GUIOdyssey: A Comprehensive Dataset for Cross-App GUI Navigation on Mobile Devices

Quanfeng Lu[4,1], Wenqi Shao[1,†], Zitao Liu[5], Lingxiao Du[3,1], Fanqing Meng[4,1], Boxuan Li[3], Botong Chen[5], Siyuan Huang[4,1], Kaipeng Zhang[1], Ping Luo[2,†]
[1]Shanghai AI Laboratory [2]The University of Hong Kong [3]Nanjing University
[4]Shanghai Jiao Tong University [5]Harbin Institute of Technology, Shenzhen
shaowenqi@pjlab.org.cn, pluo@cs.hku.hk
https://github.com/OpenGVLab/GUI-Odyssey

Abstract

Autonomous Graphical User Interface (GUI) navigation agents can enhance user experience in communication, entertainment, and productivity by streamlining workflows and reducing manual intervention. However, prior GUI agents often trained with datasets comprising tasks that can be completed within a single app, leading to poor performance in cross-app navigation. To address this problem, we present GUIOdyssey, a comprehensive dataset for cross-app mobile GUI navigation. GUIOdyssey comprises 8,334 episodes with an average of 15.3 steps per episode, covering 6 mobile devices, 212 distinct apps, and 1,357 app combinations. Each step is enriched with detailed semantic reasoning annotations, which aid the model in building cognitive processes and enhancing its reasoning abilities for complex cross-app tasks. Building on GUIOdyssey, we develop OdysseyAgent, an exploratory multimodal agent for long-step cross-app navigation equipped with a history resampler module that efficiently attends to historical screenshot tokens, balancing performance and inference speed. Extensive experiments conducted in both in-domain and out-of-domain scenarios validate the effectiveness of our approach. Moreover, we demonstrate that historial information involving actions, screenshots and context in our dataset can significantly enhances OdysseyAgent's performance on complex cross-app tasks.

1. Introduction

Smartphones have become indispensable tools in our daily lives [6]. With a growing number of mobile applications, users frequently navigate across multiple apps to complete tasks, such as sharing content between social media platforms or coordinating schedules between messaging apps and calendars. Introducing a smart assistant to streamline these workflows and reduce manual intervention would be highly beneficial, particularly for individuals with physical disabilities [35]. Nowadays, the rapid advancement of large foundation model [1, 2, 12, 18, 50] has enabled the development of intelligent agents [1, 40, 51, 69]. These agents process environmental observations, maintain multi-turn context, and execute actions to achieve specific goals, making autonomous GUI navigation increasingly feasible and practical.

While current foundation models are yet fully capable across various domains [62], they can still be effectively leveraged through GUI navigation datasets to build GUI agents that deliver more efficient and user-friendly mobile experiences. For instance, AITW [38] constructs a dataset encompassing various tasks to develop generalist agents for smartphones using large language models (LLMs) [16]. Similarly, AndroidControl [26] introduces a dataset focused on everyday tasks involving Android apps, providing both high-level and low-level instructions for GUI agents. However, these datasets primarily comprise operational actions, such as 'click' and 'scroll'. Furthermore, existing mobile GUI navigation datasets predominantly focus on tasks solvable within a single app, as depicted in Fig. 1(a). However, many real-world tasks require cross-app navigation, involving the transfer of context and data among multiple apps, as shown in Fig. 1(b). These complex workflows cannot be fully captured by single-app datasets, nor can they be decomposed without losing critical cross-app interactions. While some studies have investigated cross-app tasks, their focus has been limited to evaluation purposes [28, 39, 48, 55]. In particular, evaluations from studies [54, 55] reveal that current performance on cross-app tasks remains significantly worse than on single-app tasks. Therefore, it is crucial to develop dedicated datasets to im-

†Corresponding author.

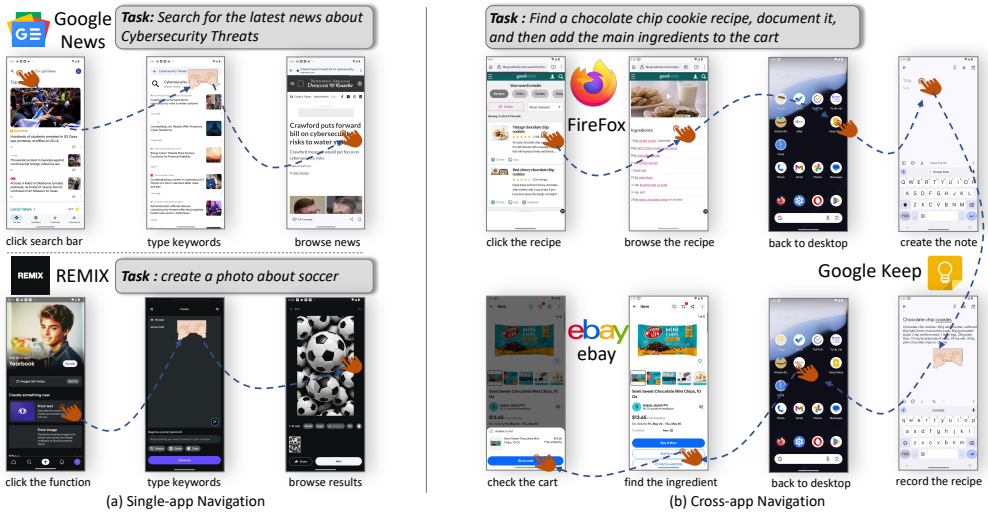


Figure 1. Illustration of single-app (a) and cross-app (b) GUI navigation. We see that cross-app navigation tasks demand the integration of multiple apps and the transfer of context and data between them, involving more complex workflows than single-app navigation.

prove the cross-app navigation capabilities of GUI agents.

To address this issue and advance the development of general GUI agents, we introduce GUIOdyssey, the first cross-app GUI navigation dataset for mobile devices, featuring task instructions designed to reflect two levels of granularity. High-level instructions emulate natural human requests to capture real-world needs, whereas low-level instructions correspond to fine-grained tasks, providing precise and unambiguous guidance to eliminate potential misunderstandings. On one hand, we propose high-level [17, 38] cross-app navigation instructions by brainstorming with human participants and GPT-4 [1], and create episode-specific user instructions to enrich task diversity. Independent annotators are then employed to annotate the entire navigation process comprehensively, including screenshots and corresponding actions, using an Android emulator[1]. On the other hand, after collecting the human-annotated data, we use GPT-4o [36] to generate low-level instructions [4, 14] for each step, providing a more fine-grained guide to task completion and facilitating deeper exploration of the GUI agent's potential for cross-app tasks.

To simulate the human approach, we further enrich GUIOdyssey with semantic annotations by breaking down each step into three components: screen comprehension, historical context review, and decision-making reasoning. GPT-4o [36] is employed to generate semantic annotations for these components. Subsequently, GUIOdyssey undergoes a quality check to ensure screenshot integrity, action accuracy, and alignment between GPT-4-generated instructions and the originals. After construction through this rigorous pipeline, designed to enhance task diversity and annotation quality, GUIOdyssey comprises $8,334$ episodes with an average of 15.3 steps, meticulously curated from 6 different mobile devices such as Pixel Pro and Tablet. It features 6 types of cross-app navigation tasks, ranging from general system tool use to media entertainment, involving navigation across 212 different apps and $1,357$ app combinations in fields such as video, music, and reading, as depicted in Fig. 2. Table 1 presents a comparison between GUIOdyssey and previous datasets.

Leveraging GUIOdyssey, we develop an exploratory cross-app multimodal agent named OdysseyAgent. Cross-app navigation tasks inherently involve long step sequences, requiring to retain numerous screenshots and actions for informed decision-making. However, processing large numbers of screenshot tokens can significantly slow inference, which is a critical concern for GUI agents frequently interacting with users. To balance performance with speed, OdysseyAgent incorporates a history resampler module that selectively attends to historical screenshot tokens while maintaining high inference throughput, thereby effectively and efficiently tackling complex cross-app tasks. We thoroughly validate our approach on GUIOdyssey in both in-domain and out-of-domain scenarios. OdysseyAgent achieves highest accuracy among existing methods, including Claude3.5-Sonnet and GPT-4o. Moreover, incorporating semantic annotations leads to further performance gains. We also conduct an in-depth analysis demonstrating that enriching historical information with actions, screenshots, and contextual information significantly improves OdysseyAgent's performance, highlighting the importance of comprehensively modeling historical information for complex cross-app navigation tasks.

