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

Yuxiang Chai[1*], Siyuan Huang[2*], Yazhe Niu[1], Han Xiao[1],
Liang Liu[3], Guozhi Wang[3], Dingyu Zhang[1], Shuai Ren[3], Hongsheng Li[1†],

[1]MMLab @ CUHK, [2]SJTU, [3]vivo AI Lab,
[*]Equal contribution, [†]Corresponding author: hsli@ee.cuhk.edu.hk

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 which are capable of completing tasks by directly interacting with the graphical user interface (GUI) on mobile devices. AMEX comprises over 104K high-resolution screenshots from popular mobile applications, which are annotated at multiple levels. Unlike existing GUI-related datasets, e.g., Rico (Deka et al., 2017), AITW (Rawles et al., 2024b), etc., AMEX includes three levels of annotations: GUI interactive element grounding, GUI screen and element functionality descriptions, and complex natural language instructions 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 finetune a baseline model SPHINX Agent and illustrate the effectiveness of AMEX.

1 Introduction

In recent years, AI-powered virtual assistants, such as Siri, Bixby, and Xiao AI, have evolved as key tools for facilitating interactions between users and mobile devices. These assistants have demonstrated to be effective in managing routine tasks such as setting alarms, performing web searches, and reporting weather conditions. However, their functionality is often restricted to system-built applications or third-party apps supported by application programming interfaces (APIs). This limitation highlights a significant gap between the capabilities of current AI assistants and the diverse, open-ended ways humans interact with mobile devices.

Unlike AI assistants, human users can accomplish tasks on mobile devices by relying solely on visual information from the screen. Humans intuitively interpret graphical user interfaces (GUIs), analyze page layouts, and infer the functionalities of interactive elements to complete tasks in various apps. Inspired by this human ability, researchers are exploring new paradigms for mobile interaction that leverage vision and natural language processing. One such paradigm involves the development of Mobile GUI-Control Agents, or GUI Agents, which aim to directly interact with screen elements based on visual and textual inputs, such as screenshots and natural language instructions. These agents hold the potential to transcend the limitations of traditional API-based assistants, enabling more flexible and universal interactions with any app or mobile interface.

Existing GUI agents show promise but face significant challenges in real-world applications. Their effectiveness is possibly limited by a lack of understanding of GUI layouts and the functionalities of interactive elements through the observation of wrong predictions. These limitations largely arise from the absence of comprehensive datasets that adequately reflect the complexity and diversity of mobile GUI environments. While several instructional datasets (Burns et al., 2021; Sunkara et al., 2022; Gubbi Venkatesh et al., 2024; Rawles et al., 2024b) have been introduced to address this issue, they struggle with limitations such as inaccurate annotations, insufficient task diversity, and a lack of representativeness for general mobile usage. Similarly, other efforts (Deka et al., 2017; Li et al., 2020; Feng et al., 2024; Bunian et al., 2021) that focus on GUI element annotation are constrained by limited dataset scale, outdated app versions, and traditional annotation styles that do not provide sufficient information for GUI agents.

Human users approach GUI-based tasks by integrating multiple cognitive processes. They in-

Findings of the Association for Computational Linguistics: ACL 2025, pages 2138–2156
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

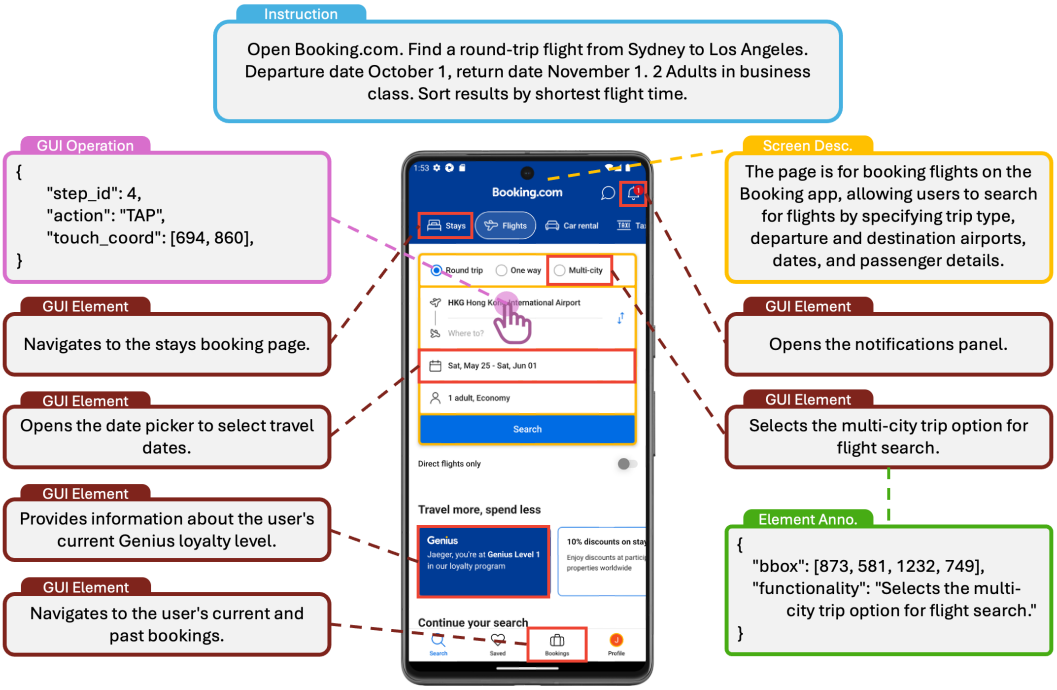


Figure 1: An example of a screenshot-instruction (Blue tab) pair illustrating the multi-level annotation of AMEX. Red boxes + Brown tabs: selected GUI interactive elements and their corresponding functionalities. Green tab: the detailed annotation of the element. Yellow tab: the description of the entire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

terpret the overall layout of the page, identify interactive elements and assess their functionality, such as recognizing that a bell icon likely leads to notifications and a magnifier icon likely triggers a search function. Users then decompose natural language instructions into actionable steps, navigating across multiple screens to achieve their goals. To design effective GUI agents, it is essential to replicate these human cognitive processes, which requires datasets that comprehensively capture the structure, functionality, and interaction patterns of mobile GUIs.

To address this need, we introduce a new dataset, the **A**ndroid **M**ulti-annotation **EX**po (AMEX), specifically designed to advance the development of GUI agents by providing a multi-level understanding of mobile GUIs. AMEX includes three levels of annotations: (i) GUI interactive element grounding, (ii) GUI screen and element functionality descriptions, and (iii) instructions with GUI-action chains. The dataset comprises over 104K high-resolution screenshots, 21K screen descriptions with 300K element-wise functionalities, and approximately 3,000 unique complex instructions, with an average of 12.8 steps (see Figure 1, Table 1, and Table 2). To ensure annotation precision and quality, all annotations, including bounding boxes, screen and element functionalities, and instructions with GUI-action chains, are verified by human annotators trained with detailed guidelines.

