UI-E2I-Synth: Advancing GUI Grounding with Large-Scale Instruction Synthesis

Xinyi Liu[1,2*†] **Xiaoyi Zhang**[1†] **Ziyun Zhang**[1,2*]
Yan Lu[1]
[1]Microsoft Research Asia
[2]School of Software and Microelectronics, Peking University

Abstract

Recent advancements in Large Vision-Language Models are accelerating the development of Graphical User Interface (GUI) agents that utilize human-like vision perception capabilities to enhance productivity on digital devices. Compared to approaches predicated on GUI metadata, which are platform-dependent and vulnerable to implementation variations, vision-based approaches offer broader applicability. In this vision-based paradigm, the GUI instruction grounding, which maps user instruction to the location of corresponding element on the given screenshot, remains a critical challenge, particularly due to limited public training dataset and resource-intensive manual instruction data annotation. In this paper, we delve into unexplored challenges in this task including element-to-screen ratio, unbalanced element type, and implicit instruction. To address these challenges, we introduce a large-scale data synthesis pipeline *UI-E2I-Synth* for generating varying complex instruction datasets using GPT-4o instead of human annotators. Furthermore, we propose a new GUI instruction grounding benchmark *UI-I2E-Bench*, which is designed to address the limitations of existing benchmarks by incorporating diverse annotation aspects. Our model, trained on the synthesized data, achieves superior performance in GUI instruction grounding, demonstrating the advancements of proposed data synthesis pipeline. The proposed benchmark, accompanied by extensive analyses, provides practical insights for future research in GUI grounding. We will release corresponding artifacts here.

1 Introduction

Graphical User Interface (GUI) agents are designed to understand and automate human instruction in the digital devices, promising a future where humans are liberated from repetitive tasks, thereby enhancing operational efficiency and productivity. The rise of Large Language Models (LLMs) (Achiam et al., 2023) made it feasible to generally understand human instructions and structured text data. Hence, early efforts (Gur et al., 2023; Zhou et al., 2023; Shaw et al., 2023; Deng

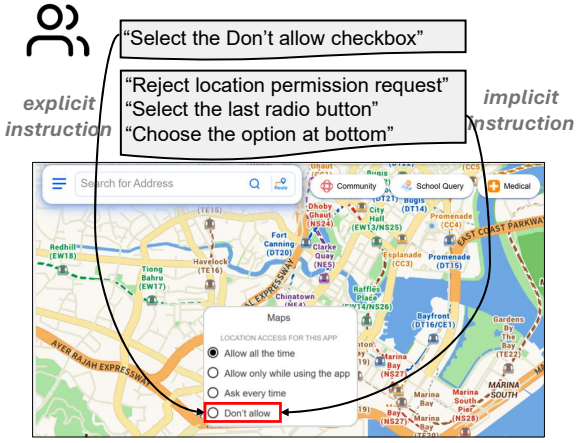


Figure 1: GUI grounding requires to localize elements in screenshots based on user instructions, with implicit instructions posing greater reasoning challenges than explicit ones.

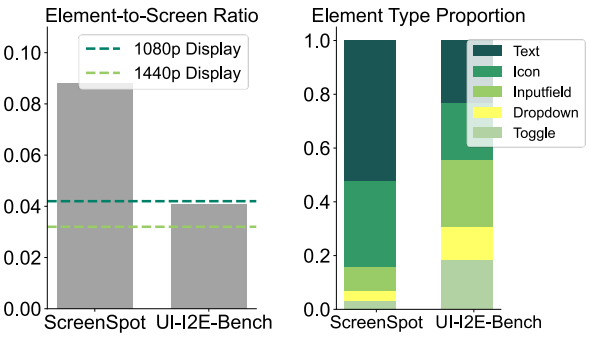


Figure 2: **Left:** Existing benchmark's element-to-screen ratio significantly deviates from typical real-world desktop displays like 1080p and 1440p, on landscape samples. **Right:** The comparison shows Text and Icon dominates existing benchmark, leaving the rest of element types omitted.

[†] Equal contribution. [*] Work done during the internship at Microsoft Research Asia.

et al., 2024) are made to operate GUI by understanding the text-rich metadata behind GUI, *e.g.*, HTML. However, some research (Fu et al., 2024; Zheng et al., 2024; WebAIM, 2024) has shown that GUI metadata is vulnerable and highly dependent on application developers for specific platforms. To overcome these limitations, more recent research (Zhang et al., 2023b; Cheng et al., 2024a; Gou et al., 2024; Wu et al., 2024; Lin et al.) has concentrated on training Vision-Language Models (VLMs) to mimic human behavior in GUI operations, manipulating GUI elements from pixels based on user instructions. This approach makes GUI grounding capability as the core, which maps the user instruction into the corresponding element by outputting specific coordinates.

Despite these works show satisfying progress on existing benchmark (Cheng et al., 2024b). Our investigation shows that there still is big gap between the existing benchmark and complicated and various GUI grounding environments. In this work, we thoroughly study the whole progress to build an element grounding dataset from screenshot collection to final instruction generation, revealing three main aspects that have not been fully explored previously: **(a) Element-to-screen ratio**: GUI instruction grounding necessitates higher resolution and smaller elements compared to natural scene object grounding. While previous studies (Cheng et al., 2024b; Gou et al., 2024) have discussed the screenshot resolution in training dataset, we argue that the key factor is the element-to-screen ratio, *i.e.*, the size of an element relative to the screenshot. This ratio is affected by both resolution and UI zoom level. As shown in the left of Figure 1, the landscape samples in existing benchmark has a larger element-to-screen ratio than typical real-world desktop displays like 1080p and 1440p, potentially leading to an overestimation of model performance. **(b) Unbalanced element type**: Different GUI element types exhibit diverse appearances and interaction designs, also with varying frequencies of occurrence. For instance, text buttons represent the most prevalent category, whereas checkboxes are comparatively less common. Unlike text buttons, whose functionality is often explicitly conveyed by their textual labels, checkboxes typically rely on surrounding elements to define their purpose. This distinction has not been adequately addressed in previous research. **(c) Implicit instruction**: In the context of GUI instruction grounding, users could convey their instructions based on their own intuitive understanding of element functionality or position, causing lack of direct correspondence with visible text on the screenshot, which we refereed as "implicit instruction". Figure 1 showcases the complexity of implicit instruction. The implicit instruction challenges the understanding and reasoning capability of VLMs.

To address the aforementioned challenges, we introduce a large-scale instruction grounding data synthesis pipeline, termed **E**lement-to-**I**nstruction **Synth**esis, abbreviated as ***UI-E2I-Synth***. This pipeline utilizes GPT-4o instead of human annotators to synthesize realistic grounding user instructions of varying complexity, which differs from the element referring expressions that previous works curated. By the principle of divide and conquer, *UI-E2I-Synth* decomposes grounding instruction synthesis into three subtasks and execute them step by step. Initially, we collect screenshot-metadata pairs from various sources at different resolutions, then using a heuristic GUI metadata parser to extract reliable element attributes. Next, With these high-quality attributes to alleviate hallucinations, both explicit and implicit referring expressions are prompted to generate from GPT-4o. In the final step, GPT-4o is employed again to simulate user behavior by generating specific action parameters. These action parameters are then combined with the REs to synthesize final realistic user instructions.

