AutoGUI: Scaling GUI Grounding with Automatic Functionality Annotations from LLMs

Anonymous ACL submission

Abstract

User interface understanding with vision-language models (VLMs) has received much attention due to its potential for enhancing software automation. However, existing datasets used to build UI-VLMs either only contain large-scale context-free element annotations or contextualized functional descriptions for elements at a small scale. In this work, we propose the **AutoGUI** pipeline for automatically annotating UI elements with detailed functionality descriptions at scale. Specifically, we leverage large language models (LLMs) to infer element functionality by comparing UI state changes before and after simulated interactions. To improve annotation quality, we propose LLM-aided rejection and verification, eliminating invalid annotations without human labor. We construct a high-quality AutoGUI-704k dataset using the proposed pipeline, featuring diverse and detailed functionality annotations that are hardly provided by previous datasets. Human evaluation shows that we achieve annotation correctness comparable to a trained human annotator. Extensive experiments show that our dataset remarkably enhances VLM's UI grounding capabilities and exhibits significant scaling effects. We also show the interesting potential use of our dataset in UI agent tasks. Please view our data at this anonymous URL: `https://huggingface.co/AutoGUI`.

1 Introduction

User interface understanding with visual language models(VLMs) (Hong et al., 2024; You et al., 2024; Wu et al., 2024) has received wide attention due to its potential in fundamentally transforming how we interact with software (Xie et al., 2024).

While recent work has made progress by employing structural mapping between UI code and visual layout, such as UI REG/REC(Hong et al., 2024; Li et al., 2020a) and layout-to-code conversion (Xia et al., 2024; Liu et al., 2023a; Baechler et al., 2024), a more critical challenge remains: understanding the semantic purpose and interactive affordance of individual UI elements, known as *functionality understanding*.

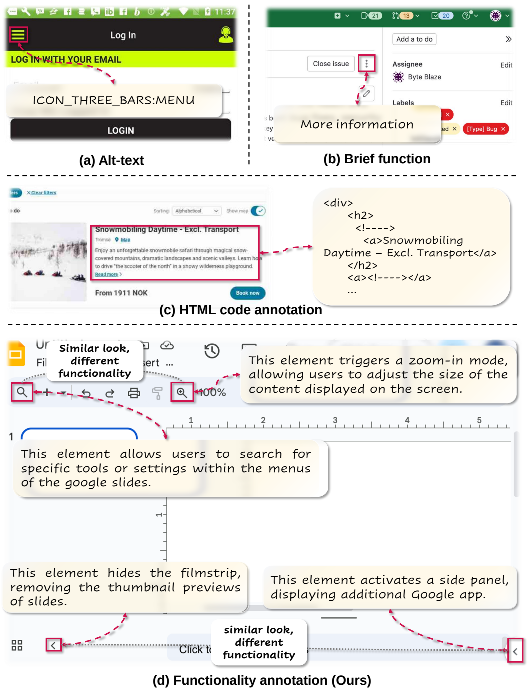


Figure 1: Our annotations are rich in functional semantics (bottom) compared with existing UI datasets.

Accurate functionality understanding requires VLMs to possess strong element grounding capabilities - the ability to connect fine-grained visual elements with their referring expressions. To enhance this capability, large-scale training data is indispensable. However, the scale of open-source datasets with detailed element annotations (Li et al., 2020a,b; Kapoor et al., 2024; Gou et al., 2024) is unsatisfactory, significantly smaller than natural image datasets such as LAION-5B (Schuhmann et al., 2022). Additionally, traditional annotation methods (Deka et al., 2017a; Li et al., 2020a) are labor-intensive, leading to prohibitive costs that hinder scalability. Moreover, existing datasets typically focus on describing either element alt-texts (Cheng et al., 2024), or brief intents weakly related to UI context (Bai et al., 2021) shown in Fig. 1. These datasets lack contextual functional descriptions of UI elements, which poses a challenge for VLMs in comprehending the roles these elements serve within specific UI contexts, such as distinguishing between two visually similar magnifier icons that may represent distinct functionalities like searching and zooming.

To address the challenge, we propose AutoGUI, an automatic annotating pipeline that provides unlimited element functionality annotations. Our pipeline collects UI interaction trajectories and leverages large language models (LLMs) to infer element functionalities based on UI state changes, eliminating the need for manual annotation by human experts. Initially, the proposed pipeline crawls a multitude of interaction trajectories on either a web browser or an Android emulator. Subsequently, we use open-source LLMs (AI@Meta, 2024) to annotate the functionalities of elements on collected GUIs based on changes to UI contents when interacting with these elements. To ensure data quality, LLM-aided rejection is utilized to eliminate invalid samples, such as incompletely rendered UIs. Additionally, inspired by LLM verification (Weng et al., 2022; Lightman et al., 2023), multiple LLMs are prompted as verifiers to identify false functionality descriptions. With both the rejection and verification processes, our pipeline removes unclear and invalid samples.

We curate the AutoGUI-704k dataset with the proposed pipeline, providing high-quality functionality grounding and referring tasks used to finetune and evaluate open-source VLMs. Pioneer experiments find that our pipeline achieves annotation accuracy of **96.7%** comparable to a trained human annotator.

Based on the collected AutoGUI-704k dataset, we finetune open-source VLMs and demonstrate that our data significantly enhances the VLMs' UI grounding accuracy and exhibits remarkable scaling effects. The results also show that our functionality annotation type is superior to the data type directly derived from web HTML code and metadata (Hong et al., 2024; Cheng et al., 2024), serving as a promising data source for building VLMs capable of UI grounding. Moreover, VLMs trained with our data can assist in GUI agent tasks by refining element grounding, which shows more potential use of our dataset.

2 Related Works

2.1 Recent Advancement of VLMs

Recent research has enhanced LLMs with the capability of processing both visual and textual information (Alayrac et al., 2022; Liu et al., 2023b; Lin et al., 2023; Chen et al., 2023; Lu et al., 2024; Wang et al., 2024a,b; Li et al., 2024b; Zhang et al., 2024a; You et al., 2024; Laurençon et al., 2024; Peng et al., 2024; Driess et al., 2023), opening the new field of VLM. Pioneering efforts Flamingo (Alayrac et al., 2022) uses interleaved visual and language inputs as prompts and shows few-shot visual question-answering capability. LLaVA (Liu et al., 2023b) and LLaMA-Adapter (Zhang et al., 2024a) have attempted to align vision encoders (Dosovitskiy et al., 2021) with LLMs to enable visual instruction following. Advanced models such as InternVL (Chen et al., 2023) and Qwen2-VL series (Wang et al., 2024a) are further equipped with impressive high-resolution and multi-lingual understanding abilities. Additionally, VLM are applied to scenarios rich in textual imagery (Ye et al., 2023b,a; Liu et al., 2024b) and embodied interactions (Driess et al., 2023; Kim et al., 2024). Despite these advancements, VLMs lag in UI understanding probably due to data scarcity. This paper contributes an automatic UI annotation pipeline to tackle this challenge, aiming to expand the data available for training VLMs in this crucial area.

2.2 Existing UI Datasets

Unlike natural image datasets (Russakovsky et al., 2014; Schuhmann et al., 2022), UI understanding datasets are much smaller. Early-stage datasets (Wang et al., 2021; Li et al., 2020a,b; Bai et al., 2021; Burns et al., 2022) primarily annotate the RICO screenshot collection (Deka et al., 2017b), which includes 72K screenshots from Android apps. Examples include Widget Captioning (Li et al., 2020a), which analyzes captions and linguistic features of UI elements, and RICOSCA (Li et al., 2020b), which maps single-step instructions to UI locations. Recently, AITW (Rawles et al., 2023) and AndroidControl (Li et al., 2024a) have been proposed to focus on interpreting high-level instructions in Android environments. To increase data scale, SeeClick (Cheng et al., 2024), CogAgent (Hong et al., 2024), and OS-ATLAS (Wu et al., 2024)

Table 1: **Comparing our AutoGUI dataset with existing UI grounding datasets.** Multi-Res means the samples are collected on devices with various resolutions. Auto Anno. means the samples are collected autonomously. #Anno. means the number of annotated samples provided by the datasets. (Methods combining open-source individual datasets are not compared.)

Dataset	UI Type	Multi Res.	Auto Anno.	Contextual Functionality Semantics	#Anno.	Task
S2W (Wang et al., 2021)	Mobile	✗	✗	✗	112k	Screen Summarization
Wid. Cap. (Li et al., 2020a)	Mobile	✗	✗	✗	163k	Element Captioning
RICOSCA (Li et al., 2020b)	Mobile	✗	✗	✗	295k	Action Grounding
MoTIF (Burns et al., 2022)	Mobile	✗	✗	✗	6k	Mobile Navigation
RefExp (Bai et al., 2021)	Mobile	✗	✗	✗	20.8k	Element Grounding
SeeClick Web (Cheng et al., 2024)	Web	✗	✓	✗	271k	Element Grounding
MultiUI (Hong et al., 2024)	Web, Mobile	✓	✓	✗	3M	Act. & Elem. Ground
UGround-Web (Gou et al., 2024)	Web	✓	✓	✗	9.4M	Element Grounding
UI REC/REG (Hong et al., 2024)	Web	✓	✓	✗	400k	Box2DOM, DOM2Box
Ferret-UI (You et al., 2025)	Mobile	✓	✓	✗	250k	Elem. Ground & Ref.
AutoGUI (ours)	Web, Mobile	✓	✓	✓	704k	Functionality Ground & Ref.

have utilized the UI metadata from Common Crawl webpages to produce massive element referring expressions. Several works (Gou et al., 2024; Lin et al., 2024; Xu et al., 2024) also filter and combine existing datasets to produce all-in-one collections that incorporate diverse training tasks. In contrast, our AutoGUI-704k dataset contributes large-scale element functionality annotations, which convey contextual functionality semantics that are hardly provided in previous datasets. The advantages of our dataset are summarized in Tab. 1.

3 AutoGUI: Automatic Functionality Annotation Pipeline

This section introduces AutoGUI, an annotation pipeline (Fig. 2) that automatically produces contextual element functionality annotations used to enhance VLMs' GUI grounding capabilities.

3.1 Collecting UI Interaction Trajectories

Our pipeline initiates by collecting interaction trajectories, which are sequences of UI contents captured by interacting with UI elements. Each step captures all interactable elements and the accessibility tree (AXTree) that briefly outlines the UI structure, which will be used to annotate functionality. To amass these trajectories, we utilize the latest Common Crawl repository as the data source for web UIs and Android Emulator for mobile UIs. The open-source trajectories from Android-Control (Li et al., 2024a) and MobileViews (Gao et al., 2024) are also included to enhance diversity. Note that illegal UIs are manually excluded from the sources. Please refer to Sec. A for collecting details and data license.

3.2 Automatic Functionality Annotation

The pipeline generates functionality annotations for elements in the collected trajectories. Interacting with an element e, by clicking or hovering over it, triggers UI content changes. In turn, these changes can be used to predict the functionality f of the interacted element. For instance, if clicking an element causes new buttons to appear in a column, the element likely functions as a dropdown menu activator (an example in Fig. D). With this observation, we utilize a capable LLM (i.e., Llama-3-70B (AI@Meta, 2024)) as a surrogate for humans to summarize an element's functionality based on the UI changes resulting from interaction. Concretely, we generate compact content differences for AXTrees before (s_t) and after (s_{t+1}) the interaction using a file-comparing library[1]. Then, we prompt the LLM to analyze the UI content changes (addition, deletion, and unchanged lines), present a detailed Chain-of-Thoughts (Wei et al., 2022) reasoning process explaining how the element affects the UI, and finally summarize the element's functionality.

In cases where element interactions significantly transform the UI and cause lengthy differences—such as navigating to a new screen—we adjust our approach by using UI description changes instead of the AXTree differences. This annotation process is formulated as: $f = \mathrm{LLM}(p_{\mathrm{anno}}, s_t, s_{t+1})$ where f is the predicted functionality, p_{anno} is the annotation prompt (Tab. B and Tab. C). Examples are depicted in Fig. 3 and more annotation details are explained in Sec. A.5.

[1]https://docs.python.org/3/library/difflib.html

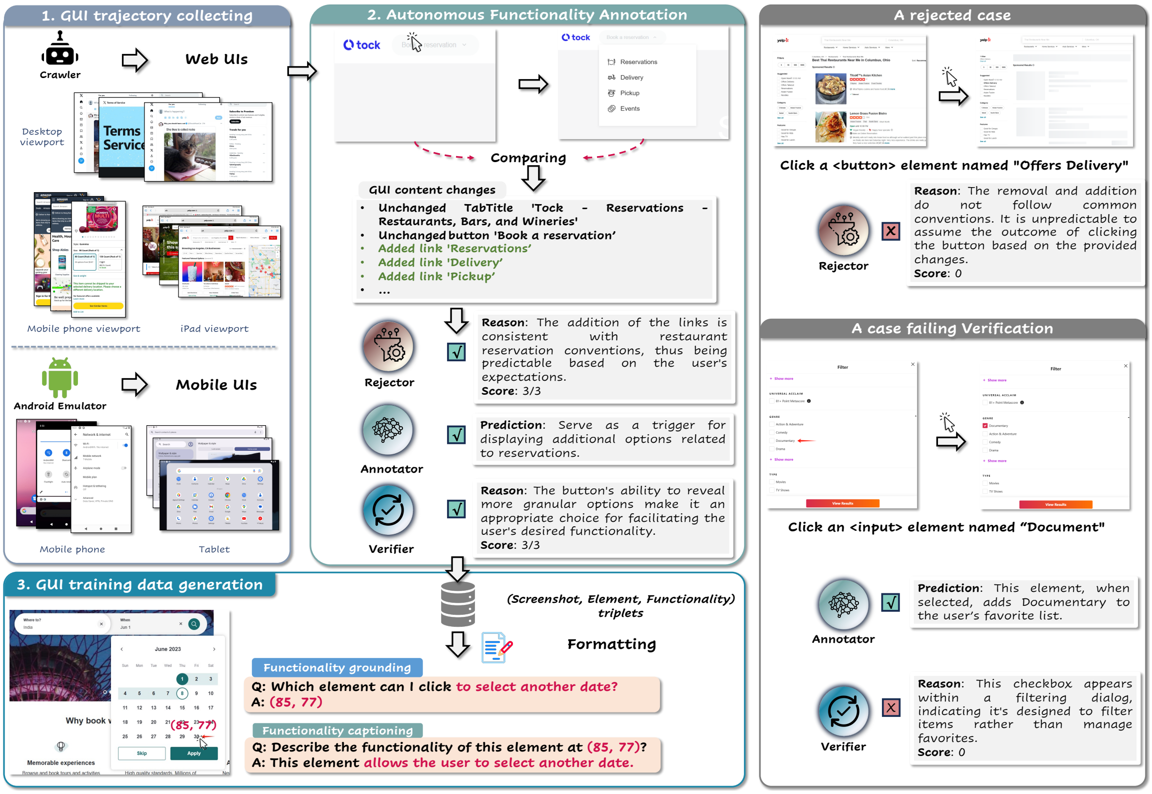


Figure 2: **The proposed pipeline for automatic UI functionality annotation.** An LLM is utilized to predict element functionality based on the UI content changes observed during the interaction. LLM-aided rejection and verification are introduced to improve data quality. Finally, the high-quality functionality annotations will be converted to instruction-following data by applying task templates.

3.3 Removing Invalid Samples via LLM-Aided Rejection

The collected trajectories may contain invalid samples due to broken UIs, such as incomplete UI loading, which can mislead the models trained with them. To filter out these invalid samples, we introduce an LLM-aided rejection approach. Initially, hand-written rules (detailed in Sec. A.6) are used to detect obvious bad cases, such as blank UIs, UIs containing elements indicating content loading, and interaction targets outside of UIs. However, a few types are difficult to detect with the rules. For instance, interacting with a “view more” button might unexpectedly redirect the user to a login page instead of the desired information page due to website login restrictions. To identify these challenging samples, we prompt the annotating LLM to also act as a rejector. Specifically, the LLM takes the UI content changes as input, provides detailed reasoning through whether the changes are meaningful for predicting the element’s functionality, and finally outputs predictability scores ranging from 0 to 3 (3 is empirically chosen for a balance between annotation efficiency and quality. More details in the Appendix). This process is formulated as follows: $score = \mathrm{LLM}(p_{\text{reject}}, e, s_t, s_{t+1})$ where p_{reject} is the rejection prompt (Tab. D).

This approach ensures that predictable samples receive higher scores, while unpredictable ones receive lower scores. For instance, if a button labeled "Show More", upon interaction, clearly adds new content, this sample will be considered to provide sufficient changes that can anticipate the content expansion functionality and will get a score of 3.

We deploy this rejector to discard the bottom 30% of samples based on score ranking to strike a balance between the elimination of invalid samples and the preservation of valid ones (Details in Sec. A.7). The samples that pass the rejection procedure are submitted for functionality annotation. Please see examples in Fig. H.