Our contributions can be summarized as follows: (a) We collect and release AMEX, a multi-level dataset, providing reliable and detailed understandings of the smartphone GUI environment, which can serve as a strong supplementary dataset to boost the performance of GUI agents; (b) We train SPHINX Agents, which can serve as the baseline models for future researches on GUI agents and illustrates the effectiveness of AMEX.

2 Related Work

2.1 GUI-related Datasets

Table 1 compares several popular GUI-related datasets on Android. While some works (Deng et al., 2024; Liu et al., 2018; Shi et al., 2017; Wu et al., 2023) focus on the web or desktop platforms, on Android OS, many works (Deka et al., 2017; Li et al., 2020; Feng et al., 2024; Bunian et al., 2021) have focused on identifying various types of GUI elements, assigning classes to different elements. This annotation style can be defined as a traditional icon classification and detection prob-

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

Dataset	Screen Desc.	Screen Element	Element Func.	Task & Action	# Screen-shots	# Element Func.	# Unique General Inst.	# Avg Steps
RICO	✗	✓	✗	✗	72K	-	-	-
RICO semantics	✗	✓	✗	✗	72K	-	-	-
VINS	✗	✓	✗	✗	4K	-	-	-
MUD	✗	✓	✗	✗	18K	-	-	-
PixelHelp	✗	✗	✗	✓	800	-	187	4.2
UGIF	✗	✗	✗	✓	3.3K	-	480	6.3
MoTIF	✗	✗	✗	✓	21K	-	480	4.5
AITW	✗	*	✗	✓	510K	-	1539	6.5
AITZ	✓	*	■	✓	18K	18K	2504	7.5
ANDROIDCONTROL	✗	*	✗	✓	99K	-	15,283	4.8
AMEX	✓	✓	✓	✓	104K	296K	3046	12.8

* 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.

Table 2: AMEX statistics

# Screenshots	# Apps	# Interactive Elements	# Functionalities	# Instructions	# Avg. Steps
104,876	192	1,659,647	296,075	3046	12.8

lem. In many cases, however, the functionality of an icon depends on the context of the GUI and many elements on mobile devices are not an icon and their functionalities are also critical. Besides, most of the datasets (Deka et al., 2017; Bunian et al., 2021; Li et al., 2020) are outdated, where the screenshot interfaces are completely different from modern apps. The most recent MUD (Feng et al., 2024) only provides 18K screenshots, which are insufficient for understanding the layouts and elements for LLMs.

Other studies (Burns et al., 2021; Rawles et al., 2024b; Zhang et al., 2024; Gubbi Venkatesh et al., 2024) primarily emphasize action-observation pairs during instructional operations, but their annotations are limited and often require supplemental View Hierarchy (VH) data for each screenshot. AITW (Rawles et al., 2024b) 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. In response, AITZ (Zhang et al., 2024) 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. ANDROIDCONTROL (Li et al., 2024) is another large-scale instruction-based dataset, however, the element-wise annotations contain heavy redundancy and misaligned cases due to the lack of human verification.

2.2 GUI-related Systems and Agents

(Wang et al., 2023) first starts to apply Large Language Model (LLM) on GUI, but it remains on tasks only interacted on single page, which are more like question-answer tasks rather than end-to-end instructional tasks. Recent advancements have leveraged the extensive world knowledge and robust embodied capabilities of LLMs (Gu and Dao, 2024; Gur et al., 2023; Touvron et al., 2023) for task planning and reasoning. Works (Zhang et al., 2023; Zheng et al., 2024) have utilized business-level models (e.g., GPT-4V), employing extensive prompt engineering to guide the LLM in executing complex tasks. The effectiveness of these methods requires a 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. For example, CogA-

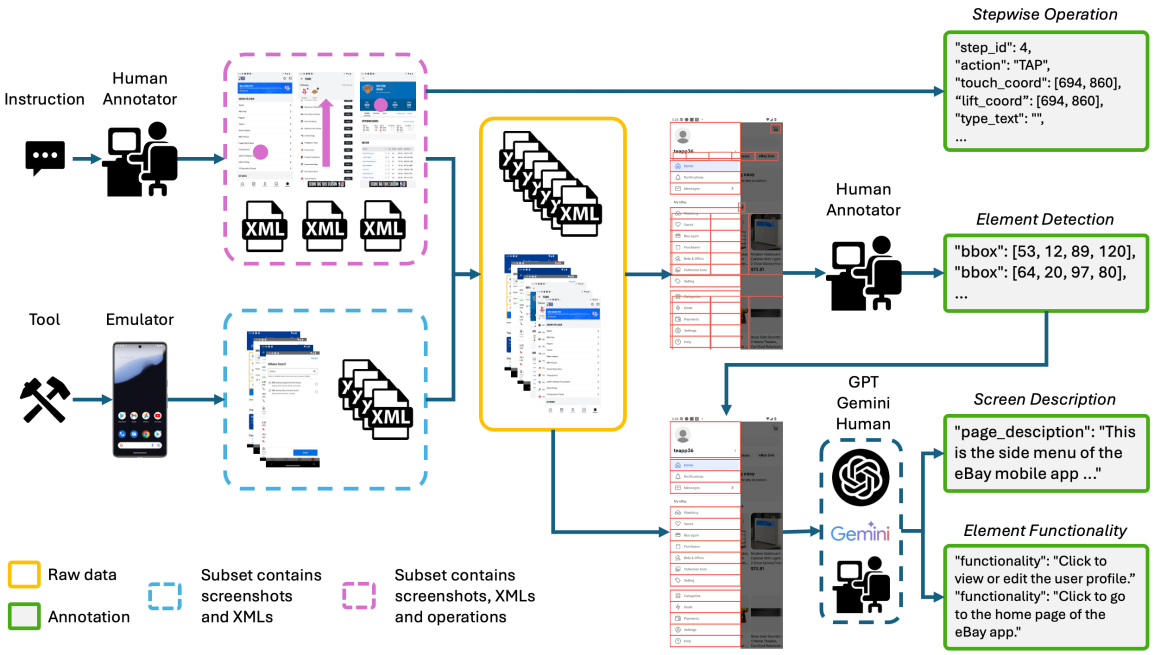


Figure 2: Overview of the data collection pipeline. The raw data is from two subsets collected by human annotators and an autonomous tool. Annotators record the GUI-action chains simultaneously while collecting the screenshots and corresponding XMLs. Then raw data is sent to annotators to filter GUI element bounding boxes, and then the boxes with corresponding raw screenshots are sent as input to GPT and Gemini to extract the GUI screen and element descriptions, which are then manually checked by humans.