To more comprehensively evaluate model performance on GUI instruction grounding, we introduce a new benchmark ***UI-I2E-Bench*** that includes detailed annotations curated through a combination of our synthesis pipeline and human annotators. These annotations encompass various dimensions such as element type, element-to-screen ratio, and the level of instruction implicitness. Through rigorous evaluation on our proposed benchmark, we demonstrate that existing VLMs are still far from being satisfactory. To address the gap, we apply the proposed *UI-E2I-Synth* to collect a synthesized GUI grounding dataset comprising 9.9M instructions. Then following OS-Atlas (Wu et al., 2024) we fine-tune two different pre-trained VLMs on the curated dataset and evaluate them. The experimental results demonstrate that our models achieve advanced performance on both the existing benchmarks and our proposed one, with substantially less instruction data. Notably, our models show outstanding capabilities in comprehending implicit instructions and handling long-tailed element types.

These findings validate the efficacy of our data synthesis framework for GUI grounding, while our comprehensive benchmark analysis provides practical insights for future research in this domain.

2 Related work

Autonomous GUI Agent and Vison-Language Models. With the development of LLMs, researchers have built autonomous agents to perform complex reasoning tasks (Sumers et al.; Yang et al., 2024; Xi et al., 2025). Early works (Gur et al., 2023; Zhou et al., 2023; Shaw et al., 2023; Deng et al., 2024) have initiated the development of GUI agent frameworks for webpages and mobile platforms using GUI metadata. To address the limitations of unstable GUI metadata, *ResponsibleTA* (Zhang et al., 2023b) introduced a vision-based GUI agent framework and *SeeClick* (Cheng et al., 2024a) highlighted the importance of instruction grounding in downstream GUI agent tasks. While works like RUIG (Zhang et al., 2023a) improved GUI grounding via reinforcement learning, current general-purpose VLMs still lack satisfactory GUI grounding capabilities. Consequently, recent studies (Gou et al., 2024; Wu et al., 2024; Lin et al.) have focused on enhancing the GUI grounding abilities of VLMs. simultaneously, in industry, closed-source solutions (OpenAI, 2025; Anthropic, 2024) have advanced rapidly but with minimal detail disclosure. Our work from data perspective contributes by synthesizing large-scale GUI grounding instructions and is intended to benefit the open-source community.

Instruction Dataset Synthesis. Instruction synthesis (Ding et al., 2023; Wang et al., 2022; Xu et al., 2023) has emerged as a prevalent strategy in the field of large language models due to its effectiveness in reducing the need for human effort in labeling datasets of millions of samples. Recent works make preliminary exploration about GUI grounding data synthesis. *SeeClick* utilizes HTML tags to generate referring expressions for webpage elements. *UGround* improves on this by employing a hybrid pipeline that creates referring expressions using both external LLMs and VLMs. *OS-Atlas* derives instructions from sequential UI changes but faces scalability issues due to the need of screenshot traces. Our data synthesis pipeline differs from previous works in several key aspects, most notably in that prior studies have focused solely on element referring expressions rather than genuine user instructions. These existing approaches overlook the complex role that users play in formulating instructions and also screen-to-element ratio and long-tailed element type.

3 Approach

Model-based large-scale GUI grounding instruction synthesis is challenging for multimodal hallucinations and varying complexity of user instruction. *UI-E2I-Synth* includes three key steps: raw data collection and parsing, referring expression generation and instruction synthesis, as illustrated in Figure 3. These steps are executed sequentially, with each step designed to address specific challenge: extracting reliable element attributes to mitigate hallucinations, generating explicit and implicit referring expressions to diversify instruction difficulty, and synthesizing instructions with user action simulation. In the following subsections we introduce each step on details.

3.1 Raw Data Collection and Parsing

This component is designed to collect diverse screenshots from multiple platforms and obtain reliable element attributes from GUI metadata for the following referring expression generation.

We first build data collection infrastructure across various platforms to obtain the GUI screenshot and metadata pairs. For Web platform, we use the dumped webpage metadata in Common Crawl (Common Crawl, 2024) and re-render as webpage screenshots. For Windows platform, we select common applications, traverse the application to obtain different UI screenshots. Specifically, we start from the inital UI of a given application, parse out clickable buttons from its metadata and click these buttons to transit to other UI in this application.All the process is conducted under a virtual machine where no personal information is involved and all corresponding UIA data is recorded. For mobile platform, we leverage the existing metadata-screenshot pairs in training set of AndroidControl (Li et al., 2024). Different platforms yield different metadata formats, such as DOM (Mozilla, 2024) for Web, UIA (Microsoft, 2021) for Windows, VH (Google, 2024) for Android. To unify element representation from different platform metadata, we build corresponding heuristic element parser to extract only three key attributes including element type, element content and element bounding box. For element type, we

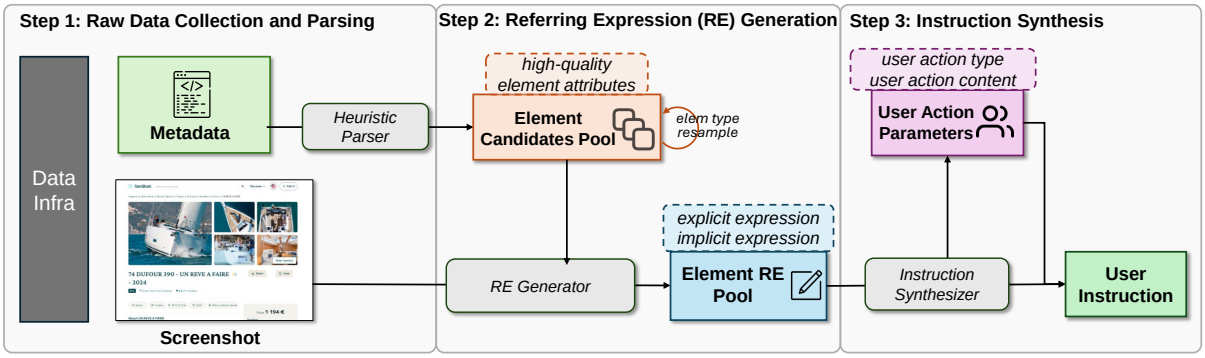


Figure 3: Overview of proposed data synthesis pipeline UI-E2I-Synth. The three steps respectively generate a high-quality pool of elements and their corresponding attributes, diverse element referring expressions, and user instructions. The output of the synthesis pipeline is format as <screenshot, user instruction, element coordinates>. GPT-4o is utilized in the component *Referring Expression Generator* and *Instruction Synthesis*.

analyze the UI element representation in different metadata format and categorize them into five main element types:

- **Text**: A button element with explicit text.
- **Inputfield**: An element requiring user input or content editing.
- **Dropdown**: A button that allows user input by selecting from provided options.
- **Icon**: A graphical button representing functionality through an image.
- **Toggle**: A two-state element, such as checkboxes, radio buttons, and switches.

Accurately parsing every element for each platform is extremely difficult, which means tons of rules need be constructed manually. However, GUI grounding data does not require perfect parsing for every element in a single screenshot. Instead, our goal is to extract sufficiently accurate element attributes from large-scale screenshot-metadata pairs. With this goal, we build our heuristic metadata parser which contains heuristic rules to only keep high precision but leaving recall aside.

After obtaining the three key element attributes, we measure the element type distribution, adjust the distribution through resampling, and compose a balanced candidate pool for the next step.