The contributions of this work are three-fold. 1) We in-

[1] https://developer.android.com/studio

troduce GUIOdyssey, a comprehensive dataset for cross-app mobile GUI navigation, comprising $8,334$ episodes with an average length of 15.3 steps. It covers a wide range of apps, tasks, and devices, with each step annotated by rich semantic reasoning to facilitate cognitive processes and enhance reasoning capabilities, thereby boosting performance on complex cross-app tasks. 2) We propose OdysseyAgent, an exploratory multimodal agent equipped with a history resampler module that balances performance and inference speed for cross-app navigation. 3) Through extensive experiments with OdysseyAgent, we demonstrate that comprehensively leveraging historical information substantially enhances performance on cross-app navigation tasks, highlighting the importance of historical information modeling.

2. Related Work

GUI Navigation Agent. Large foundation models [1, 2, 12, 30, 46, 50, 56] have recently demonstrated the capacity to utilize extensive world knowledge to solve complex autonomous tasks [34, 41, 43, 59, 61]. These advancements have paved the way for the development of GUI agents capable of autonomous device control. For instance, works such as [17, 17, 21, 67] focus on autonomous agents in the Web domain, while studies like [15, 28, 48, 49, 57] leverage powerful language models, such as GPT-4V [1], to address GUI navigation tasks on mobile devices. Additionally, other research [52, 65] explores the potential applications of OS-specific agents. This line of research often incorporates supplementary inputs, such as accessibility trees (A11y trees), to provide details like UI element coordinates or utilizes the Set-Of-Marks [58] strategy to outline bounding boxes of UI elements, supported by GUI-specific grounding models [20, 33]. An alternative approach, as exemplified by [9, 14, 19, 23, 29, 42, 53, 63, 66], employs a coordinate-based method combined with visual models to develop GUI navigation agents. This approach directly provides positional information for executing actions, without relying on additional information. While coordinate-based navigation can be fragile and may underperform in certain scenarios, it represents the ultimate solution for GUI navigation in the long run [54]. In cases where structured A11y trees are unavailable or impractical [14, 42, 54], coordinate-based navigation offers a natural and straightforward solution that enhances task and device transferability [11]. GUIOdyssey specifically adopts coordinate-based methods, aiming to create versatile, general-purpose GUI agents.

Benchmarks and Datasets for GUI Agents. Numerous benchmarks and datasets have been proposed to advance research in GUI navigation. Interactive online environments [22, 25, 39, 44, 54, 55, 60, 68] evaluate agents' GUI navigation capabilities, while other datasets [3, 4, 10, 14, 31, 63] primarily enhance UI perception and comprehension. Recent GUI datasets [8, 9, 11, 17, 24, 26, 27, 32, 38, 45, 47, 66] predominantly involve tasks confined to a single app. However, real-world usage frequently requires navigation across multiple apps, significantly increasing complexity. Cross-app tasks typically require longer action sequences (see Table 1), leading to higher error propagation risks. Additionally, cross-app interactions necessitate managing diverse working memory since key UI elements and contextual information span multiple apps. Furthermore, these tasks demand broader functional knowledge to integrate distinct interaction types like file sharing, email composition, and messaging. Additional examples of cross-app tasks are provided in Appendix Sec. 8.4. To address these challenges, we introduce GUIOdyssey, the first comprehensive cross-app GUI navigation dataset. A detailed comparison between GUIOdyssey and prior datasets is presented in Table 1.

3. GUIOdyssey Dataset

This section introduces the proposed cross-app navigation dataset. We present the metadata definition in Sec. 3.1, details in data collection in Sec. 3.2, and dataset statistics in Sec. 3.3, respectively. The dataset overview is shown in Fig. 2 and the collection process is presented in Fig. 3.

3.1. Metadata Definition

GUI Episode. A GUI episode is a recorded sequence of interactions capturing the action steps to complete the navigation task from the user's high-level instruction. Formally, given the user's high-level instruction I_{user} and the screenshot X^t at the time step t, the GUI Agent $\mathcal{G}$ will take the action $A^t = \mathcal{G}(X^t, I_{user})$ to complete this instruction. When the task is completed, the episode is defined as the sequence including all screenshots and actions denoted as $E = \{(X^t, A^t)_{t=1}^T, I_{user}\}$ where T indicates the total steps. An example of the episode is illustrated in Fig. 3. Note that the total step T of cross-app navigation is much larger than that of single-app navigation as shown in Fig. 1.

Action Set. The action set of GUIOdyssey comprises 9 kinds of actions: `CLICK`, `SCROLL`, `LONG PRESS`, `TYPE`, `COMPLETE`, `IMPOSSIBLE`, `HOME`, `BACK`, and `RECENT`. The arguments and functionalities of these actions are summarized in Table 5 of Appendix Sec. 8.2.

3.2. Data Collection

Cross-app Task Proposal. As depicted in Fig. 2, GUIOdyssey comprises six types of cross-app navigation tasks: 1) **General Tool**, which includes tasks that entail system-wide operations. 2) **Information Management**. It encompasses the activities of searching for and recording information for future utilization. 3) **Web Shopping**. Shopping encompass a variety of activities associated with online product purchases. 4) **Media Entertainment**, which revolves around engaging in activities related to video and

Figure 2. An overview of the proposed GUIOdessey. It encompasses 6 types of cross-app navigation tasks spanning 212 unique apps and 1,357 combos of multiple apps from 6 different devices.

Table 1. GUI navigation dataset comparison. GUIOdyssey is a comprehensive cross-app GUI navigation dataset with over 8k+ episodes, featuring an average of 15.3 steps, which is the longest among mobile GUI datasets, and includes a diverse range of devices such as tablets.

Dataset	# Episodes	# Unique I_{user}	# Avg. Steps	Cross-app?	Platform	# Domains	Instruction Level	Semantic Annotation
Mind2Web [17]	2,350	2,350	7.3	✗	Web	137 sites	high	✗
WebLINX [32]	2,337	2,337	43.0	✗	Web	155 sites	high	✗
PixelHelp [27]	187	187	4.2	✗	Phone	4 apps	high & low	✗
MoTIF [8]	4,707	276	4.5	✗	Phone	125 apps	high & low	✗
UGIF [47]	523	480	5.3	✗	Phone	12 apps	high & low	✓
Meta-GUI	4,684	1,125	5.3	✗	Phone	11 apps	high	✗
AITW [66]	715,142	30,378	6.5	✗	Phone	159 apps, 198+ sites	high	✗
AITZ [66]	2,504	2,504	7.5	✗	Phone	70+ apps	high	✓
AndroidControl [26]	15,283	14,548	5.5	✗	Phone	833 apps	high & low	✗
AMEX [9]	2,946	2,946	12.8	✗	Phone	110 apps	high	✓
GUIOdyssey	8,334	8,334	15.3	✓	Phone & Tablet	212 apps, 1,357 app combos	high & low	✓

music streaming applications. 5) **Social Sharing** encompasses activities where users share content across various social media platforms, and 6) **Multi-Apps**, which involve more complex operations across different domains. See Appendix Sec. 8.1 for details on these tasks.

High-Level Task Instruction. For all aforementioned cross-app tasks, we propose a flexible high-level instruction template to construct diverse GUI episodes. The instruction templates are generated by i) human participants and ii) prompting GPT-4 with task descriptions. Ultimately, we collect 91 high-level instruction templates. The diversity of instructions is implemented in three ways. First, the item in each template can be replaced with various candidates. For instance, the item in the instruction "Listen to a podcast episode on {item: yoga} for beginners and create a to-do list" can be substituted with "meditation" or "digital marketing" as shown in Fig. 3. Second, the apps used to complete the instruction can be selected from a predefined pool. For example, the podcast app can be Spotify or Google Podcast and the scheduling app can be Todoist or Microsoft To Do. Finally, we employ GPT-4 to rewrite the instruction using candidate items and apps with different expressions.

Human Demonstration. With diverse high-level instructions collected, we then engage independent annotators, experienced in using mobile devices and various apps, participate in the annotation of GUI episodes. As mentioned in Sec. 3.1, we use an Android emulator to record GUI episodes on various mobile devices such as Pixel Pro, Tablet, and Fold as shown in Fig. 2. All annotators are required to complete the instructions step-by-step and avoid clicking on anything unrelated to the task while recording their interactions. To improve data quality, annotators are trained to annotate at least twenty episodes before starting annotation. During annotation, annotators are asked to save the screenshot before each action step. As shown in Table 5 of Appendix Sec. 8.2, we use the actions `IMPOSSIBLE` and `COMPLETE` to denote the instructions that cannot be completed and those that have been completed, respectively. Specifically, when annotators select `IMPOSSIBLE`, they are required to record the reason why the task could not be completed. Upon completion of the navigation, our data annotation tools save the episode, including the user's instructions, screenshots, actions taken at each step, the apps used by the annotator, and any additional notes. An example of the annotation process is illustrated in Fig. 3.