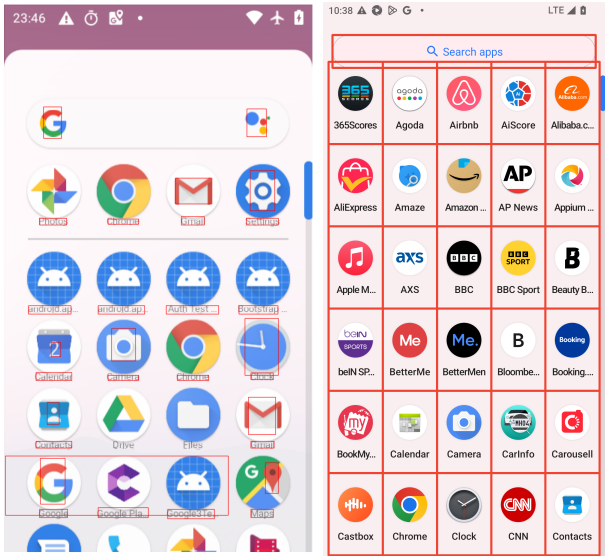
gent (Hong et al., 2024) enhances performance in GUI-related tasks by integrating a high-resolution cross-module that fuses image features from various levels. CoCo-Agent (Ma et al., 2024) and AndroidControl (Li et al., 2024), unlike other agents taking only screenshots as input, use element layouts from accessibility trees or view hierarchy as additional input to enhance the performance. However, still many apps don't support accessibility information or only provide very little of that.

3 Android Multi-annotation EXpo (AMEX)

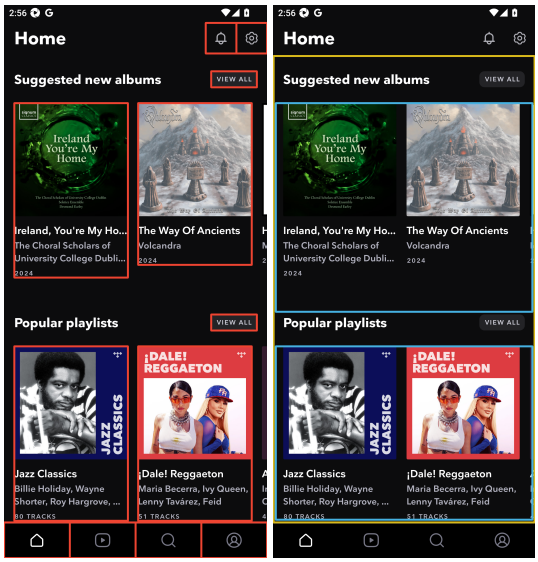
When receiving an instruction, a human user first analyzes the overall screen layout to form a basic understanding of the current Android environment. The user then identifies interactive elements and areas, and assesses the functionalities of those elements. Finally, the user breaks down the instruction 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 overall statistics of AMEX are listed in Table 2.

3.1 Data Collection Pipeline

An overview of the data collection pipeline is illustrated in Figure 2. 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 Extensible Markup Language (XML) data. 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 and Gemini generates the functionalities of the selected elements and provides descriptions for the whole screenshot. Annotators then further check the quality of the descriptions of functionalities. More details are discussed in the following sections and Appendix A.1.



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



(b) Demo of GUI interactive elements.

Figure 3: 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.

3.2 Level I: GUI Interactive Element Grounding

Existing datasets (Deka et al., 2017; Sunkara et al., 2022; Feng et al., 2024; Bunian et al., 2021) 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 3a). Specifically, interactive elements in our dataset are only categorized into two subsets: (i) clickable elements and (ii) scrollable elements. The motivation of our annotation is mainly from two facts: (i) no existing dataset provides information on the "scroll" action, which is crucial especially for agents to know the horizontally scrollable area. (ii) "click" action is the most used action and many "compound" elements cannot be simply categorized. As illustrated in Figure 3b red boxes, four "compound" elements in the middle contain both image and text and the definition of "clickable" is more suitable with the functionality in Level 2 for agents to understand the element.

Clickable Elements are the most common components in a screen. They typically include clickable icons, images, texts, and compounds that combine several categories. 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 contains many clickable elements and supports a pair of actions, such as "scroll down" and "scroll up" or "scroll left" and "scroll right." Figure 3b illustrates these two types of scrollable elements.

3.3 Level II: GUI Screen and Element Functionality Descriptions

Previous works (Deka et al., 2017; Feng et al., 2024; Bunian et al., 2021) on GUI elements often rely on predefined class names, such as "image" and "text", to convey the underlying meaning of each element. Rico Semantics (Sunkara et al., 2022) annotates the icons with slightly more detailed semantics information but the annotations are still classifications on icons. However, this classification-based method has significant limitations. For instance, a "plus" symbol typically indicates "plus" in calculator interfaces, but it means "create a new task" in a ToDo interface. This class-based annotation approach focuses on class labels rather than truly understanding the functionality of each element in the surrounding context, which may lead to a misunderstanding of the page layout.

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 icon and labeling it as "ICON_PLUS", we present the actual functionality of the element within its context (e.g., "Create a new task"). This strategy offers a clear, instructive, and detailed description of the interactive element, enhancing the dataset's applicability. Besides, the traditional categories of "IMAGE" and "TEXT" provides very little information of the actual element functionality. Our annotation will provide a much more detailed and informative functionality for those elements (e.g., "View the detail page of the 'Jazz Classics' playlist" for the red left bottom "compound" element in Figure 3b). Further elaboration on the collection and processing is provided in Appendix A.1.4.

3.4 Level III: Instructions with GUI-Action Chains

Table 1 illustrates that most instruction-based datasets provide an average number of operation steps below 7, which implies most of the instructions are simpler than tasks in real-world. For example, users often apply filters or sort by different attributes when searching, which causes more complex instructions and longer action chains. In order to address this situation, we collect a set of complex instructions with GUI-action chains on popular apps from Google play store.

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 users to fully simulate a wide range of typical use cases. Detailed collection process and instruction generation are in Appendix A.1.5.

Table 3: ScreenSpot mobile subset experiment results.

Model	Size	Icon / Widget
Fuyu	8B	1.3%
CogAgent	18B	24.0%
SeeClick	9.6B	52.0%
Qwen2-VL	7B	60.7%
GPT-4V w. OmniParser	-	57%
SphAgent	7B	**72.6%**

4 Experiments

AMEX serves as a supplementary dataset for other large-scale instruction-based datasets. To test the effectiveness of AMEX, we directly finetune SPHINX models (Liu et al., 2024) without any tricks and conduct evaluation on three other benchmark datasets. We choose ScreenSpot (Cheng et al., 2024) as the benchmark of our agent's element grounding ability and AITW and ANDROIDCONTROL as the benchmarks of our agent's GUI-control ability.

4.1 SPHINX Agent

Our SPHINX agent, SphAgent for short, is initialized from SPHINX (Liu et al., 2024) and it is finetuned for the GUI-control tasks and element grounding tasks. For both tasks, we train the model in a pure vision-based principle, which means the model understands the environment by only screenshots, without any other supplementary information such as accessibility trees or view hierarchy, due to the fact that many apps don't support or only provide very little accessibility tree or view hierarchy element information. We consider vision-based agents are more applicable in more various use cases.

