 4k hd. *arXiv preprint*
569 *arXiv:2404.06512*, 2024.
- 570 Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas
571 Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An
572 image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint*
573 *arXiv:2010.11929*, 2020.
- 574 Peng Gao, Renrui Zhang, Chris Liu, Longtian Qiu, Siyuan Huang, Weifeng Lin, Shitian Zhao, Shijie
575 Geng, Ziyi Lin, Peng Jin, et al. Sphinx-x: Scaling data and parameters for a family of multi-modal
576 large language models. *arXiv preprint arXiv:2402.05935*, 2024.
- 577 Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and
578 Aleksandra Faust. A real-world webagent with planning, long context understanding, and program
579 synthesis. In *International Conference on Learning Representations*, 2024.
- 580 Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan,
581 and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models.
582 *arXiv preprint arXiv:2401.13919*, 2024.
- 583 Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan
584 Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents.
585 In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp.
586 14281–14290, 2024.
- 587 Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang,
588 and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International*
589 *Conference on Learning Representations*, 2022.

- 594 Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair
595 Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven
596 approach for learning to control computers. In *International Conference on Machine Learning*, pp.
597 9466–9482. PMLR, 2022.
- 598
599 Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh,
600 and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist
601 autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024.
- 602
603 Andrej Karpathy, Armand Joulin, and Li F Fei-Fei. Deep fragment embeddings for bidirectional
604 image sentence mapping. *Advances in neural information processing systems*, 27, 2014.
- 605
606 Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks.
607 *Advances in Neural Information Processing Systems*, 36, 2024.
- 608
609 Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham
610 Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating
611 multimodal agents on realistic visual web tasks. In *ICLR 2024 Workshop on Large Language
612 Model (LLM) Agents*, 2024.
- 613
614 Zhengfeng Lai, Haotian Zhang, Wentao Wu, Haoping Bai, Aleksei Timofeev, Xianzhi Du, Zhe Gan,
615 Jiulong Shan, Chen-Nee Chuah, Yinfei Yang, et al. From scarcity to efficiency: Improving clip
616 training via visual-enriched captions. *arXiv preprint arXiv:2310.07699*, 2023.
- 617
618 Bo Li, Hao Zhang, Kaichen Zhang, Dong Guo, Yuanhan Zhang, Renrui Zhang, Feng Li, Ziwei Liu,
619 and Chunyuan Li. Llava-next: What else influences visual instruction tuning beyond data?, May
620 2024a. URL <https://llava-v1.github.io/blog/2024-05-25-llava-next-ablations/>.
- 621
622 Gang Li and Yang Li. Spotlight: Mobile ui understanding using vision-language models with a focus.
623 In *The Eleventh International Conference on Learning Representations*, 2022.
- 624
625 Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu,
626 and Oriana Riva. On the effects of data scale on computer control agents. *arXiv preprint
627 arXiv:2406.03679*, 2024b.
- 628
629 Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language
630 instructions to mobile ui action sequences. In *Proceedings of the 58th Annual Meeting of the
631 Association for Computational Linguistics*, pp. 8198–8210, 2020a.
- 632
633 Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. Widget captioning:
634 Generating natural language description for mobile user interface elements. In *Proceedings
635 of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp.
636 5495–5510, 2020b.
- 637
638 Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction
639 tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*,
640 pp. 26296–26306, 2024a.
- 641
642 Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee.
643 Llava-next: Improved reasoning, ocr, and world knowledge, January 2024b. URL [https://
644 llava-v1.github.io/blog/2024-01-30-llava-next/](https://llava-v1.github.io/blog/2024-01-30-llava-next/).
- 645
646 Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in
647 neural information processing systems*, 36, 2024c.
- 648
649 Chuofan Ma, Yi Jiang, Jiannan Wu, Zehuan Yuan, and Xiaojuan Qi. Groma: Localized visual
650 tokenization for grounding multimodal large language models. *arXiv preprint arXiv:2404.13013*,
651 2024.
- 652
653 Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy.
654 Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE
655 conference on computer vision and pattern recognition*, pp. 11–20, 2016.

- 648 Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi
649 Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online
650 environments. *arXiv preprint arXiv:2406.12373*, 2024.
- 651
652 Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu
653 Wei. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint*
654 *arXiv:2306.14824*, 2023.
- 655 Yijun Qian, Yujie Lu, Alexander G Hauptmann, and Oriana Riva. Visual grounding for user interfaces.
656 In *Proceedings of the 2024 Conference of the North American Chapter of the Association for*
657 *Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pp.
658 97–107, 2024.
- 659 Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal,
660 Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual
661 models from natural language supervision. In *International conference on machine learning*, pp.
662 8748–8763. PMLR, 2021.
- 663
664 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in
665 the wild: A large-scale dataset for android device control. In *Advances in Neural Information*
666 *Processing Systems*, volume 36, pp. 59708–59728, 2023.
- 667
668 Christopher Rawles, Sarah Clinckemaiillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth
669 Fair, Alice Li, William Bishop, Wei Li, Folawiyi Campbell-Ajala, et al. Androidworld: A dynamic
670 benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.
- 671
672 Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi
673 Khandelwal, Kenton Lee, and Kristina N Toutanova. From pixels to ui actions: Learning to follow
674 instructions via graphical user interfaces. *Advances in Neural Information Processing Systems*, 36:
675 34354–34370, 2023.
- 676
677 Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An
678 open-domain platform for web-based agents. In *International Conference on Machine Learning*,
pp. 3135–3144. PMLR, 2017.
- 679
680 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion:
681 Language agents with verbal reinforcement learning. *Advances in Neural Information Processing*
682 *Systems*, 36, 2024.
- 683
684 Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao
685 Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. In
686 *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024a.
- 687
688 Weiyun Wang, Min Shi, Qingyun Li, Wenhui Wang, Zhenhang Huang, Linjie Xing, Zhe Chen, Hao
689 Li, Xizhou Zhu, Zhiguo Cao, et al. The all-seeing project: Towards panoptic visual recognition
690 and understanding of the open world. In *The Twelfth International Conference on Learning*
691 *Representations*, 2024b.
- 692
693 Wenhui Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong
694 Lu, Jie Zhou, Yu Qiao, et al. Visionllm: Large language model is also an open-ended decoder for
695 vision-centric tasks. *Advances in Neural Information Processing Systems*, 36, 2024c.
- 696
697 Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-
698 attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural*
699 *Information Processing Systems*, 33:5776–5788, 2020.
- 700
701 WebAIM. The WebAIM Million. <https://webaim.org/projects/million/>, 2024. Accessed:
2024-08-04.
- Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu,
and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. In
ICLR 2024 Workshop on Large Language Model (LLM) Agents, 2024.