3.4 Improving Annotation Quality via LLM-Based Verification

The functionality annotations produced by the LLM probably contain incorrect and hallucinated samples. To improve dataset quality, we prompt LLMs to verify the annotations, inspired by works

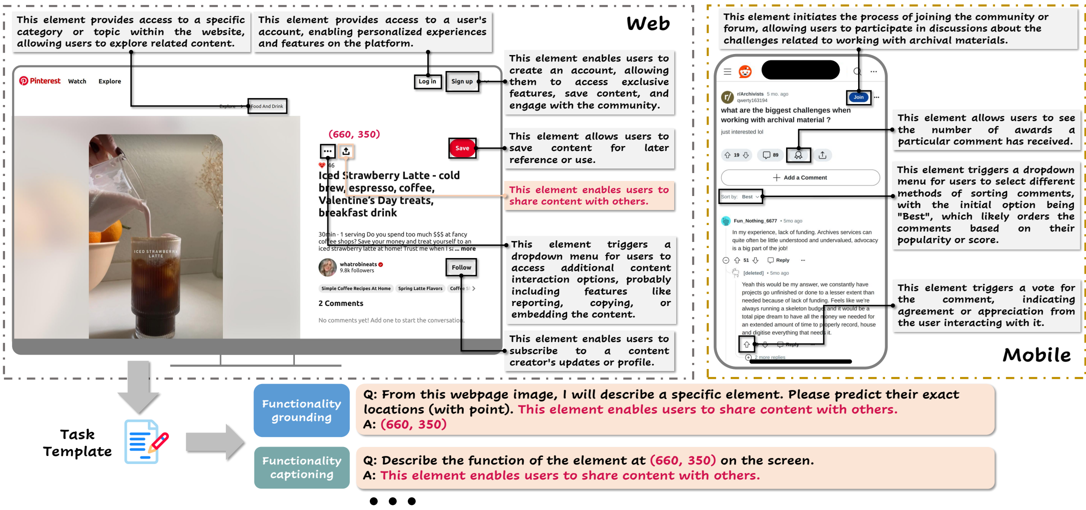


Figure 3: Element functionality annotations generated by the AutoGUI pipeline for both web and mobile domains.

that justify the feasibility of LLM-based verification (Zheng et al., 2023; Chen et al., 2024; Lee et al., 2023). This process presents the LLMs with the interacted element, its UI context, the UI changes induced by the interaction, and the functionality annotation generated in the previous stage. Then, the LLMs analyze the UI content changes and predict whether the interacted element aligns with the given functionality. If the LLMs determine that the interacted element fulfills the functionality given its UI context, the LLMs will grant a full score (An example in Fig. I). If not, this functionality will be seen as incorrect as this mismatch indicates that it may not accurately reflect the element's role within the UI context.

To mitigate the potential biases in LLMs (Panickssery et al., 2024; Bai et al., 2024), two different LLMs (i.e., Llama-3-70B and Mistral-7B-Instruct-v0.2) are employed as verifiers and prompted to output 0-3 scores (This scoring range is chosen as it empirically achieves a high verification accuracy). The scoring process is formulated as follows: $score = \mathrm{LLM}(p_{\text{verify}}, e, f, s_t, s_{t+1})$ where p_{verify} denotes the verification prompt (Tab. E). Only if the two scores are both 3s do we consider the functionality annotation correct (Details in Sec. A.8). While this approach seems stringent, we can make up the number of annotations through scaling.

3.5 Task Generation

After rejecting, annotating, and verifying, we obtain a high-quality UI functionality dataset containing triplets of {*UI screenshot, Interacted element, Functionality*}. To convert this dataset into an instruction-following dataset for training and evaluation, we generate functionality grounding and referring tasks using diverse prompt templates. The coordinates of element bounding boxes are normalized within the range $[0, 999]$ (see Fig. 3).

We finally annotate 2k grounding samples as a test set and 702k as a training set (The details of ensuring no overlap between the two sets can be seen in Sec. A.1). The statistics of our dataset in Tab. 2 and Sec. A.1 show that our dataset covers diverse UIs and exhibits variety in lengths and functional semantics of the annotations.

3.6 Analysis of Data Quality

Comparison with Human Annotation $N = 145$ samples (99 valid and 46 invalid) are randomly selected as a testbed for comparing the annotation correctness of a trained human annotator and our pipeline. Here, correctness is defined as $Correctness = C/(N - R)$, where C and R denote the numbers of correctly annotated and rejected samples, respectively. The denominator subtracts the number of rejected samples as we are more interested in the percentage of correct samples after rejecting invalid samples. The authors rigorously evaluate the annotation results based on three criteria outlined in Fig. J. Details can be found in Sec. B.1.

After experimenting with three runs, Tab. 3 shows that the AutoGUI pipeline achieves high correctness comparable to the trained human annotator (r6 vs. r1). Without rejection and verification (r2), AutoGUI is inferior as it cannot recognize invalid samples. Notably, simply using the rules written by the authors can improve the correctness, which is further enhanced with the LLM-aided re-

Table 2: **The statistics of the AutoGUI datasets.** The Anno. Tokens and Avg. Words columns show the total number of tokens and the average number of words for the functionality annotations regardless of task templates. The Domains/Apps column shows the number of unique web domains/mobile Apps involved in each split.

Split	#Tasks	Anno. Tokens	Avg. Words	Domains/Apps	Device Ratio
Train	702k	17.9M	23.1	916	Web: 54.6%, Mobile: 45.4%
Test	2k	53.4k	22.5	299	Web: 50%, Mobile: 50%

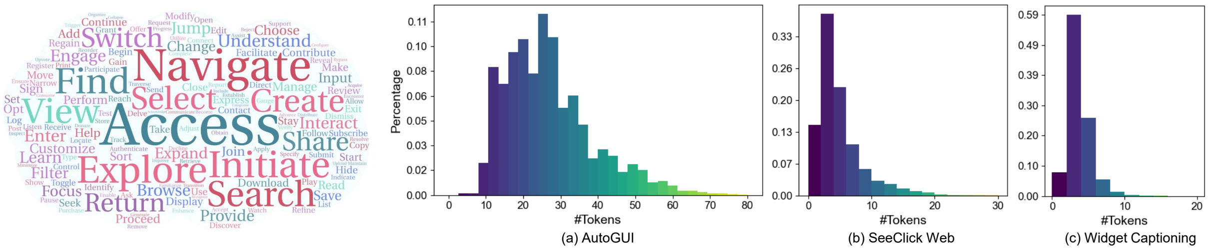


Figure 4: **Diversity of the AutoGUI dataset. Left**: The word cloud illustrates the ratios of the verbs representing the main intents in the functionality annotations. **Right**: Comparing the distributions of the annotation token numbers for our AutoGUI training split, SeeClick Web training data (Cheng et al., 2024), and Widget Captioning (Li et al., 2020a). The comparison demonstrates that our dataset covers significantly more diverse task lengths.

Table 3: **Comparing AutoGUI and human annotator.** AutoGUI with the proposed rejection and verification achieves correctness comparable to the trained human annotator. One LLM means Llama-3-70B and Two LLMs include Mistral-7B-Instruct-v0.2.

No.	Annotator	Rejector	Verifier	Correctness
r1	Human	-	-	95.5%
r2	Llama-3-70B	-	-	64.5%
r3	Llama-3-70B	Rules	-	83.1%
r4	Llama-3-70B	Rules+LLM	-	94.4%
r5	Llama-3-70B	Rules+LLM	One LLM	96.0%
r6	Llama-3-70B	Rules+LLM	Two LLMs	**96.7%**

jector (r4 vs. r3). Moreover, utilizing the annotating LLM itself to self-verify its annotations helps AutoGUI surpass the trained annotator (r5 vs. r1). Introducing another LLM verifier (i.e., Mistral-7B-Instruct-v0.2) brings a slight increase which results from Mistral recognizing Llama-3-70B's incorrect descriptions of how dropdown menu options work. Overall, these results justify the efficacy of the AutoGUI annotation pipeline.

Qualitative comparison (Fig. R) shows that our pipeline generates more detailed annotations which would take more time for the human annotator.

Impact of LLM Output Uncertainty Despite LLM output uncertainty, our pipeline achieves a high annotation consistency of 94.5%. LLM uncertainty affects rejection but has a minimal overall impact due to the low prevalence of invalid samples. More experimental details in the Appendix.

4 Fine-Tuning Experiments

This section validates that our dataset effectively enhances the GUI grounding capabilities of VLMs.

4.1 Experimental Settings

Evaluation Benchmarks We base our evaluation on the UI grounding benchmarks for various scenarios: **FuncPred** is the test split from our collected functionality dataset. This benchmark requires a model to locate the element specified by its functionality description. **ScreenSpot** (Cheng et al., 2024) and **ScreenSpot-v2** (Wu et al., 2024) require a model to locate elements based on short instructions on mobile, desktop, and web platforms. **VisualWebBench (VWB)** (Liu et al., 2024a) is a comprehensive multi-modal benchmark assessing the understanding capabilities of VLMs in web scenarios. We select the element and action grounding tasks from this benchmark. **MOTIF** (Burns et al., 2022) requires an agent to complete a natural language command in mobile Apps. Samples of the benchmarks are visualized in Fig. K. We report the grounding accuracy (%): Acc $= \sum_{i=1}^{N} \mathbf{1}\left(\text{pred}_i \text{ inside GT bbox}_i\right)/N \times 100$ where $\mathbf{1}$ is an indicator function and N is the number of test samples. This formula denotes the percentage of samples with the predicted points lying within the bounding boxes of the target elements.

Training Details We select SliME-8B (Zhang et al., 2024b), Qwen-VL (Bai et al., 2023), and Qwen2-

Table 4: **Element grounding accuracy across benchmarks.** We compare the base models fine-tuned with our AutoGUI data and representative open-source VLMs. General-purpose models (Qwen-VL, SliME-8B, and Qwen2-VL) show significant performance improvements after fine-tuning with the AutoGUI data. UI-specialized models (SeeClick and UGround) also improve when AutoGUI data is added to their fine-tuning datasets. Green text indicates gains over base models. † denotes metrics quoted from the original benchmark paper.

Type	Model	Size	FuncPred	ScreenSpot	ScreenSpot-v2	MoTIF	VWB EG	VWB AG
General	GPT-4o	N/A	9.8	17.8	20.4	30.5	5.6	6.8
	Llama-3.2-Vision-Instruct	11B	4.9	11.7	11.6	19.7	7.0	3.9
	SliME (Zhang et al., 2024b)	8B	3.2	13.0	13.4	7.0	6.1	4.9
	Qwen-VL (Bai et al., 2023)	10B	3.0	5.2†	5.6	7.8	1.7	3.9
	Qwen2-VL (Bai et al., 2023)	7B	38.7	66.4	66.9	71.1	55.9	62.1
UI Experts	CogAgent (Hong et al., 2024)	18B	29.3	47.4†	52.1	45.1	55.7	59.2
	SeeClick (Cheng et al., 2024)	10B	19.8	53.4†	54.0	66.5	39.2	27.2
	UGround-v1-7B (Gou et al., 2024)	7B	55.8	85.9	**88.0**	78.4	92.7	69.9
	OS-ATLAS (Gou et al., 2024)	7B	52.1	82.5	84.1	78.8	82.6	**71.8**
Finetuned	Qwen-VL-AutoGUI702k	10B	48.7 (+45.7)	41.2 (+36.0)	40.2 (+34.6)	44.0 (+36.2)	42.1 (+40.4)	35.9 (+32.0)
	SliME-AutoGUI702k	8B	62.6 (+59.4)	44.0 (+31.0)	42.5 (+29.1)	44.9 (+37.9)	25.4 (+19.3)	13.6 (+8.7)
	Qwen2-VL-AutoGUI702k	7B	**65.0** (+26.3)	80.0 (+13.6)	83.2 (+16.3)	72.3 (+1.2)	90.3 (+34.4)	70.9 (+8.8)
	SeeClick w/ AutoGUI702k	10B	50.0 (+30.2)	54.2 (+0.8)	54.7 (+0.7)	67.0 (+0.5)	56.2 (+17.0)	45.6 (+18.4)
	UGround w/ AutoGUI702k	7B	58.7 (+2.9)	**86.5** (+0.6)	**88.0** (+0.0)	**80.5** (+2.1)	**94.9** (+2.2)	**71.8** (+1.9)

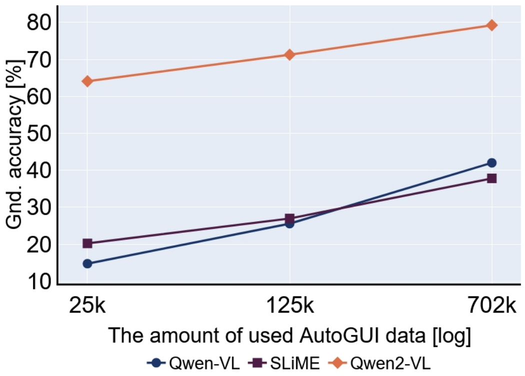


Figure 5: **Scaling effect of the AutoGUI data.** The three general-purpose VLMs are fine-tuned with three scales of AutoGUI data. Using more data consistently enhances the grounding accuracy of the three models. Note that the grounding accuracy (Y-axis) is averaged over all the element grounding benchmarks.

VL-7B (Wang et al., 2024a) as the base models and fine-tune them on 25k, 125k, and 702k samples of the AutoGUI training data to investigate how the AutoGUI data enhances their UI grounding capabilities. We fine-tune Qwen-VL and Qwen2-VL with LoRA (Hu et al., 2022) and fine-tune SliME (Zhang et al., 2024b) with only the visual encoder frozen. We also test the benefits of our dataset for UI expert VLMs, such as SeeClick (Cheng et al., 2024) and UGround (Gou et al., 2024), by adding our data to their fine-tuning data. All models are fine-tuned on 8 A100 GPUs for one epoch. (More details and hyper-parameters in Sec. B.2)

Compared VLMs We compare with both general-purpose VLMs and UI expert VLMs. During the evaluation, we manually craft grounding prompts suitable for these VLMs.

4.2 Experimental Results and Analysis

A) AutoGUI functionality annotations effectively enhance VLMs' UI grounding capabilities and achieve scaling effects. We endeavor to show that the element functionality data collected by AutoGUI contributes to high grounding accuracy. The results in Tab. 4 demonstrate that the base models embrace notable performance gains on all the benchmarks. The two general-purpose VLMs (Qwen-VL and SLiME), which perform poorly, witness huge performance increases after fine-tuning with AutoGUI data. Qwen2-VL fine-tuned with our data achieves high accuracy comparable to the expert models, i.e., UGround and OS-ATLAS. Interestingly, the two UI expert VLMs (SeeClick and UGround) also benefit from our data, with remarkable performance gains on FuncPred and VWB.

Fig. 5 shows that the three general-purpose VLMs obtain progressively rising grounding accuracy as the AutoGUI data size scales from 25k to 702k, indicating that increasing AutoGUI data amount leads to better localization performance.

In summary, our functionality data enhances VLMs UI element grounding ability and exhibits clear scaling effects as the data size increases.

B) Our functionality annotations are effective for enhancing UI grounding capabilities. To assess the effectiveness of functionality annotations, we compare this annotation type with three types: 1) **Naive element-HTML pairs**, which are directly obtained from the UI source code (Hong et al., 2024) and associate HTML code with elements in specified areas of a screenshot. Examples are shown in Fig. 1. To create these pairs, we replace the functionality annotations with the corresponding HTML code snippets recorded during trajectory collection. 2) **Alt-Texts** that are extracted from HTML code. We use the displayed texts for plain-text elements and descriptive texts associated with images and icons. 3) **Brief functionality descriptions** that are generated by prompting GPT-4o-mini[2] to condense the AutoGUI functionality annotations. For example, a full description such as *'This element provides access to a documentation category, allowing users to explore relevant information and guides'* is shortened to *'Documentation category access'*.

After experimenting with Qwen-VL (Bai et al., 2023) at the 25k and 125k scales, the results in Tab. 5 show that fine-tuning with the complete functionality annotations is superior to the other three types. Notably, our functionality annotation type yields the largest gain on the challenging FuncPred benchmark that emphasizes contextual functionality grounding. In contrast, the Elem-HTML type performs poorly due to the noise inherent in HTML code (e.g., numerous redundant tags), which reduces fine-tuning efficiency. The variant using alt-texts also performs poorly on FuncPred as short plain texts can hardly convey detailed functionality semantics, but it encounters a smaller drop on ScreenSpot probably because this benchmark contains many short-phrase localization tasks. The condensed functionality annotations are also inferior, as the consensing loses details necessary for fine-grained element grounding. In summary, the AutoGUI functionality annotations provide a clear advantage in enhancing UI grounding capabilities.

4.3 Grounding Failure Case Analysis

After analyzing the grounding failure cases, we identified several failure patterns in the fine-tuned models: a) difficulty in accurately locating small elements; b) challenges in distinguishing between similar but incorrect elements; and c) issues with recognizing icons that have uncommon shapes. Please refer to Sec. D.2 for details.

5 Potential Use of AutoGUI Data

We apply our dataset to a downstream GUI agent task to demonstrate how our dataset can benefit GUI agents based on proprietary VLMs. The used benchmark is AITW (Rawles et al., 2023) which requires an agent to complete high-level user instructions on mobile apps. The metric is step accuracy,

[2]https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

Table 5: **Comparing the AutoGUI functionality annotation type with existing types**. Qwen-VL is separately fine-tuned with the four annotation types. Our functionality annotation leads to superior grounding accuracy.

Data Size	Variant	FuncPred	MOTIF	ScreenSpot
25k	w/ Elem-HTML	5.3	11.7	5.7
	w/ Alt-Text	4.5	12.9	12.6
	w/ Condensed Func.	3.8	19.8	4.8
	w/ Func. (Ours full)	**21.1**	**22.5**	**16.4**
125k	w/ Elem-HTML	15.5	15.8	17.0
	w/ Alt-Text	12.8	16.7	24.3
	w/ Condensed Func.	14.1	23.7	23.8
	w/ Func. (Ours full)	**24.6**	**28.7**	**27.0**

where a planned step is considered correct only if the action type and arguments match ground truths. **2-stage planning** Following UGround (Gou et al., 2024), a planner model initially performs reasoning through task progress and UI content and then plans the next action. We prompt the planner to also describe the expected functionality of the target element for click actions. Next, a grounding model (Qwen2-VL-7B) trained with our functionality grounding tasks is used to locate the target according to the functionality description. See an example in Fig. L and additional details in Sec. C).

