AMEX: Android Multi-annotation Expo Dataset for Mobile GUI Agents

Yuxiang Chai* MMLab, CUHK
Siyuan Huang* SJTU, Shanghai AI Lab
Yazhe Niu MMLab, CUHK, Shanghai AI Lab
Han Xiao MMLab, CUHK
Liang Liu vivo AI Lab
Dingyu Zhang MMLab, CUHK
Peng Gao Shanghai AI Lab
Shuai Ren vivo AI Lab
Hongsheng Li✉ MMLab, CUHK

Abstract

AI agents have drawn increasing attention mostly on their ability to perceive environments, understand tasks, and autonomously achieve goals. To advance research on AI agents in mobile scenarios, we introduce the Android Multi-annotation EXpo (AMEX), a comprehensive, large-scale dataset designed for generalist mobile GUI-control agents. Their capabilities of completing complex tasks by directly interacting with the graphical user interface (GUI) on mobile devices are trained and evaluated with the proposed dataset. AMEX comprises over 104K high-resolution screenshots from 110 popular mobile applications, which are annotated at multiple levels. Unlike existing mobile device-control datasets, e.g., MoTIF [2], AITW [19], etc., AMEX includes three levels of annotations: GUI interactive element grounding, GUI screen and element functionality descriptions, and complex natural language instructions, each averaging 13 steps with stepwise GUI-action chains. We develop this dataset from a more instructive and detailed perspective, complementing the general settings of existing datasets. Additionally, we develop a baseline model SPHINX Agent and compare its performance across state-of-the-art agents trained on other datasets. To facilitate further research, we open-source our dataset, models, and relevant evaluation tools. The project is available at `https://yuxiangchai.github.io/AMEX/`.

1 Introduction

AI assistants on mobile devices, such as Siri on iPhones, Bixby on Samsung devices, and Xiao AI on Xiaomi smartphones, have become increasingly prevalent in recent years. While these assistants are adept at managing various routine tasks like setting alarms, conducting web searches, and reporting weather conditions, their capabilities are predominantly confined to interacting with system-built applications. Furthermore, although many can interface with third-party apps via APIs, the lack of universal API support across different mobile operating systems often hampers their performance.

In contrast, human users can complete tasks on mobile devices purely based on visual information from the screen. Inspired by this, researchers are exploring alternative approaches [3, 10, 30] that process vision and natural language inputs, specifically screenshots in mobile environments. We refer to these as Mobile GUI-Control Agents, or GUI Agents for short. These agents are designed to manipulate the user interface elements on the screen directly, with the abilities to interpret natural language commands and analyze screenshot layouts and element functionalities, which theoretically enable agents to execute any task on any app.

*indicates the equal contribution. And ✉ denotes the corresponding author.

Preprint. Under review.

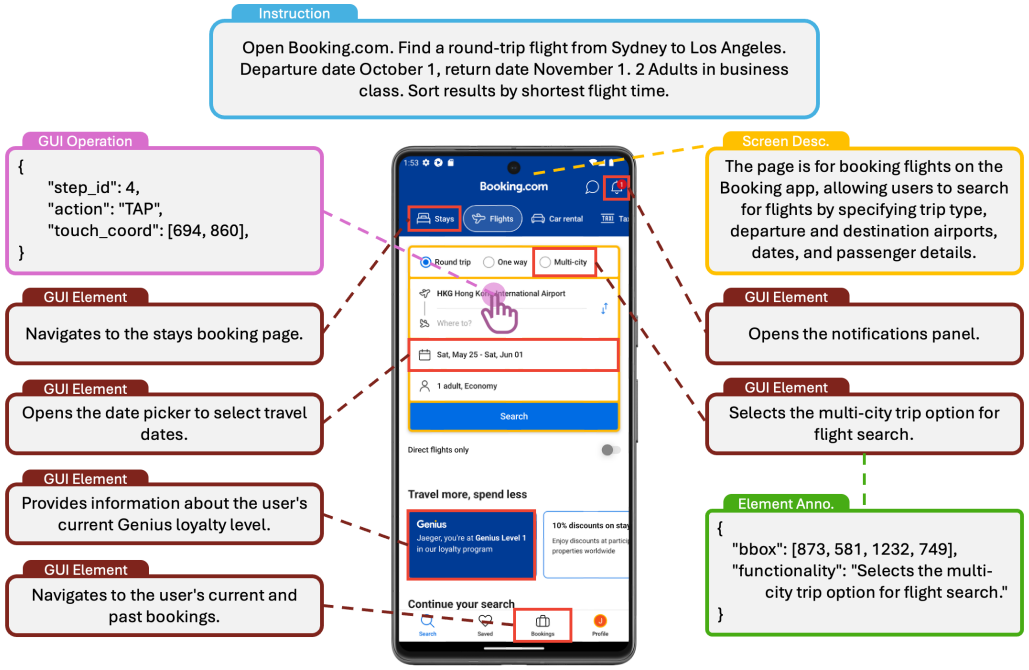


Figure 1: An example of a screenshot-instruction (Blue tab) pair illustrating the multi-level annotation of AMEX. Red boxes + Brown tabs: selected GUI interactive elements and their corresponding functionalities. Green tab: the detailed annotation of the element. Yellow tab: the description of the entire screenshot. Purple hand icon + tab: the current action and the annotation.

Table 1: Comparison of AMEX to other datasets. **Scale**: the number of unique instructions on general third-party apps, average steps per instruction and screenshots. **Diversity**: screenshot description, element labels, element functionality and the action details for stepwise operation.

Dataset	# Unique General Inst.	# Avg Steps	# Screen-shots	Screen Desc.	Screen Element	Element Func.	Action Detail
PixelHelp	187	4.2	~800	✗	✗	✗	✓
UGIF	480	6.3	~3.3K	✗	✗	✗	✓
MoTIF	480	4.5	~21K	✗	✗	✗	✓
AITW	1539	6.5	~510K	✗	*	✗	✓
AITZ	2504	7.5	~18K	✓	*	■	✓
AMEX	2946	12.8	~104K	✓	✓	✓	✓

* indicates elements are mis-annotated and the bounding boxes are not well-aligned.
■ indicates containing only action results of the element interacted at each step.

However, the latest GUI agents, such as CogAgent [10] and AppAgent [28], exhibit unsatisfactory performance when handling real-world tasks. The primary issues with these agents are their lack of awareness of page layouts and their limited understanding of the functionalities of various user interface elements. These shortcomings are largely due to the absence of a comprehensive, detailed, large-scale dataset. Recent research has introduced several related datasets for the Android operating system, featuring different types of annotations. For instance, AITW [19] is a large-scale dataset annotated with instructional operations and screenshot icon detection. However, the data quality, especially in element-level annotation which is constructed by a pretrained IconNet, is subpar due to the unregulated annotation style. Besides, only 2.8% of instructions are for general third-party apps, not representative of general usage. Additionally, AITW suffers from heavy data redundancy as 94.9% instructions, including both multi-step and single-step, are in the category of "Web-shopping". To address these issues, AITZ [32] selects a smaller subset of AITW and re-annotates it with screenshot descriptions and Chain-of-Action-Thought (CoAT) annotations. Though the CoAT paradigm is of some help, the dataset scale shrinks substantially to total 18K screen-action annotation pairs. Other datasets [2, 14, 23] also have various limitations, including the number and practicability of instructions, diversity, accuracy and reliability of annotations, coverage of apps and Android versions.

Furthermore, these datasets primarily offer supplemental tree-based representations, such as the View Hierarchy. Notably, the View Hierarchy is not universally available across all apps, nor is it readily accessible to average mobile users. This limitation restricts the practical usability and generalizability of the data, as it does not reflect the typical interaction scenarios encountered by most users.

When human users receive an instruction, they perform several steps beyond merely interpreting the instruction. A human user would comprehend the overall page layout and analyze the current page to identify interactive elements, determining which elements can be tapped and where to scroll. The functionality of those elements are then assessed, particularly the clickable ones. For example, a bell icon likely indicates navigation to a notification page, while a magnifier icon might signify a search page or activate a search bar. Equipped with this information and contextual knowledge, the user can effectively break down the instruction into specific actions for multiple screens.