3.2 Referring Expression Generation

The Referring Expression Generation step is designed to enable exsiting Large VLM to generate expressions that are both reliable and diverse in difficulty. The element referring expressions (REs) denote a element description from a specific perspective, which is independent from user action. Generating elements and their descriptions from scratch can be challenging for Large VLMs, as they often struggle to produce accurate coordinates and are prone to hallucinations (Zheng et al., 2024) under high-resolution images. A straightforward method of generating element REs is using Set-of-Marks (Yang et al., 2023) to caption every marked element. However, this method also tends to result in severe hallucinations. To address this issue, we propose a attribute-enhanced style of RE generation. We supply a list containing the element types and element content obtained from the previous step, with Set-of-Marks screenshots as context. We then leverage GPT-4o as the Large VLM to first generate a full description that explains the function of each element and the expected outcome when interacting with it. Next, GPT-4o is prompted with generating two types of REs: explicit RE and implicit RE. We define explicit RE as directly referring to the obvious features of the element, while implicit RE refers to the element by describing the semantic function or its relationship with nearby elements, thus avoiding the obvious visible features. The generated explicit and implicit REs collectively form the element RE pool, which provides user action object for the final step.

3.3 Instruction Synthesis

The obtained referring expressions are only descriptions about element, omitting the user role in the progress. User instruction implies intention from user, which can be directly or implicitly related to the element. When directly asked to generate user instruction, we observed that Large VLM trends to generate generally instruction from a role of assistant, such as “Fill your details in the input field” or “Click to check your profile”, even we ask it to generate as a computer user’s first perspective view.

Benchmark	Sample Num	Landscape element-to-screen ratio	Fine-grained type annotation	Min type proportion	Implicitness annotation	Implicit instruction proportion
ScreenSpot	1,272	0.088	✗	3.21%†	✗	-
Our *UI-I2E-Bench*	1,477	**0.042**	✓	**12.34%**	✓	**63.03%**

Table 1: Comparison between proposed benchmark and existing benchmark. Screen-to-element ratio is calculated as the ratio of the square roots of the areas of the box and the image. † denotes we re-annotate ScreenSpot-Web with our fine-grained element type and calculate the number.

This phenomenon may arise because the output message role in the Large VLM is set to "assistant", preventing it from easily simulating the user role. Hence, here we propose to a parametrization way to generate user instruction. We decompose the wanted instruction into three parameters, user action type, user action content and element object. A prompt is utilized to instruct GPT-4o to simulate a user interacting with the current application, by generate specific user actions and content. Combining previous element referring expressions as the element object, GPT-4o is further instructed with synthesizing the final instruction with all these action parameters. To maintain diversity in the generated instructions, both explicit and implicit element referring expressions are employed to create distinct user instructions. To this end, the output of the whole synthesis pipeline is format as <*screenshot, user instruction, element coordinates*>.

With this pipeline, we synthesize a large-scale grounding instruction data to train a model to demonstrate our advancement. Dataset details are explained in Section 5.1.

4 *UI-I2E-Bench*: A Comprehensive Grounding Benchmark

As shown in Figure 1, the existing benchmark ScreenSpot exhibits a lower level of difficulty in element-to-screen ratio. Additionally, it lacks detailed element annotations, such as element type and implicitness, which are crucial for assessing the element grounding capabilities of models. To address these limitations, we introduce *UI-I2E-Bench*, developed through a semi-automated workflow to provide a more comprehensive evaluation. We first apply our data synthesis pipeline into data collected from multiple platforms, then we sampled the instructions by element-to-screen ratio and element type. We manually review the validness of parsed element and correctness of generated instruction to obtain the final *UI-I2E-Bench*, which includes 1477 grounding instructions from Web, Windows and Android. The further statistical data and comparisons with *ScreenSpot* are included in Table 1. Our benchmark offers more comprehensive annotations, a lower element-to-screen ratio, and a higher proportion of implicit instructions. These features facilitate in-depth performance diagnostics in challenging GUI instruction grounding cases.

5 Experiments

To demonstrate the effectiveness of our grounding instruction synthesis pipeline, we utilize UI-E2I-Synth to collect a GUI grounding training dataset and fine-tune a pre-trained VLM to develop UI-I2E-VLM. This section is structured as follows: First, we provide details on the curation of the training dataset and model training. Next, we introduce various baselines and evaluate the performance of UI-I2E-VLM against these baselines on two GUI instruction grounding benchmarks. We analyze the results on UI-I2E-Bench to reveal the shortcomings of current VLMs across multiple aspects. We then integrate UI-I2E-VLM with the GPT-4o planner and assess its performance on the live GUI agent benchmark OSWorld (Xie et al., 2024).

5.1 Training Details

Training data curation. In this subsection, we briefly describe the final curated dataset leveraging the *UI-E2I-Synth* pipeline while leaving more details in our supplementary materials.We apply our pipeline mainly on three platforms including Web, Windows and Android to derive this training dataset. The Web is our main data source due to the variety of layouts and design styles across websites, as well as the extensive quantity of webpages available in Common Crawl. We start by extracting webpages from the top 500k domains with the highest traffic for three webpages per domain. Next, we filter out non-English and error state webpages, resulting in final 724,839 webpages. These webpages are re-rendered at seven different resolutions, including one mobile resolution to simulate mobile scenarios and left six resolutions for landscape desktops. For Windows platform, we select 90 windows apps to totally collect 15K screenshot-metadata pairs. For Android platform, we leverage existing screenshot-metadata pairs in training

Model	Size	#Train	ScreenSpot				UI-I2E-Bench			ScreenSpot-Pro	*Avg.**
			Mobile	Web	Desktop	Avg.	Explicit	Implicit	Avg.		
InternVL2-4B	4B	-	7.2	0.5	4.5	4.2	1.4	0.5	0.9	0.3	*1.8*
Qwen2-VL-7B	7B	-	51.3	27.7	49.1	42.6	53.8	45.6	48.7	1.6	*31.0*
OmniParser	-	-	78.5	63.9	79.7	73.9	54.3	52.4	53.1	8.3	*45.1*
Seeclick	9.6B	1.0M	66.1	44.7	54.5	55.8	37.1	19.9	26.4	1.1	*27.8*
UGround	7B	9.7M	72.5	75.7	74.6	74.1	65.8	47.1	54.2	16.5	*48.3*
ShowUI	2B	2.7M	84.6	73.2	69.9	76.8	51.3	35.6	41.5	7.7	*42.0*
OS-Atlas-4B	4B	13.6M	73.3	73.4	61.1	70.1	51.5	39.9	44.3	3.7	*39.4*
OS-Atlas-7B	7B	13.6M	83.8	**83.1**	79.7	82.5	63.2	55.8	58.6	18.9	*53.3*
UI-I2E-VLM-4B	4B	9.9M	70.3	70.9	70.1	70.4	61.9	48.3	53.4	12.2	*45.3*
UI-I2E-VLM-7B	7B	9.9M	**86.5**	78.0	**82.6**	**82.5**	**72.0**	**67.9**	**69.5**	**23.6**	***58.5***

Table 2: Results on GUI grounding benchmarks. #Train denotes the instructions number in training data. We calculate the average accuracy as correct samples divided by total samples. *Avg.** denotes the arithmetic mean of average accuracy across three benchmarks.