The results in Tab. J show that even the strong proprietary VLMs (e.g. Gemini) possess weak UI element grounding capability. Qwen2-VL trained with AutoGUI functionality grounding tasks can overtake the element grounding process of the proprietary models and help the planners achieve significantly higher step accuracy by correcting the target locations of the *click* actions.

Although this experiment is not designed to surpass expert models tailored for agent tasks, we hope it can facilitate further research of GUI agents with strong element grounding ability.

6 Conclusion

We propose AutoGUI, a scalable and automatic annotation pipeline aimed to produce massive UI element functionality annotations used to enhance UI grounding capabilities of VLMs. The pipeline prompts an open-source LLM to generate element functionalities based on the UI content changes induced by interacting with the elements. LLM-aided rejection and verification are used to guarantee high quality. Fine-tuned with the data collected by AutoGUI, the base models obtain stronger UI grounding ability and exhibit data scaling effects. We hope that AutoGUI will open up possibilities for advancing the field of UI agents.

Limitations

AutoGUI is dedicated to providing an autonomous way to collect scalable UI grounding/captioning data for training capable UI-VLMs. However, AutoGUI still encounters several limitations:

Lack of Diverse Mobile App Data. As many Apps implement anti-emulator code, it is extremely difficult to navigate through popular Apps, such as TikTok and WeChat, on Android emulators. To circumvent this issue, AutoGUI renders webpages at various resolutions, including smartphone resolution, to mimic diverse device types. Although mainstream websites, such as YouTube and Reddit, provide delicately designed webpage responsiveness for various resolutions, a number of less common websites do not possess such flexible responsiveness and distort severely when rendered at smartphone resolutions. Therefore, collecting UI data at a smartphone resolution probably leads to domain gaps between the collected data and real smartphone Apps that are not rendered with HTML.

AutoGUI is Not Indented to Record Task-Oriented Interaction Trajectories. AutoGUI randomly interacts with UIs to record transition trajectories and utilize the UI content changes to predict the functionalities of the interacted elements. Hence, the collected trajectories do not provide high-level task semantics. In other words, the AutoGUI dataset does not contain tasks that combine multiple low-level steps, such as selecting a check-in date and then a check-out date. These long-horizon tasks are usually generated by human annotators in the existing works (Deng et al., 2024; Rawles et al., 2023). In future work, we can also utilize capable LLMs to generate high-level tasks and then prompt the LLMs to interact with UIs according to the tasks.

AutoGUI Cannot Annotate UI Elements That Modify Content on the Internet To avoid causing potential contamination on the Internet and bearing unexpected responsibilities, we try our best to eliminate interaction samples that manipulate sensitive elements that probably modify contents on the Internet. For example, elements used to post comments, make purchases, and enter account information are discarded. Consequently, the AutoGUI pipeline mainly annotates elements that only support read-only functionalities.

References

AI@Meta. 2024. Llama 3 model card.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736.

Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. 2024. Screenai: A vision-language model for ui and infographics understanding. *arXiv preprint arXiv:2402.04615*.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. Uibert: Learning generic multimodal representations for ui understanding. In *International Joint Conference on Artificial Intelligence*.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond.

Yushi Bai, Jiahao Ying, Yixin Cao, Xin Lv, Yuze He, Xiaozhi Wang, Jifan Yu, Kaisheng Zeng, Yijia Xiao, Haozhe Lyu, et al. 2024. Benchmarking foundation models with language-model-as-an-examiner. *Advances in Neural Information Processing Systems*, 36.

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*.

Ruirui Chen, Chengwei Qin, Weifeng Jiang, and Dongkyu Choi. 2024. Is a large language model a good annotator for event extraction? *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17772–17780.

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, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Y. Li, Jeffrey Nichols, and Ranjitha Kumar. 2017a. Rico: A mobile app dataset for building data-driven design applications. *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017b. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, pages 845–854.

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.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*.

Longxi Gao, Li Zhang, Shihe Wang, Shangguang Wang, Yuanchun Li, and Mengwei Xu. 2024. Mobileviews: A large-scale mobile gui dataset. *Preprint*, arXiv:2409.14337.

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. *Preprint*, arXiv:2410.05243.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14281–14290.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2024. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. 2024. What matters when building vision-language models? *Preprint*, arXiv:2405.02246.

Dong-Ho Lee, Jay Pujara, Mohit Sewak, Ryen White, and Sujay Jauhar. 2023. Making large language models better data creators. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15349–15360, Singapore. Association for Computational Linguistics.

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

Y. Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020a. Widget captioning: Generating natural language description for mobile user interface elements. In *Conference on Empirical Methods in Natural Language Processing*.

Yang Li, Jiacong He, Xiaoxia Zhou, Yuan Zhang, and Jason Baldridge. 2020b. Mapping natural language instructions to mobile ui action sequences. *ArXiv*, abs/2005.03776.

Zhang Li, Biao Yang, Qiang Liu, Zhiyin Ma, Shuo Zhang, Jingxu Yang, Yabo Sun, Yuliang Liu, and Xiang Bai. 2024b. Monkey: Image resolution and text label are important things for large multi-modal models. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *ArXiv*, abs/2305.20050.

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

Ziyi Lin, Chris Liu, Renrui Zhang, Peng Gao, Longtian Qiu, Han Xiao, Han Qiu, Chen Lin, Wenqi Shao, Keqin Chen, Jiaming Han, Siyuan Huang, Yichi

Zhang, Xuming He, Hongsheng Li, and Yu Qiao. 2023. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models. *Preprint*, arXiv:2311.07575.

Fangyu Liu, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023a. Deplot: One-shot visual language reasoning by plot-to-table translation. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning.

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

Yuliang Liu, Biao Yang, Qiang Liu, Zhang Li, Zhiyin Ma, Shuo Zhang, and Xiang Bai. 2024b. Textmonkey: An ocr-free large multimodal model for understanding document. *arXiv preprint arXiv:2403.04473*.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. 2024. Deepseek-vl: Towards real-world vision-language understanding. *Preprint*, arXiv:2403.05525.

Arjun Panickssery, Samuel R Bowman, and Shi Feng. 2024. Llm evaluators recognize and favor their own generations. *arXiv preprint arXiv:2404.13076*.

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, Qixiang Ye, and Furu Wei. 2024. Grounding multimodal large language models to the world. In *The Twelfth International Conference on Learning Representations*.

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. 2014. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211 – 252.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. 2022. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile ui summarization with multimodal learning. *The 34th Annual ACM Symposium on User Interface Software and Technology*.

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. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *Preprint*, arXiv:2409.12191.

Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. 2024b. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *Advances in Neural Information Processing Systems*, 36.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. 2022. Large language models are better reasoners with self-verification. In *Conference on Empirical Methods in Natural Language Processing*.

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

Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Min Dou, Botian Shi, Junchi Yan, et al. 2024. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *arXiv preprint arXiv:2402.12185*.

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

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

Jiabo Ye, Anwen Hu, Haiyang Xu, Qinghao Ye, Ming Yan, Yuhao Dan, Chenlin Zhao, Guohai Xu, Chenliang Li, Junfeng Tian, Qian Qi, Ji Zhang, and Fei

Huang. 2023a. mplug-docowl: Modularized multimodal large language model for document understanding. *Preprint*, arXiv:2307.02499.

Jiabo Ye, Anwen Hu, Haiyang Xu, Qinghao Ye, Ming Yan, Guohai Xu, Chenliang Li, Junfeng Tian, Qi Qian, Ji Zhang, Qin Jin, Liang He, Xin Lin, and Fei Huang. 2023b. UReader: Universal OCR-free visually-situated language understanding with multimodal large language model. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2841–2858, Singapore. Association for Computational Linguistics.

Haoxuan You, Haotian Zhang, Zhe Gan, Xianzhi Du, Bowen Zhang, Zirui Wang, Liangliang Cao, Shih-Fu Chang, and Yinfei Yang. 2024. Ferret: Refer and ground anything anywhere at any granularity. In *The Twelfth International Conference on Learning Representations*.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2025. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *Computer Vision – ECCV 2024*, pages 240–255, Cham. Springer Nature Switzerland.

Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Yu Qiao, Hongsheng Li, and Peng Gao. 2024a. LLaMA-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *The Twelfth International Conference on Learning Representations*.

Yi-Fan Zhang, Qingsong Wen, Chaoyou Fu, Xue Wang, Zhang Zhang, Liang Wang, and Rong Jin. 2024b. Beyond llava-hd: Diving into high-resolution large multimodal models. *Preprint*, arXiv:2406.08487.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

A Appendix

The appendix comprises the following sections:

Section A: Details for implementation details for the autonomous annotation pipeline, including dataset statistics, visualized annotation pipeline, and LLM prompts.

Section B: Details for model implementation and training.

Section C: Additional experimental analysis including analysis of successful and failure cases on two benchmarks.

Section D and E: Limitations and Potential Societal Impact.

A Details of the AutoGUI Pipeline

A.1 Extra Statistics of the AutoGUI Dataset

Fig. A visualizes the verb-noun statistics of the AutoGUI dataset, highlighting its extensive coverage of diverse UI functionalities. Fig. B lists the top 50 most frequent top-level domains in the AutoGUI dataset, showing that the AutoGUI dataset involves a broad spectrum of real-world scenarios, including technology (e.g., apple.com), entertainment (e.g., tiktok.com), office (e.g., outlook.com), news (e.g., medium.org), and finance (e.g., paypal.com).

The approach to avoidding overlap between train and test data: Since our focus is on annotating contextual functionality for GUI elements, we define two elements as distinct if they serve different functions within their respective contexts. For example, two "magnifier" buttons on the same GUI might have different roles—one for zooming in and the other for searching. To ensure no contamination, we investigated whether the test elements appeared in the training set by checking if bounding box overlapping occurred on the same GUIs. After this analysis, we found no such overlap.

A.2 License of The Used Artifacts

The licenses of the data sources on which the AutoGUI dataset is built are listed in Tab. A. These sources are all allowed to be used for academic research.

A.3 Recording Interaction Trajectories on Web

Interactive Crawler for Common Crawl We design an in-house web crawler that interacts with most elements rendered on the web page. In contrast with existing methods which contain information for elements on the initial static web page for a given URL, our crawler randomly interacts with a rendered web page for **multiple steps** within a given action horizon T_{act} to collect UI data with abundant functional semantics. Fig. C compares the proposed AutoGUI and the existing annotation methods. We empirically set $T_{\text{act}} = 10$ in all our recordings. Therefore, our interactive crawler could collect functionality of elements that are not visible to static pages, including nested drop-down menus, date and location selectors, and secondary menus.

Data Source and Data Format To incorporate a wide basis of web pages, we first obtain a list of the top-200 most visited domains [3] and manually remove content delivery network (CDN) and not safe for work (NSFW) sites. We use URLs in this curated list as seeds to query the Common Crawl index [4] to find additional URLs with maximum sub-domain and path diversity. Querying URLs from the Common Crawl index ensures that our crawler respects each site's robots.txt file, making the dataset collection process legally safe. By obeying the directives in robots.txt, we avoid potential legal issues associated with unauthorized web scraping. For each web page, we collect the following data:

- Screenshot image of the rendered page
- Accessible Tree (AXTree) text representing the page's accessibility structure
- HTML source code of the page
- Accessible Node (AXNode) text for the specific element our crawler interacted with at each step

A.4 Recording Interaction Trajectories on Android Devices

We also implement an in-house crawler that interacts with multiple emulated Google Pixel phones. The phones are reset to different starting UIs before a script randomly interacts with these phones to record trajectories. To improve data diversity, the starting UIs include the home page, drop-down panel, settings page, and Apps drawer.

Similar to webpage HTML, mobile phone UIs are rendered with XML code, which is cleaned and converted to AXTree-like content before being used to annotate functionalities.

[3] https://tranco-list.eu/
[4] https://index.commoncrawl.org/

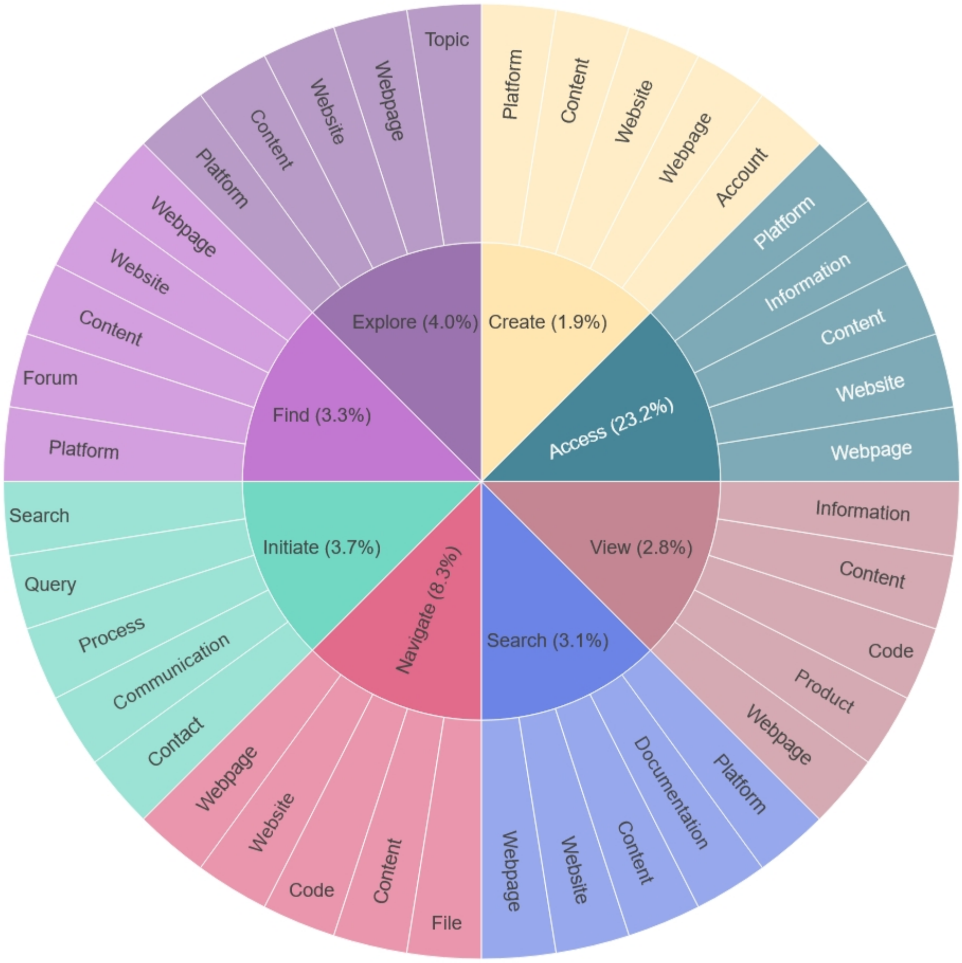


Figure A: **Diversity of the verb-noun phrases of the AutoGUI dataset.** The top 10 verbs and their top 5 following nouns are displayed. This diagram shows that our dataset contains diverse tasks that involve various UI functions.

Table A: License or terms for use and/or distribution of the used artifacts in this work.

Artifacts	License	URLs containing Term-of-Use or other license information
Common Crawl	CC BY	https://commoncrawl.org/terms-of-use
AndroidControl Trajectories	Apache 2.0	https://github.com/google-research/google-research/tree/41e4f1cbe1db648feb518a60501f638d9c8b25f2/android_control
Mobile Views Trajectories	MIT	https://huggingface.co/datasets/mllmTeam/MobileViews

A.5 Functionality Annotation Details

The AutoGUI pipeline utilizes UI content changes to predict the functionalities of the interacted elements. For interactions that manipulate the existing UI, the pipeline analyzes differences in the AXTrees to annotate functionalities. Conversely, when interactions result in navigation to a new UI, the pipeline examines changes in UI descriptions to guide the annotation process. Details on these methodologies are outlined below:

UI manipulation case We use a file-comparison library, DiffLib, to generate line-by-line differences of the AXtrees before and after interactions. To balance efficiency with annotation integrity, we limit the differences to 250 lines. In addition to the standard markings by DiffLib—addition, deletion, and unchanged status—we incorporate two additional change markers: 'Repositioning' and 'Attribute Update'. These markers provide detailed information about UI content changes, essential for representing realistic structural variations. For example, without the attribute update marker, a clicked menu icon would erroneously appear as both deleted and added in the difference output, despite the menu element remaining in place. An example of this case is shown in Fig. D. The used prompt is shown in Tab. B.

UI navigation case When an interacted element causes navigation to a new UI, the resultant changes are often extensive, potentially exceeding the con-

Table B: The functionality annotation prompt used in the AutoGUI pipeline in UI manipulation cases.

(Requirements for annotation)
Objective: As an Internet expert, your task is to describe the usage and functionality of a webpage element based on the changes observed in the webpage contents before and after interacting with the element.
Instructions:
1. You will be shown line-by-line differences between the webpage content before and after interacting with the element. Here's what each prefix indicates:
Unchanged: Lines that are identical before and after the interaction.
Added: New lines that appear after the interaction.
Deleted: Lines that were present before the interaction but removed afterward.
Renaming: Lines indicating elements that were renamed due to the interaction.
Attribute Update: Lines showing elements whose attributes were updated during the interaction.
Repositioned: Elements that were moved to a different part of the webpage.
2. You MUST thoroughly analyze the changes in webpage content (Added, Deleted, Unchanged lines) caused by interacting with the element, present a detailed reasoning process elucidating how the element affects the webpage, and finally summarize the element's overall purpose based on your analysis
3. Avoid detailing every specific functionality of the webpage element. Instead, focus on describing its broader impact on the webpage experience. For example, if interacting with a "Products" button reveals a dropdown menu, do not catalog the subsequent webpage changes in exhaustive detail.
4. Your output MUST follow this format:
Reasoning: (Examine the webpage variation carefully to figure out how the interacted element changes the webpage)
Summary: This element ... (Provide a concise high-level description of the element's function. This description should contain the meaningful feature of the element in its context.)
5. Avoid mentioning specific elements from the webpage before interaction in the Summary. Instead, focus directly on the outcome of the interaction. For instance, rather than detailing a transition from one element to another, like "This element switches the focus from the 'Destination' input field to the 'Check-in Date' field, providing a date picker," simplify it to "This element triggers a date picker for users to select their check-in date." This approach keeps the description concise and focused on functionality, eliminating unnecessary details.
6. For simplicity, you MUST not mention the text of the element in the summary.
Now it's your turn.