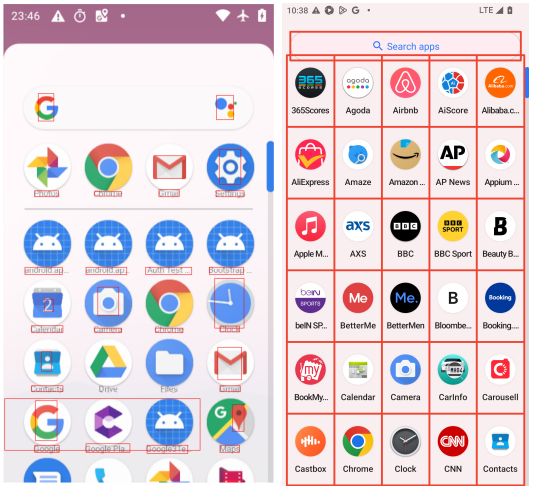
To align an GUI agent with humans, we design a comprehensive multi-level annotation dataset for the Android OS, named the **A**ndroid **M**ulti-annotation **EX**po (AMEX). AMEX consists of three levels of annotations (see Figure 1): (i) GUI interactive element grounding, (ii) GUI screen and element functionality descriptions, and (iii) instructions with GUI-action chains. In total, AMEX consists of annotations on over 104K high-resolution screenshots, 711K element-wise functionalities, and around 3,000 unique instructions with stepwise GUI-action chains, with an average of 13 steps per instruction (see Table 1 and Table 2). The bounding boxes of GUI interactive elements are filtered and verified by human annotators. Screen descriptions and element functionalities, generated by GPT with built-in app descriptions, are also manually checked. Given random apps and instructions, annotators, all trained with specific and precise guidelines for each action, conduct the operations in a natural manner to achieve the objective. The data is collected from 110 third-party apps with Android 13 on Pixel 7 Pro and Samsung S10 emulators, ensuring up-to-date and prevalent screen resolutions and aspect ratios. The key features of AMEX can be summarized as providing knowledge of the modern mobile GUI environment from multiple levels, and complex human logics and operations on third-party apps.

Our contributions can be summarized as follows: (a) We collect and release AMEX for training and evaluating generalist mobile GUI agents, which has multi-level annotations, providing reliable understandings and complex instructions of the smartphone UI environment; (b) We release the SOTA agent SPHINX Agent, which can serve as the baseline model for future researches on GUI agents.

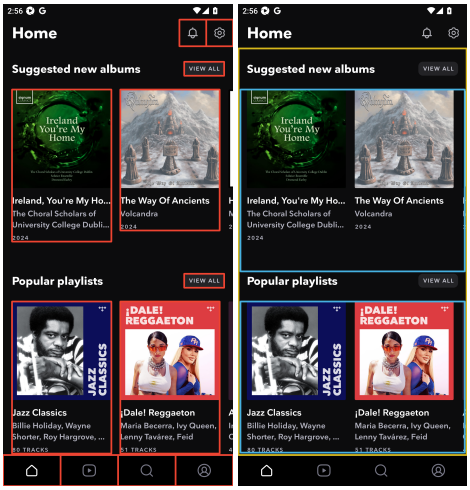
2 Related Work

GUI-Control Datasets. Table 1 compares several popular GUI-control datasets on Android. While some works [5, 15, 20] focus on the web platform, on Android OS, many works [19, 29, 33] have focused on identifying various types of GUI elements, often assigning numerous classes to different elements. Other studies [2, 14, 23] primarily emphasize action-observation pairs during instructional operations, but their annotations are limited and often require supplemental View Hierarchy (VH) data for each screenshot. Furthermore, these instructions are typically too simplistic for real-world tasks. AITW [19] provides both screen GUI element annotations and instructions, but it includes only a small portion of instructions for third-party apps, with most operations conducted on Chrome and other system-built apps. Additionally, each instruction is repeated multiple times, resulting in significant data redundancy. It also relies on the pre-trained IconNet for automatic annotations, which unfortunately leads to numerous mis-annotated types and misaligned bounding boxes. In response, AITZ [32] filters AITW thoroughly, selecting 2.5K unique instructions and episodes, and introduces the Chain-of-Action-Thought framework to annotate action results and page descriptions better. Despite this refinement, AITZ contains only 18K screen-action pairs.

GUI-Control Agents. Recent advancements have leveraged the extensive world knowledge and robust embodied capabilities of Large Language Models (LLMs) [1, 8, 9, 11, 12, 22] for complex task planning and reasoning within GUIs. A notable approach involves employing business-level generalist models like GPT-4v directly as GUI-control agents. Works [28, 34] have utilized these models, employing extensive prompt engineering to guide the LLM in executing complex tasks. However, the effectiveness of these methods is inherently limited by the capabilities of the available generalist MLLMs [4, 13, 16, 24] and requires meticulous prompt design to achieve optimal results. Alternatively, another research line focuses on fine-tuning smaller LLMs on GUI-specific datasets to imbue them with domain-specific knowledge, thereby enhancing their operational efficiency. For example, CogAgent [10] enhances performance in GUI-related tasks by integrating a high-resolution cross-module that fuses image features from various levels. Similarly, MobileAgent [6]



(a) Element annotation on AITW (left) and AMEX (right).



(b) Demo of GUI interactive elements.

Figure 2: Demonstrations of element annotations of AITW and AMEX. (a) Element bounding boxes in AITW (left) and AMEX (right). Boxes in AMEX are well aligned but boxes in AITW might be misaligned and mis-annotated. (b) GUI interactive elements in AMEX. Red boxes: clickable elements. Blue boxes: horizontally scrollable elements. Yellow box: vertically scrollable element.

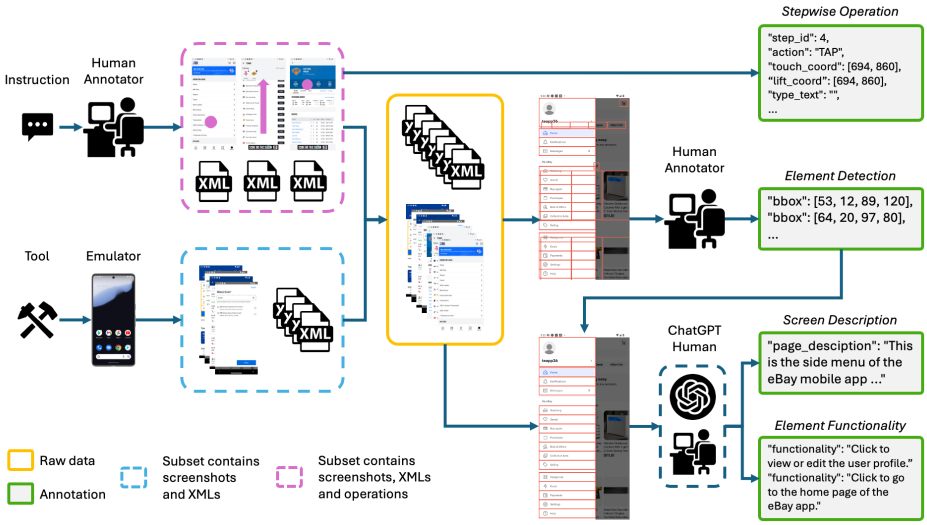


Figure 3: Overview of the data collection pipeline. The raw data is from two subsets collected by human annotators and an autonomous tool respectively, while annotators record the GUI-action chains simultaneously. Then raw data is sent to annotators to filter GUI element bounding boxes, then they with raw screenshots are sent as input to GPT to extract the GUI screen and element descriptions, which are then manually checked by humans.

improves sample handling and structuring of input data to make it more consistent and LLM-friendly. Auto-UI [30] utilizes a multimodal encoder-decoder language model based on T5 and BLIP and CoCo-Agent [17] takes element OCR layouts as additional input to enhance the performance. Expanding beyond traditional mobile phone GUI domains, recent studies [26, 31] are exploring the potential of LLM-Agents in operating system-level task execution.

3 Android Multi-annotation EXpo (AMEX)

When receiving an instruction, a human user first analyzes the overall screen layout to form a basic understanding of the current Android environment. The user then identifies interactive elements and areas, and assesses the functionalities of those elements. Finally, the user breaks down the instruction into simple, step-by-step actions on each screen. Based on this human cognitive process, we design three levels of annotations in our proposed AMEX training set: (i) GUI interactive element grounding

(see Section 3.2), (ii) GUI screen and element descriptions (see Section 3.3), and (iii) instructions with GUI-action chains (see Section 3.4).