4.2 ScreenSpot

ScreenSpot (Cheng et al., 2024) is a benchmark dataset which contains over 600 UI screenshots from mobile devices, desktops and websites. It is designed to evaluate the model's element grounding capability, where each human-annotated functionality corresponds to an interactive element. We train SPHINX agent on AMEX level 1 and level 2 data to evaluate the element grounding capability.

Table 3 lists the evaluation results on the ScreenSpot mobile subset. The models (Bavishi

Table 4: Experiment results on ANDROIDCONTROL. "Click & Long Press" is specially computed to show the improvements on these actions. The best results are in bold for different subsets and levels.

Agent	# data	Training Data	Task Level	IDD	Category Unseen	App Unseen	Task Unseen	Overall	Click & Long Press
SphAgent	178K	ANDROIDCTRL	High Lv.	58.3	40.6	39.4	52.2	49.8	34.1
			Low Lv.	79.8	71.0	71.1	83.5	75.8	53.0
SphAgent	178K	70% ANDRCTRL + 10% AMEX	High Lv.	60.4	41.7	43.4	59.3	52.8	37.3
			Low Lv.	81.9	73.2	72.6	87.5	77.7	56.7
SphAgent	712K	ANDROIDCTRL + AMEX	High Lv.	**70.5**	**51.6**	**55.6**	**70.2**	**61.7**	**50.8**
			Low Lv.	**88.9**	**81.8**	**76.9**	**92.6**	**84.2**	**67.0**

et al., 2023; Hong et al., 2024; Cheng et al., 2024; Wang et al., 2024) in first four rows are LVLMs specially trained on GUI data. OmniParser (Wan et al., 2024) is a combination of an icon detection model, an icon description model and an OCR module, which serves as a screenshot parser for other LVLMs such as GPT-4V. During evaluation, our SphAgent surpasses four LVLMs by a large margin on the "Icon / Widget" subset without any training tricks, proving that the functionality understanding can largely boost the performance of GUI grounding for agents. The above results and comparisons strongly prove the effectiveness and applicability of AMEX in GUI element grounding tasks.

4.3 ANDROIDCONTROL

ANDROIDCONTROL (Li et al., 2024) is a large-scale benchmark dataset to evaluate the agent's GUI-control ability to complete tasks on general apps (see more detailed description of the dataset and splits in Appendix A.2.1). We evaluate three SphAgents trained on (i) only ANDROIDCONTROL, (ii) random 70% ANDROIDCONTROL data mixed with random 10% AMEX level 1 and 2, (iii) a mix of ANDROIDCONTROL and AMEX level 1 and 2. The (ii) experiment ensures the same number of data point as in (i) experiment. The action space of ANDROIDCONTROL is different from AMEX level 3 and AITW, so we only use level 1 and level 2 data during the experiments (ii) and (iii).

Table 4 lists the evaluation results on ANDROIDCONTROL test set. The evaluation results of experiment (i) and (iii) strongly support the effectiveness of AMEX level 1 and level 2. Adding full data of AMEX level 1 and level 2 leads to an average 10% overall performance gain and strongly improves the "click" and "long press" actions by more than 14%. Also, the comparative results from experiment (i) and (ii) shows that even using the same number of data points, replacing down-stream instructional task data with environment understanding data would also lift the performance of agents on both low-level and high-level at an average of 2.5%. This performance gain indicates the effectiveness of our multi-level annotations.

4.4 AITW

AITW is another large-scale benchmark dataset to evaluate the agent's GUI-control ability to complete task goals. We evaluate three SphAgents trained on (i) only AITW, (ii) a mix of AITW and AMEX. Since the GUI-action chains in AMEX share the same action space with AITW, we utilize all three levels of AMEX in the mixed data.

Table 5 lists the evaluation results on AITW test set, which has five subsets for different types of tasks. SphAgent trained on the mixed data (AITW + AMEX) achieves the highest overall accuracy among all agents, and when compared to the SphAgent trained on only AITW, the accuracies in "General" and "Single" categories have a notable gain of around 5% and the overall accuracy lifts by 2.5%. That the gains in other categories are not remarkable is possibly due to the mis-aligned element-wise annotation and unregulated instruction operation annotation. In Appendix A.2.2 we further explore the dataset evaluation and explain the reason why the evaluation results might be misleading.

4.5 Cross-Domain Experiments

Since AMEX Level 3 shares the same action space only with AITW, we conducted two experiments: (i) training SphAgent on AMEX and testing it on AITW, and (ii) training SphAgent on AITW and testing it on AMEX. The results, as shown in the last row of Table 5, indicate poor performance when the agent is trained only on AMEX and tested on AITW. Similarly, Table 6 demonstrates poor per-

Table 5: Experiment results on AITW.

Agent	Training Data	General	Install	G-Apps	Single	WebShopping	Overall
SphAgent	AITW	68.2	80.5	73.3	85.4	74	76.28
SphAgent	AITW + AMEX	73.1	80.6	73.4	90.8	75.8	78.72
SphAgent	AMEX	51.5	55.6	57.1	61.9	55.1	56.2

Table 6: Experiment results on AMEX. The test apps are excluded during the training of SphAgent on AMEX.

Agent	Training Data	Gmail	Booking	YT-Music	SHEIN	NBC	CM	ToDo	Signal	Yelp	Overall
SphAgent	AITW	32.1	45.9	46.1	35.1	48.3	61.1	55.9	43.3	42.9	45.6
SphAgent	AMEX	63.7	67.4	76.1	70.0	68.5	62.7	78.6	68.2	68.9	69.3

formance when the agent is trained on AITW and tested on AMEX. This significant discrepancy in evaluation results is likely due to the domain gap between the two datasets. First, the styles of collecting GUI-action chains differ between the two datasets. For instance, annotators in AITW tend to directly search for apps, whereas annotators in AMEX prefer navigating through the app library to locate apps. This inconsistency in interaction strategies poses a challenge for the agent, which evaluates tasks based on static frames rather than dynamically adapting to different usage patterns. Second, although both datasets share the same action space, they differ in terms of visual appearance and versioning. The apps in AMEX are updated to their latest versions, while those in AITW are based on older versions. These versioning differences lead to changes in the visual appearance of GUI elements (e.g., color schemes, icons, or button designs), which can negatively impact the agent's ability to recognize and interact with them.

5 Discussions

5.1 Limitations and Future Work

Multi-lingual Most existing datasets are limited to English, with UGIF (Gubbi Venkatesh et al., 2024) 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 multilingual screenshots, functionalities, and instructions to create a more robust and comprehensive multi-lingual environment for GUI agents.

Online Evaluation Due to the limitation that SPHINX is not deployable, we encountered issues trying online evaluation such as AndroidWorld (Rawles et al., 2024a). In future work, online dynamic evaluation is more accurate and simulate the real-world agent usage.

Multi-Platform Android is an open-source OS and the community provides well-developed tools for interactions and emulations. iOS is another widely used mobile device OS but it is restricted and hard to run test on. In the future, a dataset extended to other OS or platforms would provide more general information.