- 702 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing
703 Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal
704 agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*,
705 2024.
- 706 Ruyi Xu, Yuan Yao, Zonghao Guo, Junbo Cui, Zanlin Ni, Chunjiang Ge, Tat-Seng Chua, Zhiyuan Liu,
707 Maosong Sun, and Gao Huang. Llava-uhd: an lmm perceiving any aspect ratio and high-resolution
708 images. *arXiv preprint arXiv:2403.11703*, 2024.
- 709 An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu
710 Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for
711 zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.
- 712 Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark
713 prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*,
714 2023.
- 715 Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable
716 real-world web interaction with grounded language agents. *Advances in Neural Information
717 Processing Systems*, 35:20744–20757, 2022.
- 718 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
719 React: Synergizing reasoning and acting in language models. In *International Conference on
720 Learning Representations (ICLR)*, 2023.
- 721 Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei
722 Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *ArXiv*,
723 abs/2404.05719, 2024.
- 724 Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context
725 in referring expressions. In *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam,
726 The Netherlands, October 11–14, 2016, Proceedings, Part II 14*, pp. 69–85. Springer, 2016.
- 727 Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents.
728 *arXiv preprint arXiv:2309.11436*, 2023.
- 729 Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei
730 Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint
731 arXiv:2402.07939*, 2024a.
- 732 Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu
733 Tang. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*,
734 2024b.
- 735 Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web
736 agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024.
- 737 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,
738 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica.
739 Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information
740 Processing Systems*, volume 36, pp. 46595–46623, 2023.
- 741 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
742 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building
743 autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.
- 744
745
746
747
748
749
750
751
752
753
754
755

756	Table of Contents in Appendix	
757		
758		
759	A Related Work	16
760		
761	B Examples	17
762	B.1 Multimodal-Mind2Web	17
763		
764	B.2 AndroidControl	17
765		
766	B.3 OmniACT	18
767		
768	B.4 Training Data	19
769		
770	C Data Construction	20
771	C.1 Web-Hybrid	20
772		
773	C.2 Web-Direct	21
774		
775	C.3 Open-source Data	22
776		
777	D Model and Training Details	22
778	D.1 Overview	22
779		
780	D.2 AnyRes	22
781		
782	D.3 Training	22
783		
784	E Evaluation Details	23
785	E.1 Model Endpoints	23
786		
787	E.2 Multimodal-Mind2Web	23
788		
789	E.3 AndroidControl	23
790		
791	E.4 OmniACT	23
792		
793	E.5 Mind2Web-Live	24
794		
795	E.6 AndroidWorld	24
796		
797		
798		
799		
800		
801		
802		
803		
804		
805		
806		
807		
808		
809		
	F Prompts	25

A RELATED WORK

GUI Agents. LLMs and MLLMs have demonstrated great capabilities and potentials in GUI automation, working as digital agents in various GUI environments (Yan et al., 2023; Kim et al., 2024; Wang et al., 2024a; Zheng et al., 2024; Xie et al., 2024). Despite the growing number of studies focused on building multimodal agents (Koh et al., 2024; Zhou et al., 2024; Cao et al., 2024), most work still relies on HTML or a11y trees for grounding, even when they are not used for observation. In this work, we advance an alternative line of research: pixel-level visually grounded GUI agents (Shaw et al., 2023; Zhan & Zhang, 2023; Hong et al., 2024; Cheng et al., 2024). Unlike nearly all previous work of this line, we propose a generic two-stage approach that separates planning and visual grounding to build vision-only GUI agents, which perform remarkably well on realistic agent benchmarks with vision-only inputs, and offers the flexibility to the choices of planning and grounding models.

Visual Grounding. Visual grounding has been long studied on natural images (Karpathy et al., 2014; Mao et al., 2016; Yu et al., 2016). More recently, with the advancements of MLLMs, their visual grounding capabilities on natural images have attracted significant attention (Bai et al., 2023; Chen et al., 2023a;b; Peng et al., 2023; Wang et al., 2024b;c; Ma et al., 2024). However, due to significant gaps in image resolution and GUI understanding, these models trained on natural contexts work poorly on GUI visual grounding (Cheng et al., 2024). One of the most popular approaches, SoM (Yang et al., 2023), proposes a visual prompting method that adds marks such as boxes and numbers to images and instructs MLLM to identify the referred objects by the labels. It is widely adopted in GUI scenarios (Yan et al., 2023; He et al., 2024; Koh et al., 2024), but still suffers from problems including reliance on complete object information or object segmentation. Only few studies have been conducted for visual grounding on GUI screenshots. Based on Rico (Deka et al., 2017), Bai et al. (2021) annotates referring expressions by humans; RicoSCA (Li et al., 2020a) generate a larger synthetic referring expression dataset; and Li et al. (2020b) collect human-labeled captions of UI elements. They have been primary resources for GUI grounding for a long time (Li & Li, 2022; Banerjee et al., 2023). Later on, Qian et al. (2024) synthesize referring expressions from Rico by heuristic rules and train a vision language model by a new layout-aware contrastive learning technique. CogAgent (Hong et al., 2024) compiles HTML documents and screenshots from real websites to GUI grounding data for the pretraining stage, and finetunes on open-source and in-house human-labeled data, to build a 19B MLLM with strong pixel-level GUI grounding capabilities. Ferret-UI (You et al., 2024) develop a UI generalist MLLM trained on a series of UI-related tasks including grounding. The most similar effort to ours is SeeClick (Cheng et al., 2024), which enhances Qwen-VL (Bai et al., 2023) by finetuning on GUI grounding data, including simplistic synthetic data compiled from real websites. It still falls shorts of the small image resolution of Qwen-VL, as well as the simplistic nature of the training data. Cheng et al. (2024) also create a new grounding benchmark for GUIs, which benefits our evaluation and analysis.