The raw data is collected using Android emulators, specifically Android Virtual Device and Genymotion[2]. We developed tools to autonomously perform emulator operations as well as record human actions. These tools leverage Appium[3], an open-source, cross-platform test automation tool for Android, iOS, and web apps. Appium collects device screenshots during operations, each of which also corresponds to an Extensible Markup Language (XML) file recording the basic screen layout with element attributes, such as bounding boxes and in-app descriptions.

3.1 Data Collection Pipeline

An overview of the data collection pipeline is illustrated in Figure 3. The raw data is collected through two methods: human instruction-following GUI manipulations and autonomous GUI controls. Human GUI manipulations involve recording stepwise operations for each instruction, and simultaneously storing screenshots and each screen's XML data before each stepwise operation. In parallel, an autonomous script controls emulators to collect additional screenshots and their XMLs. These two subsets comprise the entire raw dataset. Then for each screenshot, initial bounding boxes of interactive elements and their in-app descriptions (if available) are parsed from the corresponding XML. Human annotators then review each screenshot to filter out all the misaligned boxes, which serve as the interactive element grounding annotations. With the in-app descriptions, GPT generates the functionalities of the selected elements and provides descriptions for the whole screenshot. Human annotators then further check the descriptions of functionalities. More detailed methods are discussed in the following sections and Appendix A.1.3.

3.2 Level I: GUI Interactive Element Grounding

Existing datasets [19, 32] typically classify elements on the screen, such as icons, texts, and images, based on their types. Instead of adhering to the traditional classification paradigm, we define interactive elements more broadly as any elements that users can interact with, regardless of their specific types (see Figure 2a). Specifically, interactive elements in our dataset are only categorized into two subsets: (i) clickable elements and (ii) scrollable elements.

Clickable Elements are the most common components in a screen. They typically include clickable icons, images, texts, and compounds that combine several categories. Figure 2b illustrates various clickable elements in red boxes. We also include certain "typeable" elements, such as search bars, because most typeable elements require a prior click action to enable typing.

Scrollable Elements typically occupy larger areas on the screen. In most cases, a scrollable element area supports a pair of actions, such as "scroll down" and "scroll up" or "scroll left" and "scroll right." Generally, the vertical scrollable area forms the main frame in the middle of the screen, while the horizontal scrollable area serves as a "carousel" for listing different categories or activities. Figure 2b illustrates these two types of scrollable elements.

As mentioned in Section 3.1, the collection of screenshots and their XML data utilizes two methods. Here, we provide a more detailed pipeline for executing an autonomous script to perform operations on an Android emulator. The script is designed to traverse an app in an unconstrained manner and collect data. It performs actions (see Appendix A.1.1) at regular time intervals to allow each page to fully load. After waiting, the script captures the current screenshot along with the corresponding XML file. To ensure efficiency and prevent redundancy, we limit the maximum number of operation steps per script execution. Additionally, a break mechanism is implemented to exit potential dead loops, ensuring the script progresses effectively. Upon the raw data (yellow part in Figure 3), we extract bounding boxes for interactive elements from the XML associated with each screenshot. While scrollable elements are typically parsed with high quality, clickable elements are more problematic. This issue is from the fact that app developers often design elements with overlapping or layered coverage, leading to the generation of bounding boxes even for elements that are visually hidden from the user's view. To address this, annotators manually inspect each screenshot, identifying and selecting only the interactive elements that are actually visible within the interface (see Appendix A.1.2).

3.3 Level II: GUI Screen and Element Functionality Descriptions

Previous works on GUI elements often rely on predefined class names to convey the underlying meaning of each element. However, this classification-based method has significant limitations.

[2]https://www.genymotion.com/

[3]https://appium.io/docs/en/latest/

Table 2: AMEX statistics

# Screenshots	# Interactive Elements	# Functionalities	# Instructions	# Avg. Steps	# Apps
103,684	1,658,439	711,823	2,946	12.8	110

For instance, a small triangle symbol typically indicates "play" in video or music interfaces, but it might also appear as part of a content creator's profile image. In such cases, the model erroneously interprets the symbol and repeatedly attempts to tap the icon to play the video. This issue arises because the class-based annotation approach focuses on class labels rather than truly understanding the functionality of each element in the surrounding context.

To ensure the dataset is truly instructive rather than merely icon detection, we focus on describing screen status and element functionalities. Consider the above mentioned example: instead of providing a bounding box for the small triangle and labeling it as "ICON_PLAY," we present the actual functionality of the element within its context (e.g., "Opens the creator's personal page"). This strategy offers clear, instructive, and detailed description, enhancing the dataset's applicability. We adopt GPT-4o for the element-wise annotation to describe screens and interpret element functionalities. Given the compact and dense nature of GUI elements, we apply Set of Markers (SoM) techniques [27] to bolster GPT-4o's capability for visual localization. The coordinates of these elements are obtained and cleaned in the preceding phases (Section 3.2). Additionally, recognizing that some elements possess abstract functionalities that are challenging to discern, we supplement the prompts for GPT-4o with in-app descriptions (if available) extracted from XMLs to enhance GPT-4o's comprehension of the screen. After the data generation, human annotators manually check 20% of the descriptions of the element functionality. Further elaboration on this methodology is provided in Appendix A.1.3.

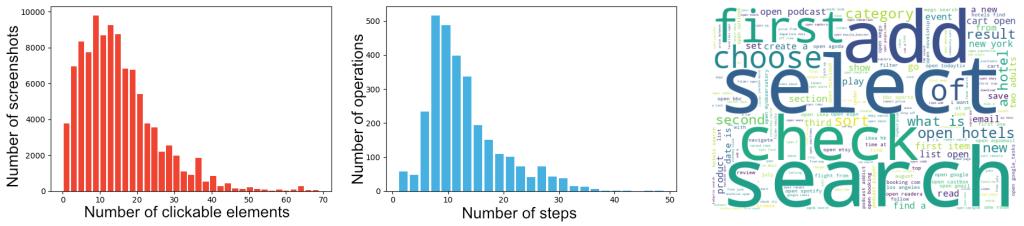
3.4 Level III: Instructions with GUI-Action Chains

Our focus for GUI agents is predominantly on third-party applications rather than system-built apps. We select an average of three apps per category from the Google Play Store. Due to the significant increase in app functionalities over the past years, we estimate that a sample size of approximately 50 apps can adequately represent most common use scenarios.

The instruction generation process comprises three phases. Initially, human annotators create 5-10 complex instructions for each target app, considering its specific purposes and capabilities. These initial instructions, combined with relevant app metadata collected online, serve as input for ChatGPT. It then generates a larger set of 80-100 instructions that exhibit similar structure and intent to the human-provided examples. For example, if a human annotator provides the instruction *"Open Google Maps. Find the shortest route from the current location to Empire State Building by car drive,"* ChatGPT might produce *"Open Google Maps. Find the fastest route from the current location to Rockefeller Center by public transit."* However, due to ChatGPT's limitations on understanding real-world constraints, human filtering is adopted to remove any unreasonable or impractical instructions. Notably, instructions within AMEX are more complex than those in the AITW dataset, with an average step count of 12.8, nearly double that of AITW.

We define our action space for stepwise GUI operations similarly to AITW: {`TAP`, `SCROLL`, `TYPE`, `PRESS_BACK`, `PRESS_HOME`, `PRESS_ENTER`, `TASK_COMPLETE`, `TASK_IMPOSSIBLE`}. `TAP` actions are characterized by identical touch and lift coordinates, while `SCROLL` actions involve distinct touch and lift coordinates. `TYPE` actions are annotated with a `type_text` attribute specifying the input text. The three `PRESS` actions correspond to system-level button presses (back, home, enter). `TASK_COMPLETE` and `TASK_IMPOSSIBLE` serve as terminal flags for instructions. Additionally, for instructions that involve information query (e.g., "What is the lowest price of the men's belt?"), we associate the `TASK_COMPLETE` action with a region of interest, which is defined as the bounding box of the area on the current screenshot, where the answer is expected to appear (see examples in Appendix A.3). This comprehensive action space allows us to fully simulate a wide range of typical use cases.