5.2 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.

6 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 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

Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. 2023. Introducing our multimodal models.

Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. Vins: Visual search for mobile user interface design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14.

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

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, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual Symposium on User Interface Software and Technology*, UIST '17.

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.

Sidong Feng, Suyu Ma, Han Wang, David Kong, and Chunyang Chen. 2024. Mud: Towards a large-scale and noise-filtered ui dataset for modern style ui modeling. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA. Association for Computing Machinery.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*.

Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. 2024. UGIF-DataSet: A new dataset for cross-lingual, cross-modal sequential actions on the UI. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1390–1399, Mexico City, Mexico. Association for Computational Linguistics.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2023. Understanding HTML with large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2803–2821, Singapore. Association for Computational Linguistics.

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

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8198–8210, Online. Association for Computational Linguistics.

Dongyang Liu, Renrui Zhang, Longtian Qiu, Siyuan Huang, Weifeng Lin, Shitian Zhao, Shijie Geng, Ziyi Lin, Peng Jin, Kaipeng Zhang, Wenqi Shao, Chao Xu, Conghui He, Junjun He, Hao Shao, Pan Lu, Yu Qiao, Hongsheng Li, and Peng Gao. 2024. SPHINX-x: Scaling data and parameters for a family of multimodal large language models. In *Forty-first International Conference on Machine Learning*.

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

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2024. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9097–9110.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2024. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2024a. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2024b. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36.

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

Srinivas Sunkara, Maria Wang, Lijuan Liu, Gilles Baechler, Yu-Chung Hsiao, Jindong Chen, Abhanshu Sharma, and James W. W. Stout. 2022. Towards better semantic understanding of mobile interfaces. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5636–5650, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities.

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

Jianqiang Wan, Sibo Song, Wenwen Yu, Yuliang Liu, Wenqing Cheng, Fei Huang, Xiang Bai, Cong Yao, and Zhibo Yang. 2024. Omniparser: A unified framework for text spotting key information extraction and table recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15641–15653.

Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17.

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

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. 2023. 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.

Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. 2023. Webui: A dataset for enhancing visual ui understanding with web semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14.

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

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *CoRR*, abs/2312.13771.

Jiwen Zhang, Jihao Wu, Teng Yihua, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024. Android in the zoo: Chain-of-action-thought for GUI agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12016–12031, Miami, Florida, USA. Association for Computational Linguistics.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61349–61385. PMLR.

A Appendix / supplemental material

A.1 Pipeline Details

The raw data is collected using Android emulators, specifically Android Virtual Device and Genymotion[1]. We developed tools to autonomously perform emulator operations and record human actions. These tools leverage Appium[2], an open-source, cross-platform test automation tool for Android, iOS, and web apps, under Apache license. Appium collects device screenshots during operations, each of which also corresponds to an XML file recording the basic screen layout with element attributes, such as bounding boxes and in-app descriptions.

A.1.1 Collection details

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.1.2) at regular time intervals to allow each page to fully load. Then the script captures the current screenshot along with the corresponding XML file. Upon the raw data (yellow part in Figure 2), we extract bounding boxes for interactive elements from the XML associated with each

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

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

screenshot. Then annotators manually inspect each screenshot, identifying and selecting only the elements that are actually visible within the interface (see Appendix A.1.3 for detailed reasons).

A.1.2 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.1.3 GUI clickable element filtering

The raw XML information sometimes contains the elements that are covered by other elements or layers. Figure 4 illustrates the same image before and after filtering. Blue boxes in Figure 4a 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 4b.

A.1.4 GUI screen and element functionality description collection details

We adopt GPT-4o and Gemini 1.5 Pro to describe screens and interpret element functionalities separately. Given the compact and dense nature of GUI elements, we apply Set of Mark (SoM) techniques (Yang et al., 2023) to boost GPT and Gemini'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 and Gemini with in-app descriptions extracted from XMLs to enhance their comprehension of the screen. After the generation, we further utilize GPT-4o to compare the functionalities of the same element generated by two models and keep them if two functionalities have the same meaning. Finally, human annotators verify the quality and the functionality accuracy rate is above 97%.

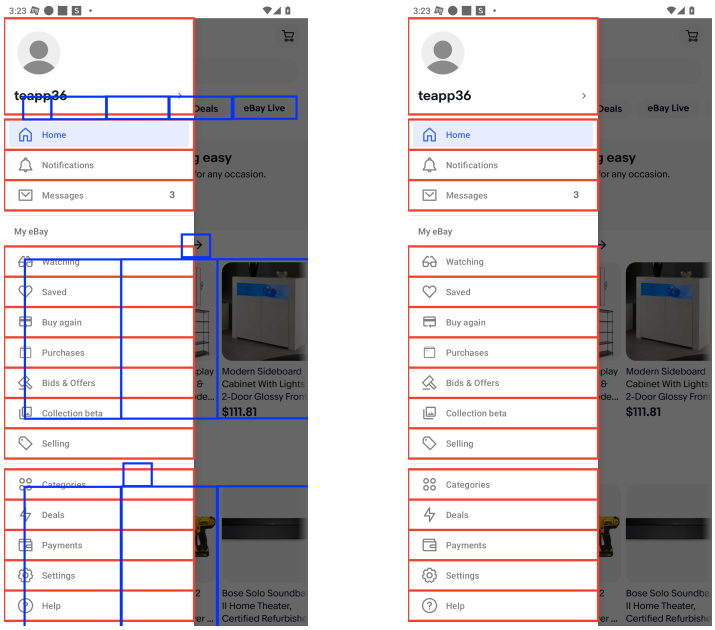
We apply the Set-of-Mark (SoM) technique when using GPT and Gemini as the description generator. The SoM technique is a visual prompting method designed to enhance the visual grounding capabilities of large multimodal models (LMMs), such as GPT-4V, by overlaying visual marks on image regions. This involves 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 5 shows the screenshot with SoM technique.

The most recurring mistake is the misinterpretation of the content description. For instance, some "compound" elements, which may contain several text, rating stars, image, etc., have couple of content descriptions. Those mixed content descriptions may cause the misunderstanding of the real functionality of the compound element. In IMDB app, a row of movie entry sometimes would consist of a poster, a text name, ratings of the movie, and the price of renting. The desired functionality of the entry could be "view the detail page of the movie xxx", while GPT might generate "view the price for renting the movie xxx", which is not exactly what we want. However, we use two LLMs to double check their generations on the same element functionality and we only keep the elements with the same functionalities generated by two LLMs.

The prompt is in the following Listing 1 format.