Model	Size	#Train	Platform			Element Type				
			Web	Desktop	Mobile	Button	Icon	Dropdown	Input	Toggle
OmniParser	-	-	30.8	45.5	67.6	68.4	60.5	65.9	58.9	26.9
Seeclick	9.6B	1.0M	18.2	15.8	37.2	31.6	26.2	22.5	29.6	22.1
Uground	7B	9.7M	53.0	44.3	61.8	57.3	49.7	76.4	64.2	37.0
ShowUI	2B	2.7M	29.6	30.4	53.9	52.0	44.1	51.1	52.8	18.9
OS-Atlas-4B	4B	13.6M	54.6	19.9	58.6	43.5	44.1	46.6	46.3	42.2
OS-Atlas-7B	7B	13.6M	52.2	48.9	68.1	69.1	58.7	80.3	70.1	32.3
UI-I2E-VLM-4B	4B	9.9M	60.9	38.9	61.4	54.3	50.0	61.2	68.6	39.0
UI-I2E-VLM-7B	7B	9.9M	**62.1**	**64.0**	**76.2**	**77.0**	**68.2**	**84.8**	**86.2**	**44.4**

Table 3: Detailed results on UI-I2E-Bench. #Train denotes the instruction number in training data.

set of AndroidControl (Li et al., 2024). Then we run the *UI-E2I-Synth* pipeline on all the collected screenshot-metadata pairs to derive our synthsized dataset. Combined wuth the existing mobile GUI dataset MOTIF (Burns et al., 2022) and Widget-Caption (Li et al., 2020) for grounding purpose, we final training dataset includes 1.6M screenshots and 9.5M instructions. The detailed training data statistics are provided in AppendixA. Our data has passed our ethics review where personal information has been examined and removed.

Base models. Following OS-Atlas (Wu et al., 2024), We consider InternVL2-4B (Chen et al., 2023) and Qwen2-VL-7B (Wang et al., 2024) as our base models. The pretraining data of Qwen2-VL-7B incorporate GUI screenshots, whereas InternVL2 does not. Both models support multi-resolution image processing. We name our models as UI-I2E-VLM-4B and UI-I2E-VLM-7B, with training details provided in Appendix B.

5.2 GUI Instruction Grounding Evaluation

In this section, we compare UI-I2E-VLM with OmniParser (Lu et al., 2024), SeeClick (Cheng et al., 2024b), UGround (Gou et al., 2024), ShowUI (Lin et al.) and OS-Atlas (Wu et al., 2024) on multiple GUI grounding benchmarks. OmniParser is a screen parsing tool containing a series of models. We combine OmniParser with GPT-4o to perform GUI grounding as its original paper suggests. SeeClick, UGround, ShowUI and OS-Atlas are VLMs fine-tuned for GUI instruction grounding tasks. Following their inference setting, we prompt SeeClick, UGround and ShowUI to generate (x, y) coordinates. For OS-Atlas and UI-I2E-VLM, we prompt the model to generate a bounding box and use its center point as the predicted coordinate.

For the evaluation benchmarks, we not only include the proposed benchmark but also the popular ScreenSpot even a concurrent benchmark ScreenSpot-Pro. ScreenSpot is a popular GUI grounding benchmark consisting of 1,272 cross-platform samples. ScreenSpot-Pro is a very recent benchmark focusing on professional desktop applications with high resolution. We use accuracy as the evaluation metric, where a prediction is considered as a correct one if the coordinate falls within the ground-truth bounding box.

Main result comparison. We present a comprehensive performance comparison across the aforementioned three benchmarks to evaluate the models' cross-domain generalization capabilities. As demonstrated in Table 2, UI-I2E-VLM-7B achieves superior performance across all benchmarks, surpassing the previous state-of-the-art model, OS-Atlas-7B, with a 9.7% relative improvement in average performance. Notably, the robust improvement on all the three distinct benchmarks is attained with a training set containing only 72% of the instruction quantity used in OS-Atlas, which makes our curated data quality more impressive. For our UI-I2E-VLM-4B, despite its base model being pretrained without GUI-specific data, still achieves impressive performance compared to the models of similar parameter size. These findings highlight the effectiveness of our data synthesis pipeline.

By comparing the numerical results between ScreenSpot and UI-I2E-Bench, our findings indicate that the reported progress in GUI grounding capabilities has been substantially overestimated. A notable example is ShowUI, which demonstrates seemingly impressive performance metrics on ScreenSpot despite utilizing significantly fewer model parameters and training data. However, its performance drops markedly when evaluated on UI-I2E-Bench and ScreenSpot-Pro, revealing critical limitations in the complexity and representativeness of the original ScreenSpot benchmark.

Teardown diagnostic on UI-I2E-Bench. Leveraging the diverse annotation aspects provided in UI-I2E-Bench, we conducted a detailed teardown analysis of model performance on UI-I2E-Bench. We report the detailed performance on the three dimensions separately in Table 2 and 3. Compared to the explicit instruction category, we observe more significant improvement happening to the implicit ones. The most outstanding model, OS-Atlas-7B falls behind of us for 12.1 percentage on this dimension, which means previous works underrate on the complexity of instructions. Another interesting observation is that OmniParser shows a competitive result on implicit instruction with understanding abilities of GPT-4o. However, it falls behind on simpler explicit instructions. It suggests that the primary bottleneck of OmniParser lies in localizing smaller and long-tailed element.

To further study the impact of element-to-screen ratio in GUI grounding, we analyze the accuracy across different ranges of the element-to-screen ratio between our UI-I2E-VLM-4B and UGround, as shown in Figure 4. Grounding accuracy drops as the element size ratio decreases, highlighting the importance of benchmarks that emphasize smaller elements and higher-resolution images. Benefiting from our training data and larger number of input image tokens, UI-I2E-VLM achieves better performance on small elements.

Model	Mobile		Desktop		Web		Avg.
	Text	Icon	Text	Icon	Text	Icon	
OS-Atlas-Web	35.2	24.9	67.0	30.7	70.9	39.8	44.9
UI-E2I-Synth-Web	**89.0**	34.1	**93.3**	**42.9**	**79.6**	**44.7**	**65.8**
- Inst. Synth.	85.4	20.5	84.0	18.6	69.6	30.1	54.3(↓ 11.5)
- GPT-4o	58.6	**37.1**	52.1	37.9	46.1	44.2	46.9(↓ 18.9)

Table 4: Data curation ablation experiment on ScreenSpot. Different 500k instruction data is sampled to separately fine-tune and evaluate the same model.

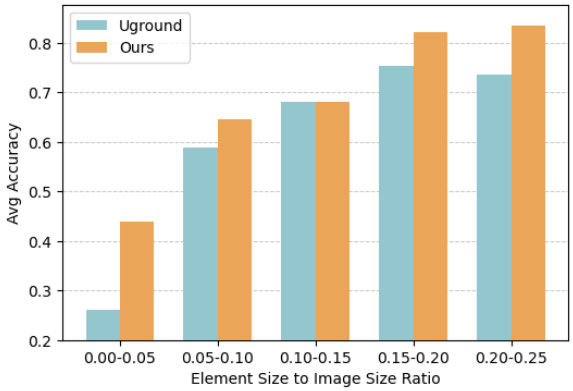


Figure 4: Accuracy comparison by the proportion of element size to image size, evaluated on UI-I2E-Bench.

In element type analysis of Table 3, we observe a significant performance gap in the Icon and Inputfield types compared to previous models, indicating that prior work has largely overlooked these long-tail categories. It supports our data synthesis pipeline to use a relatively balanced distribution of element types when curating the training dataset.