Element interaction: After clicking a <link> element named "Community submenu" (an example of the interaction target), the webpage exhibits the following variations:

(UI content changes)
Unchanged button 'Extend submenu' expanded: False
Unchanged StaticText 'Extend'
Unchanged button 'Learn submenu' expanded: False
Unchanged StaticText 'Learn'
Before Attribute Update button 'Community submenu' expanded: False
After Attribute Update button 'Community submenu' focused: True expanded: True
Unchanged StaticText 'Community'
Added link 'Make WordPress'
Added link 'Photo Directory'
Added link 'Five for the Future'
Added link 'Events'
Added link 'Job Board'
Unchanged button 'About submenu' expanded: False
Unchanged StaticText 'About'
Unchanged button 'Open Search' hasPopup: dialog
Unchanged link 'Get WordPress'

Now please think step-by-step, examine the content variation in detail, and finally summarize the high-level usage of the interacted element. Do follow the output format "Reasoning: ... Summary: This element ...".

(LLM response)
Reasoning: After interacting with the "Community submenu" button, the webpage content changes significantly. The button's "expanded" attribute is updated to "True", indicating that the button is now expanded. Additionally, the button gains focus, as indicated by the "focused: True" attribute. The most notable change, however, is the appearance of new. These new elements suggest that the button has triggered the display of a submenu or dropdown list related to the "Community" theme.
Summary: This element reveals a submenu of community-related links and resources.

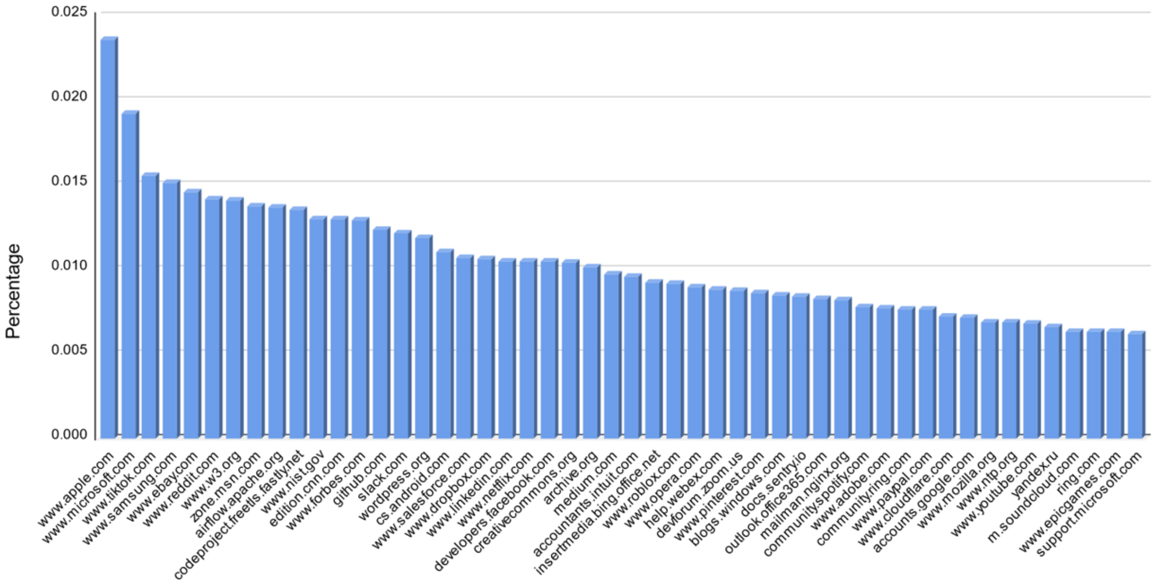


Figure B: The top-50 most frequent top-level domains in the AutoGUI dataset.

text limit of an LLM and complicating the analysis of these changes. To handle this situation, UI description changes are used to predict functionalities. Concretely, the LLM is initially prompted to describe the UIs before and after interaction given UI AXTrees as input. Subsequently, the LLM then uses these descriptions to analyze content changes and predict the functionality of the interacted element. The description length of the AXTree is limited to 150 lines. An illustration of this process is shown in Fig. E. The corresponding prompt is detailed in Tab. C.

A.6 Details of Rejecting Invalid Samples via Hand-Written Rules

To clarify the hand-written rules used in the process of removing invalid samples: (1) **Removing blank GUIs**. We remove blank GUIs by verifying whether the accessibility tree contains more than one node. If no nodes are present, the GUI is considered blank. (2) **Removing UIs containing elements indicating content loading**. GUIs containing elements indicative of content loading (e.g., keywords such as "loading", "please wait", or "refreshing") are excluded. These keywords typically suggest that the content has not fully loaded and may affect the validity of the sample. (3) **Removing interaction targets outside of screens**. Occasionally, part of the UI, including the interacted element, may fail to be captured. We filter out GUIs where interaction targets appear outside of the visible screen area. This is determined by checking whether the interacted element exists within the bounds of the recorded accessibility tree. Note that these rules are designed mainly for the domains from which we collected GUI metadata. Nevertheless, one can extend the rules flexibly according to the noise characteristics of new domains.

A.7 Details of Rejecting Invalid Samples via LLMs

Rejection process To eliminate invalid samples before functionality annotation, the AutoGUI pipeline prompts the annotating LLM to also determine the validity of samples by analyzing the predictability of the UI content changes. The LLM evaluates each sample against three criteria: 1) Explicitness of Changes: This measures how clearly the changes indicate the element's functionality. Changes that directly suggest functionality receive higher scores, while vague or irrelevant changes are not scored. 2) Relevance of Changes: This criterion assesses the significance of the modifications in relation to the element's intended function. Highly related modifications obtain a high score. No scores for irrelevant or unrelated content changes. 3) Predictability of Outcome: This involves determining how anticipated the interaction outcome is based on the changes, considering common web conventions and user experience principles. Highly predictable changes obtain a high score, whereas moderate, unexpected, or counter-intuitive outcomes receive no

Table C: The functionality annotation prompt used in the AutoGUI pipeline in UI navigation cases. This example shows how the LLM

(Requirements for annotation)
Objective: Your mission, as a digital navigation specialist, is to deduce and articulate the function and usage of a specific webpage element. This deduction should be based on your analysis of the differences in webpage content before and after interacting with said element.
Instructions:
1. You will be given descriptions of a webpage before and after interaction with an element. Your primary task is to meticulously analyze the differences in content resulting from this interaction to understand what the functionality of the element is in the webpage context.
2. You must present a detailed reasoning process before finally summarizing the element's overall purpose based on your analysis.
3. Prioritize examining changes in the webpage's regional content over individual element variations. This approach will provide a more holistic view of the element's impact on the webpage.
4. You should emphasize on the main content changes and pay less attention to less meaningful regions, such as headers, navigation bars, and footers.
5. Your output MUST follow this format:
Reasoning: (Examine the webpage variation carefully to figure out how the interacted element changes the webpage)
Summary: This element ... (Provide a high-level description of the element's functionality. This description should contain the meaningful feature of the element in its context.)
6. Avoid mentioning specific elements from the webpage prior to interaction in the Summary. Instead, focus directly on the outcome of the interaction. For instance, rather than detailing element changes, like "This element triggers the disappearance of the header and language selector elements and the emergence of a login form", simplify it to "This element triggers the display of a login page." This approach keeps the summary concise and focused on functionality, eliminating unnecessary details.
7. The final summary must emphasize the distinctive functionality of the element and not be too vague. For example, "This element redirects the user to a specific website, allowing them to access its content and services" is too broad and lacks meaningful detail. Instead, it should precisely describe how the element provides users with unique value
Now it's your turn.
The description of the initial webpage:

(An example description of the UI before the interaction)
Region 1 (Header): Navigation and login functionality
Region 2 (Main Content): Announcing updates to Instagram Direct Message, providing a blog post with a title, date, and content
Region 3 (Footer): Additional links and buttons (less meaningful region, summarized in a single sentence)
Overall Functionality: A blog post webpage announcing updates to Instagram Direct Message, providing a title, date, and content, with navigation and login functionality at the top.
After clicking a <link> element named "" (an example of the interaction target), we jump to a new webpage.
The description of the new webpage is:

(An example description of the UI after the interaction)
Region 1 (Header): Displays the webpage title and heading.
Region 2 (Login Form): Allows users to log in to their Facebook account by providing their email address or phone number and password.
Region 3 (Forgot Password): Provides a link to recover a forgotten account password.
Region 4 (Create New Account): Offers an option to create a new Facebook account.
Overall Functionality: Enables users to access their existing Facebook account or create a new one, providing a gateway to the social media platform.

Now please think step-by-step, examine the content variation in detail, and finally summarize the high-level usage of the interacted element. Do follow the output format "Reasoning: ... Summary: This element ...".

(LLM response)
Reasoning: Upon examining the webpage variations, I notice that the initial webpage is a blog post announcing updates to Instagram Direct Message, with navigation and login functionality at the top. After clicking the '<link>' element named "", the webpage transitions to a new page with a Facebook title and heading. The main content of the new webpage is a login form, allowing users to log in to their Facebook account. Additionally, there are options to recover a forgotten password and create a new Facebook account.
Summary: This element redirects the user to a Facebook login page, allowing them to access their existing account or create a new one.

score.

Rationale for setting the scoring range Given the UI content changes as the input, the LLM first presents detailed reasoning processes about the three criteria and then outputs an overall score summing the individual scores for each criterion, with each contributing 0 to 3 points for a maximum of 9 points. The LLM presents three rejection results with temperature = 1.0 for each sample. Samples falling in the bottom 30% of average scores are considered invalid and discarded. This method ensures a balance between high recall of actual invalid samples and retention of valid samples. The prompt is shown in Tab. D, the rejection process is illustrated in Fig. G, and several representative rejection examples are shown in Fig. H. Note that UI content changes are represented as line-by-line differences in UI manipulation cases, and as descriptive changes in navigation scenarios.

To validate the effectiveness of the chosen score range 0-3, we test the ranges 0-2, 0-3, and 0-4 to select a range that helps to reduce false positives (valid but rejected) and increase true positives (invalid and rejected). We used 216 tasks, including 147 valid and 69 invalid samples as the test bed. We then drew a line chart illustrating the rejection ratios (Y-axis) for both valid and invalid samples against various threshold settings (X-axis) (Note that a sample whose score ranks below the threshold will be discarded). The selection criteria: the area under the curve (AUC) for the valid samples should be as small as possible, while the AUC for invalid samples should be large, ensuring valid samples rank higher. The results in Fig. F show that when using the score range 0-3, the AUC for invalid samples is the largest while the value for valid ones is small, which suggests that this range achieves a better tradeoff between retaining valid samples and rejecting as many invalid samples as possible.

A.8 Details of LLM-Based Verification

Verification process To improve the quality of functionality annotations, the AutoGUI pipeline prompts two LLMs (i.e.g, Llama-3-70B and Mistral-7B-Instruct-v0.2) as verifiers to assign scores to samples based on how well the target elements adhere to their functionality annotations. The LLMs receive as the input a) the target element along with its surrounding UI content (up to 20 lines), b) the functionality annotation of this element, and c) the outcome of interacting with the element, either being the UI line-by-line differences (at most 250 lines) in manipulation cases or the UI description after the interaction in navigation cases. Given these inputs, the two LLMs generate two responses containing a score. Samples that do not achieve two full scores are discarded for higher quality of the AutoGUI dataset. The used prompt is shown in Tab. E and an example is illustrated in Fig. I.

Rationale for setting the scoring range To choose a score range that achieves a good tradeoff between rejecting incorrect annotations as many as possible while reducing the number of mistakenly rejected correct ones, we test three ranges: 0-2, 0-3, and 0-4. The selection metric is verification accuracy, which is defined as the number of correctly classified annotations divided by the total. The accuracy values for the three ranges are 0.93, 0.97, and 0.95, respectively. Therefore, we choose 0-3 as the scoring range of the LLM-based verifier.

A.9 Details of Grounding/Captioning Task Generation

After collecting the element-functionality pairs, the AutoGUI pipeline converts these pairs into functionality grounding and captioning tasks by formatting a multitude of task templates (several examples are shown in Tab. F). A functionality grounding task requires a VLM to output point coordinates of the element fulfilling the given functionality, while a captioning task demands that the VLM articulate a functionality description for an element, given its coordinates. It is important to note that each element-functionality pair is utilized to generate both a grounding task and a captioning task.

B Implementation Details

B.1 Human Evaluation Details

To justify the efficacy of the AutoGUI pipeline, we conducted a comparative evaluation of annotation correctness between a trained human annotator and the AutoGUI system. The human annotator was a graduate student proficient in using digital devices, ensuring familiarity with diverse user interfaces. This annotator was normally and adequately paid a monthly wage (roughly 600 dollars) as a student researcher.

We selected a set of 30 invalid samples, each showcasing a variety of element functionalities, to prepare the annotator for the annotation process. These functionalities included drop-down menu expansions, menu item selections, date-pickers, fil-

Table D: The rejection prompt used in the AutoGUI pipeline in UI manipulation cases. This example shows how the LLM assigns a low score to a sample that exhibits meaningless and unpredictable UI content changes.

(Requirements for rejection)
Your primary objective is to determine whether the changes in the webpage's content are sufficient for predicting the functionality of the webpage element causing these changes after being interacted with.
Instructions:
1. You will be shown the outcome (webpage changes) resulting from interacting with the element. The outcome can take one of two forms: changes to the webpage description, or line-by-line differences. For the latter form, here's what each prefix indicates:
Unchanged: Lines that are identical before and after the interaction.
Added: New lines that appear after the interaction.
Deleted: Lines that were present before the interaction but removed afterward.
Renaming: Lines indicating elements that were renamed due to the interaction.
Attribute Update: Lines showing elements whose attributes were updated during the interaction.
Repositioned: Elements that were moved to a different part of the webpage.
2. Analyze the provided outcome and provide detailed reasoning for whether this outcome helps to predict the element's functionality, considering the following stringent criteria:
1) Explicitness of Changes: Rate how directly the changes suggest the element's functionality. Score 1-3 for clear, unambiguous changes. Clearer changes obtain a higher score. No scores for vague, meaningless, or non-specific changes.
Positive Example: A button labeled "Show More" that, upon interaction, clearly adds new content below it. The direct addition of content clearly indicates a content expansion functionality. Score: 3
Negative Example: After clicking a "Details" button, the page layout changes subtly without adding relevant information or altering content in a meaningful way. The changes do not clearly relate to the button's presumed functionality. Score: 0
2) Relevance of Changes: Evaluate the significance of the modifications in relation to the element's intended function. Score 1-3 for changes that enhance understanding of the element's role. Highly related modifications obtain a high score. No scores for irrelevant or unrelated content changes.
Positive Example: Clicking on a "Contact Us" button opens a form to fill out, which is highly relevant to the button's intended functionality. Score: 3
Negative Example: Clicking on a "View Profile" link leads to a page refresh without displaying the profile or any related information, making the change irrelevant to the link's intended purpose. Score: 0
3) Predictability of Outcome: Assess how anticipated the interaction outcome is based on the changes, considering common web conventions and user experience principles. Score 1-3 for highly predictable outcomes. Highly predictable changes obtain a high score. No scores for outcomes that are moderate, unexpected, or counterintuitive.
Positive Example: Clicking or hovering over a "Help" icon reveals a tooltip with information, an outcome that users can easily predict given the icon's universal symbol for help. Score: 3
Negative Example: Hovering over a "Settings" icon changes its color but does not display any settings options, tooltips, or lead to a settings page, making the outcome unpredictable and the icon's purpose unclear. Score: 0
Provide a detailed analysis based on these criteria, ending with an overall score between 0-9.
4. Your output MUST follow this format:
Reasoning:
- Explicitness of Changes: [Detailed reasoning and score]
- Relevance of Changes: [Detailed reasoning and score]
- Predictability of Outcome: [Detailed reasoning and score]
Overall score: <score>[criterium 1 score] + [criterium 2 score] + [criterium 3 score] = [total score]</score> (output an additive equation enclosed with a pair of <score></score>)
Now it's your turn.
The interacted element is a <button> element named "Cancel" (interaction target)
Outcome:

(An example of interaction outcome)
After clicking a <button> element named "Cancel", the webpage exhibits the following variations:
Unchanged StaticText 'Email Address'
Unchanged textbox 'Email Address' required: True
Unchanged button 'Submit'
Unchanged StaticText 'Submit'
Before Attribute Update button 'Cancel'
After Attribute Update button 'Cancel' focused: True
Unchanged StaticText 'Cancel'
Unchanged contentinfo ''
Unchanged StaticText 'Your contact information is used to deliver requested updates or to access your subscriber preferences.'

Now please think step-by-step, examine the content variation in detail, and finally calculate scores according to the criteria. Do follow the output format "Reasoning: ... Overall score: <score>...</score>".