B EXAMPLES

B.1 MULTIMODAL-MIND2WEB

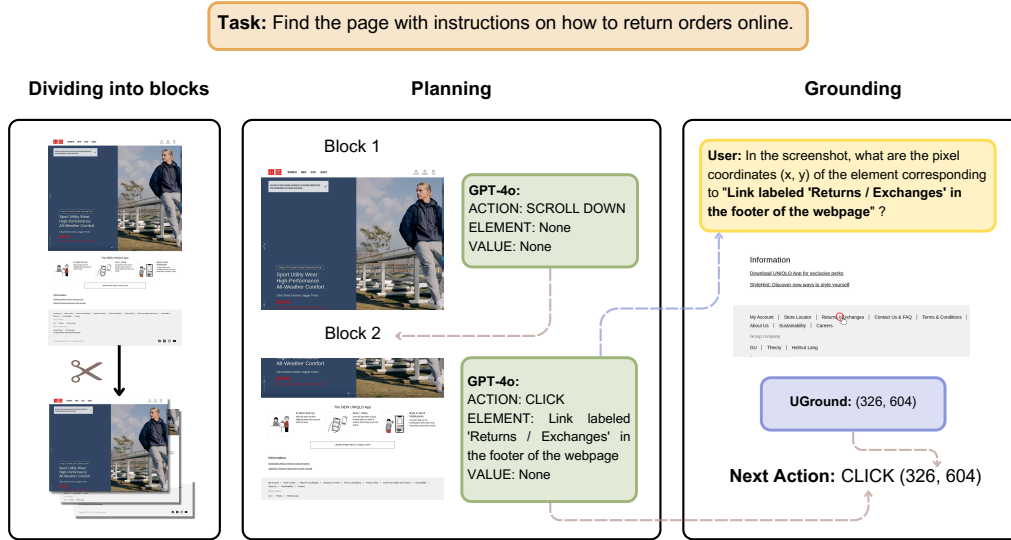


Figure 1: Example of the Mind2Web Evaluation Pipeline.

B.2 ANDROIDCONTROL

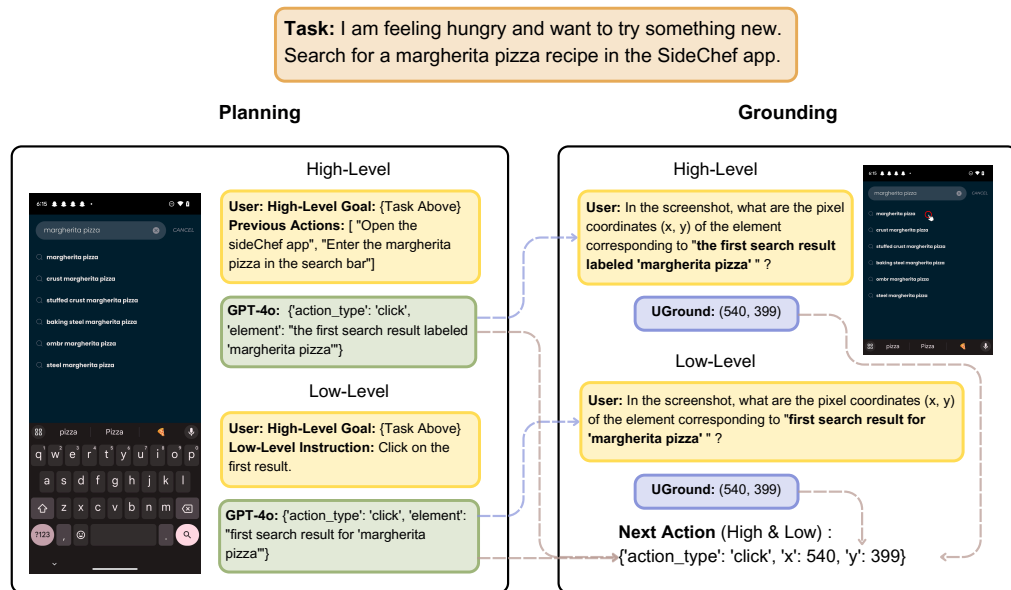


Figure 2: Example of the AndroidControl Evaluation Pipeline .

B.3 OMNIACT

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

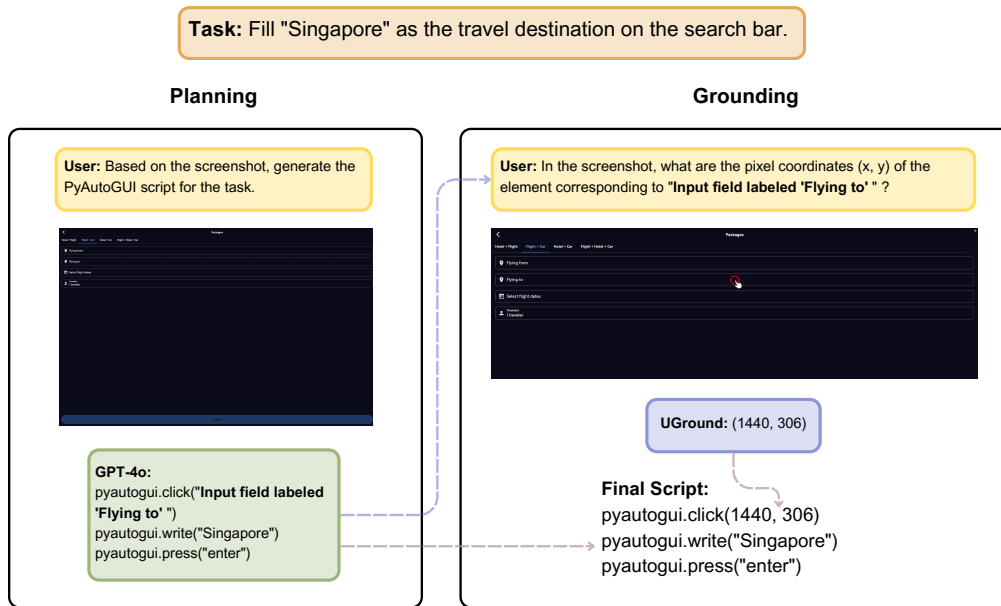


Figure 3: OmniACT Evaluation Pipeline Example.