Data curation ablation study. To fairly evaluate the quality of our curated dataset and analyze components in our data synthesis pipeline, we compare our synthesized data with public released data from OS-Atlas (Wu et al., 2024) and variations from our own synthesis pipeline. Since OS-Atlas only constructs data on the web at a large scale, we also curate data on web domain. The comparison includes: (1) **OS-Atlas-Web**, the synthesized data from OS-Atlas dataset, derived from FineWeb (Penedo et al., 2024); (2) **UI-E2I-Synth**, 500k instructions sampled from the web portion of our final training dataset; (3) **- Inst. Synth.**, a variant that skips

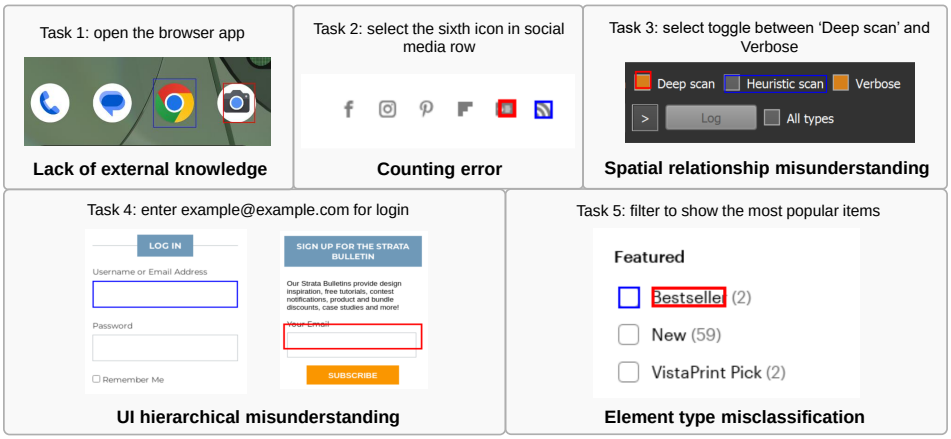


Figure 5: Typical failure cases in UI-I2E-Bench. Blue box denotes the correct elements while red ones are wrong prediction from our model.

Setting	OS	Office	Daily	Profess.	Workflow	Overall
Screenshot only*	8.33	2.64	6.54	0	2.06	3.62
+UGround	12.5	3.41	3.98	4.08	4.58	4.80
+UI-I2E-VLM	8.33	5.27	6.54	2.03	3.59	**5.12**

Table 5: Task success rate on OSWorld. *Report as the performance reproduced by us. In this experiment, the environment and settings are consistent.

the instruction synthesis step and uses referring expressions as instructions; and (4) **- GPT-4o**, a configuration using only raw element attributes as instructions. We sample 500k instruction from above configurations and fine-tuned the same InternVL2-4B model separately on each dataset and evaluated the trained models on the ScreenSpot benchmark. As shown in Table 4, the results demonstrate that our dataset yields significant performance improvement compared to OS-Atlas-Web. While this advantage diminishes with OS-Atlas larger training data in Table 2, it also demonstrate the effectiveness of our curated data. Furthermore, the performance degradation observed when eliminating instruction synthesis and GPT-4o components provides evidence for our UI-E2I-Synth design.

Failure case analysis. Figure 5 illustrates common errors of UI-I2E-VLM on UI-I2E-Bench, including: (1) failure to recognize icons without text due to limited knowledge, (2) incorrect positioning of elements within rows or columns, (3) misinterpretation of spatial relationships, (4) misunderstanding hierarchical relationships, and (5) misclassifying element types, such as confusing checkboxes with adjacent text.

5.3 Agent Task Online Evaluation

OSWorld. We evaluate our model on OSWorld (Xie et al., 2024), a live benchmark built in real computer environment for GUI agents. OSWorld consists of 369 open-ended computer tasks across various operating systems, including Ubuntu, Windows, and macOS. Following its original screenshot-only design, we use GPT-4o as the planner, providing only screenshots as the observation space. For every step, the planner is given the user task and the screenshot of current stage to predict the next action to take in next step. Then for the screenshot and the predicted action, we leverage VLMs to provide corresponding element coordinates to compare the effect of different VLMs. For screenshot only baseline, we use GPT-4o to directly generate element coordinates. The prompt used in evaluation is provided in Appendix F. The result comparison is shown in Table 5. The results show that UI-I2E-VLM helps to address GPT-4o's limitations in grounding and shows competitive performance, which prove our data synthesis pipeline can benefit the GUI agent development.

6 Conclusion

This work tries to identify and address critical challenges in GUI instruction grounding, such as element-to-screen ratio, unbalanced element types, and implicit instructions, through the introduction of a large-scale data synthesis pipeline, *UI-E2I-Synth*. Moreover, we propose a new GUI instruction grounding benchmark, *UI-I2E-Bench*, which overcomes the limitations of existing benchmarks by incorporating diverse annotation aspects. Our model, trained on the synthesized data, outperforms the previous state-of-the-art grounding models with less data and a smaller model size. We hope this work provides valuable insights and tools for the future development of GUI instruction grounding.

Limitations

While the introduction of a large-scale data synthesis pipeline (*UI-E2I-Synth*) has significantly improved the generation of extensive instruction datasets, there is still potential for further scaling the data size and model size. Increasing the dataset size could enhance the robustness and generalizability of the model, addressing more diverse and complex GUI scenarios. Another limitation is this work only focuses on English instruction, we hope to further apply our data synthesis pipeline on other lanugages.

Ethical considerations

Automated GUI agents can liberate human from repetitive tasks in digital devices but also could be used to launch robot attacks, such as brute force login attempts, automated spam, or denial-of-service attacks, by interacting with web interfaces at a speed and scale beyond human capability. They might be used to automate phishing attacks, where the agent mimics legitimate user interactions to steal sensitive information. It necessitates the maintaining transparency about the capabilities and limitations of GUI agents.

References

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

Anthropic. 2024. Claude 3 model card.

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. 2022. A dataset for interactive vision language navigation with unknown command feasibility. In *European Conference on Computer Vision (ECCV)*.

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

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

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

Common Crawl. 2024. Common crawl - open repository of web crawl data.

Wenliang Dai, Nayeon Lee, Boxin Wang, Zhuolin Yang, Zihan Liu, Jon Barker, Tuomas Rintamaki, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. Nvlm: Open frontier-class multimodal llms. *arXiv preprint arXiv:2409.11402*.

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

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

Jingwen Fu, Xiaoyi Zhang, Yuwang Wang, Wenjun Zeng, and Nanning Zheng. 2024. Understanding mobile gui: From pixel-words to screen-sentences. *Neurocomputing*, 601:128200.

Google. 2024. Profile your layout with hierarchy viewer.

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

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*.

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget captioning: Generating natural language description for mobile user interface elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5495–5510.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*.

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

Microsoft. 2021. Ui automation overview - .net framework.

Mozilla. 2024. Document object model (dom) - web apis - mdn web docs.

OpenAI. 2025. Operator system card.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina N Toutanova. 2023. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *Advances in Neural Information Processing Systems*, 36:34354–34370.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *Preprint*, arXiv:2409.12191.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

WebAIM. 2024. Webaim: The webaim million - the 2024 report on the accessibility of the top 1,000,000 home pages. Accessed on September 30, 2024.

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

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *Preprint*, arXiv:2310.11441.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zhizheng Zhang, Wenxuan Xie, Xiaoyi Zhang, and Yan Lu. 2023a. Reinforced ui instruction grounding: Towards a generic ui task automation api. *arXiv preprint arXiv:2310.04716*.

Zhizheng Zhang, Xiaoyi Zhang, Wenxuan Xie, and Yan Lu. 2023b. Responsible task automation: Empowering large language models as responsible task automators. *arXiv preprint arXiv:2306.01242*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

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

Appendix

The writing of this paper was limitedly assisted by AI, with refinement focused on language and style, and no original ideas generated by the AI.

A Data Statistics

We present a detailed summary of the statistics for the complete training dataset we used in Table 6. All the external dataset used are publicly available with Apache 2.0 License.

Dataset	Platform	#Screenshots	#Instructions
UI-E2I-Synth-Web	Web	1,536,200	9,097,736
UI-E2I-Synth-Desktop	Desktop	14,087	334,397
UI-E2I-Synth-AndroidControl	Mobile	40,199	109,126
MOTIF(Burns et al., 2022)	Mobile	30,699	320,219
WidgetCaption(Li et al., 2020)	Mobile	14,409	38,103
Total	-	1,635,594	9,899,581

Table 6: Grounding training datasets statistics.