(LLM response)
Reasoning:
- Explicitness of Changes: The changes resulting from interacting with the "Cancel" button are mostly unchanged elements, except for the "Cancel" button itself, which now has a "focused: True" attribute. This change is not directly related to the button's functionality, as it does not provide clear insight into what the button does when clicked. Score: 1
- Relevance of Changes: The changes do not provide any significant information about the button's intended function. The button's name "Cancel" suggests that it might cancel some action or process, but the changes do not reveal what exactly is being canceled or how the button's functionality relates to the surrounding elements. Score: 1
- Predictability of Outcome: Given the name "Cancel" and the context of a form with an "Email Address" field and a "Submit" button, a user might expect the "Cancel" button to reset the form, clear the input field, or abort a submission process. However, the outcome does not provide any clear indication of what happens when the button is clicked. The changes are not predictable based on common web conventions and user experience principles. Score: 1
Overall score: <score>1 + 1 + 1 = 3</score>

Table E: The self-verification prompt used in the AutoGUI pipeline in UI manipulation cases. This example shows how the LLM assigns a low score to the incorrect functionality.

(Requirements for self-verification)
Given the following inputs:
1) Webpage content: This input represents the hierarchical structure of a webpage's elements, emphasizing semantic information and relationships. Each node in the tree includes details such as the element's role (e.g., button, link, heading), relevant attributes (e.g., expanded), and hierarchical relationships with other elements.
2) Task Description: This describes the action a user intends to perform (such as submitting a form, navigating to a particular section, or adjusting settings) or the information they seek (such as a specific content piece or form field). It also introduces a candidate element for evaluation and then presents the webpage changes caused by interacting with this element. Your task is to assess whether this element effectively facilitates the specified user action.
Your job is to:
1) Analyze the provided webpage content to understand the structure and semantics of the webpage's elements.
2) Evaluate the Candidate Element: Determine the suitability of the specified candidate element for the described action. Consider the element's role, attributes, and position within the hierarchy. Your evaluation should be grounded in how well these aspects align with the required functionality for the user's intended action.
3) Score the Element: Assign a score ranging from 0 to 3, enclosed within <score></score> tags. This score should reflect the degree to which the candidate element meets the action's requirements:
0: The element does not support the action in any capacity.
1: The element provides minimal support for the action.
2: The element supports the action but with limitations.
3: The element fully supports the action without significant limitations.
4) Provide Reasoning: Before presenting your score, offer a detailed explanation of your reasoning. This should cover your analysis of the webpage content, the relationship between the candidate element and the specified action, and how these factors informed your scoring decision.
5) Format for Your Answer:
Reasoning: (Provide a comprehensive analysis covering the webpage's insights, the relationship between the specified action and the candidate element, and the rationale behind your scoring decision.)
Score: <score>$[YourScoreHere]$</score>
(An in-context exemplar)
Example:
Webpage content:
[0] RootWebArea 'Rental Cars at Low, Affordable Rates'
[1] dialog 'Vehicle Class' modal: True
[2] radiogroup 'Vans'
[3] radio 'Minivans' checked: false
[4] radio 'Passenger Vans' checked: false
[5] radio 'Cargo Vans' checked: false
[6] button 'Cancel'
[7] button 'Apply Filter'
Task Description: Please identify the target element. The element helps users narrow down their vehicle choices to minivans specifically
Candidate element: [3] radio 'Minivans'
After interacting with the candidate element, the webpage exhibits these changes:
Upon clicking the "Minivans" <input> element, a new "Remove Filter" button is added to the dialog modal. The radio buttons for different vehicle classes remain unchanged, but the "Minivans" radio button is checked after the interaction.
Reference response:
Reasoning: The provided webpage content outlines a clear hierarchical structure for selecting vehicle types on a car rental webpage, categorized into sections like Cars, Vans, and Trucks, each with its own set of options represented as radio buttons.
The task involves narrowing vehicle choices to minivans. The candidate element is part of the 'Vans' radiogroup on a car rental webpage. This directly supports the user's action of narrowing choices to minivans. The element's interaction leads to its checking and introduces a "Remove Filter" button. Its role, position, and functionality support the user's intent.
Score: <score>3</score>
(The self-verification task)
Now it's your turn.
Current webpage content: (A UI content example)
[0] RootWebArea 'X for Business | X tips, tools, and best practices' focused: True
[2] link 'Get started in your ads account'
[4] link 'Campaign setup'
[6] link 'Editing and optimization'
[8] link 'Measurement'
[9] heading "Be what's happening"
[10] StaticText 'Spread your message, grow your audience, and get more traffic with Twitter Ads.'
=> [11] link 'Learn more' (The interaction target)
[13] heading 'Twitter is where people come to discover what's happening.'
[14] StaticText 'With over 500 million Tweets being sent each day, ...'
[15] StaticText 'Whether you're a CEO strengthening their personal brand...'
[16] link 'Get started with Twitter for Business'
[17] button 'Get started with Twitter for Business'
[18] heading "What's new on Twitter Business Outlook"
[20] link 'Google'
Task Description: Please identify the target element. This element redirects users to a non-existent page, providing a custom error message and alternative content instead of the expected information. (The functionality annotation of the element)
Candidate element: [11] link 'Learn more' (The interaction target)

(The interaction outcome)
After interacting with the candidate element, we navigate to a new webpage that contains these contents:
Region 1 (Header): Contains the website's logo and navigation links.
Region 2 (Error Message): Displays an error message indicating that the page does not exist, accompanied by a humorous image and a link to potentially related content.
Region 3 (Footer): Provides links to various policy and information pages, as well as copyright information.
Overall Functionality: This webpage serves as a custom 404 error page, providing a lighthearted and engaging way to notify users that the requested page is not available.

Now please do your job. Your answer:

(LLM response)
Reasoning: The candidate element, a link labeled 'Learn more', is part of the webpage's main content, surrounded by descriptions and headings related to Twitter for Business. However, upon interacting with this element, the webpage exhibits significant changes, redirecting users to a custom 404 error page with a humorous image and a link to potentially related content. In fact, the element's interaction leads to a completely different webpage with an error message and alternative content, which is unrelated to the initial webpage's purpose. The element's role, position, and functionality do not align with the user's intended action.
Score: <score>0</score>

Table F: Examples of the task templates used by the AutoGUI pipeline to convert element-functionality pairs to functionality grounding/referring tasks. {functionality} and {point} are placeholders to be replaced with the real functionalities and target points.

Task Type	Examples
Grounding	For the element I describe in this page capture, can you predict their positions (with point)? {functionality} In this web page image, please locate the element as I describe it (with point). {functionality} Given a screenshot, I will describe a specific element; your task is to predict their locations (with point). {functionality} I want to click the element that {functionality}. Please locate the target element.
Referring	Describe the function of the element at {point} on the screen. Detail the functionality of the UI element positioned at {point}. What kind of input or interaction is expected at the point marked {point}?

tering options, pop-up modals, webpage navigation, and zooming in/out buttons. The purpose of this selection was to expose the annotator to a broad spectrum of potential UI interactions, enhancing their ability to accurately assess element functionality based on UI content changes.

During the training phase, we provided the annotator with detailed guidelines, including three specific criteria outlined in Fig J, to ensure the clarity and correctness of their annotations. Additionally, we incorporated 15 invalid samples to instruct the human annotator on how to identify and exclude these cases during the evaluation process. These invalid samples encompassed scenarios such as incompletely loaded UIs, network failure incidents, login restrictions, and UIs displaying inappropriate content.

Following the training stage, the human annotator evaluated a total of 146 samples. Remarkably, the annotator successfully identified all invalid samples, achieving an overall annotation correctness rate of 95.5%. The few incorrect annotations were categorized as such due to vagueness or instances of hallucination, where the descriptions did not accurately reflect the UI elements.

B.2 Fine-Tuning Details

Qwen-VL-Chat (Bai et al., 2023), SliME (Zhang et al., 2024b), and Qwen2-VL-7B (Wang et al., 2024a) are selected as the base models in the experiments. To investigate the scaling effects of our dataset, 25k, 125k, and the entirety of the 702k samples in the training split are used as training data in the three scaling experiments. For the first two smaller-scale experiments, a subset of the 702k data is randomly sampled.

Pilot experiments find that the non-UI training data (i.e., LLaVA-instruct-150k and the Cauldron) significantly outnumber the 25k and 125k UI training data, resulting in data imbalance that biases the trained UI-VLM towards the general Q&A tasks in

Table G: The training hyper-parameters used for fine-tuning Qwen-VL in the experiments.

Hyper-Parameter	Value
Epoch	1
Global batch size	128
#GPUs	8
LoRA Rank	64
LoRA Alpha	16
Learning rate	3e-5
weight decay	0.1
ADAM Beta2	0.95
Warm-up ratio	0.01
LR scheduler	Cosine
Model max length	768
LoRA	ViT + LLM
DeepSpeed	ZeRO-2
#Parameters	Trainable params: 234,500,864 All params: 9,891,436,032 Trainable%: 2.3707
Data type	BFloat16

the non-UI data and leads to inferior UI grounding performance. To tackle this issue, the 25k/125k samples are resampled to the same number of the non-UI training data to enable the UI-VLM to acquire more supervising signals from the UI data. This resampling approach is not employed in the 702k experiment as this experiment does not encounter the imbalance issue.

We train our models based on the HuggingFace Transformers[5] and the PEFT library[6]. The training configurations are shown in Tab. G, Tab. H, and Tab. I.

Fine-tuning Qwen-VL-AutoGUI702k, SLiME-AutoGUI702k, Qwen2-VL-7B-AutoGUI702k, SeeClick w/ AutoGUI702k, UGround w/ AutoGUI702k consumed approximately 25 hours, 36 hours, 20 hours, 25 hours, 46 hours, respectively.

The framework used to fine-tune Qwen-VL and SeeClick w/ AutoGUI702k is the SeeClick codebase (Cheng et al., 2024); The framework used

[5]https://huggingface.co/docs/transformers/index
[6]https://huggingface.co/docs/peft/index

Table H: The training hyper-parameters used for fine-tuning SliME in the experiments.

Hyper-Parameter	Value
Epoch	1
Global batch size	128
#GPUs	8
Learning rate	3e-5
weight decay	0.0
ADAM Beta2	0.95
Warm-up ratio	0.03
LR scheduler	Cosine
Model max length	2048
Frozen module	ViT
DeepSpeed	ZeRO-2
#Parameters	Trainable params: 7535796224 All params: 8364644352 Trainable%: 90.09
Data type	BFloat16

Table I: The training hyper-parameters used for fine-tuning Qwen2-VL in the experiments.

Hyper-Parameter	Value
Epoch	1
Global batch size	128
#GPUs	8
LoRA Rank	128
LoRA Alpha	256
Learning rate	3e-5
weight decay	0.0
ADAM Beta2	0.95
Warm-up ratio	0.03
LR scheduler	Cosine
Model max length	2048
Frozen module	ViT
DeepSpeed	ZeRO-0
#Parameters	Trainable params: 322,961,408 All params: 8,614,337,024 Trainable%: 3.75
Data type	BFloat16

to fine-tune SLiME-AutoGUI702k is the SLiME codebase (Zhang et al., 2024b); The framework used to fine-tune Qwen2-VL-7B-AutoGUI702k and UGround w/ AutoGUI702k is LLaMA-Factory (Zheng et al., 2024).

B.3 Samples of Benchmarks

For clarity, the benchmarks' samples are visualized in Fig. K.

C Potential Use of AutoGUI Dataset

We mainly conduct 2-stage planning on the AITW (Rawles et al., 2023) benchmark to assess the benefits of our AutoGUI data on downstream agent tasks.

As illustrated in Fig. L, a planner is utilized to conduct reasoning and step prediction while a grounding model locates target elements for the actions that require targets (click, long-press, and hover). For other actions like swipe, back, home, and input-text, the grounding model is not involved. As this experiment requires the planner to describe the expected functionality of target elements, we use strong proprietary VLMs, such as GPT-4o-mini and Gemini-2.0 as the planners. Expert models, such as OS-ATLAS (Wu et al., 2024), are not used as they typically have lost general instruction following capability after large-scale fine-tuning.

The results in Tab. J show that Qwen2-VL-7B trained with our functionality grounding tasks can help the planners to more accurately locate target elements.

D Additional Experimental Analysis

D.1 Growing Grounding Performance Brought by Scaling Data Size

To further investigate the benefit of scaling the AutoGUI functionality data, the histogram of distance from a predicted point to the ground truth box center is plotted for the 25k, 125k, and 702k experiments. The results in Fig. M demonstrate that the distance distributions become denser at lower ranges, suggesting that increasing the AutoGUI training data leads to consistently improved grounding performances.

D.2 Case Analysis on FuncPred Test Split

Successful cases Fig. N demonstrates several examples of the grounding results from Qwen-VL trained with the 25k, 125k, and 702k AutoGUI data. The model trained with the 702k data (ours-702k) exhibits more accurate functionality grounding performance. For instance, Fig. N (a) shows that ours-702k predicts the point right on the target (The 'Get an account' button) while the other two models slightly miss the target. Case (c) shows that ours-702k correctly understands the functional intent to locate the WordPress logo, in contrast to the other models, which incorrectly focus on the text 'Get WordPress'. Additionally, case (f) illustrates that ours-702k successfully locates the three-dot menu icon, aligning with the intent to expand a dropdown menu. These results suggest that increasing the AutoGUI training data enhances the model's ability to understand complex functional intents and to recognize diverse iconic elements accurately.

Failure cases To explore the limitations of our model, we analyze several failure cases across the scaling experiments, as shown in Fig. O. The pri-

Table J: **Applying our AutoGUI dataset to 2-stage GUI agent task planning on AITW benchmark.** The planner model is prompted to describe the expected functionality of target elements in the reasoning content at each step. Then the grounding model locates the target element according to the functionality description by outputting specific coordinates. The results show that strong proprietary models possess weak element grounding capability. Qwen2-VL trained with AutoGUI functionality grounding tasks can overtake the element grounding process of the proprietary models to achieve significantly higher step accuracy. **Step Acc.** means the percentage of correctly planned actions regardless of action types while **Click acc.** means the percentage of correctly planned click actions.

Planner	Grounding Model	**General** **Step acc. / Click acc.**	**Install** **Step acc. / Click acc.**	**Google Apps** **Step acc. / Click acc.**	**Single** **Step acc. / Click acc.**	**Webshopping** **Step acc. / Click acc.**	Avg Step acc.
GPT-4o-mini	GPT-4o-mini	14.85 / 9.58	11.17 / 5.76	12.08 / 6.85	21.09 / 11.24	10.89 / 11.22	14.01
	Qwen2-VL-7B SFT w/ AutoGUI	20.43 / 20.56	25.59 / 22.49	15.25 / 12.33	25.59 / 22.49	16.15 / 20.53	**18.37** (+4.36)
Gemini-2.0-flash-exp	Gemini-2.0-flash-exp	26.37 / 18.16	28.49 / 26.91	30.30 / 22.88	41.94 / 28.95	20.22 / 22.65	29.50
	Qwen2-VL-7B SFT w/ AutoGUI	36.34 / 36.54	50.95 / 48.95	40.99 / 40.52	50.95 / 48.95	32.83 / 43.52	**39.23** (+9.73)

mary failure cases comprise (1) Difficulty in accurately locating very small target elements, as illustrated by the tiny 'Policy' button in case (a); (2) Misunderstanding functional intents, as shown in case (b) where the three models fail to locate the element for account creation and case (g) where ours-702k mistakenly focuses on navigating to previous content instead of subsequent content; (3) Challenges in recognizing abstract iconic elements, as seen with the map style icon in case (d) and the compass icon in case (f).

Despite these challenges, the enhanced performance observed with ours-702k supports the potential of the AutoGUI pipeline to further improve functionality grounding. The successful cases underscore that increasing the size of the training dataset not only boosts the model's ability to interpret functional intents but also its capability to process a variety of textual and iconic elements effectively.

D.3 Case Analysis on MoTIF Test Split

We evaluate the instruction following ability on MoTIF dataset. Our analysis focuses on two aspects: (1) what improvements our model can achieve with the scaling of our functionality dataset (Fig. P); and (2) in which scenarios our model still fails to achieve correct grounding (Fig. Q).

Fig. P shows that the model can more accurately understand the action instruction and make meaningful localization as scaling improves from 125k to 702k. For instance, when the objective is to *click sleep noise recording and click enable*, the model can comprehend the semantics of this global objective and identify *turn on*. Additionally, the model can mitigate localization errors, such as the 702k being more accurately positioned on the target element (e.g., the icon of *reservation*) than the 125k. However, MoTIF still struggles with certain tasks. For example, as shown Fig. Q, it has difficulty with localization in fine-grained steps for the instruction *search for Kingston Drive and show me the route to it*. It can be seen that the model does not effectively understand situations involving widget pop-ups (e.g., protocol and advertisement). This may be attributed to the weak semantic connection between pop-ups and the instruction. Furthermore, the model still falls short in precise localization. Enriching the dataset further could alleviate this issue.

E Potential Societal Impact

The potential societal impacts of the proposed AutoGUI can be considered across various dimensions:

Accessibility Enhancements VLMs trained with the AutoGUI data obtain stronger UI grounding capabilities, thereby possessing the potential to act as UI agents. By enabling context-aware understanding of UI functionalities, the VLMs can help users locate elements on complex UIs, significantly improving accessibility features in software. This could lead to the development of applications that are more intuitive for users with disabilities, such as those requiring screen readers or other assistive technologies.

Research Impact: By reducing the labor and time required for annotating UI data via the AutoGUI, the industry and academia could lower costs to easily build UI agents. This could also shift labor demands towards more creative and strategic roles rather than repetitive annotation tasks.

Privacy and Security Concerns: Although we employ precautions of eliminating samples related to sensitive UI elements (e.g., avoid interacting with elements modifying the Internet and use only popular public websites without exposing privacy), corner cases still exist on the vast Internet. UI data involving either content modification or personal information are hard to discern as UI designs are distinct and no universal detection rules exist. Therefore, it is essential for cyber-security research to consider the potential leakage of personal information in the collected data and devise preemptive protective approaches.

Potential for Bias and Fairness: The bias of the LLMs used in the AutoGUI annotation pipeline is probably reflected in the collected data, leading to a trained UI-VLM that inherits the bias. Therefore, mitigating bias in the LLM's annotations will be important for developing fair VLM agents that align with the values of users from diverse cultures.