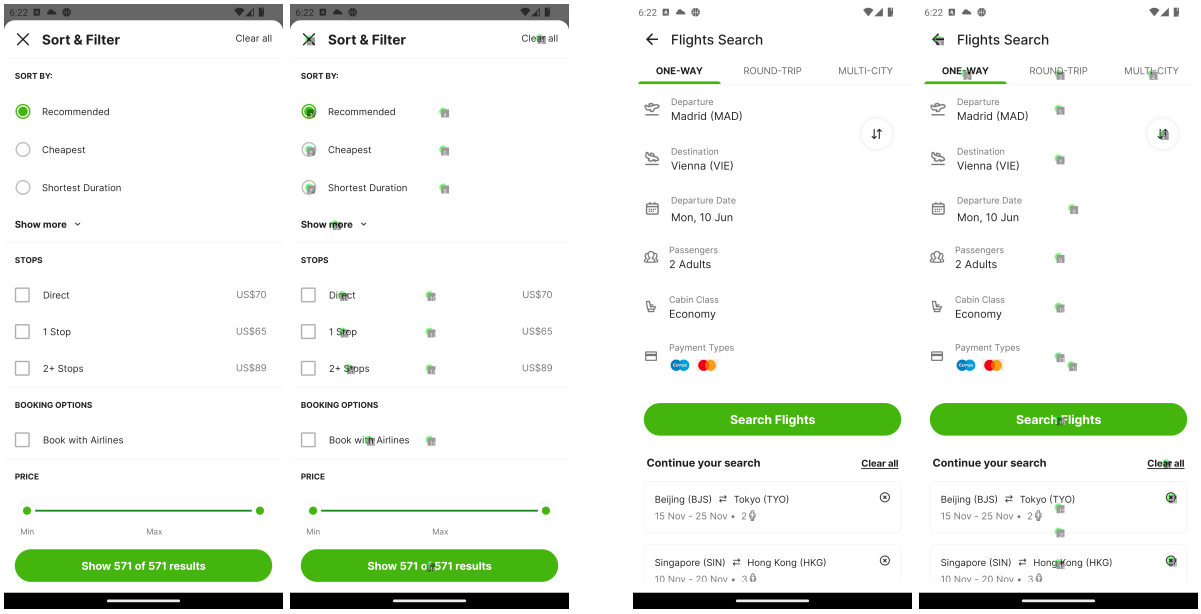
A.1.5 GUI-Action chain collection details

The instruction generation process comprises three phases. Initially, human annotators create 5-10 complex instructions for each target app, considering its specific functions and capabilities. These initial instructions, combined with relevant app metadata collected online, serve as input for GPT. It then



(a) (b)

Figure 4: Demonstration of human annotator filtering. (a) before filtering. (b) after filtering.



(a) (b)