We further analyzed the statistics of the synthetic data. In our synthetic data, the proportion of non-text elements is 23.0%. For comparison, we randomly sampled 100 elements from the SeeClick web data, manually labeled the non-text elements, and calculated their proportion. In SeeClick, non-text elements only account for 8.7%. Our dataset exhibits a more balanced distribution across different element types.

Additionally, we randomly sampled 1,000 elements from landscape screenshots in both datasets and calculated their element-to-screen ratio. The distribution in shown in Table 7. Compared to SeeClick, we constructed our dataset with more data for smaller and more challenging targets, in order to optimize the model's performance in high-resolution real-world scenarios.

Element-to-screen Ratio Range	Proportion (%) (SeeClick)	Proportion (%) (Ours)
0.00 - 0.02	11.49	36.92
0.02 - 0.04	43.30	40.43
0.04 - 1.00	45.21	22.65

Table 7: Element-to-screen ratio comparison between SeeClick web dataset and our synthetic web dataset.

B Training Details

UI-I2E-VLM-7B. To maintain consistency with the Qwen2-VL training data, we converted the format of the bounding boxes to `<|box_start|>(x1,y1), (x2,y2)<|box_end|>`, where (x1, y1) and (x2, y2) represent the coordinates of the upper left and bottom right corners of the box, normalized within the range [0, 1000). `<|box_start|>` and `<|box_end|>` are treated as special tokens. We pack the samples corresponding to the same image into a single conversation, with no more than 15 samples per conversation. To achieve better performance on high-resolution images, we set the `max_pixels` to 1500*1500. We perform full-parameter fine-tuning on our dataset. The whole training process costs around 60 hours on 16 A100 GPU cards.

UI-I2E-VLM-4B. Following the pre-training setting, we convert bounding boxes into the format `<box>[[x1, y1, x2, y2]]</box>`, where [x1, y1, x2, y2] are the original coordinates on the image without normalization. `<box>` and `</box>` are treated as special tokens. Through experiments we found that packing the data caused a performance degradation of InternVL2-4B, so we did not perform packing.

We apply the 2-D bounding box tile tag proposed in (Dai et al., 2024) to improve the model's visual ability. Since InternVL-2 leverages Dynamic Aspect Ratio to handle images of varying resolutions, during both training and inference, we transform the original coordinates to align with the resized image size within the model. To accommodate the high-resolution nature of the UI interface, we set `patch_num` to 12, which means that each image is divided into 12 tiles of 448x448 pixels.

We perform full-parameter fine-tuning on the language model on our dataset. The whole training process costs around 90 hours on 64 A100 GPU cards.

C Details of *UI-I2E-Bench* Annotation

Error Category	Severity	Proportion (%)
Element Box	Serious	9.7
	Slight	24.1
Element Instruction	Serious	17.3
	Slight	7.9

Table 8: Distribution of error types of *UI-E2I-Synth* generated data in *UI-I2E-Bench*

To reduce the workload of manual annotation, we adopt our *UI-E2I-Synth* pipeline on multiple platforms to construct the *UI-I2E-Bench*. We do balanced sampling for every element type, resulting in 1,987 elements. Then we manually annotated each data, including box quality, instruction quality and instruction type. For box quality, we consider bounding boxes that do not enclose any interactive UI elements as serious errors. Boxes that contain the correct element but are slightly misaligned or have inaccurate boundaries are labeled as slight errors. We retained data with slight errors in our benchmark and manually adjusted the boxes. For instruction quality, we consider instructions that are completely unrelated to any elements to be serious errors. Instructions that are unclear or ambiguous are labeled as slight errors. Similarly, we modified the slightly erroneous instructions and retained them. In addition, we annotated the instruction type for each data. Instructions that directly refer to an element are labeled as explicit, while those describe the element indirectly by position, appearance, or other references are implicit.

After manual annotation, we removed data with serious errors, resulting in a benchmark containing

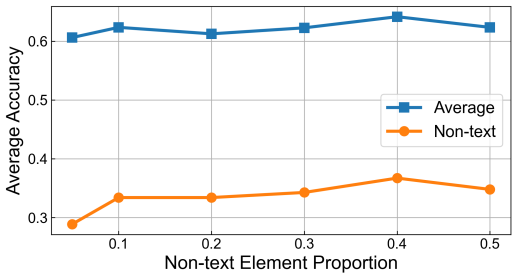


Figure 6: Performance of model trained on 500k UI elements with different proportions of non-text elements in training dataset. The Average represents accuracy on the entire ScreenSpot, while the Non-text represents accuracy specifically on icons/widgets in ScreenSpot.

1,477 element-instruction pairs. The distribution of error types of data are shown in Table 8. We present some examples of our benchmark in Figure 7. All annotators are authors of this work and they all are Asian using English.

D Ablation on Non-Text Element Proportion

In unfiltered data, the quantity of non-text elements is usually smaller than that of text elements. However, identifying non-text elements is much more challenging. When computational resources are limited, adjusting the proportion of non-text elements in the total dataset can help to improve model performance. We investigate the impact of the proportion of non-text elements on accuracy. We use InternVL2-4B as the base model, and sample 500K elements from 250K images for each proportion. The results are shown in Figure 6. Within a certain range, increasing the proportion of non-text elements can effectively improve the accuracy of non-text elements.

E Related literature discussion

GUI agent is a multimodal agent designed for GUI scenarios, which requires multi-step planning and reasoning ability as general language agents (Sumers et al.; Yang et al., 2024; Xi et al., 2025) do. However, due to the incompleteness and opacity of UI metadata, the information within a GUI cannot be fully represented through pure language. It is necessary to leverage the multimodal perception capabilities of models to understand the current task state via GUI screenshots. During action execution, multimodal capabilities are also required to extract the coordinates of interactive elements from the current GUI screenshot, with GUI grounding playing a crucial role. Nevertheless, current general-purpose VLM, especially open-sourced leading MLLMs such as InternVL2 (Chen et al., 2023) and Qwen2-VL (Wang et al., 2024), exhibit relatively limited perception and planning capabilities for GUI tasks. Hence, the collection of corresponding training data is essential and warrants community attention.

F Prompt templates

We display the prompts for the instruction synthesis in UI-E2I-Synth, prompts for OmniParser evaluation and prompts given to GPT-4o in OSWorld evaluation as follows.

G Insights on failure cases in UI-I2E-Bench

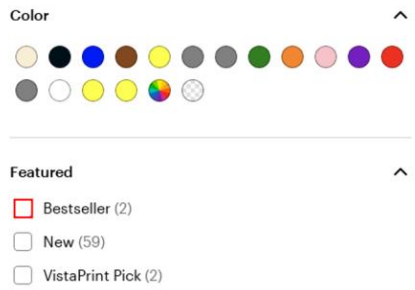
Here we provide preliminary analysis about the typical failure cases in UI-I2E-Bench separately:

Counting error and spatial relationship misunderstanding. The spatial understanding has always been two of the challenges faced by MLLMs. Due to the bias in MLLM pre-training data, models are typically better at detecting whether an object exists in the image but struggle with identifying its quantity and location. We believe this issue can be mitigated by adding data with instructions related to quantities and spatial relationships during training. In fact, through our synthetic data pipeline, our model has already made considerable progress compared to the baseline models in addressing these issues.