The collection tool is designed to simultaneously record the screen and action pair together with the XML file. The details of GUI-action chain collection can be found in Appendix A.1.4.



(a) (b) (c)

Figure 4: (a) The number of screenshots by the number of GUI clickable elements. (b) The number of instructions by the number of GUI operations steps. (c) The word cloud of the instructions.

Table 3: Task template examples.

Tasks	Examples of Task Templates	Source
Screen Description	User: Provide a one-sentence caption for the provided GUI image. Agent: This page is a film section of the IMDb app, displaying a list of movies or shows with sorting and filtering options.	AMEX
Element Grounding	User: Identify all clickable elements and provide their 2D bounding boxes. Agent: List of Elements with **BBox**es.	AMEX
Functionality Description	User: What is the function of the icon at the coordinates: **BBox**. Agent: Click to expand or collapse the sort options.	AMEX
GUI-Action Prediction	User: Please execute the task described by the following instruction: Delete the first email in the inbox. Agent: CLICK[[0.59, 0.92]].	AMEX & AITW

3.5 AMEX Statistics and Test Set

Data statistics. Table 2 lists the statistics of the AMEX dataset. AMEX has approximately 104K screenshots, 1.6M GUI interactive elements for grounding, 712K GUI element functionality descriptions, and 3K instructions. Figure 4 illustrates the distribution of GUI interactive elements per screen and GUI operation steps along with the word cloud of the instructions.

Test set. Instructions and their annotated screen-action chains from 9 out of 55 apps are chosen as the test set, including those on `Booking`, `Gmail`, `NBC News` (News), `SHEIN`, `Citymapper` (CM), `Microsoft To Do` (ToDo), `Signal`, `Yelp` and `YouTube Music` (Music), covering different scenarios. The number of test instructions is 362, around 10% of the total instructions.

The evaluation of AMEX test set is similar to the existing dataset AITW [19]. We perform the action matching to get the partial action match score. The partial action matching score is the number of correct actions divided by the episode length [14]. To ensure the correctness of the test set, we carefully review the trajectory annotations in the unseen test set, where we drop the former step of `PRESS_BACK`, which indicates the wrong operation of the annotator. However, we keep the `PRESS_BACK` step to evaluate the agent's self-correction ability.

4 SPHINX-GUI Agent

Integrating data from AMEX with the existing AITW dataset, we train the SPHINX-GUI Agent tailored for GUI-related tasks as the baseline for AMEX dataset. This Agent is initialized from SPHINX [7] and is specifically trained to generate actions to manipulate the GUI effectively, which are different from the more flexible output formats typical of standard MLLM tasks such as QA. To incorporate GUI-specific intelligence into SPHINX, we have implemented several key modifications:

Instruction Dataset Construction. We transform the AMEX data and the publicly available AITW dataset into a VQA format conducive to instruction understanding, thereby adapting the MLLM as a GUI agent. To embed GUI-specific knowledge within the MLLM, we developed four distinct VQA tasks, detailed in the instruction templates provided in Table 3.

Structured Representation. To effectively guide the agent in task completion, we incorporate the goal into the input prompt. To prevent redundant actions, we also include a history of actions formatted as $X_{\text{history}} = [a_{t-k}, \ldots, a_t]$, where each a denotes an action type, accompanied by a touch point for actions such as `SCROLL` or `TAP`. The parameter k denotes the length of the history. Action trajectories from both AITW and AMEX datasets are converted into this uniform representation.

Given the extensive nature of GUI screenshots and the small relative size of some target elements within the screen, we adopt the "any resolution" approach from SPHINX [7]. This method initially partitions the input image into sub-images, then encodes separately. The LLM processes all the

Table 4: Experiment results on AITW. $^+$ indicates CoCo-Agent takes screen element OCR layouts as additional inputs.

Agent	# Params	Training Data	General	Install	G-Apps	Single	WebShopping	Overall
Auto-UI	5B	AITW	68.24	76.89	71.37	84.58	70.26	74.27
SeeClick	9.6B	AITW+External	67.6	79.6	**75.9**	84.6	73.1	76.2
CogAgent	18B	AITW+External	65.38	78.86	74.95	**93.49**	71.73	76.88
CoCo-Agent$^+$	7B	AITW	70.96	81.46	76.45	91.41	75.00	79.05
SphAgent	7B	AITW	68.2	80.5	73.3	85.4	74	76.28
SphAgent	7B	AITW + AMEX	**73.1**	**80.5**	73.4	90.8	**75.8**	**78.72**

Table 5: Experiment results on AMEX

Agent	Training Data	Gmail	Booking	Music	SHEIN	News	CM	ToDo	Signal	Yelp	Overall
SeeClick	AITW+External	28.2	29.4	18.1	20.0	30.0	53.1	30.7	37.1	27.4	30.44
SphAgent	AITW	32.1	45.9	46.1	35.1	48.3	61.1	55.9	43.3	42.9	45.63
SphAgent	AMEX	61.7	68.2	77.7	72.0	71.9	64.6	79.6	71.3	69.6	70.71
SphAgent	AITW + AMEX	62.4	68.1	76.3	71.9	68.6	67.3	77.6	66.0	64.1	69.14

resulting visual tokens to ensure comprehensive understanding and interaction capabilities. We follow all other default training receipts in [7] to obtain the final agent.

5 Experiments

5.1 Evaluation Setup and Compared LVLMs

We evaluate different GUI agents including SPHINX Agent developed in Section 4 on AITW and our AMEX dataset respectively. For AITW, we simultaneously train SPHINX Agent on all the subsets and then assess all test sets. We downsample GoogleApps subset to 10% to avoid data imbalance. The training and test split setting follows the one in [19]. For AMEX, we evaluated our SPHINX agents on the test set as described in Section 3.5. Our models are benchmarked against several agents including Auto-UI [30], SeeClick [3], CogAgent [10] and CoCo-Agent [17].

5.2 Experiment Results

The experiment outcomes for AITW and AMEX are presented in Tables 4 and 5 respectively. As indicated in Table 4, SPHINX Agent (SphAgent) trained on AITW exhibit competitive performance on the AITW dataset relative to the baselines. Notably, the introduction of the AMEX dataset enhanced the overall performance by approximately 2.5%, with significant gains observed in the "General" (5%), "Single" (5%) and "WebShopping" (2%) tasks, which indicates the strong complement from AMEX dataset. The GUI element functionality descriptions are served as the "Single" step instructions which strongly promote the performance in the "Single" tasks, while 3K instructions from general third-party apps also boost the performance in the "General" tasks. "WebShopping" is the category with the most complicated tasks in AITW, which also benefits from the complex instructions provided by AMEX.

5.3 Cross-Domain Experiments

We also examine the generalization capabilities of our agent across different domains. Specifically, the agent trained exclusively on the AITW dataset is tested on the AMEX dataset. The results, as detailed in the first row of Table 5, reveal significant findings. Agent trained only on AITW, shows satisfactory performance on the AITW dataset itself. However, when evaluated on the AMEX dataset, there is a notable decrease in performance. This performance degradation can likely be attributed to several factors: *1) Higher complexity in AMEX*. The AMEX dataset generally involves longer operational sequences compared to those in AITW, presenting a higher complexity level that the agent might not have been exposed to during training. *2) Domain gap*. There are substantial differences in the visual textures and language instructions between the two datasets. These variations could hinder the agent's ability to effectively transfer its learned knowledge to a new, unfamiliar domain. Additionally, we observed a significant performance degradation for the SeeClick agent, originally trained on AITW. Beyond the previously mentioned reasons, this decline may also stem from the fact that the original SeeClick model was designed to process only 224×224 image inputs. In contrast, the AMEX dataset comprises high-resolution images and requires interactions with tiny icons. The high-definition images in AMEX introduce new challenges for GUI agents.

Table 6: Human evaluation results on 5%-subsampled AITW test sub-set. "MMR" stands for "Mis-Match Random" subset.