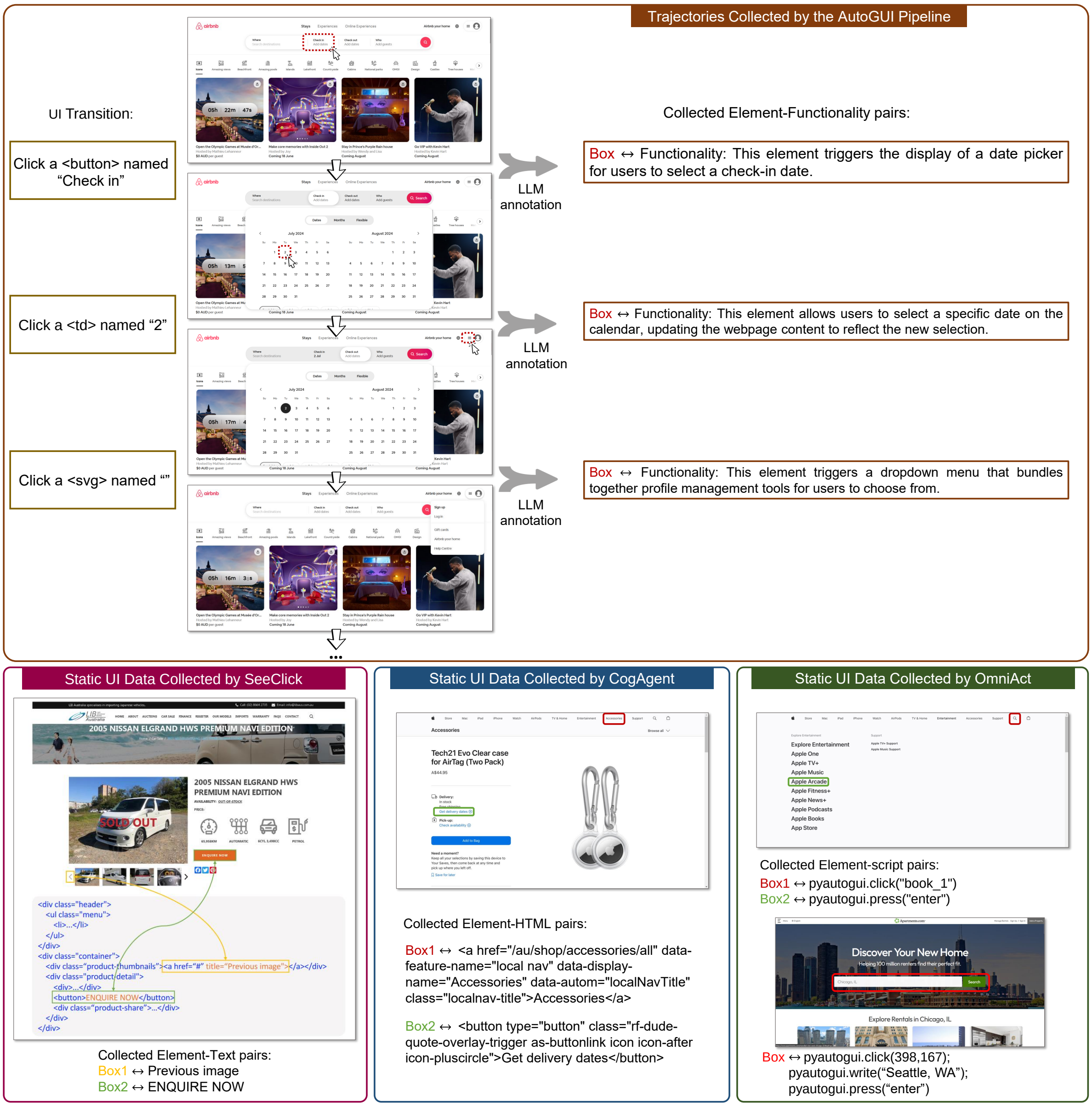


Figure C: **Comparing the proposed AutoGUI annotation pipeline with existing methods.** AutoGUI is able to manipulate real UIs and interact with elements hidden beneath deeper levels (e.g., the buttons hidden in collapsed dropdown menus), thereby collecting considerably rich element-functionality annotations from the immense UI resources on the Internet. In contrast, SeeClick (Cheng et al., 2024) only uses static webpages and collects static element-text pairs. Likewise, CogAgent collects static element-HTML pairs while OmniAct generates Python scripts only for visible elements. These three existing methods can only annotate visible static UI elements and ignore the rich UI functional semantics entailed in interaction trajectories which are provided by our AutoGUI pipeline in abundance.

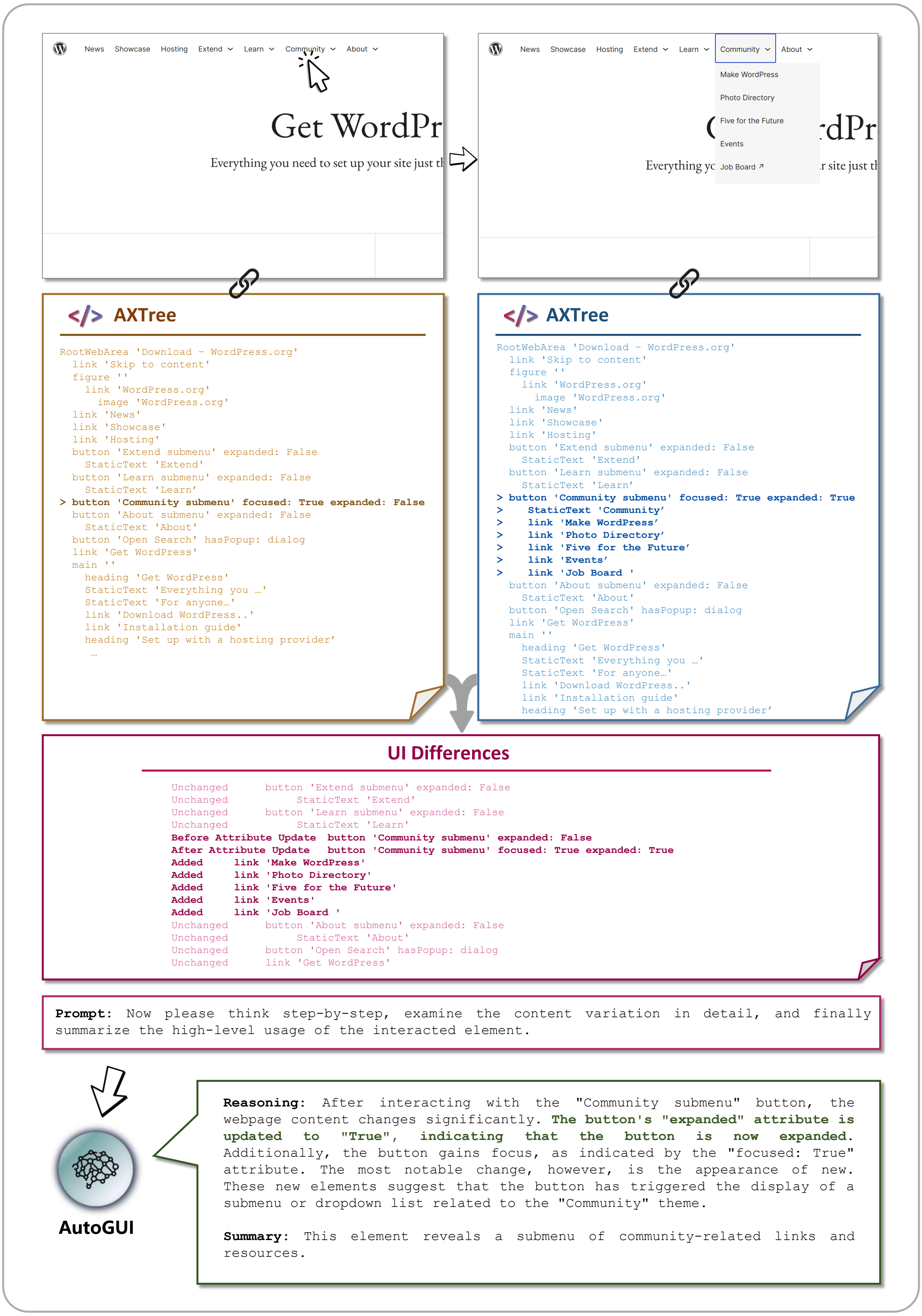


Figure D: An example of the AutoGUI functionality annotation using UI AXTree differences. AutoGUI records the AXTrees before and after interaction and then generates line-by-line differences with our custom change markers. Subsequently, the LLM takes the differences as input to predict the element functionality.

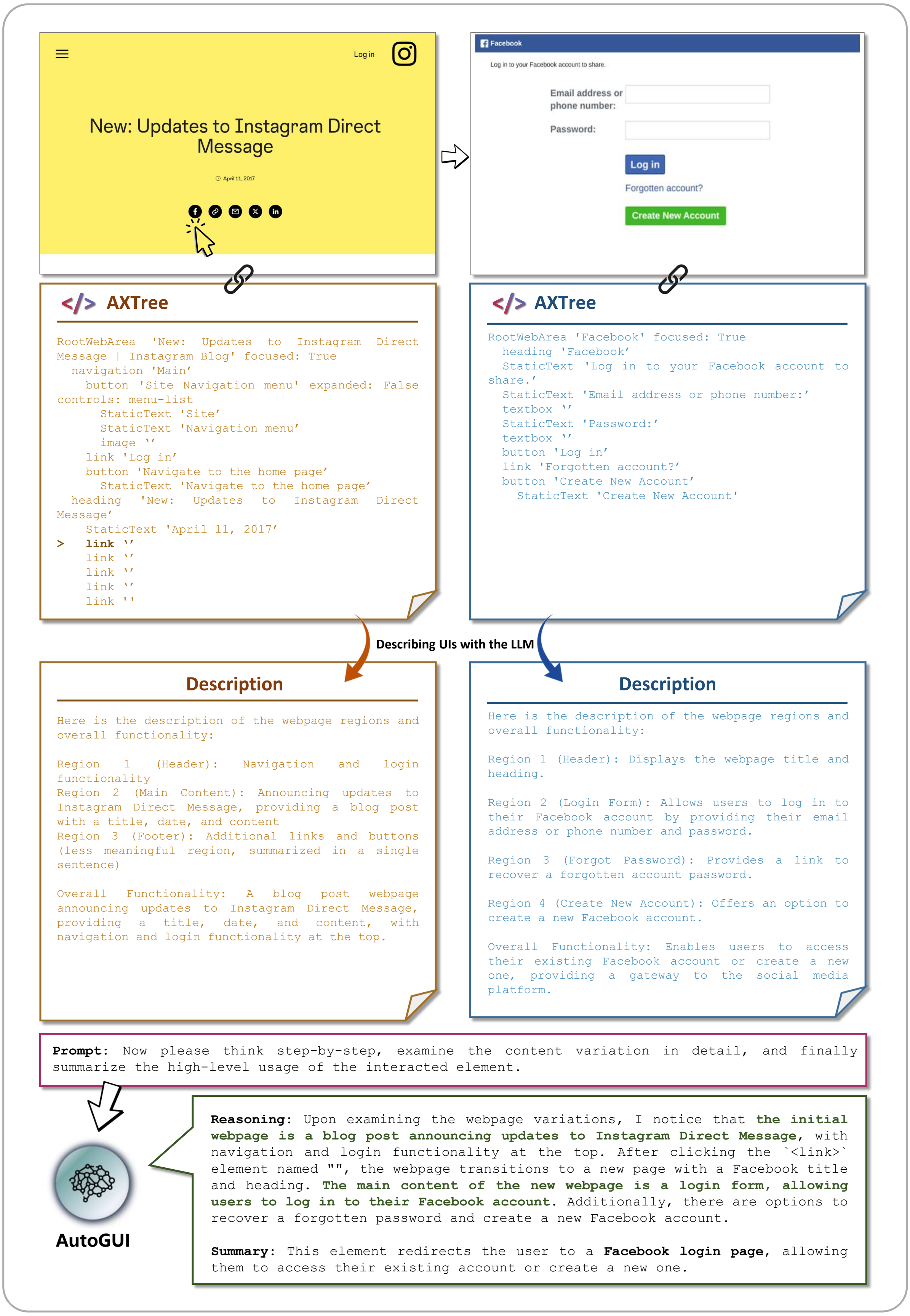


Figure E: An example of the AutoGUI functionality annotation using UI descriptions. AutoGUI records the AXTrees before and after interaction and then prompts the LLM to describe the AXTrees in detail. Subsequently, the LLM takes the two descriptions as input to predict the element functionality.

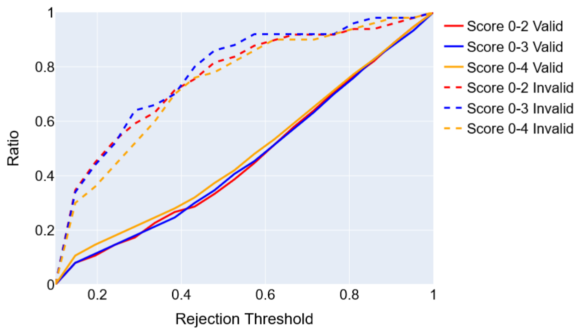


Figure F: **Rejection outcomes of the three score ranges used by the LLM-based rejector**. We plot the ratios of rejected valid and invalid samples when setting the rejection threshold to different values from 0.0 to 1.0. As the rejection threshold increases, more invalid and valid samples will be rejected. We expect the area under the curve (AUC) for rejecting invalid samples to be as large as possible while the AUC for rejecting valid ones to be as small as possible. The figure shows that using a score range of 0-3 leads to the largest AUC for invalid samples and a small AUC for valid ones.

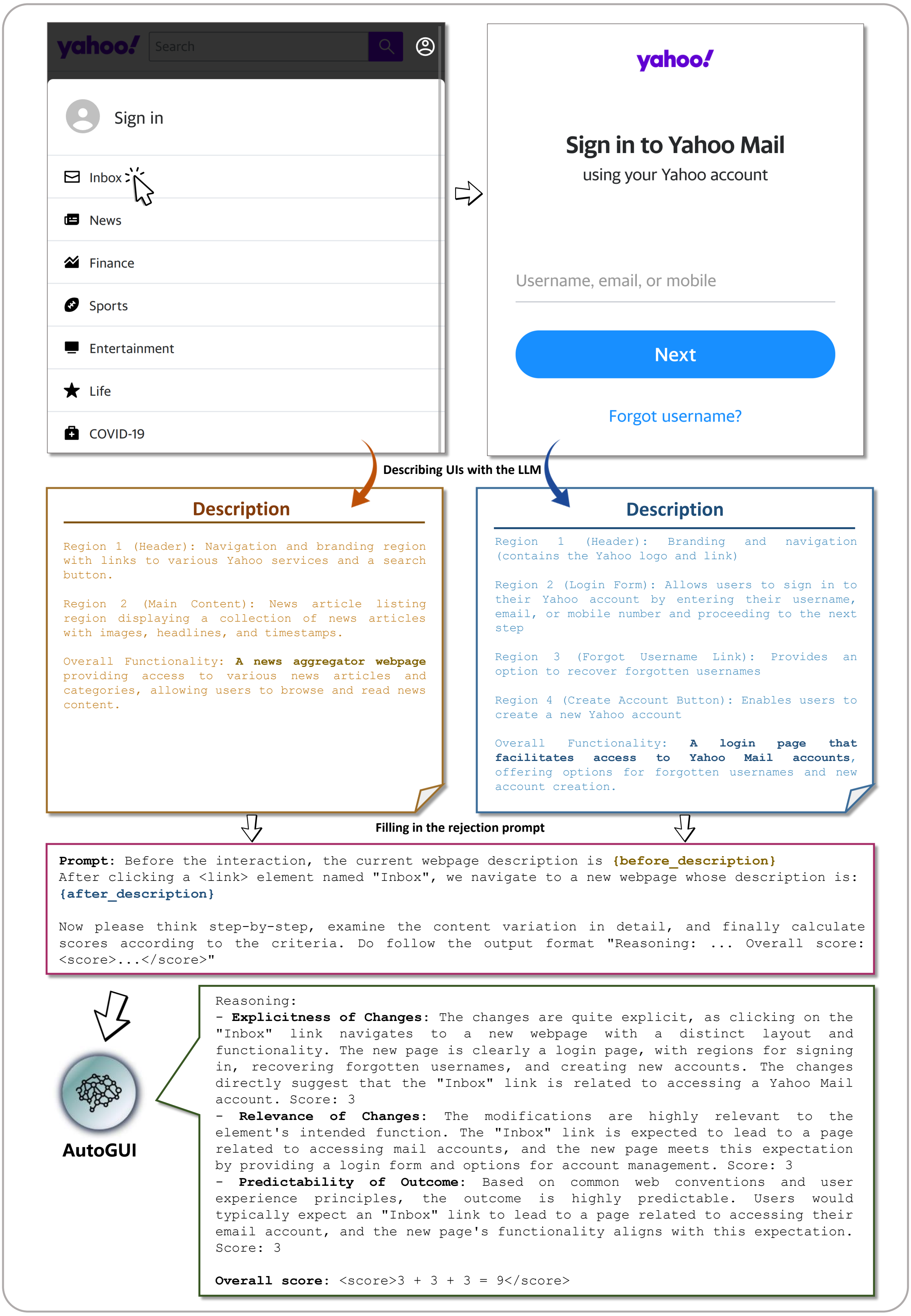


Figure G: An example of AutoGUI prompting the LLM as a rejector to determine whether a sample shows meaningful UI content changes sufficient for predicting the functionality of the interacted element. The sample shown is a navigation case in which AutoGUI uses UI descriptions, instead of line-by-line differences, to make decisions.