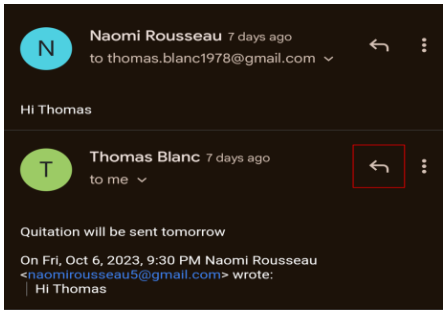
Element type misclassification. In UI environment, the interactive region of different element types naturally varies. In general, large-scale training data contains elements with the same contents but different types, with some types of elements dominating in quantity(e.g., text). As a result, the model is more likely to assume that elements belong to these dominate types, leading to incorrect judgements on their interactive regions.

Lack of external knowledge. While some common cases can be addressed by scaling up the range of training data, it is important to note that covering all icons from a variety of specialized software in the training data is challenging. We believe that for specific software, creating explanatory documentation for its icons and providing it as contextual reference for the model could be a potential solution in the future.

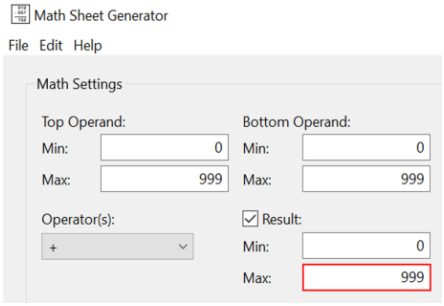
Instruction: Select to change the website language.
Annotation:
- Source: Web
- Element type: Dropdown
- Instruction type: Implicit



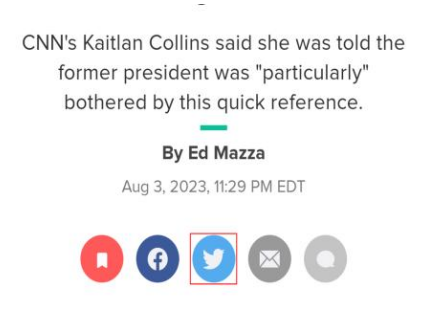
Instruction: Filter to show most popular items.
Annotation:
- Source: Web
- Element type: Toggle
- Instruction type: Implicit



Instruction: Click the reply icon near the second email.
Annotation:
- Source: Mobile
- Element type: Icon
- Instruction type: Implicit

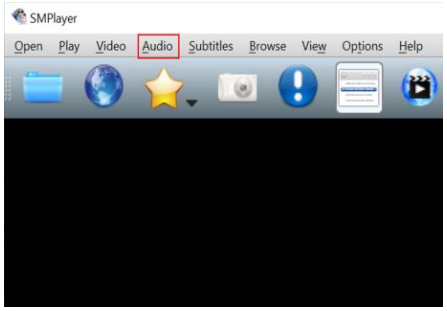


Instruction: Define the max value of result as 999.
Annotation:
- Source: Desktop
- Element type: Input
- Instruction type: Implicit



Instruction: Share article on Twitter.

Annotation:
- Source: Mobile
- Element type: Icon
- Instruction type: Explicit



Instruction: Click the button to access audio settings.
Annotation:
- Source: Desktop
- Element type: Button
- Instruction type: Explicit

Figure 7: Examples in our benchmark UI-I2E-Bench. Red boxes denote the target elements. In the examples shown in this figure, we cropped only a portion of the original image.

Model	Size	#Train	Mobile		Desktop		Web		Avg.
			Text	Icon/Widget	Text	Icon/Widget	Text	Icon/Widget	
InternVL2-4B	4B	-	9.2	4.8	4.6	4.3	0.9	0.1	4.2
Qwen2-VL-7B	7B	-	61.3	39.3	52.0	45.0	33.0	21.8	42.6
OmniParser	-	-	87.4	67.9	93.2	60.9	77.8	48.3	73.9
CogAgent	17B	37M*	67.0	24.0	74.2	20.0	70.4	28.6	47.4
SeeClick	9.6B	1.0M	78.0	52.0	72.2	30.0	55.7	32.5	55.8
UGround	7B	9.7M	82.8	60.3	82.5	63.6	80.4	70.4	74.1
ShowUI	2B	2.7M	92.3	75.5	76.3	61.1	81.7	63.6	76.8
OS-Atlas-4B	4B	13.6M	85.7	58.5	72.2	45.7	82.6	63.1	70.1
OS-Atlas-7B	7B	13.6M	93.0	72.9	91.8	62.9	90.9	74.3	82.5
UI-I2E-VLM-4B	4B	9.9M	87.6	49.8	87.6	45.7	85.7	54.4	70.4
UI-I2E-VLM-7B	7B	9.9M	94.1	77.3	90.7	71.4	87.4	67.5	82.5

Table 9: Full results on ScreenSpot. *Train data for CogAgent includes both pre-training and multi-task finetuning data.

Model	Development	Creative	CAD	Scientific	Office	Operating Systems	Overall Avg
InternVL2-4B	0.3	0.0	0.0	0.4	0.4	0.5	0.3
Qwen2-VL-7B	1.3	0.9	0.4	3.5	3.0	0.5	1.6
OmniParser+GPT-4o	13.7	1.5	7.7	9.4	14.3	4.6	8.3
SeeClick	0.3	0.6	1.9	2.0	0.9	1.5	1.1
UGround	14.7	17.0	11.1	19.3	27.0	9.7	16.5
ShowUI	9.4	5.3	1.9	10.6	13.5	6.6	7.7
OS-Atlas-4B	3.7	2.3	1.5	7.5	4.8	3.1	3.7
OS-Atlas-7B	17.7	17.9	10.3	24.4	27.4	16.8	18.9
UI-I2E-VLM-4B	12.7	12.6	5.7	13.4	15.2	14.3	12.2
UI-I2E-VLM-7B	24.1	23.5	9.2	27.6	38.3	19.9	23.6

Table 10: Full results on ScreenSpot-Pro(simple view).

UI hierarchical misunderstanding. Most current grounding models directly output bounding boxes, lacking an analysis and reasoning progress regarding UI hierarchies. In fact, when dealing with grounding tasks that need page semantic understanding, the model sometimes needs to perform multi-step reasoning. In the example of Figure 5 Task 4, the model needs to first locate the login module and then identify the e-mail input field. A potential mitigation strategy is to use CoT reasoning.

H GPT-4o response example in Instruction Synthesis

For the parameter-based instruction synthesis, we do not simply repeat the already generated action parameters and assemble them to instruction. Instead, we use them as reference to guide the final generation of first-person perspective instructions. We here provide examples of the action parameter and corresponding synthesized instructions sampled from generated dataset. Rather than merely rigidly assembling the given parameters, the synthesized instructions generated from GPT-4o demonstrate naturalness and fluency, exhibiting newly generated content that diverges from the provided action parameters.

Prompts for instruction synthesis in *UI-E2I-Synth*

Step1:

You are a screen UI expert. Here is a UI screenshot image with highlighted bounding boxes and corresponding labeled ID overlayed on bottom of them, and here is a list of corresponding UI element (icon/button/inputfield) box description within these bounding boxes. You should first ensure you understand the screenshot's status and every annotated UI element bounding box on it. Then output the updated element list with below template as JSON format.