Subset	Action Sources	General	Install	G-Apps	Single	WebShopping	Overall
MMR	SphAgent	90.91	83.65	87.50	87.68	82.39	86.43
	AITW Anno	88.18	75.00	82.69	84.78	75.57	81.25
Random	SphAgent	92.31	93.75	94.33	94.83	94.55	93.95
	AITW Anno	94.23	90.34	93.62	93.97	95.00	93.43

Table 7: Ablation study of each level of annotations.

Tasks			Gmail	Booking	Music	SHEIN	NBC	CM	ToDo	Signal	Yelp	Avg
L1	L2	L3										
		✓	45.9	64.5	74.4	71.8	70.3	67.4	79.3	64.9	66.3	67.2
✓		✓	58.6	68.3	74.4	68.8	68.1	61.1	79.3	73.2	67.4	68.8
	✓	✓	60.3	66.1	77.2	70.2	72.0	64.0	78.1	72.3	65.7	69.5
✓	✓	✓	61.7	68.2	77.7	72.0	71.9	64.6	79.6	71.3	69.6	70.7

5.4 AITW Human Evaluation Results

As specified in AITZ [32], several types of error cases are identified in the AITW test set (see Figure5). To assess the unreliability of the original AITW test set and its evaluation methods, we propose AITW-HE (Human Evaluation), a refined subset of the AITW test set. Human annotators evaluate two subsets derived from the original test set. One subset is randomly chosen from episodes where SphAgent receives low scores (`Mis-Match Random`), i.e., about 66 score compared to the overall 78.72 shown in Table 4, indicating a high mismatch between inference results and AITW annotations. The other subset is randomly selected from the remaining episodes (`Random`). Annotators first filter out repeated and redundant screenshots (see Figure 5a). For each remaining screenshot, they then record whether the SphAgent-inferred action and the original AITW annotation are correct. Figure 6 illustrates cases where annotators mark both the inferred action and the original annotation as correct, even though they interact with different elements.

Table 6 presents the accuracy scores from human evaluation. The table indicates that the low SphAgent scores in the MMR subset are primarily due to unsatisfactory annotations (AITW Anno), which are used as ground truth during the evaluation. Both SphAgent and AITW Anno in the "MMR" subset have lower scores than those in the "Random" subset, highlighting that the tasks in the MMR subset are relatively more challenging. Furthermore, SphAgent achieves better human evaluation results than the original annotations, demonstrating its effectiveness and close alignment with human judgment. Comparing the results from Table 4 and Table 6, the notable differences in overall scores (i.e., 78.72%, 86.43%, and 93.95%) underscore the unreliability and misleading nature of the AITW test set and its evaluation methods.

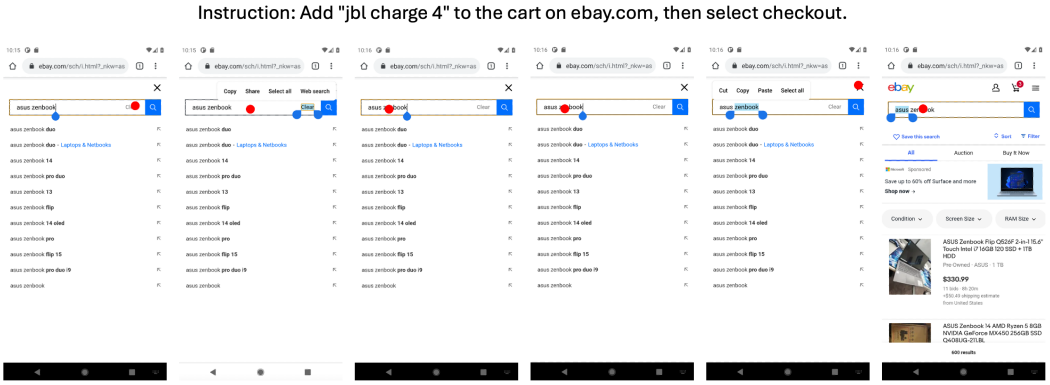
5.5 Ablation Study on Multi-level Annotations

To validate the effectiveness of the multi-level annotations proposed in the AMEX dataset, we conducted an ablation study. As shown in Table 7, the agent achieved a performance gain of approximately 1.0 when aided by L1 annotations. When trained with L1 and L3 annotations, the agent exhibited a performance gain of about 1.6 compared to the baseline. The gain from L2 annotations is approximately 2.3. Furthermore, training the agent with the complete set of annotations resulted in a performance gain of approximately 3.5. The ablation study results demonstrate that each level of annotations in AMEX enhances the final agent's performance.

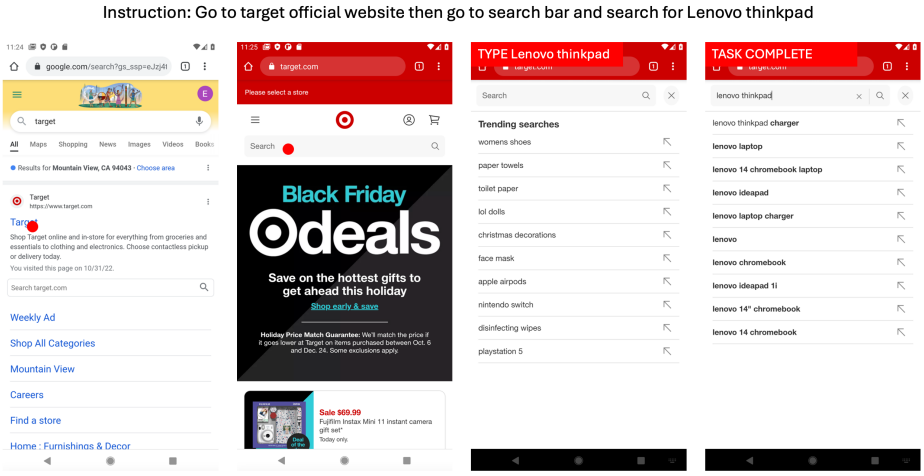
6 Discussions

6.1 Limitations and Future Work

Multi-lingual Most existing datasets are limited to English, with UGIF [23] being a notable exception, as it includes instructions and screenshots in eight languages. The AMEX dataset contains a small number of screenshots in Chinese and Spanish, primarily due to strict registration and login requirements for Chinese apps and a lack of expertise in other languages. Future work should incorporate multi-lingual screenshots, functionalities, and instructions to create a more robust and comprehensive multi-lingual environment for GUI agents.



(a) Repeating useless screenshots and actions.



(b) Wrong task complete.

Figure 5: Error cases in AITW test set. Red dots indicate the actions from the AITW annotations.

Evaluation The current evaluation method employed in both AMEX and AITW is straightforward: the model predicts the current action based on given instructions, previous operations, and screenshots. However, this evaluation approach has significant limitations in real-world scenarios. It fails to account for loading pages, waiting times for page loading, etc. An ideal evaluation would be an "Arena" (a mobile version of the environment in WEBARENA [35]) — an independent local virtual device equipped with Android and static background apps. This setup would ensure consistent results for the same queries, avoiding variations due to differing dates, prices, and other dynamic data.

6.2 Ethical Considerations

- The accounts registered and logged in are all for testing purposes, not including any personal information. The dataset doesn't contain any private or personal information.
- The dataset, if misused, could be exploited for undesirable purposes, such as anti-fraud mechanisms and anti-script verification codes (see Appendix A.4), potentially leading to harm.
- Annotators received remuneration in line with local wage standards for their annotation works.