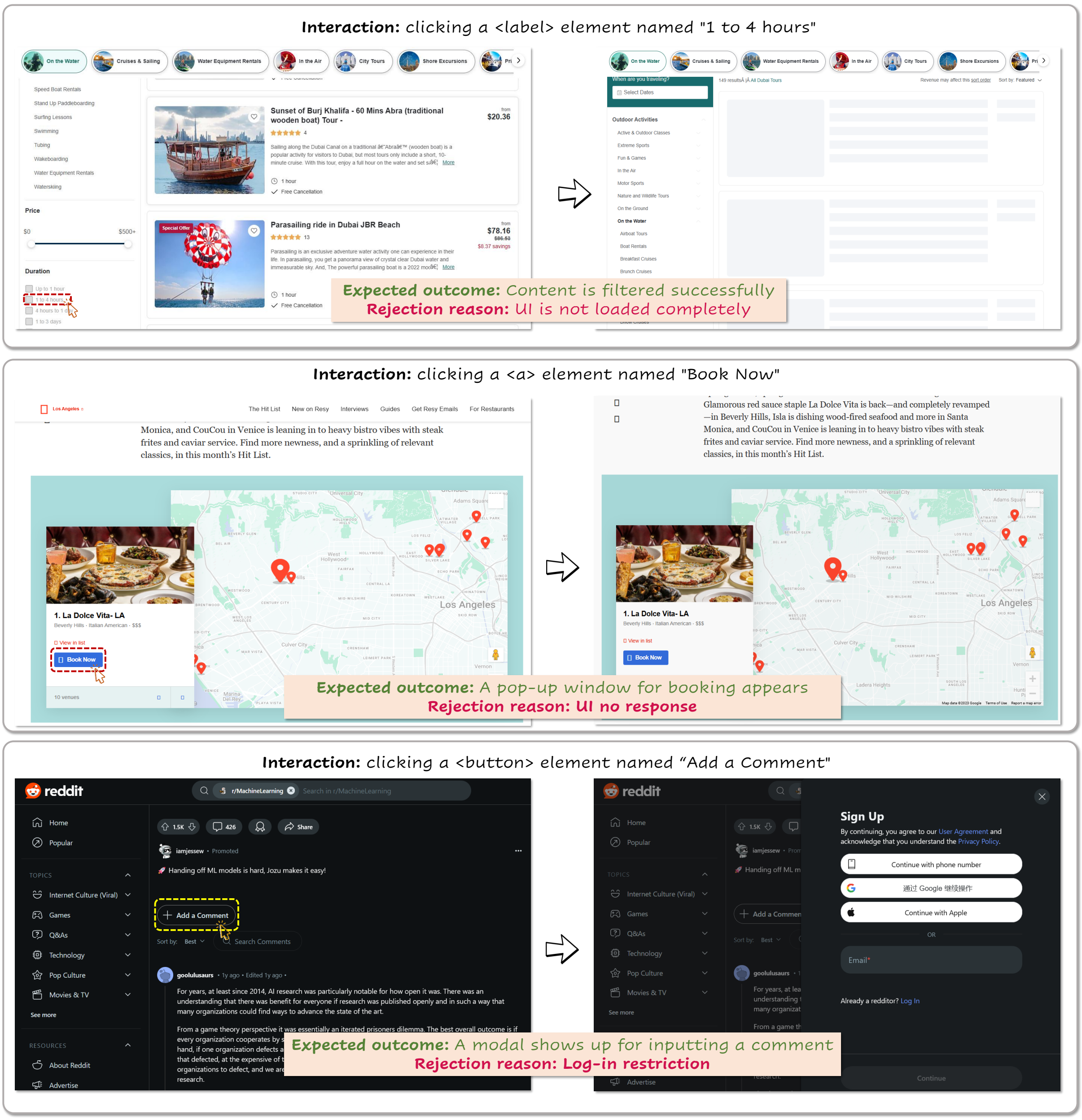


Figure H: **Examples of samples rejected by the AutoGUI pipeline.** The first sample encounters incompletely loaded content that interferes LLM annotation. The second encounters a no-response issue where the pop-up window fails to appear. The third shows a case where an unexpected log-in page pops up to interrupt the functionality of the "Add a Comment" element.

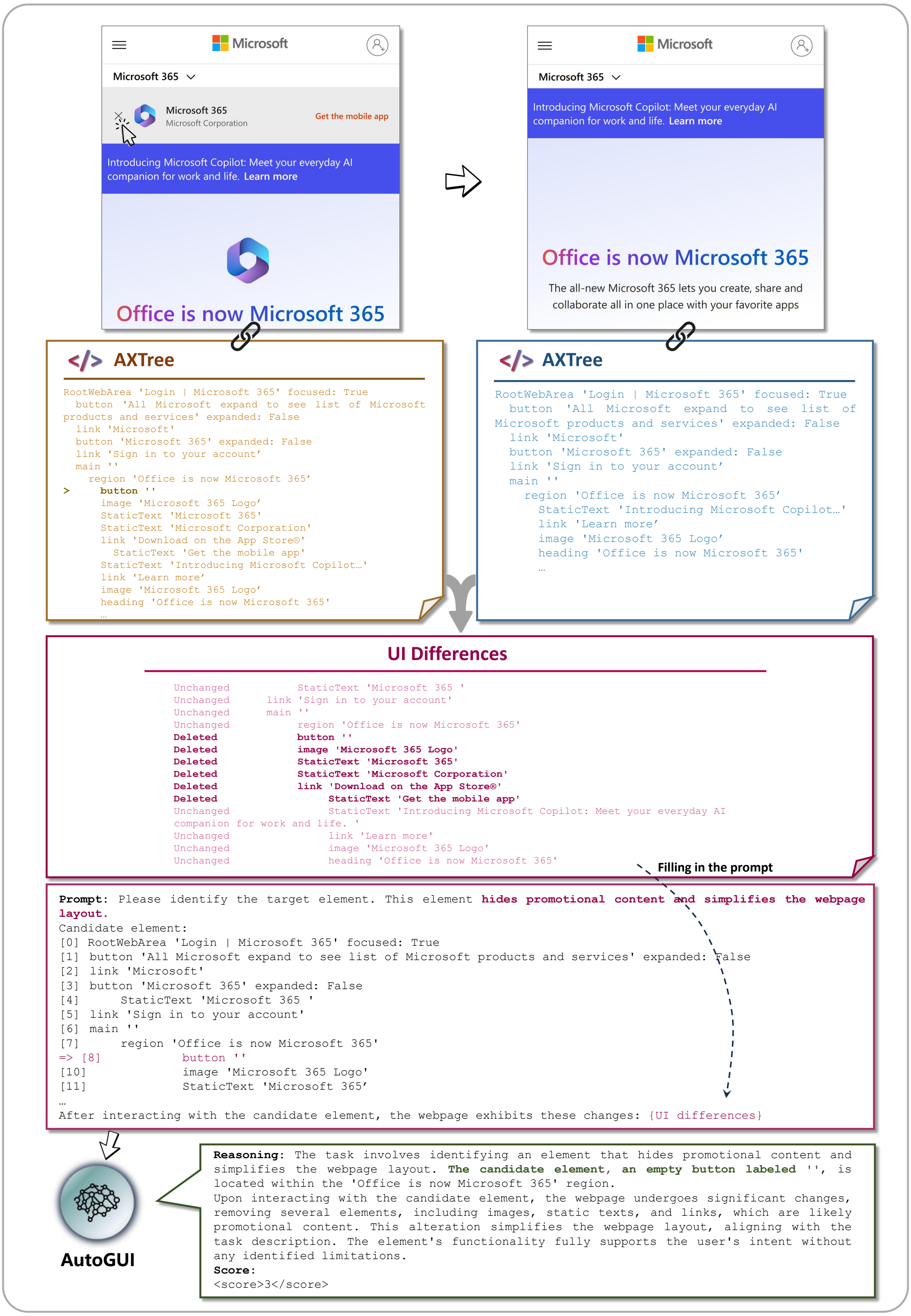


Figure I: An example of AutoGUI prompting the LLM as a self-verifier to determine whether an element supports its functionality annotation. The sample shown is a manipulation case in which AutoGUI uses UI line-by-line differences to make decisions about whether a button fulfills the intent of hiding promotional content.

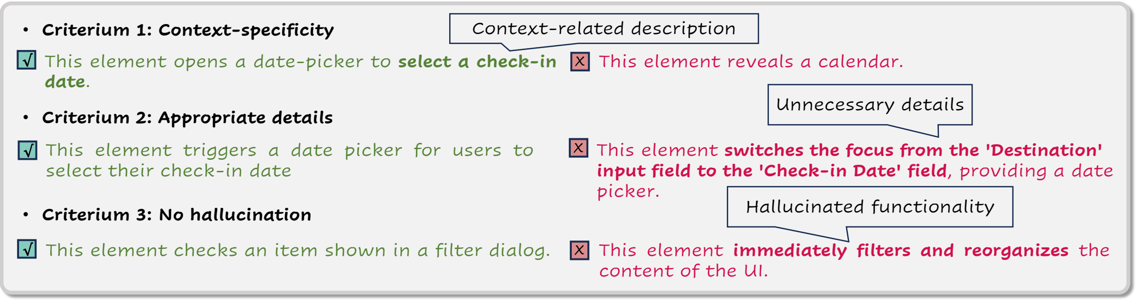


Figure J: The checking criteria used for comparing AutoGUI pipeline and the human annotator.

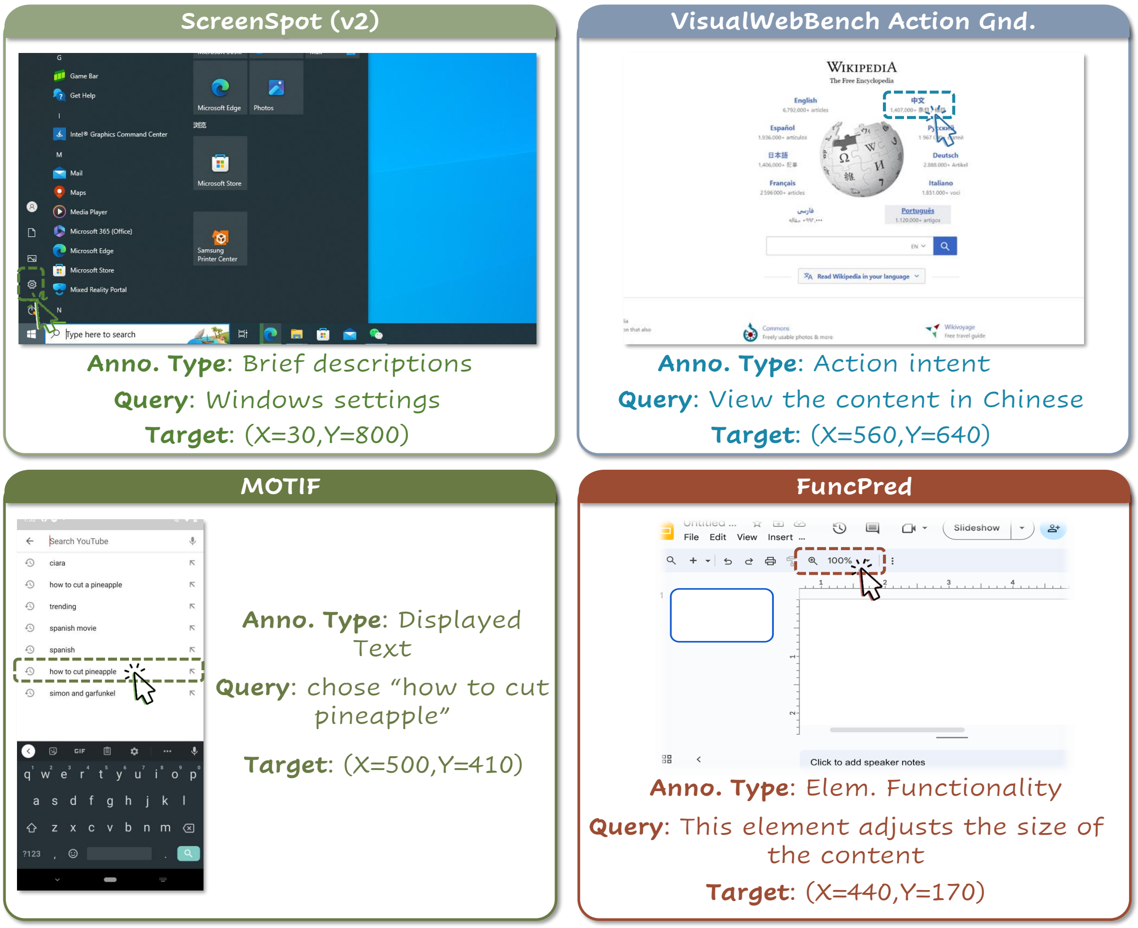


Figure K: Samples of the UI grounding benchmarks used in the experiments

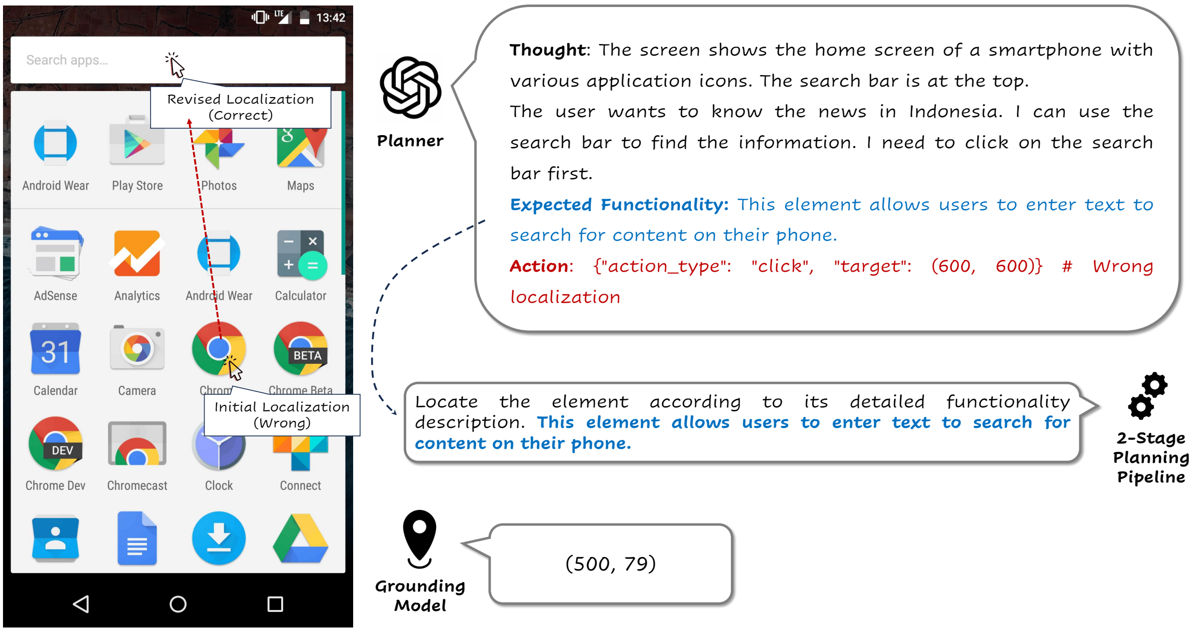


Figure L: An example of the 2-stage planning setting used to demonstrate the potential use of AutoGUI data. The planner (a proprietary VLM like Gemini) is bad at outputting numeric coordinates when locating elements. The grounding model, finetuned with AutoGUI functionality grounding tasks, can correctly locate the task-related target element according to the expected functionality description output by the planner.