NOTE: Ensure all referring exprression should UNIQUELY correspond to the element when generating them.
Here is the output template:

```
{
    "elements": [
        {
            "id": "<copy corresponding labeled ID on the bottom of element>",
            "shortDescription": "<copy from given input list>",
            "fullDescription": "<a comprehensive description that explains the function of the
            ↪  element and the expected outcome when interacting with it>",
           "explicitRefer": "<a referring expression that explicitly refers to the element, from a
            ↪  computer user's first perspective>",
            "implicitReferByElementFunction": "<a referring expression that does NOT explicitly
            ↪  refer to this element content or obvious visual feature, but implicitly refers to
            ↪  it by its function in the whole page or expected outcome after interacting with
            ↪  it>",
            "implicitReferByNearElement": "<a referring expression that does NOT explicitly refer
            ↪  to this element content or obvious visual feature, but implicitly refers to it by
            ↪  its relationship with near elements or emphasizes it from similar elements by
            ↪  spatial order>"
        }
    ]
}
```

Step2:
You are a computer expert user. You will be given a UI element list from a UI screenshot which includes elements and their referring expressions as shown in the following input template. You should simulate a user using the UI screenshot, generate possible action type and action content, then generate instructions for every element according to the given element referring expressions, as shown in the following output template. Return as JSON format.

NOTE:
1. For inputfield element type, the generated instruction should contain the possible input content from a computer user's first perspective. For example, "fill James as last name", "enter Boeing747 in search field", "select article in recent six months".
2. Ensure all instruction should UNIQUELY correspond to the element when generating them.
Here is the input template:

```
{
    "elements": [
        {
            "id": "<copy corresponding labeled ID on the bottom of element>",
           "shortDescription": "<a short description about the element, including element type and
            ↪  element content>",
            "fullDescription": "<a comprehensive description that explains the function of the
            ↪  element and the expected outcome when interacting with it>",
           "explicitRefer": "<a referring expression that explicitly refers to the element, from a
            ↪  computer user's first perspective>",
            "implicitReferByElementFunction": "<a referring expression that does NOT explicitly
            ↪  refer to this element content or obvious visual feature, but implicitly refers to
            ↪  it by its function in the whole page or expected outcome after interacting with
            ↪  it>",
            "implicitReferByNearElement": "<a referring expression that does NOT explicitly refer
            ↪  to this element content or obvious visual feature, but implicitly refers to it by
            ↪  its relationship with near elements or emphasizes it from similar elements by
            ↪  spatial order>"
        }
    ]
}
```

Here is the output template:

```
{
    "elements": [
```

(cont.) Prompts for instruction synthesis in *UI-E2I-Synth*

```
        {
            "id": "<copy corresponding labeled ID on the bottom of element>",
            "shortDescription": "<copy from given input list>",
            "instructionArgs": {
                "actionType": "<choose appropriate action type>",
               "actionContentDescription": "<describe what is the possible action content about. if
                ↪  CLICK: leave empty string. if TYPE: specific input content which is appropriate
                ↪  for current UI screenshot. if Toggle, Checkbox and Switch, ON or OFF>",
                "actionContent": "<According to `actionContentDescription` to fill the specific
                ↪  content, if CLICK: leave empty string>"
            },
            "convertedUserInstructionByElementFunction": "<including above `actionContent` if not
            ↪  empty, convert above `implicitReferByElementFunction` into a computer user's first
            ↪  perspective instruction, concise and less than 15 words>",
            "convertedUserInstructionByNearElement": "<including above `actionContent` if not
            ↪  empty, convert above `implicitReferByNearElement` into a computer user's first
            ↪  perspective instruction, concise and less than 15 words>"
        }
    ]
}
```

Prompts for OmniParser Evaluation

Here is a UI screenshot image with bounding boxes and corresponding labeled ID overlayed on top of it, and here is a list of icon/text box description: `{parsed_local_semantics}`.
Your task is `{task}`. Which bounding box label you should operate on? Give a brief analysis, then put your answer in the format of:
Box with label ID: [xx]

Prompts for GPT-4o in OSWorld

You are an agent which follow my instruction and perform desktop computer tasks as instructed.
You have good knowledge of computer and good internet connection and assume your code will run on a computer for controlling the mouse and keyboard.

For each step, you will get an observation of an image, which is the screenshot of the computer screen and you will predict the action of the computer based on the image.
You are required to use ‘pyautogui‘ to perform the action grounded to the observation, but DONOT use the ‘pyautogui.locateCenterOnScreen‘ function to locate the element you want to operate with since we have no image of the element you want to operate with. DONOT USE ‘pyautogui.screenshot()‘ to make screenshot.
Return one line or multiple lines of python code to perform the action each time, be time efficient. When predicting multiple lines of code, make some small sleep like ‘time.sleep(0.5);‘ interval so that the machine could take; Each time you need to predict a complete code, no variables or function can be shared from history.
You need to to specify the coordinates of by yourself based on your observation of current observation, but you should be careful to ensure that the coordinates are correct.

Important NOTE: when you use pyautogui.click() / pyautogui.moveTo() / pyautogui.dragTo() / pyautogui.rightClick() / pyautogui.middleClick() / pyautogui.doubleClick() / pyautogui.tripleClick() / pyautogui.scroll() / pyautogui.mouseDown / pyautogui.mouseUp() , you should add a detailed description of the target in annotation after the code,following the demands below:
1. Provide a description of the element you want to operate. (If ACTION == SCROLL DOWN, this field should be None.)
2. It should include the element’s identity, type (button, input field, dropdown menu, tab, etc.), and text on it (if have).
3. Ensure your description is both concise and complete, covering all the necessary information and less than 30 words.
4. If you find identical elements, specify its location and details to differentiate it from others.
5. Example: “‘python pyautogui.click(100, 200) # target description: Tab labeled S̈ervicesïn the pop-up information box “‘

You ONLY need to return the code inside a code block, like this: “‘python # your code here “‘
Specially, it is also allowed to return the following special code:
When you think you have to wait for some time, return “‘WAIT“‘;
When you think the task can not be done, return “‘FAIL“‘, don’t easily say “‘FAIL“‘, try your best to do the task;
When you think the task is done, return “‘DONE“‘.
My computer’s password is ’password’, feel free to use it when you need sudo rights.
First give the current screenshot and previous things we did a short reflection, then RETURN ME THE CODE OR SPECIAL CODE I ASKED FOR. NEVER EVER RETURN ME ANYTHING ELSE.

(cont.) Prompts for GPT-4o in OSWorld

You are asked to complete the following task: {task}
Given the screenshot as below. What's the next step that you will do to help with the task?

GPT-4o response example in UI-E2I-Synth


```
[
    {
        "instructionArgs": {
            "actionType": "TYPE",
            "actionContent": "Gangsta-Groove"
        },
        "convertedUserInstructionByElementFunction": "Enter 'Gangsta-Groove' in the search bar",
        "convertedUserInstructionByNearElement": "Type 'Gangsta-Groove' in the search field above
        ↪  the article",
    },
    {
        "instructionArgs": {
            "actionType": "CLICK",
            "actionContent": ""
        },
        "convertedUserInstructionByElementFunction": "Review price based on net area.",
        "convertedUserInstructionByNearElement": "Check near rent/total price toggle.",
    },
    {
        "instructionArgs": {
            "actionType": "SELECT",
            "actionContent": "preferred language"
        },
        "convertedUserInstructionByElementFunction": "Select your preferred language",
        "convertedUserInstructionByNearElement": "Choose a language next to Contact Us",
    },
    {
        "instructionArgs": {
            "actionType": "CLICK",
            "actionContent": ""
        },
        "convertedUserInstructionByElementFunction": "Click to learn more about us",
        "convertedUserInstructionByNearElement": "Click the second button on the menu"
    },
    {
        "instructionArgs": {
            "actionType": "CLICK",
            "actionContent": ""
        },
        "convertedUserInstructionByElementFunction": "Return to the top of the page",
        "convertedUserInstructionByNearElement": "Click the bottom right corner icon",
    }
]
```