7 Conclusion

As AI agents become more prevalent, mobile GUI agents are emerging as a research hotspot. To address the lack of fundamental understanding of GUI elements in existing datasets, we present the Android Multi-annotation EXpo (AMEX) dataset, which includes three levels of annotation to

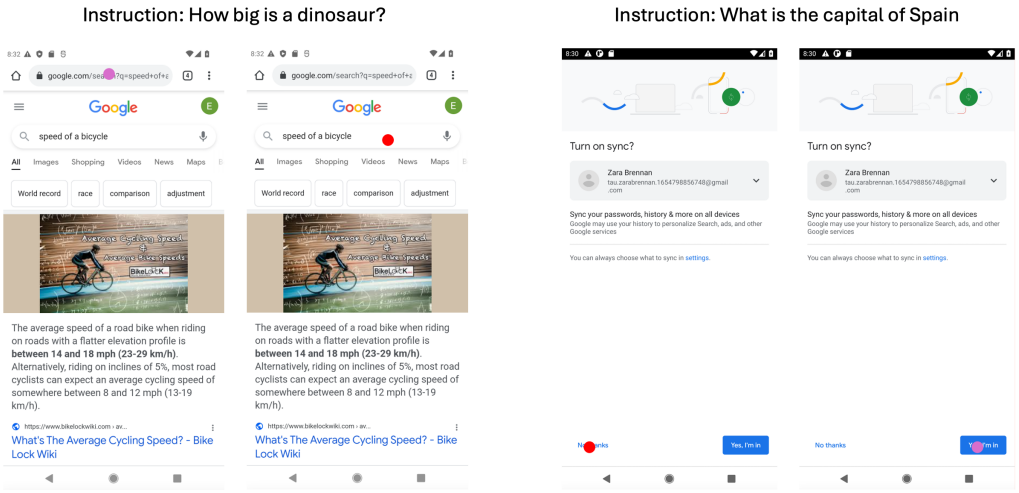


Figure 6: Purple dots indicate the SphAgent inferred actions and red dots indicate the AITW annotations. Human annotators mark them both correct even though they are clicking different elements.

provide a more instructive and detailed understanding of UI screens and elements. Additionally, we introduce a state-of-the-art SPHINX Agent, which can serve as a baseline model for future research.

References

[1] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[2] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments. *arXiv preprint arXiv:2104.08560*, 2021.

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

[4] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[6] Tinghe Ding. Mobileagent: enhancing mobile control via human-machine interaction and sop integration. *arXiv preprint arXiv:2401.04124*, 2024.

[7] Peng Gao, Renrui Zhang, Chris Liu, Longtian Qiu, Siyuan Huang, Weifeng Lin, Shitian Zhao, Shijie Geng, Ziyi Lin, Peng Jin, et al. Sphinx-x: Scaling data and parameters for a family of multi-modal large language models. *arXiv preprint arXiv:2402.05935*, 2024.

[8] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[9] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding html with large language models. *arXiv preprint arXiv:2210.03945*, 2022.

[10] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*, 2023.

[11] Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Barun Patra, et al. Language is not all you need: Aligning perception with language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[13] Chunyuan Li, Zhe Gan, Zhengyuan Yang, Jianwei Yang, Linjie Li, Lijuan Wang, Jianfeng Gao, et al. Multimodal foundation models: From specialists to general-purpose assistants. *Foundations and Trends® in Computer Graphics and Vision*, 16(1-2):1–214, 2024.

[14] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.

[15] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

[16] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[17] Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. Comprehensive cognitive llm agent for smartphone gui automation. *arXiv preprint arXiv:2402.11941*, 2024.

[18] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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

[20] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

[21] InternLM Team. Internlm: A multilingual language model with progressively enhanced capabilities, 2023.

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[23] Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. Ugif: Ui grounded instruction following. *arXiv preprint arXiv:2211.07615*, 2022.

[24] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, et al. Cogvlm: Visual expert for pretrained language models. *arXiv preprint arXiv:2311.03079*, 2023.

[25] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

[26] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

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

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

[29] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *arXiv preprint arXiv:2404.05719*, 2024.

[30] Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023.

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

[32] Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents, 2024.

[33] Xiaoyi Zhang, Lilian De Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, et al. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systautoems*, pages 1–15, 2021.

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

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

A Appendix / supplemental material

A.1 Pipeline Details

A.1.1 Autonomous script details

The autonomous script controls the emulator using three actions: `TAP`, `SCROLL`, `TYPE`.

- For the `TAP` action, we employ two algorithms. The first randomly selects a clickable element on the current screen, while the second computes the index of a clickable element using a formula to ensure the elements chosen are likely unique. We apply one of these algorithms randomly for different executions.
- For the `SCROLL` action, we classify whether an area is vertically or horizontally scrollable by setting a width-height ratio threshold, $\mathcal{R}$. If the element's ratio exceeds $\mathcal{R}$, it is considered horizontally scrollable; otherwise, it is vertically scrollable. We then randomly select a scrollable element on the current screen and perform a scroll action based on its type.
- For the `TYPE` action, we pre-define a list of phrases relevant to the category of apps being tested. For example, ["Women's dress", "Nike sneakers", ...] for clothing shopping apps. These phrases are primarily used in search scenarios.

A.1.2 GUI clickable element filtering

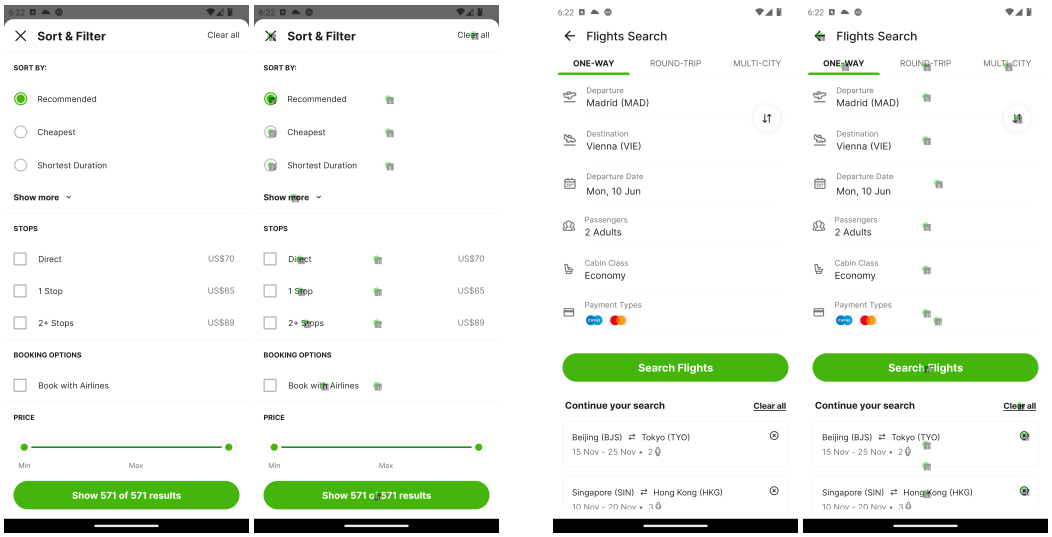
(a) (b)

Figure 7: Demonstration of human annotator filtering. (a) before filtering. (b) after filtering.

The raw XML information sometimes contains the elements that are covered by other elements or layers. Figure 7 illustrates the same image before and after filtering. Blue boxes in Figure 7a are those elements under the current active layer and they still show up from XML parsing. Thus we need human annotators to filter out these blocked elements to get Figure 7b.

A.1.3 GUI screen and element functionality description collection details

We apply the Set-of-Mark (SoM) [27] technique when using GPT-4o as the description generator. The SoM technique is a visual prompting method designed to enhance the visual grounding capabilities of large multimodal models (LMMs), such as GPT-4V, by overlaying visual marks on image regions. This involves partitioning an image into semantically meaningful regions and adding distinct marks (e.g., alphanumeric characters, masks, or boxes) to these regions. It demonstrates significant improvements in precision and accuracy over traditional prompting methods and other state-of-the-art models. Figure 8 shows the screenshot with SoM technique.



(a) (b)

Figure 8: Demonstrations of SoM technique on screenshots.

The prompt is in the following format.

```
Based on the screenshot of an Android mobile phone from the APP_NAME, please follow the instructions:

1.  Understand the Page Content:
- Analyze the overall content of the page.
- Provide a brief summary of the page content in 1-2 sentences.
2.  Explain Highlighted Areas:
- Each highlighted area is either clickable or scrollable.
- Treat each highlighted area as a unique and separate entity, using identifiers such as <Region 1> etc.
- If the highlighted area is a general icon, provide its type first in the format ICON_Magnifying_Glass.
- If the highlighted area is more complex, provide a brief description in the format Element('a poster of
the movie named <La La Land>').
- Explain the purpose or functionality of each highlighted area.  In other words, what result will happen
or what's the user's intention when the marked <@area is clicked or scrolled?
- Some functionality may require an overall analysis, and try to give the functionality specifically and
related to the current screenshot.
3.  Additional Information for Highlighted Areas:
There are total NUMBER elements to annotate.
MarkerInformation
4.  Output Format:
The output should be in JSON format as follows:
{
"overall_page_content":  "1-2 sentences summarizing the page content",
"Region 1":  "ICON_XXX_XXX <functionality>:  xxxxx",
"Region 2":  "Element('a poster of a movie xxxx')
 <functionality>:  click to see the details and forward the purchase page of the movie xx",
}
```

Listing 1: An example prompt for guiding GPT4o to generate the element functionalities for the given Screenshot.