Fine-grained Episode Annotation. After collecting human-demonstrated GUI episodes, we utilize the state-of-the-art model GPT-4o to generate fine-grained episode annotations, consisting of two main components. The first component is the **Low-Level Instruction**, which refers to a set of fine-grained instructions that serve as atomic decompositions of high-level instructions, providing detailed steps for executing the next action on the current page. The second component is **Semantic Annotation**, which includes: (1) **Screen Description**, offering a detailed depiction of the content displayed in the screenshot; (2) **Contextual Information**, summarizing the preceding steps that led to the current stage of the task; and (3) **Decision Rationale**, ex-

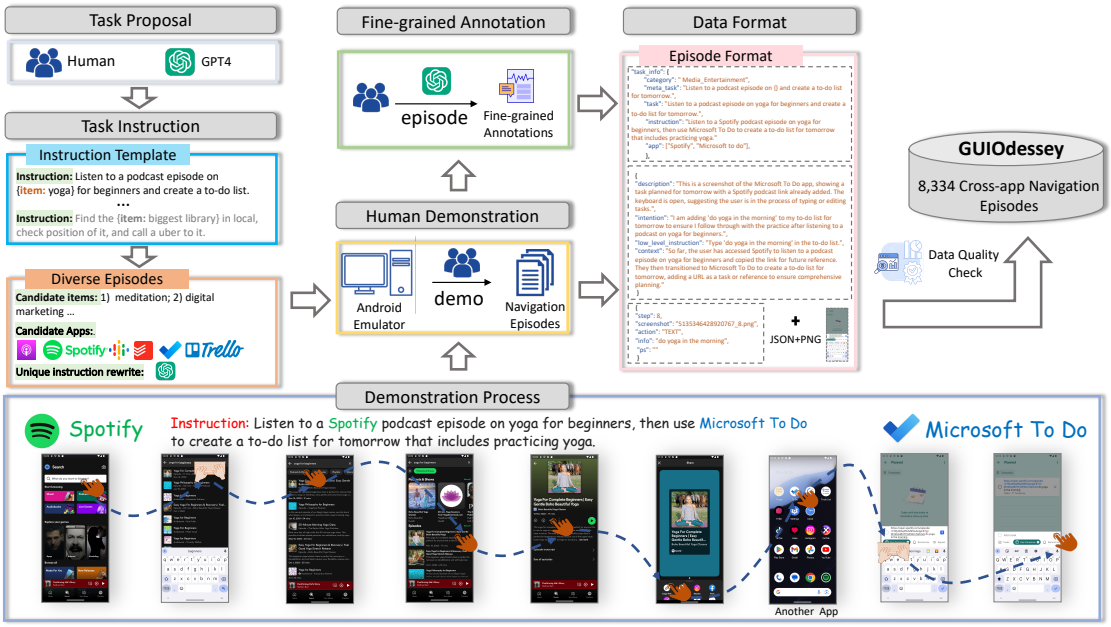


Figure 3. Data collection pipeline of GUIOdyssey for cross-app GUI navigation. GUIOdyssey comprises six categories of navigation tasks. For each category, we construct instruction templates with items and apps selected from a predefined pool, resulting in a vast array of unique instructions for annotating GUI episodes. Human demonstrations on an Android emulator capture the annotation of each episode in a comprehensive format. After rigorous quality checks, GUIOdyssey includes 8,334 validated cross-app GUI navigation episodes.

plaining the reasoning behind the next action based on both historical context and the current screen content. Further details can be found in Appendix Sec. 8.3, while an example of the semantic annotation process is illustrated in Fig. 3.

Data Quality Check. With all episodes collected, we perform a data quality check. The episode is thought to be accurate and complete if it satisfies the following three criteria: i) whether any screenshot in the episode is regularly saved; ii) whether the sequence of screenshots and actions can complete the instruction; iii) whether the instruction rewritten by GPT-4 is equivalent to the original one. After filtering low-quality data, we obtain our cross-app navigation dataset called GUIOdyssey.

3.3. Dataset Statistics

GUIOdyssey targets cross-app navigation, a more practical scenario than single-app navigation in real-world settings. It comprises 8,334 episodes with an average of 15.3 steps per episode, making it the mobile GUI navigation dataset with the longest average episode length. Compared to existing datasets, GUIOdyssey encompasses a broader range of navigation tasks and more complex workflows, featuring six types of cross-app tasks that span 212 apps across domains such as video, music, and reading. It also includes six types of electronic devices, including foldable phones and tablets, which were not covered in previous datasets. Visual statistics are presented in Fig. 4, where Fig. 4 (c) highlights its significantly longer episode lengths compared to single-app datasets[26, 38]. Other provide additional insights into app combination and usage frequency (Fig. 4 a, b), episode length distribution across task types (Fig. 4 d), the presence of 25 app categories (Fig. 4 e), and the diversity of device types (Fig. 4 f).

4. Method: OdysseyAgent

Building upon GUIOdyssey, we introduce OdysseyAgent, an exploratory framework for cross-app navigation tasks powered by Large Vision-Language Models (LVLMs). A key challenge in cross-app tasks is balancing the need to process numerous historical screenshots and lengthy action sequences with the requirement for fast inference in frequent user interactions. To address these demands, we fine-tune Qwen-VL [5] on GUIOdyssey, incorporating a history replay module to optimize both performance and efficiency.

As illustrated in Fig. 5, OdysseyAgent inherits from Qwen-VL-Chat [5] and comprises a vision encoder, a large language model (LLM), and a vision-language (VL) adapter. Crucially, we introduce a history resampler to compress historical screenshot tokens before they reach the LLM. This design alleviates the overhead of stacking all past screenshots while still leveraging essential contextual information. In Appendix Sec. 10.1, we compare the history resampler with a straightforward multi-image concatenation approach, demonstrating that the history resampler achieves

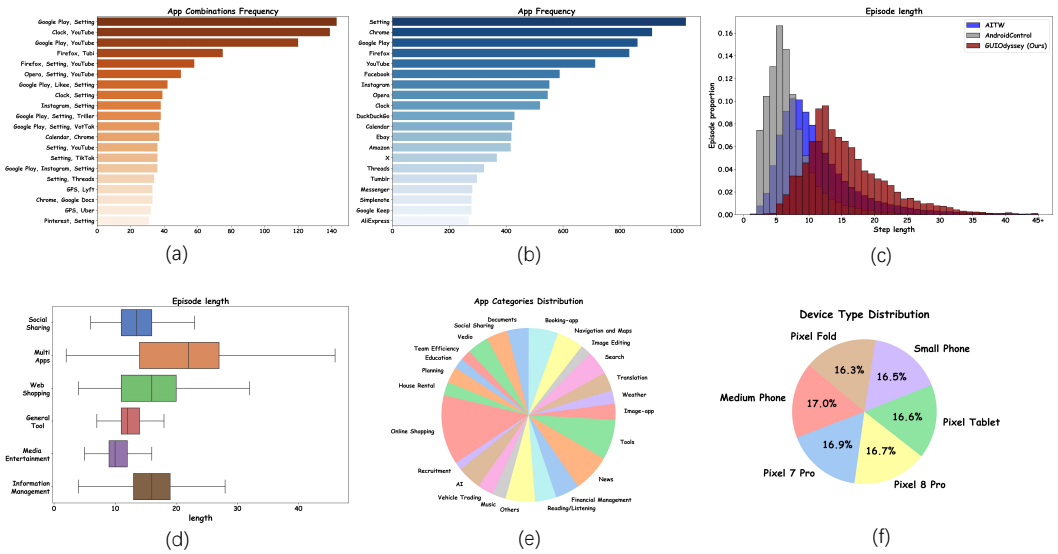


Figure 4. Statistics for GUIOdyssey, zoom in to view details. (a) App Combinations Frequency. (b) App Frequency. (c) Episode length of AITW, AndroidControl and GUIOdyssey. (d) Episode length distribution. (e) App categories distribution. (f) Device statistics.

a more favorable balance between performance and inference efficiency.