B.4 TRAINING DATA

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

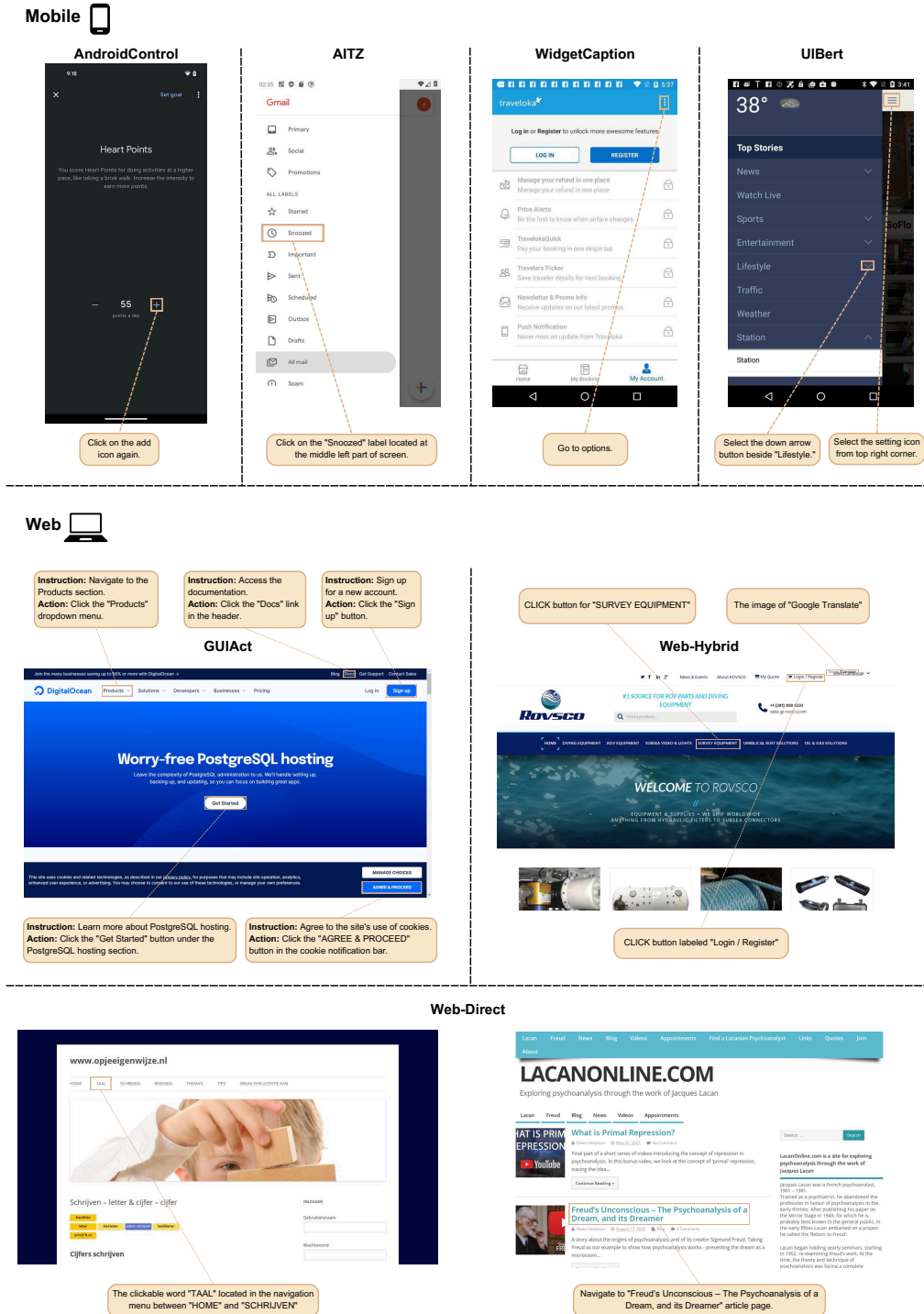


Figure 4: Examples of Training Data from different sources.

C DATA CONSTRUCTION

We describe the details of our data construction in this section. Illustrative examples of all our training data are given in Figure 4.

C.1 WEB-HYBRID

Following prior work (Hong et al., 2024; Cheng et al., 2024), we download and randomly sample from the latest Common Crawl¹. We apply few filtering methods, excluding non-webpage files based on URLs, and removing non-English pages based on labels of language provided by Common Crawl. We use Playwright to load and render the webpages, capturing screenshots and collecting metadata for web elements. The metadata includes bounding box coordinates and potentially useful HTML attributes, such as the element’s tag, text (`inner_text`), and alternative text (e.g., `alt`).

During the rendering process with Playwright, we randomly apply different image sizes to cover a wide range of resolutions and aspect ratios. Specifically, approximately one-third of the data uses mobile-friendly aspect ratios, where the webpages are rendered in mobile web mode. By doing this, some of the websites automatically switch to their mobile versions, which helps improve the coverage of mobile UI environments. For each long webpage, we randomly sample a block of content within a viewport-sized area to ensure diversity in the captured content.

As detailed in §2.2, we employ a hybrid strategy to generate referring expressions (REs) for webpage elements. Below, we firstly describe how we leverage MLLMs (LLaVA-NeXT-13B) and LLMs (Llama-3-8B) to generate concise, element-level descriptions without positional or contextual information.

We extract the bounding box regions from the webpage screenshots corresponding to the elements and pass these smaller cropped element images along with their main HTML attributes into LLaVA. Using the prompts outlined below, we prompt LLaVA to generate an element description based on its internal knowledge, the element’s image, and relevant HTML attributes:

Based on the attached image of a web element, please provide a short description of the web element displayed. The goal is to capture the intuitive and visual appearance of the element. Use the accompanying HTML information as context but focus more on describing what is visually observable. Avoid directly referencing HTML attributes; instead, interpret their possible visual implications if they can be inferred from the image. Be cautious of potential inaccuracies in the HTML attributes and use them to enhance understanding only when they align reasonably with what can be inferred visually.

HTML: {A list of salient HTML attributes}

We observe that since the input to LLaVA is a small cropped image, the model tends to have less hallucinations compared to directly caption an element with a bounding box overlaid in the image. However, due to the limited language capabilities of 13B LLaVA, it often generates lengthy interpretations rather than concise referring expressions.

To address this, we then pass the generated description to Llama-3-8B, using the following prompt to instruct it to condense the interpretation into a brief referring expression:

Here is a description to an element in a webpage. Using the detailed description provided, create a concise phrase that captures the essential visual and functional characteristics of the web element. The rephrased description should be straightforward, simple and precise enough to allow humans quickly spot this element in a webpage screenshot. Focus on the most prominent visual features and any critical function indicated by the text.

Description: {}

Leave only your final description in the answer, without any explanation.

Next, we describe the generation process for each crawled element.

¹CC-MAIN-2023-50

We begin by categorizing the webpage elements based on their tags into two groups: interactive elements (e.g., a, input, select, etc.) and pure text elements (e.g., p, h1, h2, etc.). Referring expressions are only generated for interactive elements, while pure text elements are used as potential sources for generating referring expressions. The primary reason is that interactive elements are the main targets in GUI grounding tasks. Additionally, the bounding boxes of certain pure text elements, as crawled, tend to have a few mismatches, which can introduce noise into the dataset.