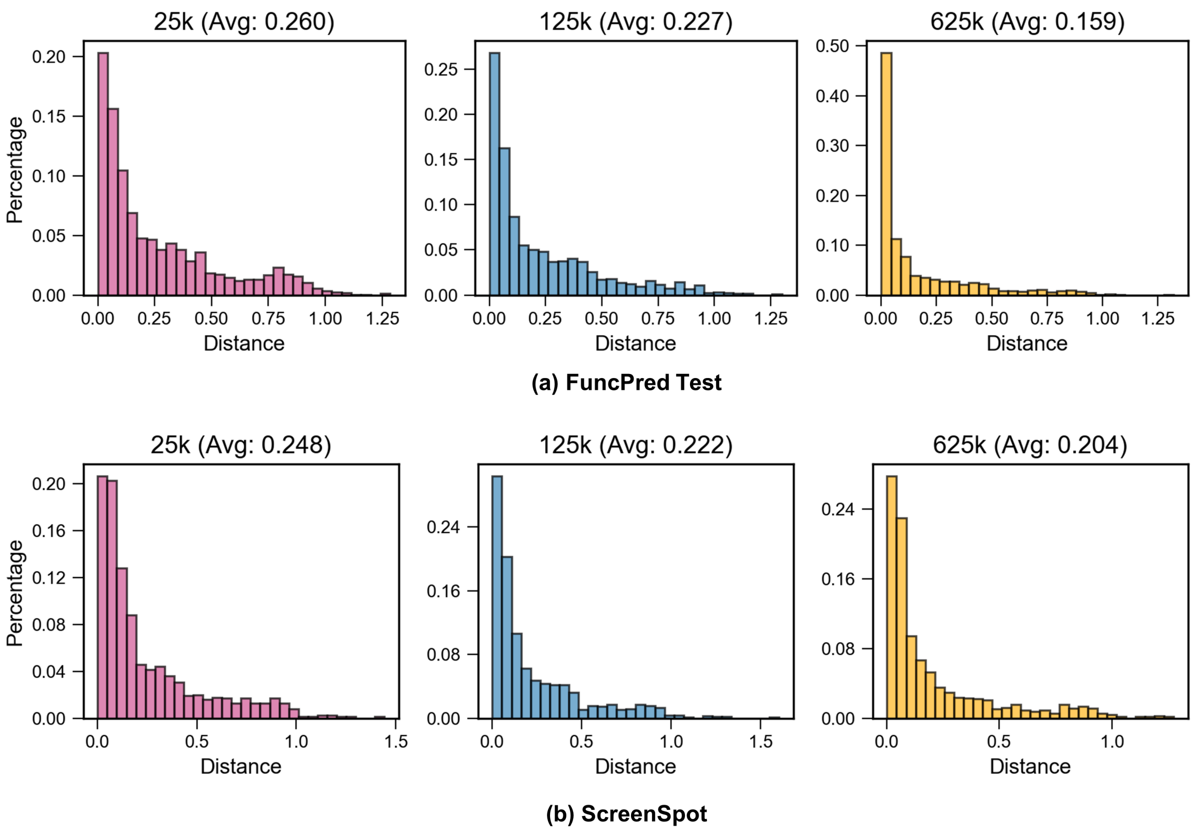
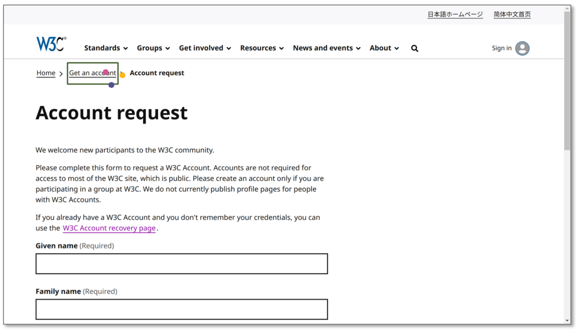
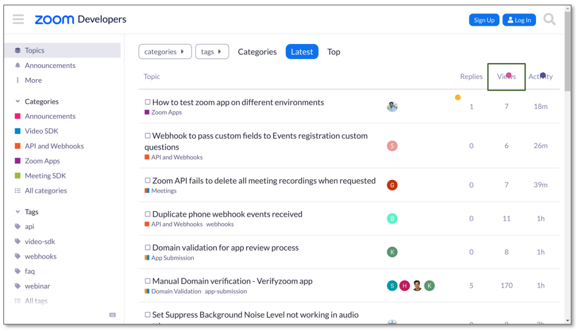


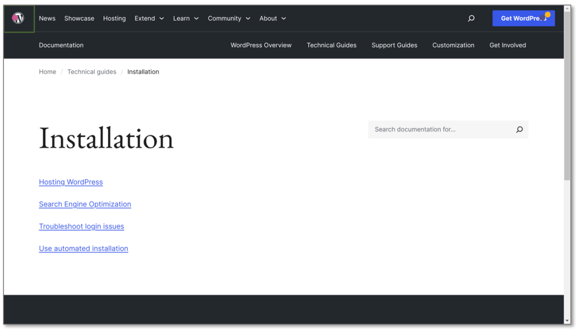
Figure M: **Histograms of distances from predicted points to ground truth box centers.** The distance from the normalized coordinate of a predicted point to its corresponding GT box center is calculated for all samples. Then, the histograms of these distances are illustrated to demonstrate the growing grounding performances brought by scaling the AutoGUI data size. The averaged distance for each experiment is displayed on the subplot title.



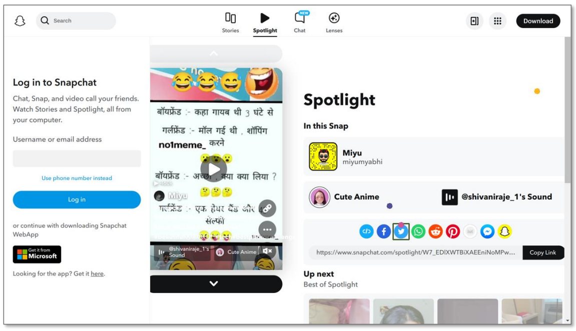
(a) Functionality: This element navigates to a page for creating or obtaining an account.



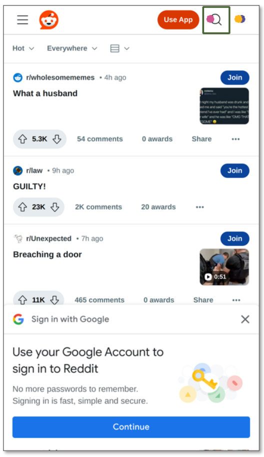
(b) Functionality: This element allows users to reorder the topic list by view count, making it easier to find popular or frequently viewed topics.



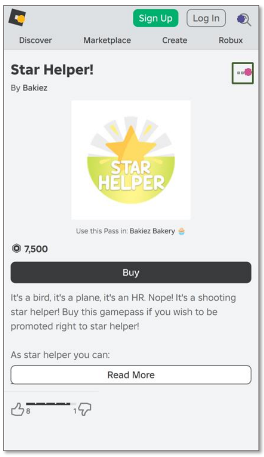
(c) Functionality: This element represents the primary brand or logo of the webpage, providing users with a direct access point to the homepage of the 'WordPress.org' website.



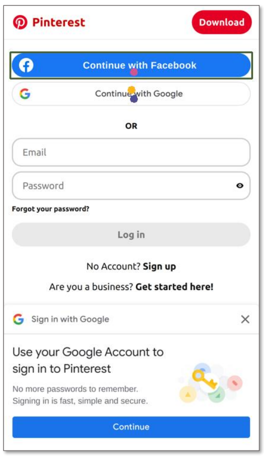
(d) Functionality: This element enables users to share content on Twitter.



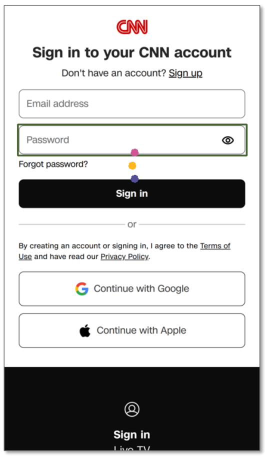
(e) Functionality: This element triggers the expansion of the search functionality on the webpage, allowing users to access more extensive search options.



(f) Functionality: This element triggers additional functionality or navigation within the webpage, such as revealing a dropdown menu.

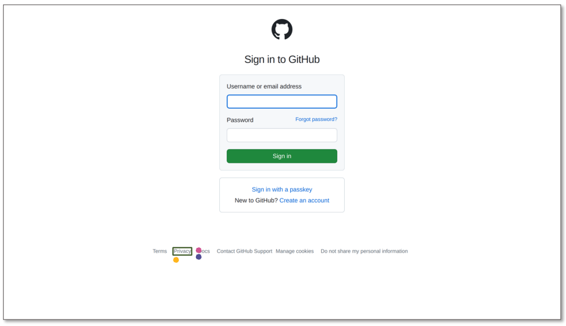


(g) Functionality: This element serves as a login gateway for the Pinterest app, allowing users to authenticate their accounts using Facebook.

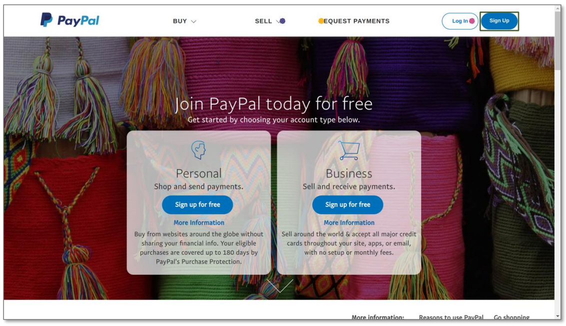


(h) Functionality: This element is a password input field, allowing users to securely enter their account password for authentication during the login process on the CNN website.

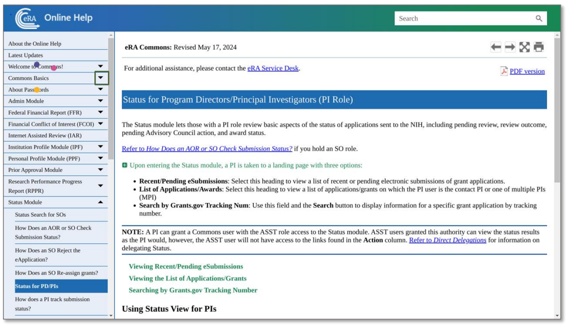
Figure N: **Visualization of the successful functionality grounding examples for ours-625k.** The ground truth bounding boxes, ours-625k predictions, ours-125k predictions, and ours-25k predictions are drawn in green, pink, blue, and orange, respectively.



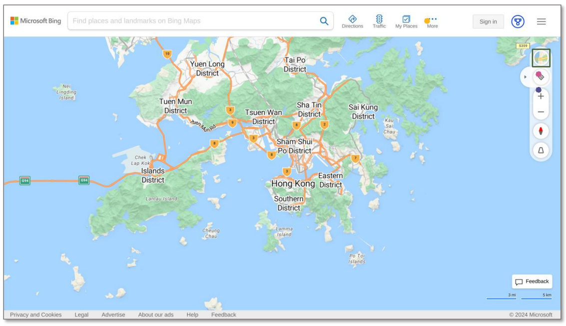
(a) Functionality: This element provides access to the privacy policy of GitHub, giving users important information about how their data is managed and handled.



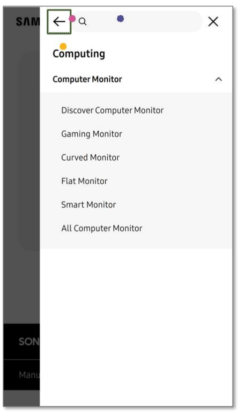
(b) Functionality: This element initiates the account creation process for new users.



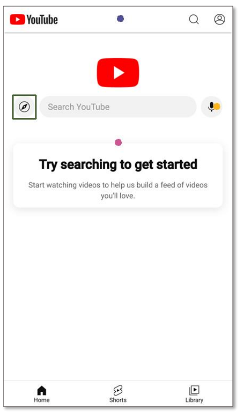
(c) Functionality: This element provides access to basic information and resources about the Commons system.



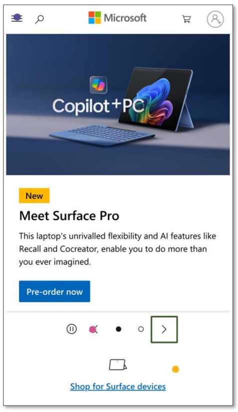
(d) Functionality: This element allows users to customize the map's visual style.



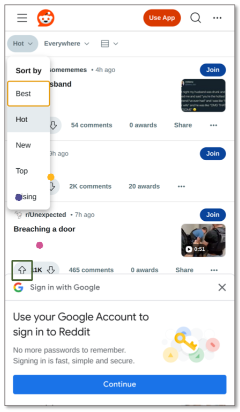
(e) Functionality: This element allows users to navigate back to the previous menu or page within the SONOACE R3 | Samsung Support Bangladesh webpage.



(f) Functionality: This element allows users to discover and explore the platform's trending and popular content, providing a gateway to various sections and categories of the video-sharing platform.

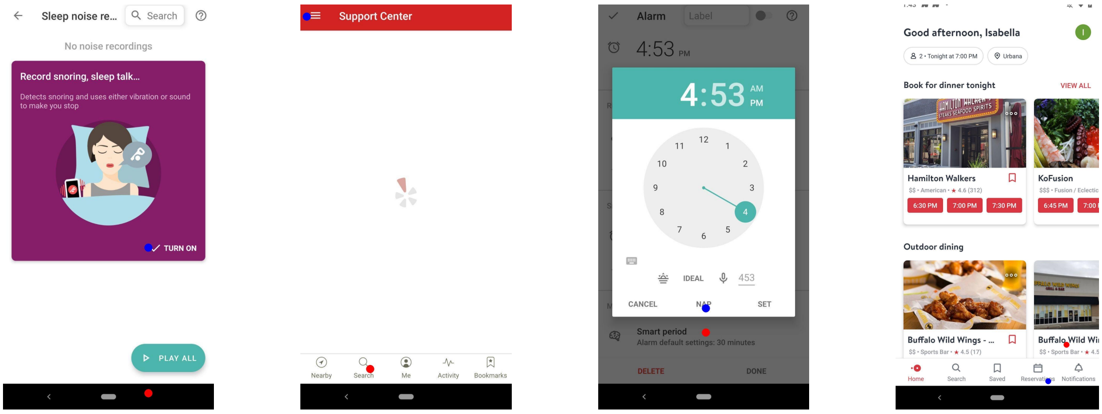


(g) Functionality: This element advances the user to the subsequent slide within the slideshow of featured products and announcements, providing a means for users to browse through the displayed content.



(h) Functionality: This element is an upvote button for users to express their approval of an article.

Figure O: **Visualization of failure examples in the scaling experiments.** The ground truth bounding boxes, ours-625k predictions, ours-125k predictions, and ours-25k predictions are drawn in green, pink, blue, and orange, respectively.



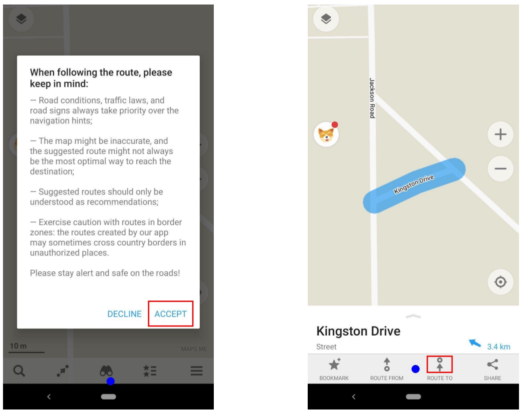
Click Sleep noise recording and click enable

Open settings and open push notifications and turn off deals and announcements

Set an alarm for a nap that is 30 minutes long

Click Reservations Tab

Figure P: Evaluation results of the model trained on 625k (blue dot) and 125k (red dot).



Search for Kingston Drive and show me the route to it

Figure Q: Bad cases on MoTIF.

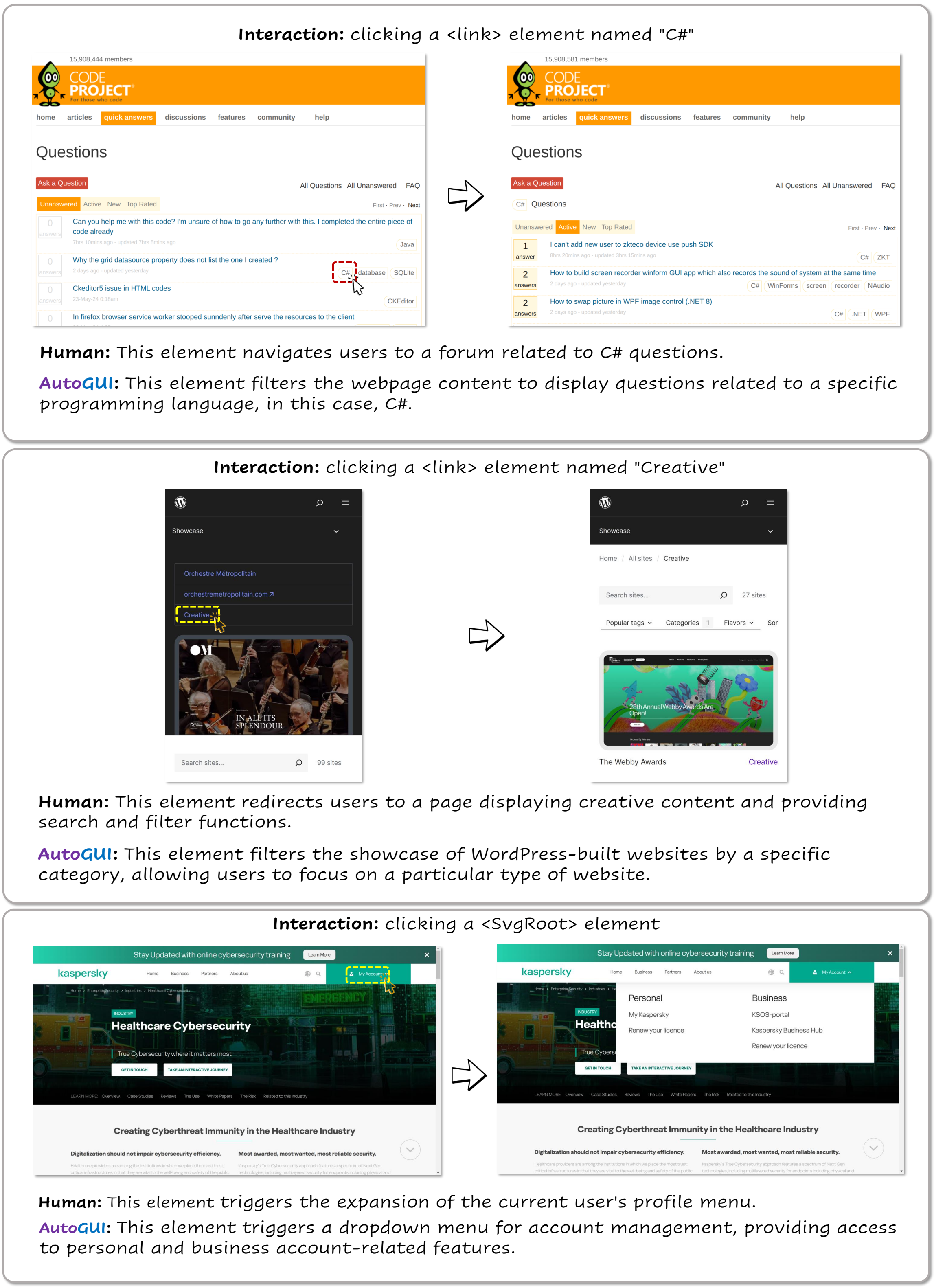


Figure R: Comparing the annotations generated by a trained human annotator and the proposed AutoGUI pipeline. We can see that AutoGUI annotations are more detailed and clear than those by the human annotator.