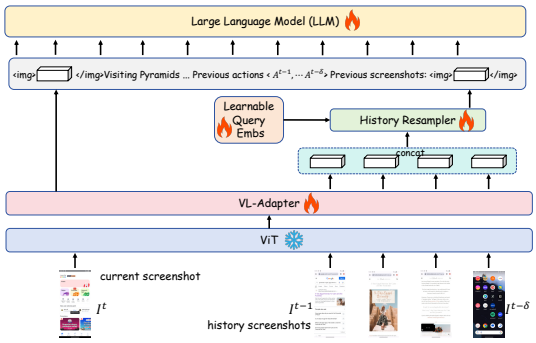


Figure 5. The architecture of OdysseyAgent. Beyond Qwen-VL's standard components, OdysseyAgent introduces a history resampler that enables efficient attention to historical screenshots.

Specifically, the history resampler is implemented as a single-layer cross-attention module, where learnable embeddings serve as the query and historical screenshot tokens function as both key and value. After resampling, the compressed historical screenshot tokens are concatenated with the current screen image token, user instruction, and previous actions. This fused representation is then fed into the LLM to predict the next action. Formally, the next-word prediction objective $\mathcal{L}$ is defined as: $\mathcal{L} = \sum_{i=1}^{N} P_\theta(A_i^t | X^{\{t,t-1,\cdots,t-\delta\}}, I_{user}, A_{<i}^t)$, where N is the number of tokens in action A^t, δ denotes the historical image window, and θ represents the trainable parameters in OdysseyAgent (namely the VL adapter, history resampler, and LLM as shown in Fig. 5).

5. Experiment

The experimental setup is detailed in Sec. 5.1. In Sec. 5.2, we evaluate OdysseyAgent's performance under both in- and out-of-domain settings. Sec. 5.3 further explores the role of historical information in cross-app tasks.

5.1. Experimental Setup

We leverage the comprehensiveness of GUIOdyssey to evaluate OdysseyAgent's performance in both in- and out-of-domain scenarios. To this end, we divide GUIOdyssey into four distinct setups. The first is an in-domain split: (i) **Train-Random & Test-Random**. The remaining three are out-of-domain splits: (ii) **Train-App & Test-App**, (iii) **Train-Task & Test-Task**, and (iv) **Train-Device & Test-Device**. These setups are designed to assess the agent's generalizability across different app, task, and device scenarios. A detailed description of the four setups is provided in Appendix Sec. 9.1, while the training details are available in Sec. 9.2.

Evaluation Metrics. To ensure reproducibility and efficiency, we adopt an offline evaluation method to benchmark performance. We use the **Action Matching Score (AMS)** as our metric, inspired by the approaches presented in AITW [38] and AutoUI [64]. An action is considered correct if its action type matches the ground-truth type. Additionally, for `CLICK` and `LONG PRESS` actions, we consider them correct if they fall within 14% of the screen distance from the reference gesture. Furthermore, we utilize SAM2 [37] to determine the coordinates of the target element, and if the predicted coordinates lie within the region segmented by SAM2, the action is also deemed cor-

Table 2. Results of different LVLMs on Test-Random split. The evaluation metric is the action matching score (AMS). 'HL' and 'LL' indicate that the task instruction is high-level and low-level, respectively. * indicates that agent's training also includes semantic annotations.

Model	Tool		Information		Shopping		Media		Social		Multi-Apps		Overall	
	HL	LL	HL	LL	HL	LL	HL	LL	HL	LL	HL	LL	HL	LL
zero-shot														
GPT-4V	14.93	40.86	14.69	38.56	12.17	36.04	10.80	48.40	16.79	43.21	11.54	40.61	13.49	41.28
GPT-4o	14.15	38.11	13.86	**42.40**	11.69	40.57	12.00	52.40	**18.14**	**44.44**	9.28	38.35	13.19	42.71
Claude3.5-sonnet	**22.99**	40.28	14.69	34.56	12.17	31.03	14.00	37.60	16.79	30.62	14.14	31.00	15.80	34.18
InternVL2-Pro	19.45	**49.51**	15.86	40.23	**17.18**	**41.05**	13.20	**51.60**	14.81	40.00	**15.72**	**41.52**	**16.04**	**43.98**
CogAgent	18.81	33.52	12.35	29.79	13.02	26.89	12.63	25.80	14.72	33.54	12.15	33.09	13.95	30.44
SphAgent	22.24	36.81	**17.43**	29.95	13.60	25.08	**15.54**	33.03	14.20	28.83	12.89	26.32	15.98	30.00
zero-shot with OmniParser														
GPT-4V	24.37	56.03	22.44	51.10	17.16	46.75	18.65	62.18	32.28	59.18	24.21	54.58	23.18	54.97
GPT-4o	26.63	55.28	23.45	53.91	18.34	47.63	19.17	63.21	31.01	61.08	23.39	55.95	23.67	56.18
Claude3.5-sonnet	**39.20**	**64.07**	**28.46**	**61.92**	**27.22**	**56.51**	**28.50**	**66.84**	**40.82**	**68.99**	**33.11**	**65.12**	**32.88**	**63.91**
InternVL2-Pro	16.58	58.04	16.03	51.30	8.88	49.70	16.58	60.62	16.14	53.80	13.95	52.39	14.69	54.31
fine-tuned														
Qwen-VL	85.55	90.79	68.04	83.36	62.28	80.67	77.56	88.15	80.29	**88.36**	74.27	86.56	74.67	86.32
Qwen-VL*	86.35	90.99	**72.01**	85.77	67.31	82.86	80.33	89.48	82.39	88.01	77.52	**89.40**	77.65	87.78
OdysseyAgent	86.01	91.21	69.83	83.37	65.19	82.63	77.10	88.55	81.47	87.66	75.13	87.84	75.79	86.88
OdysseyAgent*	**86.82**	**91.25**	71.79	**86.58**	**68.58**	**83.74**	**80.93**	**89.66**	**82.88**	88.27	**78.47**	89.39	**78.24**	**88.15**

Table 3. OdysseyAgent's performance on out-of-domain tasks. 'Semantic?' indicates whether the model's training includes semantic annotations. 'HL' and 'LL' indicate that the task instruction is high-level and low-level, respectively.

Semantic?	Task Level	Test-Task		Test-Device		Test-App		Overall	
		AMS	SR	AMS	SR	AMS	SR	AMS	SR
✗	HL	54.36	0.09	61.20	1.88	63.03	7.70	59.53	3.22
	LL	78.97	2.20	79.66	8.47	84.24	20.70	80.96	10.46
✓	HL	56.19	0.26	66.63	5.07	65.89	8.81	62.90	4.71
	LL	80.19	2.29	79.93	11.66	83.47	20.02	81.20	11.32

rect. As for SCROLL actions, we compare whether the direction (*i.e.*, up, down, left, or right) matches the gold gesture's direction. For TYPE actions, we evaluate the Average Normalized Levenshtein Similarity (ANLS) [7] between the predicted and gold gestures. If the ANLS is below a certain threshold (set to 0.5 in our experiments), we consider it correct. We then calculate **Success Rate (SR)** for the whole episode. A task is considered successful only if all actions are correct. Success Rate (SR) is a rigorous metric. It would be harder to achieve higher SR in tasks with more action steps.

5.2. Comprehensive evaluation on the GUIOdyssey

We evaluate OdysseyAgent's performance in both in-domain and out-of-domain scenarios. For each step in the dataset, we construct prompts using high-level and low-level instructions separately for training and evaluation. High-level instructions reflect the model's capability to handle real-world GUI navigation tasks, while low-level instructions break down each step of the high-level tasks, assessing the model's ability to follow simpler commands. Naturally, high-level instructions are more challenging than low-level instructions.