For each interactive element, we first apply an OCR model (EasyOCR²) to extract text from the element’s bounding box. If the similarity between the OCR-extracted text and the element’s inner-text exceeds a threshold of 0.7, we treat the element as a textual element and skip the MLLM-based synthesis pipeline. This helps us avoid generating trivial data, such as “Gray links labeled by *link text*”. Furthermore, for textual elements, we filter out those that share identical text with other elements on the same page to prevent grounding ambiguities, which could arise when multiple elements share the same label within a single screenshot.

Based on handwritten rules, we label each element’s neighboring elements in various directions (multiple neighbors are allowed), mark the nearest upper h1, h2, or h3 elements (titles), and determine its absolute position (e.g., center of the screenshot, top, top-left corner) to generate absolute position-based referring expressions. We randomly select 0-2 relative elements in different directions and randomly pick elements whose distance from the target is within 500 pixels (empirically, always selecting the closest element does not yield the best performance). These are used to generate relative position descriptions. Some of the relative descriptions are further randomly modified to common terms such as “next to” or “between”. For contextual references, we create the following rules: If an element is detected as a checkbox or radio button based on its HTML properties, we assume it has a corresponding label (e.g., “radio button for Yes”). If such labels are provided in the HTML attributes, we use them directly; otherwise, we select the nearest element on the same row as the label (or the nearest element in the same column if none exist in the same row). Similarly, we add potential labels for input fields and select boxes. We then generate expressions like “under”, “in”, or “under section A” based on the hierarchical structure of the titles (primarily h1, h2, and h3). If an element has title, alt, or aria-label attributes, they are always utilized as potential descriptors, typically covering both visual and functional REs, with most being functional REs.

Finally, for each element, we randomly combine any descriptors (from accessibility labels, the element’s own text, or MLLM-based descriptions) with absolute position descriptions (randomly included, not always) and randomly add 0-2 relative or contextual descriptions (for radio buttons or similar elements, the label is always included; in other cases, 0-2 descriptors are randomly added) to generate the final referring expression. For each webpage, we use up to 100 elements. When randomly selecting elements, we prioritize those with accessibility labels or those annotated by MLLMs. We limit the total number of pure text elements to no more than three times the sum of elements with accessibility labels and MLLM-annotated elements (with a minimum of 10, or the actual number of available elements, whichever is lower) to reduce the number of pure text elements. We also count the occurrences of all unique accessibility labels and their respective frequencies. In total, our training set contains approximately 1.9M unique accessibility labels. For labels appearing more than 1,000 times, we downsample them to appear only 1,000 times in the training set. For example, the label “Next” appears 13K times, but is downsampled to 1K occurrences in our training data.

C.2 WEB-DIRECT

For the Web-Direct dataset, we directly employ GPT-4o to generate referring expressions. We observed that, due to its limited grounded understanding capabilities, simply enclosing an element in the image with a bounding box often leads to notable hallucinations, even with GPT-4o. This is especially prevalent when it provides descriptions of nearby elements. However, we aim to avoid the high cost of manual post-verification. Through empirical studies, we found that highlighting an element with an arrow, in addition to a bounding box, helps mitigate hallucinations. Therefore, for each webpage and any arbitrary element on the page, we annotate the element with a red bounding box and a red arrow pointing to it. Additionally, we explicitly ask GPT whether it can identify the element, which further reduces potential hallucinations and helps filter out a small number of crawling errors or occluded elements.

We use the following prompt to generate free-form referring expressions:

Here is supposed to be an interactive element (button, link, dropdown, text box etc) in the red box pointed by an arrow in the screenshot. Can you find it? Is it visible from the screenshot? Can you write a concise



Figure 5: Example of the image annotations of a bounding box and an arrow

²<https://github.com/JaidedAI/EasyOCR/>

description that is sufficient for humans to locate it from the screenshot? Your response should be a json. For example, “visible”: true, “description”: “your description here”.

Furthermore, we use the following prompt to generate more functionally oriented referring expressions:

Here is supposed to be an interactive element (button, link, dropdown, text box etc) in the red box pointed by an arrow in the screenshot. Can you find int? Is it visible from the screenshot? What unique function does this element enable? Your response should be a json. For example, “visible”: true, “action”: “subscribe the latest updates”.

C.3 OPEN-SOURCE DATA

We leverage several high-quality open-source referring expression datasets in Android, as well as GUIAct, as an additional source of web data. Specifically:

1. **GUIAct**: We use the annotated data from GUIAct (web-single). Any steps that do not involve coordinates or are marked as multi-step operations (for example, “click ... then type”) are filtered out. We use both the *Instruction* and *Action* annotations for grounding (i.e., each element is seen in training twice with different expressions).
2. **AndroidControl**: Similarly, we use the human-annotated actions from the training set. We filter out any actions that do not have associated coordinate data, ensuring that only steps with specific visual grounding targets are included in the dataset.
3. **Widget Caption**: For each element in the training set, multiple functional captions are provided. To enhance data variety, we randomly select two captions per element from the available set of functional captions during data construction.
4. **UIBert**: We use the training set elements from UIBert without any additional special processing, directly utilizing the referring expressions provided by this dataset.
5. **AITZ**: We incorporate the annotated actions (Thought) from AITZ, using each step’s action annotation for grounding in the dataset. These annotations contribute to a more diverse set of referring expressions, particularly for action-oriented grounding tasks.

D MODEL AND TRAINING DETAILS

D.1 OVERVIEW

For flexible investigation about the model architecture, we build the architecture based on LLaVA-NeXT (Liu et al., 2024b), and train from scratch using opensource data from Liu et al. (2024a). We use CLIP-ViT-L-14 (224px) as our base image encoder for more flexible splitting of AnyRes, and freeze it during training. We use Vicuna-1.5-7b-16k (Zheng et al., 2023) as the language backbone as a long-context LM backbone for handling long visual contexts.

D.2 ANYRES

As described in §2.3, AnyRes allows convenient scaling up of image resolutions, although it’s not always beneficial to enlarge image resolutions (Li et al., 2024a). We keep the main pipeline of AnyRes, splitting images into 224px grids. However, to keep identical image aspect ratios, we resize only by width and pad to the bottoms if needed, and use pixel-level coordinates in numbers that are compatible with this design. We allow at most 36 grids, for a maximum resolution of 1344*1344 and 896*2016. We empirically find AnyRes does not generalize to unseen image resolutions for visual grounding. Therefore, we resize images by width to keep them within the training resolution ranges when needed. We remove the low-resolution image for providing global context, because it intuitively does not providing informative contexts when images are larger than 1000px, and we empirically find it slightly hurt the performance.