Figure 5: Demonstrations of SoM technique on screenshots.

```
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.

generates a larger set of 80-100 instructions that exhibit similar structure and intent to the human-provided examples. However, due to GPT's limitations on understanding real-world constraints, human filtering is adopted to modify or remove any unreasonable or impractical instructions. For example, the human-proposed instruction "*Open Booking.com, search for stays in Washington DC from 2024 June 11 to June 15.*" would generate a similar but impractical "*Open Booking.com, search for stays in New York from 2024 May 3 to May 7*", which needs modification since the dates are not selectable when the task is generated.

Since the meta information online doesn't reflect all functions of an app and sometimes may include misleading information, the instruction generated by GPT may have two types of impracticabilities: (i) the task is unrelated to the app activity (ii) some of the steps in the instruction are impractical. For instance, an app "SEPHORA" only sells beauty related products, but GPT may generate tasks that searches for electronics. Another example would be about a ticket booking app "SeekGeek", which doesn't provide a sort option "sort by price from high to low". While human created task would include "sort by price from low to high", GPT would generate something with"sort by price from high to low". This step is impractical so we would filter it out.

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.1.6 Collection resources details

- GUI interactive element grounding takes approximately 3000 human-hour to filter bounding boxes described in Appendix A.1.3.
- GUI screen and element descriptions use GPT-4o and Gemini API, which consumes about 800 dollars.
- Instructions with GUI-action chains take approximately 300 human-hour.

A.2 Experiment details

In our implementation, we utilize the internlm-7b variant of the SPHINX model, as detailed in (Liu et al., 2024). The pre-trained checkpoint for this model was sourced from the official repository mentioned in (Team, 2023). 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 (Oquab et al., 2024) and ConvNext (Woo et al., 2023). 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 4.1, 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.2.1 ANDROIDCONTROL details

ANDROIDCONTROL splits the test set into four categories, i.e., in-domain data (IDD), category unseen, app unseen, and task unseen. Besides, the instructions are split into two levels. The high-level instructions are simply the goal of the task and the low-level instructions are the guides for each step during the task execution. Low-level tasks are easier for agents due to the fact that the agent only need to perform the action specified by the step-wise low-level instruction, while the high-level tasks require the agents to perform actions based on the overall instruction. For example, a step-wise low-level instruction is "*click the profile element at the bottom right of the screen.*", compared with high-level instruction "*Change my profile name to Unknown111*".

A.2.2 AITW details