In-domain Performance. We compare OdysseyAgent against three types of methods on the Test-Random split of GUIOdyssey: (1) LVLMs zero-shot, including closed-source proprietary LVLMs (GPT-4V [1], GPT-4o [36], Claude3.5-Sonnet [2], InternVL2-Pro [13]) and open-source GUI-specific models (SphAgent [9], CogAgent [23]); (2) closed-source LVLMs zero-shot with OmniParser [33]; and (3) fine-tuned LVLMs Qwen-VL[5]. Due to budget constraints, for closed-source models, we sample 200 episodes from the original test set to serve as their evaluation set. Note that Qwen-VL is effectively OdysseyAgent without the history resampler, meaning it does not incorporate historical screenshots. The result is shown in Table 2. InternVL2-Pro achieves the best overall performance among all coordinated-based models. Despite being trained on other GUI navigation datasets, CogAgent and SphAgent exhibit poor performance on GUIOdyssey, which we attribute to a significant domain gap between cross-app and single-app tasks, resulting in substantial performance disparities. Supported by OmniParser's robust GUI grounding, most closed-source LVLMs substantially improve their cross-app performance, with Claude3.5-Sonnet achieving the best results. In addition, OdysseyAgent surpasses the fine-tuned Qwen-VL, indicating that the proposed history resampler module enhances cross-app navigation. After incorporating semantic annotations during training, OdysseyAgent further improves its performance on all cross-app tasks, achieving 78.24 and 88.15 AMS in high-level and low-level instruction tasks, respectively, thereby demonstrating the effectiveness of our dataset.

Out-of-domain Performance. We further assess the OdysseyAgent's generalization capability in unseen scenarios. As shown in Table 3, OdysseyAgent's out-of-domain performance declines by 16.26 and 5.92 for high- and low-level instructions, respectively, compared to in-domain performance without semantic annotations. With semantic an-

Table 4. The impact of different historical components in GUIOdyssey across four splits. High-level instructions are used for both training and evaluation, and performance is measured by AMS and SR.

	Historical Information			Test-Random		Test-Task		Test-Device		Test-App		Overall	
	action	screenshot	context	AMS	SR	AMS	SR	AMS	SR	AMS	SR	AMS	SR
(1)	✗	✗	✗	66.13	1.65	47.62	0.00	54.15	0.72	54.49	3.59	55.60	1.49
(2)	✓	✗	✗	74.67	9.70	55.00	0.00	62.03	2.03	62.06	**8.98**	63.44	5.18
(3)	✗	✓	✗	71.22	6.69	51.69	0.09	59.12	2.24	59.16	7.78	60.30	4.20
(4)	✗	✗	✓	75.25	9.50	57.66	**0.62**	62.35	2.24	63.82	7.87	64.77	5.06
(5)	✓	✓	✗	75.79	9.38	54.36	0.09	61.20	1.88	63.03	7.70	63.60	4.76
(6)	✓	✓	✓	**77.06**	**11.61**	**58.83**	0.18	**65.85**	**5.00**	**65.63**	8.47	**66.84**	**6.32**

notations, these declines become 15.34 and 6.95. This suggests that high-level instructions are more challenging to generalize in cross-app tasks compared to low-level instructions. Furthermore, incorporating semantic annotations during training improves performance in most scenarios, with especially notable gains on high-level instruction tasks, underscoring the value of semantic annotations for unseen domain. Compared to in-domain performance, the performance gap between high- and low-level instructions is even larger in out-of-domain tasks. This implies that the model currently lacks sufficient reasoning and planning capabilities to effectively handle unseen high-level instruction tasks.

5.3. The effect of different historical information.

We now conduct a detailed experiment to deeply explore the role of historical information components. Currently, two main types of historical information are used in GUI agents: historical actions and historical screenshots. Note that the Contextual Information included in the semantic annotations of GUIOdyssey serves as a summary of previous steps, providing a more comprehensive textual representation of historical information. Therefore, we also include it in our experiments. Detailed results are presented in Table 4. Comparing experiments (2)–(4) with the baseline experiment (1), we observe that all three types of historical information significantly improve model performance, with contextual information producing the most substantial enhancement: improving AMS by 9.17 and SR by 240% compared to the baseline. A comparison between experiments (4) and (5) shows that using contextual information alone significantly improves out-of-domain performance compared to employing both actions and screenshots as historical input. This suggests that summarizing and abstracting historical information can better help the model generalize to unseen GUI scenarios. Additionally, experiment (6) shows that incorporating all types of historical information as input further enhances the model performance. We hypothesize that cross-app tasks inherently require more sophisticated memory mechanisms due to the dependencies and interactions between multiple apps. For example, as illustrated in Fig. 6 (Appendix Sec. 8.4), completing a cross-app task—such as identifying properties of triangles from Chrome and subsequently recording them in Google Docs—requires effectively remembering and transferring key information across apps. This example highlights the critical role historical information plays, underscoring the importance of comprehensive historical context modeling for GUI agents in complex cross-app scenarios.

More experiments. To deepen our analysis using GUIOdyssey, we conduct additional experiments detailed in Appendix Sec. 10. These include investigations of various strategies for handling historical screenshots, different semantic annotation components, transferability across devices, different instruction granularities, and the relationship between cross-app and single-app tasks.

6. Conclusion

In this work, we address the limitations of existing GUI navigation agents for cross-app tasks by introducing GUIOdyssey, the first comprehensive cross-app mobile GUI navigation dataset enriched with semantic annotations. Leveraging this dataset, we develop OdysseyAgent, a multimodal cross-app navigation agent equipped with a history resampler module that efficiently processes historical image tokens to balance performance and inference speed. We conduct extensive experiments with OdysseyAgent to evaluate our approach on both in-domain and out-of-domain scenarios. Our results further indicate that richer utilization of historical information can substantially enhance OdysseyAgent's performance. We hope GUIOdyssey and OdysseyAgent can drive the research in the field of general GUI Agents.

Acknowledgments and Disclosure of Funding

We thank Zhouheng Yao, Zihao Zhao for their help in data collection. This paper is partially supported by the National Key R & D Program of China No.2022ZD0160101 & No.2022ZD0161000.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1, 2, 3, 7

[2] Anthropic. Claude, 2023. Accessed: 2023-04-18. 1, 3, 7

[3] Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. Screenai: A vision-language model for ui and infographics understanding. *arXiv preprint arXiv:2402.04615*, 2024. 3

[4] Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, et al. Uibert: Learning generic multimodal representations for ui understanding. *arXiv preprint arXiv:2107.13731*, 2021. 2, 3

[5] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023. 5, 7

[6] Louise Barkhuus and Valerie E Polichar. Empowerment through seamfulness: smart phones in everyday life. *Personal and Ubiquitous Computing*, 15:629–639, 2011. 1

[7] Ali Furkan Biten, Ruben Tito, Andres Mafla, Lluis Gomez, Marçal Rusinol, Ernest Valveny, CV Jawahar, and Dimosthenis Karatzas. Scene text visual question answering. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4291–4301, 2019. 7

[8] 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, 4

[9] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*, 2024. 3, 4, 7

[10] Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Liuyi Chen, Yilin Bai, Zhigang He, Chenlong Wang, Huichi Zhou, Yiqiang Li, et al. Gui-world: A dataset for gui-oriented multimodal llm-based agents. *arXiv preprint arXiv:2406.10819*, 2024. 3

[11] Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, et al. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*, 2024. 3

[12] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023. 1, 3

[13] Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*, 2024. 7

[14] 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. 2, 3

[15] Zhang Chi, 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. 3

[16] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023. 1

[17] 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. 2, 3, 4

[18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1

[19] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023. 3

[20] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024. 3

[21] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023. 3

[22] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. 3

[23] 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. 3, 7

[24] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024. 3

[25] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. 3

[26] Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*, 2024. 1, 3, 4, 5

[27] 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. 3, 4

[28] Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024. 1, 3

[29] Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui 2: Mastering universal user interface understanding across platforms. *arXiv preprint arXiv:2410.18967*, 2024. 3

[30] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024. 3

[31] Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding? *arXiv preprint arXiv:2404.05955*, 2024. 3

[32] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024. 3, 4

[33] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024. 3, 7

[34] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[35] Amal Nanavati, Vinitha Ranganeni, and Maya Cakmak. Physically assistive robots: A systematic review of mobile and manipulator robots that physically assist people with disabilities. *Annual Review of Control, Robotics, and Autonomous Systems*, 7, 2023. 1

[36] OpenAI. Gpt4o, 2024. 2, 7

[37] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 6, 2

[38] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023. 1, 2, 3, 5, 6

[39] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024. 1, 3

[40] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1

[41] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[42] Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. In *Advances in Neural Information Processing Systems*, 2023. 3

[43] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[44] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017. 3

[45] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: towards multi-modal conversational agents on mobile gui. *arXiv preprint arXiv:2205.11029*, 2022. 3

[46] 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. 3

[47] Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. Ugif: Ui grounded instruction following. *arXiv preprint arXiv:2211.07615*, 2022. 3, 4