D.3 TRAINING

Our training primarily consists of two stages:

1. **LLaVA-1.5 Pretraining and Finetuning:** We follow the exact pretraining in Liu et al. (2024a). Then, in the instruction finetuning stage, we change the grounding data from normalized coordinates to absolute coordinates as we wish, and start to use our modified AnyRes setting.

2. **GUI Visual Grounding:** Then we train UGround on our training datasets.

Due to the huge computation cost of handling high resolution images, we use LoRA (Hu et al., 2022) for instruction finetuning in the two stages, with a device batch size of 4.

The first stage takes about 50 hours on a single 4x NVIDIA A100 machine. And for the large scale GUI data training, we use 112 NVIDIA H100 GPUs and finish the training in about 6 hours.

E EVALUATION DETAILS

E.1 MODEL ENDPOINTS

As studied in (Pan et al., 2024), MLLM endpoints could pose slight effects to the performance of GUI tasks. Hence, we provide the specific endpoint names we use in our evaluation, as well as those of the baselines we use (if available).

- Ours (across every benchmark): gpt-4-turbo-2024-04-09 and gpt-4o-2024-05-13
- Multimodal-Mind2Web: gpt-4-1106-vision-preview
- OmniACT: gpt-4-0613 and gpt-4-1106-vision-preview
- Mind2Web-Live: gpt-4-0125-preview and gpt-4o-2024-05-13
- AndroidWorld: gpt-4-turbo-2024-04-09

E.2 MULTIMODAL-MIND2WEB

Many screenshots in Multimodal-Mind2Web have giant vertical heights (e.g., $1,280 \times 10,000$ pixels). Similar to Zheng et al. (2024), to avoid the overly long screenshots, we divide whole webpage screenshots into viewport-sized blocks, and simulate scrolling down to the next block when agents either determine that no valid action can be taken or explicitly choose to scroll. Specifically, we divide each full-page screenshot into $1,280 \times 1,000$ pixel blocks, except for the final block, which may be shorter depending on the page’s total height. Most of the target elements are within the first block (about 80%). See Figure 1 for an illustrative example of the pipeline.

We report element accuracy on the benchmark, and an element grounding is considered to be correct if the output coordinates fall in the box coordinates of the ground truth element.

E.3 ANDROIDCONTROL

We adopt the M3A (Multimodal Autonomous Agent for Android) prompt (Rawles et al., 2024), the state-of-the-art zero-shot method in Li et al. (2024b). We make minor modifications to integrate UGround to M3A.

We follow the standard data processing steps outlined in Li et al. (2024b). During evaluation, coordinates generated by grounding models are translated to the smallest visible element that includes the coordinates.

E.4 OMNIACT

We follow the method in Kapoor et al. (2024) for prompt design and the selection of five in-context examples. The prompt is slightly modified to generate element descriptions as function parameters for PyAutoGUI scripts, instead of directly outputting coordinates. The examples are selected based on task similarity using MiniLM (Wang et al., 2020) (MiniLM-L12-H384) embeddings. After generating the PyAutoGUI script with element descriptions, we use grounding models to predict the corresponding coordinates and substitute them back into the original script. See Figure 3 for an illustrative example of the pipeline.

The evaluation metrics in OmniACT include Sequence Score and Action Score. The Sequence Score measures whether the predicted action sequence exactly matches the ground truth. If the sequence is correct, the Action Score is calculated by penalizing inaccuracies in how the script performs the task, including: *Click Penalty* (M_p), applied when predicted coordinates fall outside the target element’s bounding box; *Key Penalty* (K_p), triggered when the predicted key set differs from the ground truth for press and hotkey actions; and *Write Penalty* (W_p), which measures discrepancies between the predicted and actual typed text using the BLEU score.

We compare our method with DetACT (Kapoor et al., 2024), the state-of-the-art method in Kapoor et al. (2024), which extracts UI elements and their coordinates through a combination of OCR, icon matching, and color

1242 detection. These elements are filtered by task relevance and passed to LLMs or MLLMs to generate the
1243 PyAutoGUI script. In contrast, our method does not use a pre-generated elements list. The planner model
1244 focuses on generating precise element descriptions based solely on the screenshot. Additionally, we corrected
1245 basic errors in the public evaluation scripts (for example, wrong file paths and wrong calculation of distances).

1246 E.5 MIND2WEB-LIVE

1247 The baseline agent in Pan et al. (2024) is text-only, perceives and interacts with webpages by hundreds of textual
1248 HTML elements at a time. To study vision-only agents, we change the observation to pure screenshots. We also
1249 make necessary changes to the standard action space to entirely isolate HTML from the planning, grounding, and
1250 execution: 1) We add `Scroll_Up` and `Scroll_Down` to the action space to better support vision-only agents with
1251 viewport-sized observation. 2) We remove `Fill_Form` and `Fill_Search` from the action space, which use an
1252 additional judgment model to determine whether to press enter after typing through HTML information. Instead,
1253 we use `Type` and `Press_Enter` to let the agent make its own decisions autonomously. 3) We disable API-based
1254 `Select`, and force agents to select options merely through clicking and makes the action more challenging. We
1255 admit some select buttons cannot be easily operated with only `Click`. We compromise this point to fulfill the
1256 motivation of this study.

1257 E.6 ANDROIDWORLD

1258 We build SeeAct-V agents based on the M3A agent in Rawles et al. (2024), which receives both raw and SoM
1259 images, and reason about the next action in a ReAct style (Yao et al., 2023) and choose the next target element
1260 from the element list. It also adopts self-reflection (Shinn et al., 2024) in the agent pipeline to instruct agents to
1261 summarize the current move and facilitate the following steps.

1262 We mainly remove SoM images and textual list of elements from a lly tree in the observation (in both planning
1263 and reflection phases), and change element-based actions to pixel-level actions.

1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

1296 F PROMPTS

1297

1298

1299

1300

Table 1: Prompt used for the planning model in **Multimodal-Mind2Web**, modified from the prompt in (Zheng et al., 2024)

1301

System Role

1302

You are imitating humans doing web navigation for a task step by step.

1303

At each stage, you can see the webpage like humans by a screenshot and know the previous actions before the current step through recorded history.

1304

You need to decide on the first following action to take.