A.1.4 GUI-Action chain collection details

Human annotators are each assigned a random selection of apps and their associated instructions, which they are asked to complete in a natural manner. In contrast to the AITW dataset, our collection methodology allows annotators to make errors and take incorrect steps, leading to a greater prevalence of PRESS_BACK actions. This is motivated by our observation that agents trained on existing datasets exhibit difficulty navigating back to previous pages due to insufficient experience with the PRESS_BACK action. Additionally, after completing an information query task, annotators are asked to manually mark the region of interest on the screenshot using our annotation tool.

A.1.5 Collection resources details

- GUI interactive element grounding takes approximately 3000 human-hour to filter bounding boxes described in Appendix A.1.2.
- GUI screen and element descriptions use GPT-4o API, which consumes about 600 dollars.
- Instructions with GUI-action chains take approximately 200 human-hour.

A.2 Experiment details

In our implementation, we utilize the internlm-7b variant of the SPHINX-X model, as detailed in [7]. The pre-trained checkpoint for this model was sourced from the official repository mentioned in [21]. For image processing, the input images, each sized 1024×1024, are segmented into sub-images. Visual features from these sub-images are extracted using two distinct visual encoders: DINOv2 [18] and ConvNext [25]. To ensure compatibility in feature dimensions across different modules, linear projection layers are employed to align the channel dimensions. Regarding the model's parameter settings, as outlined in Section 4, we configure the history window size to four. Additionally, we introduce a special token, `<ICON>`, specifically designed to identify interactive elements within the interface, strengthening the model's interpretability and responsiveness to user interactions. The agent is trained on a cluster with 3 nodes, each with eight NVIDIA A100 (80GB) GPUs. The fine-tuning was completed in four epochs.

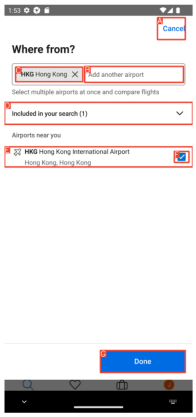
A.3 More AMEX examples

See Figure 9 for more examples of GUI interactive elements grounding and description.

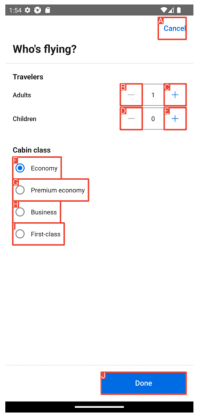
See Figure 10 and Figure 11 for more examples of instruction with GUI-action chains.

A.4 Examples of Ethical Problems

Figure 12 shows examples of anti-script mechanism where the agent can correctly enter the verification codes.



A. Click to cancel the current operation and go back to the previous screen.
B. Click to add another airport to the search criteria.
C. Click to remove the selected airport (HKG Hong Kong) from the search criteria.
D. Click to expand or collapse the list of included airports in the search.
E. Click to select or deselect this airport for the search.
F. Indicates that the HKG Hong Kong International Airport is currently selected.
G. Click to confirm the selected airports and proceed to the next step.

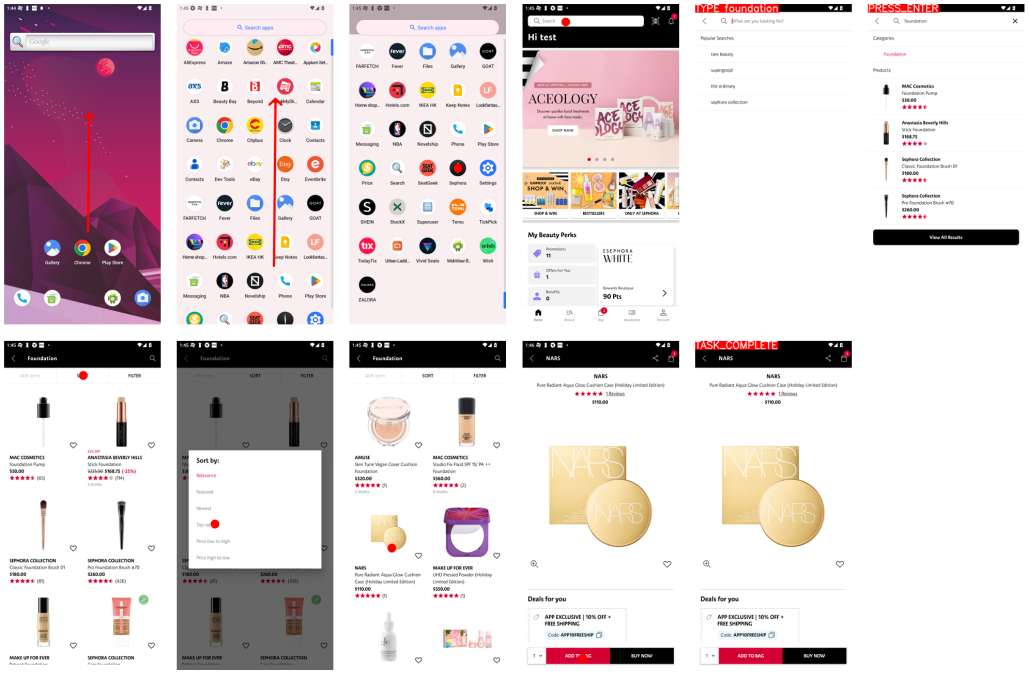


A. Click to cancel the current action and go back to the previous screen.
B. Click to decrease the number of adults traveling.
C. Click to increase the number of adults traveling.
D. Click to decrease the number of children traveling.
E. Click to increase the number of children traveling.
F. Click to select Economy class for the flight.
G. Click to select Premium Economy class for the flight.
H. Click to select Business class for the flight.
I. Click to select First-class for the flight.
J. Click to confirm the selections and proceed to the next step.

A. Go back to the previous page.
B. Open the search bar to search for products on Amazon.
C. Open the camera feature, likely for scanning barcodes or searching by image.
D. Open the voice search feature.
E. Click to view the product details and purchase options for the Spigen phone case.
F. Indicates that the product listing is a sponsored advertisement.
G. Click to view customer reviews and ratings for the product.
H. Click to visit the Samsung store on Amazon.
I. Click to view details about the sustainability feature of the product.
J. Click to view a larger image or more images of the product.
K. Add the product to the wishlist.
L. Click to view the product in 3D model.
M. Click to view the product in 3D showroom.
N. Go to the home page of the Amazon app.
O. Open the categories menu to browse different product categories.
P. View or edit user profile and account settings.
Q. View the shopping cart which contains 2 items.
R. Open the main menu for more options and settings.

Figure 9: More examples

Open Sephora. Search for "foundation". Sort the results by highest rating. Select the third one. Add it to cart.



Open IMDB. Search the movie "Titanic". Who is the top-billed cast member?