[48] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024. 1, 3

[49] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024. 3

[50] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. 1, 3

[51] Zhiping Paul Wang, Priyanka Bhandary, Yizhou Wang, and Jason H Moore. Using gpt-4 to write a scientific review arti-

cle: a pilot evaluation study. *bioRxiv*, pages 2024–04, 2024. 1

[52] 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. 3

[53] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024. 3

[54] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. 1, 3

[55] Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. Understanding the weakness of large language model agents within a complex android environment. *arXiv preprint arXiv:2402.06596*, 2024. 1, 3

[56] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023. 3

[57] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023. 3

[58] 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. 3

[59] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023. 3

[60] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022. 3

[61] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 3

[62] Kaining Ying, Fanqing Meng, Jin Wang, Zhiqian Li, Han Lin, Yue Yang, Hao Zhang, Wenbo Zhang, Yuqi Lin, Shuo Liu, et al. Mmt-bench: A comprehensive multimodal benchmark for evaluating large vision-language models towards multitask agi. *arXiv preprint arXiv:2404.16006*, 2024. 1

[63] 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. In *European Conference on Computer Vision*, pages 240–255. Springer, 2025. 3

[64] Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023. 6

[65] 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. 3

[66] 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. *arXiv preprint arXiv:2403.02713*, 2024. 3, 4

[67] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024. 3

[68] 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. 3

[69] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023. 1

GUIOdyssey: A Comprehensive Dataset for Cross-App GUI Navigation on Mobile Devices

Supplementary Material

7. Ethical Discussion

Privacy. We use temporary accounts and virtual usernames to register various apps and ensure no personal information is entered. The dataset contains no authentic personal information.

Ethical Consent in Data Collection. A formal consent process is implemented, wherein participants explicitly agree to the inclusion of their human-annotated data in the dataset. All data are collected with informed consent and in full compliance with ethical guidelines.

Security Concerns. The development of intelligent agents trained on datasets like this offers significant potential for automating tasks and enhancing accessibility. However, it also raises important ethical and security concerns. Sensitive operations, such as financial transactions or privacy management, pose vulnerabilities without robust safeguards. Additionally, malicious actors could exploit these agents to bypass security protocols or manipulate applications for unethical purposes. To mitigate these risks, it is crucial to implement secure model designs, privacy-preserving techniques, and establish clear ethical guidelines. Addressing these challenges will help ensure the responsible deployment of such technology while maximizing its societal benefits.

8. Details of GUIOdyssey

8.1. Description of Task Categories

The specific details of the six task categories are as follows:

General Tool. This category encompasses tasks that involve navigating through system-wide operations such as managing system settings or notifications for apps. An instruction example of a general tool task is "Adjust the notification settings for the YouTube app on your phone using Settings, then proceed to open YouTube".

Information Management. Information management tasks involve searching for information and recording it for future use. This might include looking up information on search engines, reading articles on news apps, checking facts on educational or reference apps, and then saving or organizing this information in note-taking apps.

Web Shopping. Shopping tasks encompass a range of activities related to purchasing products online. Users may start by searching for a product on one app, comparing prices on different e-commerce platforms, checking reviews and ratings on review apps or websites, and finally making a purchase.

Media Entertainment. Media entertainment tasks are about activities involving video and music streaming apps. Users may browse for new content on video platforms like YouTube or Netflix, stream music on services like Spotify or Apple Music, and switch between different media apps to manage playlists or download content.

Social Sharing. This task involves activities where users share content across different social media platforms. This could include taking photos or videos with the camera app, editing them using a photo or video editing app, and then sharing them on multiple social media platforms like Instagram, Facebook, Twitter, or TikTok.

Multi-Apps. Multiple-app tasks involve more complex operations that require three or more apps to complete. For example, cooking food with an online recipe might involve finding the recipe of the food, recording the recipe to a note-taking app, and buying the ingredients online(Fig. 1).

8.2. Action Set

Our recording system utilizes Android Studio to simulate GUI navigation and virtualize various devices. We use the Android Debug Bridge (ADB) to retrieve device information and status, such as the coordinates of click events, and to monitor a wide range of functional keys. The details of the action set in our Android emulator are presented in Table 5.

8.3. Fine-grained Episode Annotation Generation

Fine-grained episode annotations consist of two components: low-level instructions and semantic annotations. Examples of the fine-grained annotations can be found in Fig. 7.

Low-Level Instruction. For each step within an episode, we provide GPT-4o with the high-level instruction corresponding to the episode, along with the action and screenshot associated with the current step. Additionally, for actions such as `CLICK` and `LONG PRESS`, we supply an additional image featuring a bounding box to indicate the click coordinates. All images are configured with the fidelity parameter set to 'high'. The prompt utilized is provided in Fig. 11.

Semantic Annotation. We use GPT-4o to generate semantic annotations in an alternating and iterative manner, following the sequential order of steps within each episode. Specifically, the process begins by providing the current episode's high-level instruction along with the actions and decision rationale from previous steps, prompting GPT-4o

Table 5. The argument and functionality of different actions in GUIOdyssey. 'pos1' and 'pos2' denote the position (x, y).

Action	Argument	Functionality
CLICK	[pos1]	click the on-screen position
LONG PRESS	[pos1]	press the screen for a long time to copy texts or download images
SCROLL	[pos1, pos2]	scroll the screen from position 1 to position 2
TYPE	text	type text with keyboard
COMPLETE	-	the sign that the instruction has been completed
IMPOSSIBLE	-	the sign that the instruction cannot be completed
HOME	-	go to the home screen
BACK	-	go to the previous screen
RECENT	-	go to the previous App

to generate the contextual information for the current step. Subsequently, using the generated contextual information, the high-level instruction, the screenshot image, and the action corresponding to the current step, GPT-4o is prompted step-by-step to generate the screen description and decision rationale for the current step. This iterative process continues until all semantic annotations for each step within the episode are completed in sequence. Similarly, for actions such as CLICK and LONG PRESS, we supply an additional image with a bounding box indicating the click coordinates. All images are configured with the fidelity parameter set to 'high' to ensure precision. The prompts used for generating these annotations are provided in Fig. 12 and Fig. 13.

8.4. Examples

An example of episodes in our GUIOdyssey is shown in Fig. 6, while examples of semantic annotations can be found in Fig. 7. An example of an annotation for a task that could not be successfully completed and ends with the IMPOSSIBLE action can be found in Fig. 8 and Fig. 9.

As mentioned in Sec. 5.1, we use SAM2 [37] to assist in evaluating whether the model's output actions are correct. Fig. 10 provides examples of bounding boxes for clicked elements obtained through SAM2 segmentation.

8.5. Data Format

Each field of annotation is as follows.

episode_id: the unique identifier of this episode.

device_info: the detailed information of the virtual device from which the episode was collected, including the device model, screen resolution, and other device-related details.

task_info: the detailed information of the task from which the episode was collected, including the task category, the app used, the high-level instruction, and other task-related details.

step_length: the total number of steps in this episode.

steps: a list of steps in this episode. Each step in the list includes the file path of the screenshot, executed action and its corresponding parameters (*e.g.*, the coordinates for a click action), the low-level instruction, the semantic annotation, the bounding box obtained from SAM2 segmentation, and additional recorded information such as the overall scroll trajectory for scroll actions and annotator notes.

9. Experiment Details

9.1. Detailed description of four different setups.

The following details the four different setups in GUIOdyssey.

i) Train-Random & Test-Random. We randomly partitioned all the episodes in the dataset into training and testing sets using a ratio of 80% to 20% as the standard approach to divide the dataset. It can assess the in-domain performance of OdysseyAgent.

ii) Train-Task & Test-Task. In this setup, We proportionally sampled meta-tasks from six categories, maintaining approximately a $6:1$ ratio for the training and test sets. The tasks in the test set differ significantly from those in the training set. This partitioning method allows for a robust assessment of an agent's generalization capabilities across diverse tasks.

iii) Train-Device & Test-Device. To evaluate an agent's generalizability across different and unseen devices, we selected episodes annotated on the Tablet, which differs significantly from other devices, as the test set. We obtained $1,381$ episodes as the test set and $6,953$ episodes as the training set.

iv) Train-App & Test-App. This split is aimed at evaluating the agent's performance on unseen Apps and App combinations. First, we calculated the frequency of app usage in the dataset and categorized the apps into 25 classes

Table 6. The impact of different semantic annotations on OdysseyAgent across four different splits. We use high-level instructions for both training and evaluation. Performance is assessed using AMS and SR as metrics. SD, CI, and DR denote screen description, contextual information, and decision rationale, respectively.