1305

1306

You can click an element with the mouse, select an option, type text with the keyboard, or scroll down.

1307

1308

Task Description

1309

You are asked to complete the following task: {Task description}

1310

Previous Actions: {List of previous actions, if any}

1311

The screenshot below shows the webpage you see.

1312

Useful Guidelines

1313

First, observe the current webpage and think through your next step based on the task and previous actions.

1314

1315

To be successful, it is important to follow the following rules:

1316

1. Make sure you understand the task goal to avoid wrong actions.

1317

2. Ensure you carefully examine the current screenshot and issue a valid action based on the observation.

1318

3. You should only issue one action at a time.

1319

4. The element you want to operate with must be fully visible in the screenshot. If it is only partially visible, you need to SCROLL DOWN to see the entire element.

1320

1321

5. The necessary element to achieve the task goal may be located further down the page. If you don't want to interact with any elements, simply select SCROLL DOWN to move to the section below.

1322

1323

1324

Reasoning

1325

Explain the action you want to perform and the element you want to operate with (if applicable).

1326

Describe your thought process and reason in 3 sentences.

1327

Output Format

1328

Finally, conclude your answer using the format below.

1329

Ensure your answer strictly follows the format and requirements provided below, and is clear and precise.

1330

The action, element, and value should each be on three separate lines.

1331

1332

ACTION: Choose an action from CLICK, TYPE, SELECT, SCROLL DOWN. You must choose one of these four, instead of choosing None.

1333

1334

1335

ELEMENT: Provide a description of the element you want to operate. (If ACTION == SCROLL DOWN, this field should be none.)

1336

It should include the element's identity, type (button, input field, dropdown menu, tab, etc.), and text on it (if applicable).

1337

Ensure your description is both concise and complete, covering all the necessary information and less than 30 words.

1338

1339

If you find identical elements, specify its location and details to differentiate it from others.

1340

1341

VALUE: Provide additional input based on ACTION.
The VALUE means:

1342

If ACTION == TYPE, specify the text to be typed.

1343

If ACTION == SELECT, specify the option to be chosen.

1344

Otherwise, write 'None'.

1345

1346

1347

1348

1349

1350 Table 2: Prompts used for the planning model in **AndroidControl**, modified from the prompt in (Li
 1351 et al., 2024b) and (Rawles et al., 2024)

1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403

General Instruction

You are an agent who can operate an Android phone on behalf of a user.

Based on user’s goal/request, you may complete some tasks described in the requests/goals by performing actions (step by step) on the phone.

When given a user request, you will try to complete it step by step. At each step, you will be given the current screenshot and a history of what you have done (in text). Based on these pieces of information and the goal, you must choose to perform one of the action in the following list (action description followed by the JSON format) by outputting the action in the correct JSON format.

- If you think the task has been completed, finish the task by using the status action with complete as goal_status: {"action_type": "status", "goal_status": "successful"}

- If you think the task is not feasible (including cases like you don’t have enough information or cannot perform some necessary actions), finish by using the 'status' action with infeasible as goal_status: {"action_type": "status", "goal_status": "infeasible"}

- Click/tap on an element on the screen, describe the element you want to operate with: {"action_type": "click", "element": <target_element_description>}

- Long press on an element on the screen, similar with the click action above: {"action_type": "long_press", "description": <target_element_description>}

- Type text into a text field: {"action_type": "type_text", "text": <text_input>, "element": <target_element_description>}

- Scroll the screen in one of the four directions: {"action_type": "scroll", "direction": <up, down, left, right>}

- Navigate to the home screen: {"action_type": "navigate_home"}

- Navigate back: {"action_type": "navigate_back"}

- Open an app (nothing will happen if the app is not installed): {"action_type": "open_app", "app_name": <name>}

- Wait for the screen to update: {"action_type": "wait"}

Useful Guidelines

Here are some useful guidelines you need to follow:

General:

- Usually there will be multiple ways to complete a task, pick the easiest one. Also when something does not work as expected (due to various reasons), sometimes a simple retry can solve the problem, but if it doesn’t (you can see that from the history), SWITCH to other solutions.

- If the desired state is already achieved (e.g., enabling Wi-Fi when it’s already on), you can just complete the task.

Action Related:

- Use the 'open_app' action whenever you want to open an app (nothing will happen if the app is not installed), do not use the app drawer to open an app unless all other ways have failed.

- Use the 'type_text' action whenever you want to type something (including password) instead of clicking characters on the keyboard one by one. Sometimes there is some default text in the text field you want to type in, remember to delete them before typing.

- For 'click', 'long_press' and 'type_text', the element you pick must be VISIBLE in the screenshot to interact with it.

- The 'element' field requires a concise yet comprehensive description of the target element in a single sentence, not exceeding 30 words. Include all essential information to uniquely identify the element. If you find identical elements, specify their location and details to differentiate them from others.

- Consider exploring the screen by using the 'scroll' action with different directions to reveal additional content.

- The direction parameter for the 'scroll' action specifies the direction in which the content moves and opposites to swipe; for example, to view content at the bottom, the 'scroll' direction should be set to 'down'.

Text Related Operations:

Continued on the next page

Table 2 – Continued from the previous page

1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

- Normally to select certain text on the screen: (i) Enter text selection mode by long pressing the area where the text is, then some of the words near the long press point will be selected (highlighted with two pointers indicating the range) and usually a text selection bar will also appear with options like 'copy', 'paste', 'select all', etc. (ii) Select the exact text you need. Usually the text selected from the previous step is NOT the one you want, you need to adjust the range by dragging the two pointers. If you want to select all text in the text field, simply click the 'select all' button in the bar.

- At this point, you don't have the ability to drag something around the screen, so in general you can not select arbitrary text.

- To delete some text: the most traditional way is to place the cursor at the right place and use the backspace button in the keyboard to delete the characters one by one (can long press the backspace to accelerate if there are many to delete). Another approach is to first select the text you want to delete, then click the backspace button in the keyboard.

- To copy some text: first select the exact text you want to copy, which usually also brings up the text selection bar, then click the 'copy' button in bar.

- To paste text into a text box, first long press the text box, then usually the text selection bar will appear with a 'paste' button in it.

- When typing into a text field, sometimes an auto-complete dropdown list will appear. This usually indicating this is a enum field and you should try to select the best match by clicking the corresponding one in the list.