We observe that there are several types of error cases in the AITW test set (see Figure 6). To assess the unreliability of the original AITW test set annotations, we hire human annotators to evaluate two subsets derived from the original test set. One subset is randomly chosen from episodes where SphAgent receives low accuracy (named `Mis-Match Random`), i.e., about 66% compared to the overall 78.72% shown in Table 5, indicating a high mismatch between inference results and AITW annotations. The other subset is randomly selected from the remaining episodes (named `Random`). Annotators first filter out repeated and redundant screenshots (see Figure 6a). For each remaining screenshot, the annotators then record the accuracies of SphAgent-inferred actions and the original AITW annotations versus our human evaluations. Figure 7 illustrates cases where annotators mark both the inferred action and the original annotation as correct, even though they interact with different elements. Table 7 presents the accuracy from human evaluation. The table indicates that the low SphAgent accuracy 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 accuracies 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 5 and Table 7, 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 annotations.

A.3 More AMEX examples

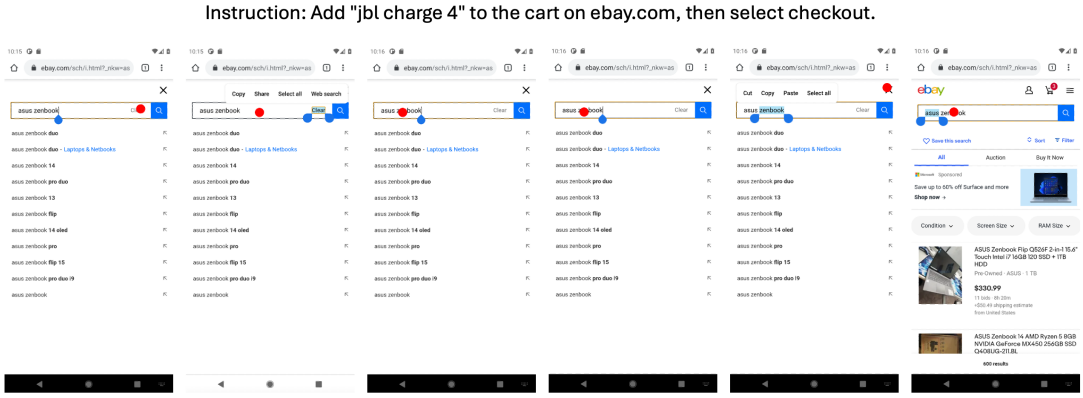
See Figure 8 for more examples of GUI interactive elements grounding and description. See Figure 9 for more examples of instruction with GUI-action chains.

A.4 Examples of Ethical Problems

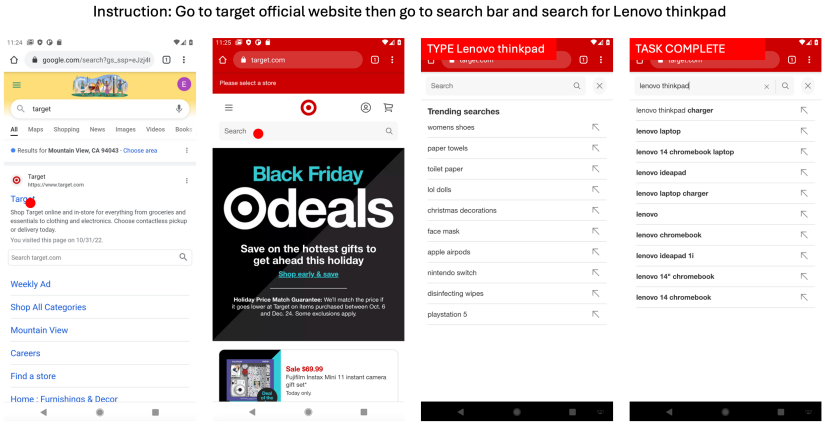
Figure 10 shows examples of anti-script mechanism where the agent can correctly enter the verification codes.

Table 7: 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 vs. Human	90.91	83.65	87.50	87.68	82.39	86.43
	AITW Anno. vs. Human	88.18	75.00	82.69	84.78	75.57	81.25
Random	SphAgent vs. Human	92.31	93.75	94.33	94.83	94.55	93.95
	AITW Anno. vs. Human	94.23	90.34	93.62	93.97	95.00	93.43



(a) Repeating useless screenshots and actions.



(b) Wrong task complete.

Figure 6: Error cases in AITW test set. Red dots indicate the actions from the AITW annotations.

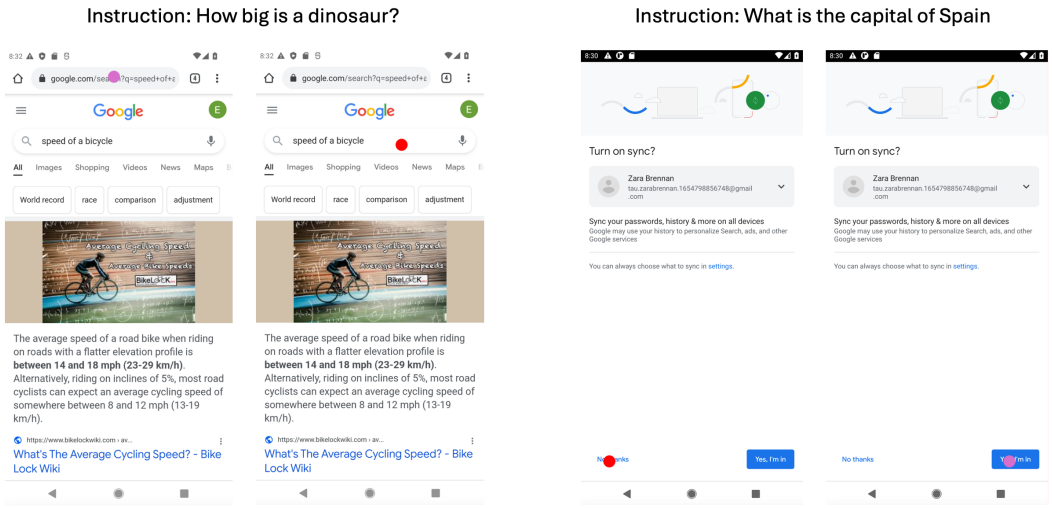
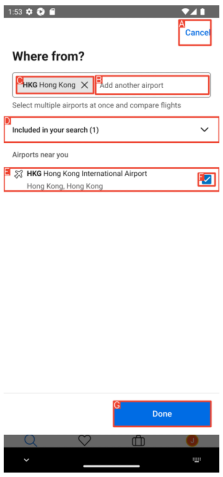
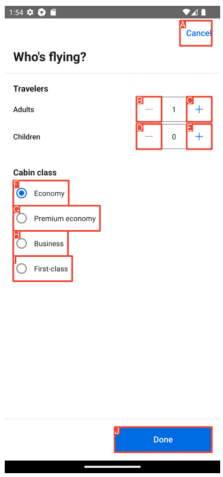


Figure 7: Purple dots indicate the SphAgent inferred actions and red dots indicate the AITW annotations. Human annotators mark them both correct even though they are clicking different elements.



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.

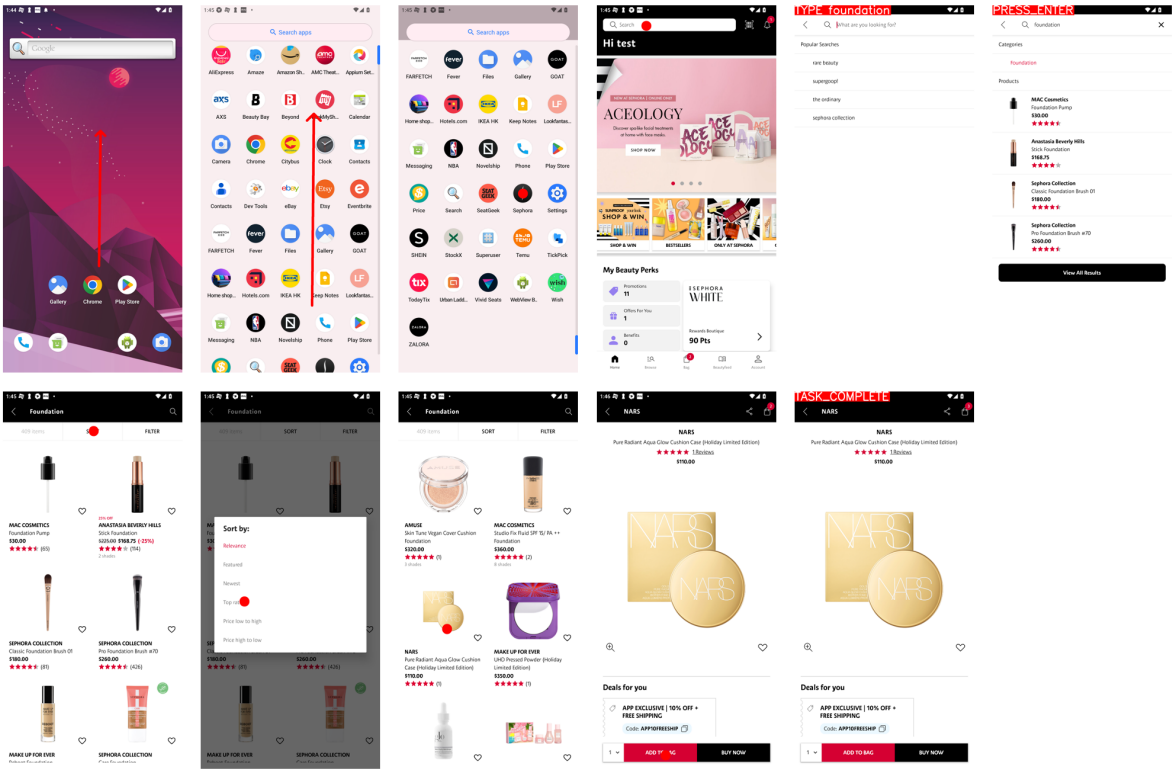


A. Click to cancel the current action and go back to the previous screen.
B. Click to decrease the number of adults traveling.
C. Click to increase the number of adults traveling.
D. Click to decrease the number of children traveling.
E. Click to increase the number of children traveling.
F. Click to 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 8: 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?

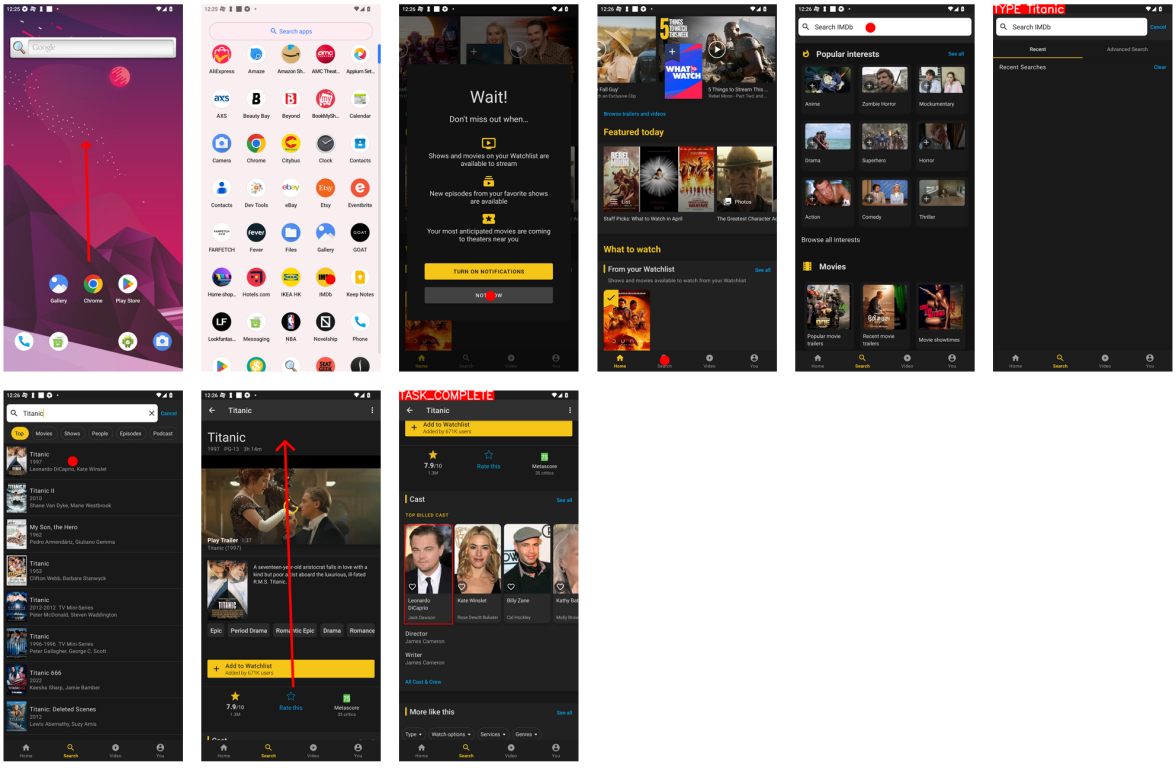


Figure 9: More examples

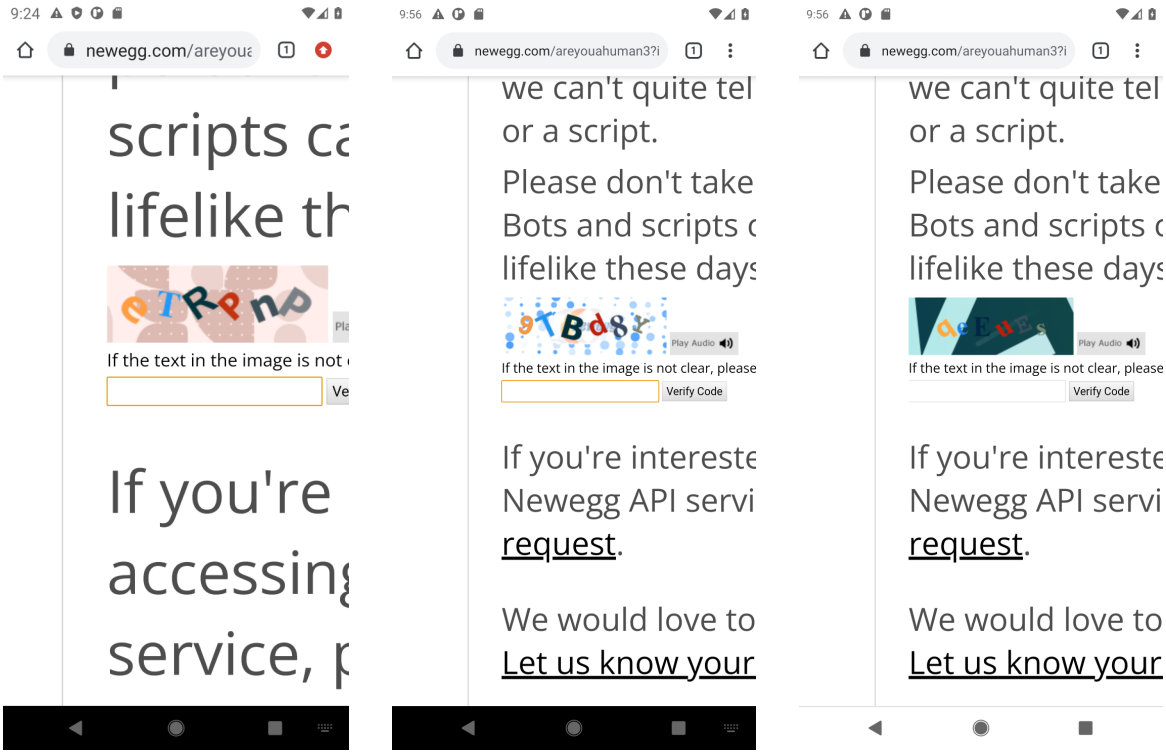


Figure 10: Demonstration of anti-script mechanism where agents can enter the verification codes correctly.