	Semantic Annotation			Test-Random		Test-Task		Test-Device		Test-App		Overall	
	SD	CI	DR	AMS	SR	AMS	SR	AMS	SR	AMS	SR	AMS	SR
(1)	✗	✗	✗	75.79	9.38	54.36	0.09	61.20	1.88	63.03	7.70	63.60	4.76
(2)	✓	✗	✗	75.18	8.94	54.06	0.00	64.41	2.03	64.91	8.47	64.64	4.86
(3)	✗	✓	✗	75.42	10.04	55.71	0.00	62.52	3.19	64.24	5.30	64.47	4.63
(4)	✗	✗	✓	77.71	11.44	55.60	**0.26**	65.88	4.63	65.74	7.96	66.23	6.07
(5)	✗	✓	✓	77.23	11.16	56.93	0.18	63.87	2.24	66.32	7.87	66.09	5.36
(6)	✓	✗	✓	77.24	10.88	**57.15**	0.00	63.55	2.17	**67.04**	**9.67**	66.24	5.68
(7)	✓	✓	✗	76.58	10.14	57.13	**0.26**	64.48	3.91	66.27	7.96	66.11	5.57
(8)	✓	✓	✓	**78.24**	**11.62**	56.19	**0.26**	**66.63**	**5.07**	65.89	8.81	**66.74**	**6.44**

(*e.g.*, Video, Music) based on their characteristics. Then, we selected a few apps with the lowest occurrence from each class to form the test app set. Subsequently, we partitioned the episodes that utilized the app in the test app set into the Test-App set, maintaining an approximately 85% to 15% ratio between the training set and the test set.

9.2. Training Details.

To train OdysseyAgent, we employ the AdamW optimizer with a learning rate of $2e{-}5$ and utilize a cosine learning rate schedule. We set β_1 and β_2 to 0.9 and 0.95, respectively, and use a weight decay of 0.1. Additionally, we utilize a global batch size of 128 and implement DeepSpeed ZERO2-style data parallelism. During training, OdysseyAgent treats each action step as an individual training sample. The input consists of the task instruction, the current screenshot, and the previous 4 actions and screenshots (*i.e.*, $\delta = 4$), while the output corresponds to the action for the current step. By default, OdysseyAgent is trained separately on Train-Random/Task/Device/App for one epoch, excluding the semantic annotation component. When training includes semantic annotations, these annotations are converted into single-turn QA pairs, which serve as additional training samples (*i.e.*, semantic annotations are introduced only during training-time). Any training configuration that incorporates semantic annotations is explicitly noted. The entire training process requires approximately 32 A100 hours to complete.

9.3. Prompt for Evaluation.

We utilize the prompt shown in Fig. 14 to evaluate the performance of GPT-4V, GPT-4o, Claude3.5-sonnet, and InternVL2-Pro. For SphAgent and CogAgent, we tested them following their officially recommended methods [9, 23].

10. More Experiments

10.1. History Resampler vs. Multi-Image Training.

We evaluate different approaches for processing historical screenshot images. Qwen-VL supports multi-image input by interleaving image and text tokens, but this leads to a high token overhead (*e.g.*, 1024 tokens for four historical steps). Our history resampler compresses this to 256 tokens, greatly improving efficiency. As shown in Table 7, both approaches achieve comparable performance, but the history resampler significantly enhances training and inference efficiency.

Table 7. The average AMS for HL and LL instructions across 4 splits, along with the number of historical screenshot tokens, inference metrics (Time to First Token (TTFT) and Tokens per Second (TPS)), and training GPU hours.

strategy	HL	LL	Token Count	TTFT ↓	TPS ↑	GPU Hours
history resampler	63.60	82.44	256	0.71	20.27	32
multi-image	65.04	82.34	1024	0.98	17.05	48

10.2. The effect of different semantic annotations.

We assess the impact of different semantic annotations in GUIOdyssey (*i.e.*, screen description, contextual information and decision rationale) on model performance in both in-domain and out-of-domain settings. The results are presented in Table 6. A comparison of experiments (1)–(4) shows that all three components contribute positively, but engaging in detailed reasoning before making decisions is more important than understanding current screen information or summarizing historical processes in cross-app tasks. Experiments (5)–(8) further indicate that using two or more types of semantic annotations generally outperforms using a single annotation type. Specifically, using all semantic annotations yields the best results and improves AMS by 3.14

and SR by 35% compared to training without any semantic annotations. These findings suggest that teaching the model to understand the reasoning behind each action—similar to how humans observe, understand, review completed steps, and reason thoroughly before deciding—can be beneficial for improving performance in both in-domain and out-of-domain cross-app tasks.

10.3. Transferability of instructions at different levels of granularity.

As shown in Table 8, models trained on high-level instructions exhibit significantly better transferability across different levels of instruction granularity compared to those trained on low-level instructions. Furthermore, training on both instruction granularities outperforms training on a single granularity, a phenomenon similar to what has been observed in single-app tasks [26].

10.4. Transferability across different devices.

We utilize our GUIOdyssey dataset to conduct additional experiments to evaluate the generalization capabilities of OdysseyAgent beyond the initial experimental setup. we test the OdysseyAgent's adaptability by using data from one device as the test set while training on data from the remaining five devices. The results of these experiments are presented in the Table 9, demonstrating the model's performance across different devices. The model exhibits the weakest transferability on tablet devices, which we attribute to the significant differences in user interface layouts between tablets and smartphones. Furthermore, the model's transferability on small phones and foldable devices is also suboptimal. We surmise that the disparity in screen resolution compared to other phone models may contribute to this underperformance.

Table 8. The results for OdysseyAgent trained and tested on Train-Random/Test-Random with both high-level and low-level instructions are presented, with AMS as the evaluation metric. HL and LL denote high-level and low-level instructions, respectively.

Testing Instructions	**Training Instructions**		
	HL	LL	HL + LL
HL	75.79	29.39	78.96
LL	71.26	86.88	88.84

10.5. Whether cross-App tasks benefit single-App tasks.

We further investigate whether cross-app tasks benefit single-app performance by evaluating the impact of different training data compositions under controlled conditions. Specifically, we randomly sample 50k training samples each from GUIOdyssey, AITW, and AndroidControl (denoted as Ody50k, AITW50k, and AC50k, respectively) and evaluate their performance on AndroidControl, which provides both in-domain and out-of-domain scenarios. As shown in Table 10, we find that incorporating cross-app data from GUIOdyssey consistently enhances performance in most single-app scenarios, whereas adding AITW data surprisingly yields limited improvements or even performance degradation. This suggests that the more complex cross-app tasks in GUIOdyssey can benefit single-app tasks.

Table 9. Performance Evaluation of OdysseyAgent Across Different Devices. Each Device serves as a test set while the remaining five devices are used as training sets.

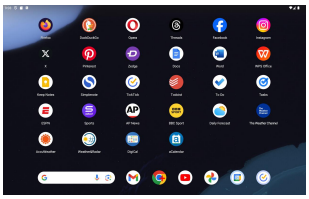
Evaluation Device	**Resolution**	**AMS**	**SR**
Pixel 7 Pro	$1,440 \times 3,120$	75.91	7.44
Pixel 8 Pro	$1,344 \times 2,992$	74.67	6.05
Small Phone	$720 \times 1,280$	71.68	3.77
Medium Phone	$1,080 \times 2,400$	73.05	5.45
Pixel Fold	$2,208 \times 1,840$	67.67	4.48
Pixel Tablet	$2,560 \times 1,600$	61.20	1.88

Table 10. Effectiveness of Different Training Data on the AndroidControl. The evaluation metrics are the action matching score (AMS).

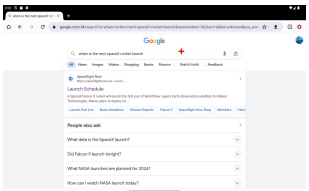
Training Data	IDD	category_unseen	app_unseen	task_unseen	Overall
AC50k	60.43	54.46	50.00	72.10	59.25
AC50k + AITW50k	60.69	**55.26**	45.19	68.84	57.50
AC50k + Ody50k	**61.48**	54.61	**50.96**	**72.46**	**59.88**

instruction: Utilize Chrome to research the key property of a Triangle and compile your findings into a concise document using Google Docs.