High-Level Prompt

{General Instruction}
The current user goal/request is: {High-level goal}
Here is a history of what you have done so far: {History}

The current raw screenshot is given to you.
{Useful Guidelines}

Now output an action from the above list in the correct JSON format, following the reason why you do that. Your answer should look like:

Reason: ...
Action: {"action_type": ...}

Your Answer:

Low-Level Prompt

{General Instruction}
The user's high-level goal/request is: {High-level goal}
The current next step's low-level goal is: {Low-level goal}

The current raw screenshot is given to you.
{Useful Guidelines}

Now output an action from the above list in the correct JSON format, following the reason why you do that. Your answer should look like:

Reason: ...
Action: {"action_type": ...}

Your Answer:

1458 Table 3: Prompt used for the planning model in **OmniACT**, modified from the prompt in (Kapoor
 1459 et al., 2024)

1461 **General Instruction**

1462 You are an excellent robotic process automation agent who needs to generate a PyAutoGUI script for
 1463 the tasks given to you.
 1464 You will receive some examples to help with the format of the script that needs to be generated.

1465 There are some actions that require you to provide an element description for the elements you want
 1466 to operate on. For the description, follow the requirements below:
 1467 Element Description Requirements:
 1468 Provide a concise description of the element you want to operate.
 1469 It should include the element’s identity, type (button, input field, dropdown menu, tab, etc.), and text
 1470 on it (if have).
 1471 If you find identical elements, specify their location and details to differentiate them from others.
 1472 Ensure your description is both concise and complete, covering all the necessary information and less
 1473 than 30 words, and organize it into one sentence.

1474 [IMPORTANT!!] Stick to the format of the output scripts in the example.
 1475 [IMPORTANT!!] Use only the functions from the API docs.
 1476 [IMPORTANT!!] Follow the output format strictly. Only write the script and nothing else.

1477 **API Reference**

1478 Here is the API reference for generating the script:

```
1479 def click(element=description):
1480     """Moves the mouse to the element corresponding to the description and performs a left click.
1481     Example:
1482     High Level Goal: Click at the rectangular red button labeled "Next".
1483     Python script:
1484     import pyautogui
1485     pyautogui.click("Rectangular red button labeled "Next" ")
1486     """
1487     pass
```

```
1488 def rightClick(element=description):
1489     """Moves the mouse to the element corresponding to the description and performs a right click.
1490     Example:
1491     High Level Goal: Right-click at link labeled "vacation rentals"under the "housing"section.
1492     Python script:
1493     import pyautogui
1494     pyautogui.rightClick("Link labeled "vacation rentals"under the "housing"section")
1495     """
1496     pass
```

```
1497 def doubleClick(element=description):
1498     """Moves the mouse to the element corresponding to the description and performs a double click.
1499     Example:
1500     High Level Goal: Double-click at folder named "courses".
1501     Python script:
1502     import pyautogui
1503     pyautogui.doubleClick("Folder named "courses" ")
1504     """
1505     pass
```

```
1506 def scroll(clicks=amount_to_scroll):
1507     """Scrolls the window that has the mouse pointer by float value (amount_to_scroll).
1508     Example:
1509     High Level Goal: Scroll screen by 30.
1510     Python script:
1511     import pyautogui
1512     pyautogui.scroll(30)
1513     """
1514     pass
```

Continued on the next page

Table 3 – Continued from the previous page

```

1512
1513
1514 def hscroll(clicks=amount_to_scroll):
1515     """Scrolls the window that has the mouse pointer horizontally by float value (amount to scroll).
1516     Example:
1517     High Level Goal: Scroll screen horizontally by 30.
1518     Python script:
1519     import pyautogui
1520     pyautogui.hscroll(30)
1521     """
1522     pass
1523
1524 def dragTo(element=description, button=holdButton):
1525     """Drags the mouse to the element corresponding to the description with (holdButton) pressed. hold-
1526     Button can be 'left', 'middle', or 'right'.
1527     Example:
1528     High Level Goal: Drag the screen from the current position to recycle bin with the left click of the
1529     mouse.
1530     Python script:
1531     import pyautogui
1532     pyautogui.dragTo("Recycle bin with trash can shape", "left")
1533     """
1534     pass
1535
1536 def moveTo(element = description):
1537     """Takes the mouse pointer to the element corresponding to the description.
1538     Example:
1539     High Level Goal: Hover the mouse pointer to search button.
1540     Python script:
1541     import pyautogui
1542     pyautogui.moveTo("Request appointment button")
1543     """
1544     pass
1545
1546 def write(str=stringType, interval=secs_between_keys):
1547     """Writes the string wherever the keyboard cursor is at the function calling time with
1548     (secs_between_keys) seconds between characters.
1549     Example:
1550     High Level Goal: Write "Hello world" with 0.1 seconds rate.
1551     Python script:
1552     import pyautogui
1553     pyautogui.write("Hello world", 0.1)
1554     """
1555     pass
1556
1557 def press(str=string_to_type):
1558     """Simulates pressing a key down and then releasing it up. Sample keys include 'enter', 'shift', arrow
1559     keys, 'f1'.
1560     Example:
1561     High Level Goal: Press the enter key now.
1562     Python script:
1563     import pyautogui
1564     pyautogui.press("enter")
1565     """
1566     pass
1567
1568 def hotkey(*args = list_of_hotkey):
1569     """Keyboard hotkeys like Ctrl-S or Ctrl-Shift-I can be done by passing a list of key names to hotkey().
1570     Multiple keys can be pressed together with a hotkey.
1571     Example:
1572     High Level Goal: Use Ctrl and V to paste from clipboard.
1573     Python script:
1574     import pyautogui

```

Continued on the next page

Table 3 – Continued from the previous page

```
pyautogui.hotkey("ctrl", "v")
"""
pass
```

Examples

Here are some examples similar to the tasks you need to complete. However, these examples use coordinate format for actions like click, rightClick, doubleClick, moveTo, dragTo, instead of element description. You should only refer to the actions in these examples, and for the output format, stick to the content in the API reference. For example, do not output "pyautogui.click(100,200)", instead output "pyautogui.click("Gray Tools menu button with a downward arrow in the top right corner") ". Omit "import pyautogui", do not include any comments or thoughts. Your output should only contain the script itself.
{Example list}

Task Description

Based on the screenshot, generate the PyAutoGUI script for the following task: {Task description} You should list all the necessary steps to finish the task, which could involve multiple steps. Also, ensure simplifying your steps as much as possible, avoid dividing a single task into multiple steps if it can be completed in one.

Table 4: Prompt used for the planning model in **ScreenSpot (Agent Setting)**.**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 description}

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".