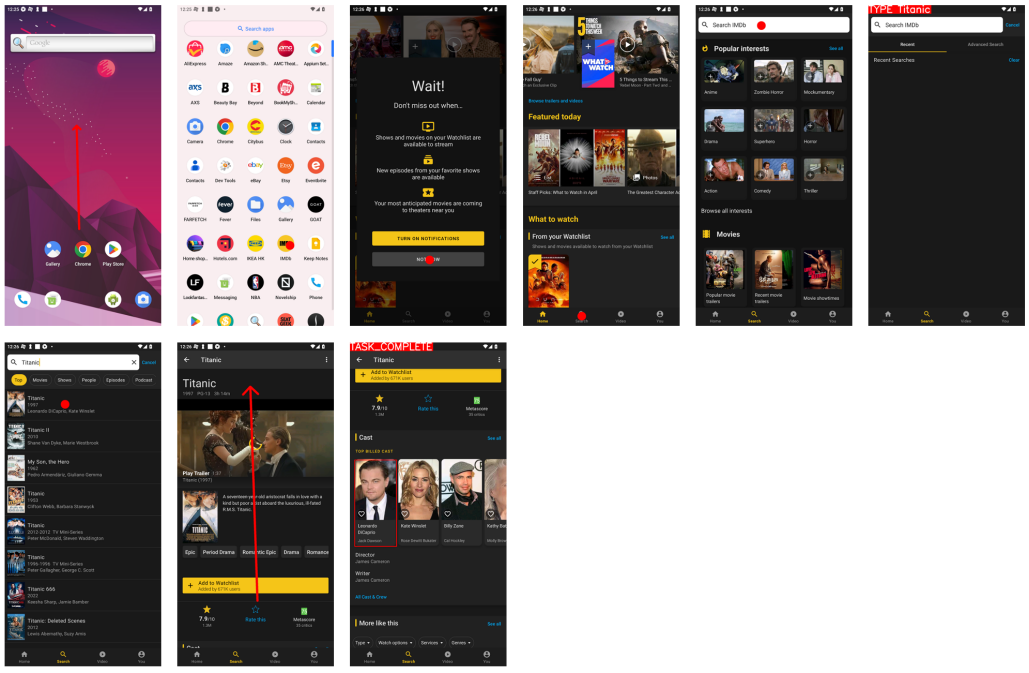
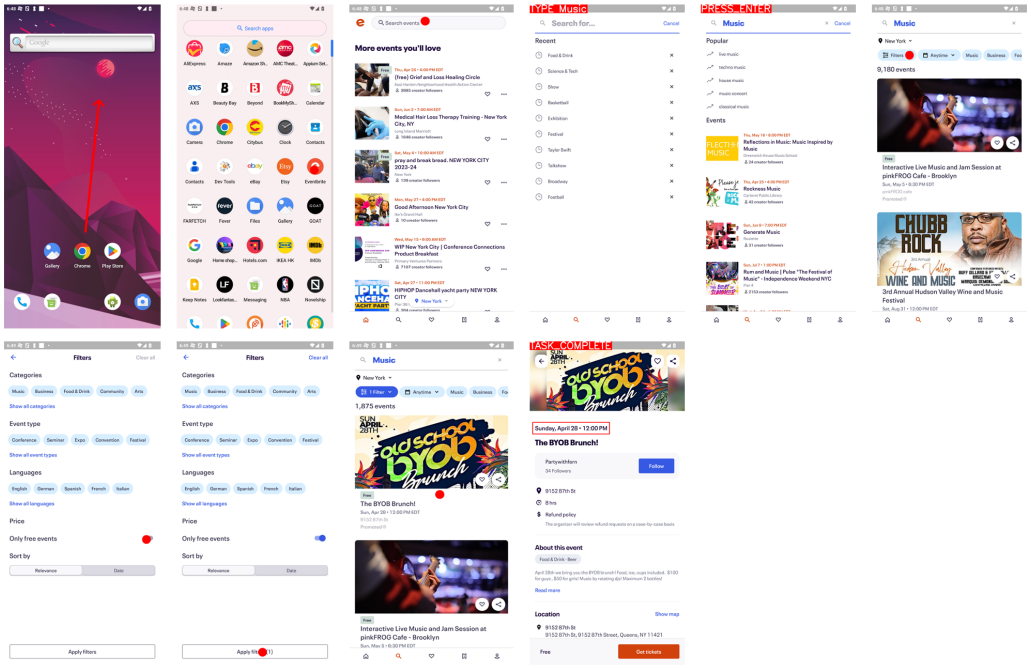


Figure 10: More examples

Open Eventbrite. Search for "Music". Filter only free events. Choose the first event. When is the date and timing?



Open TodayTix. Search "Six" in New York. Specify the date as May 15th. Want 2 tickets. Proceed to check ticket prices range.

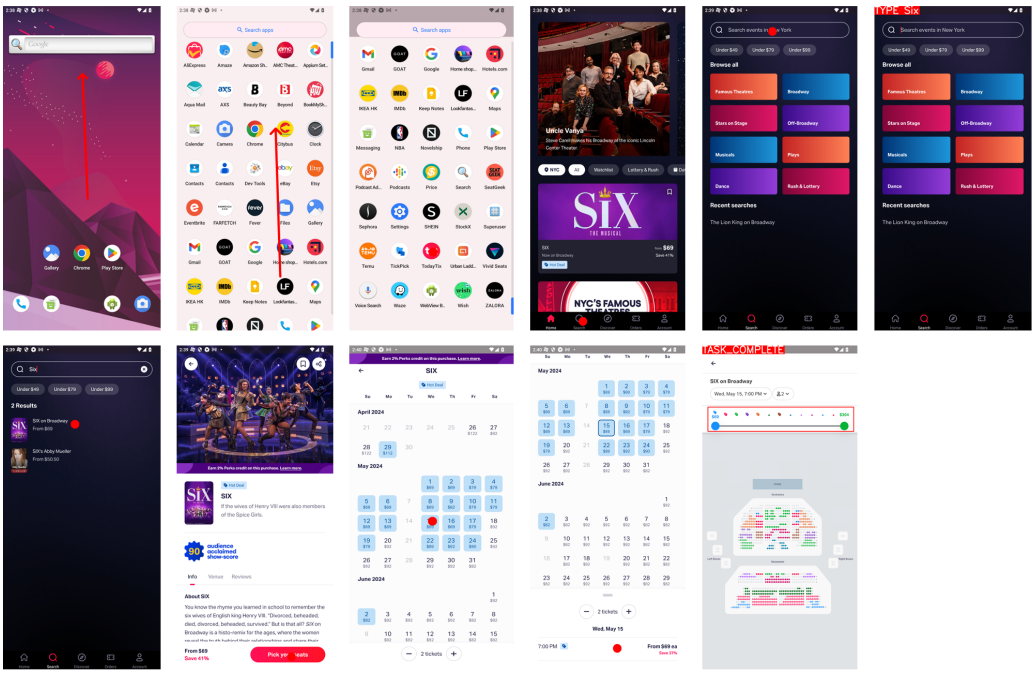


Figure 11: More examples

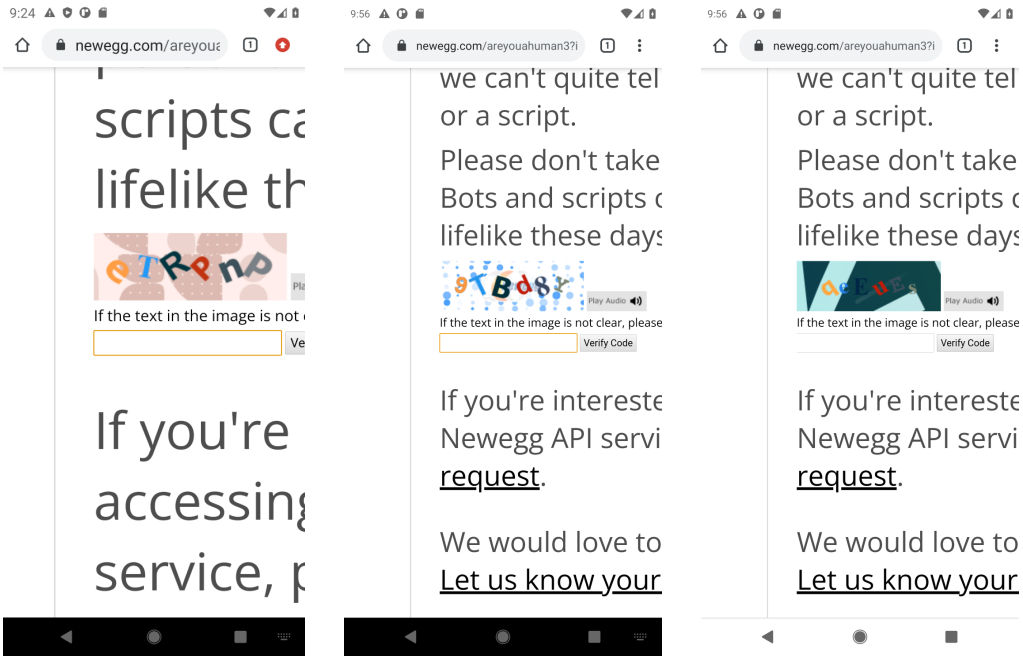


Figure 12: Demonstration of anti-script mechanism where agents can enter the verification codes correctly.