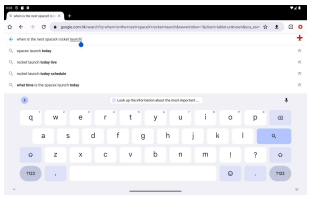
device_name: Pixel Tablet
category: Information_Management
app: ["Chrome", "Google Docs"]
step_length: 16



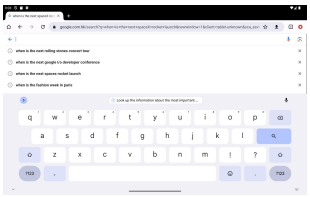
"Open the Chrome browser."
CLICK: [525, 912]



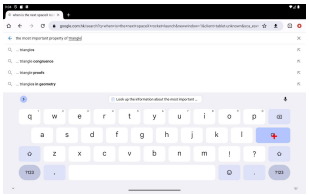
"Tap the search bar."
CLICK: [589, 244]



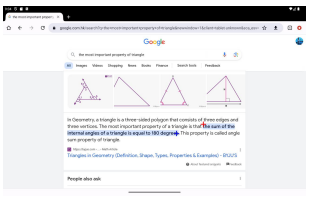
"Clear the current search query."
CLICK: [976, 172]



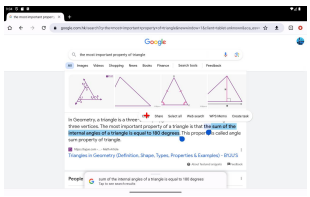
"Type 'the most important property of triangle' in the search bar."
TEXT: the most important property of triangle



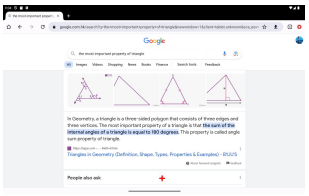
"Initiate the search query."
CLICK: [895, 696]



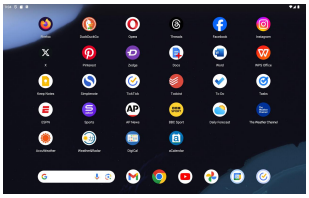
"Scroll to copy the related results."
SCROLL: [661, 622], [573, 671]



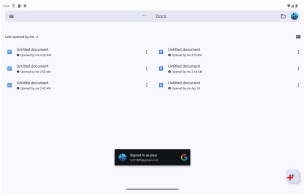
"Copy the highlighted text."
CLICK: [471, 576], [471, 576]



"Return to the home screen."
CLICK: HOME

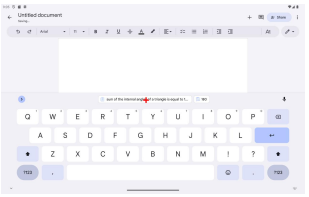


"Open the Google Docs app."
CLICK: [580, 272]

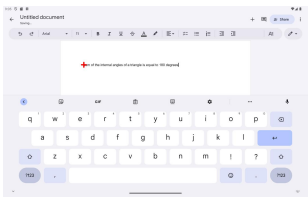


"Create a new document in Google Docs."
CLICK: [954, 919]

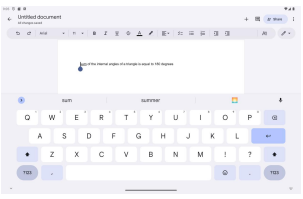
"Create a new document."
CLICK: [967, 826]



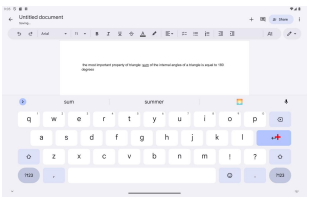
"Select the copied text from the keyboard."
CLICK: [477, 506]



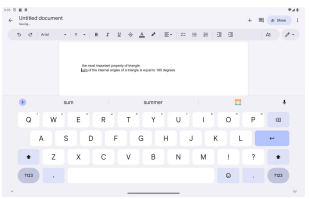
"Select the text at the beginning of the document."
CLICK: [265, 308]



"Type 'the most important property of a triangle:' into the document."
TEXT: the most important property of triangle:



"Press the Enter key."
CLICK: [904, 686]

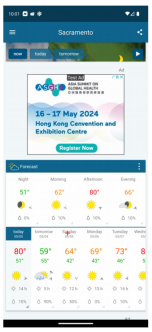


"Task completed."
COMPLETE

Figure 6. An example of episodes in our GUIOdyssey.

High-Level Instruction: Use Chrome to search for a new hiking trail, then check the weekend weather forecast using Weather & Radar. Finally, invite katsunaksu to join the adventure via Tumblr.

step: 13



CLICK: [444, 707]

low_level_instruction: Select the weather forecast for Sunday.

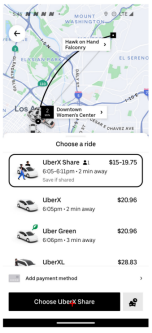
description: This is a screenshot of a weather forecast application displaying the weather for Sacramento. It shows detailed weather information for different parts of the day and upcoming days.

intention: I am selecting the weather forecast for Sunday to verify the conditions for the planned hiking trip. This action allows me to ensure the weather will be suitable for outdoor activities on that day, aiding in the planning process.

Context: So far, a search for new hiking trails has been conducted using the Chrome browser, and detailed information about popular trails in California has been gathered. The Weather & Radar app was then used to check the weekend weather forecast for Sacramento, CA, providing necessary weather insights for planning the hiking adventure.

High-Level Instruction: Using Waze Navigation & Live Traffic, locate a nearby zoo, then request a ride using the Uber app.

step: 10



CLICK: [476, 927]

low_level_instruction: Select the 'Choose UberX Share' button.

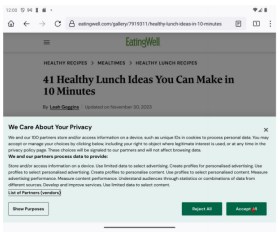
description: This is a screenshot of the Uber app showing ride options to a destination labeled 'Hawk on Hand Falconry.' The user is presented with various ride choices including UberX Share, UberX, Uber Green, and UberXL, along with their estimated arrival times and prices.

intention: I am selecting the 'Choose UberX Share' option to confirm my ride to 'Hawk on Hand Falconry.' This action is appropriate because it finalizes my transportation arrangement, ensuring a cost-effective and timely arrival at my destination.

Context: So far, the task has involved using the Waze app to locate a nearby zoo by searching for it and confirming the search results. After identifying the desired zoo location, the Uber app was accessed to request a ride, specifying 'Hawk On Hand Falconry' as the destination to ensure a seamless transportation arrangement.

High-Level Instruction: Choose a nutritious lunch plan for tomorrow, then jot it down in Google Keep. After that, use Firefox to find and watch a Tiktok video on how to prepare one of the dishes.

step: 8



CLICK: [916, 901]

low_level_instruction: Accept the privacy terms to continue browsing the webpage.

description: This is a screenshot of a webpage from EatingWell displaying an article titled '41 Healthy Lunch Ideas You Can Make in 10 Minutes.' A privacy notice is visible, offering options to accept or reject data processing terms.

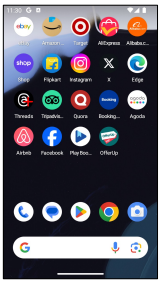
intention: I am choosing to click 'Accept All' to quickly dismiss the privacy notice and proceed with viewing the article. This action allows me to focus on selecting a nutritious lunch plan without interruptions, which is essential for completing my task efficiently.

Context: So far, the task has involved using the Firefox browser to search for a TikTok video on how to prepare a nutritious meal as part of a healthy lunch plan. The search included typing a query, initiating the search, and scrolling through the results to find relevant content. A link titled '41 Healthy Lunch Ideas You Can Make in 10 Minutes' was clicked to explore specific meal options that are both nutritious and quick to prepare.

Figure 7. Examples of fine-grained annotations in GUIOdyssey.

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 1



low_level_instruction: Open the Threads app.

description: This is a screenshot of an Android home screen displaying various app icons, including Threads, Booking.com, and Airbnb.

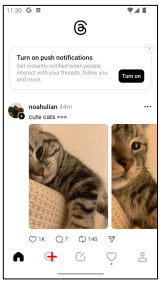
intention: I am selecting the Threads app to gather ideas and recommendations for my travel itinerary to New York City. This step is crucial for accessing social media content that might offer insights or suggestions for the trip.

Context: Task just started, nothing has been done.

CLICK: [162, 353]

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 2



low_level_instruction: Open the search section.

description: This is a screenshot of a social media app showing a post featuring pictures of a cute cat. There is also a prompt to turn on push notifications at the top of the screen.

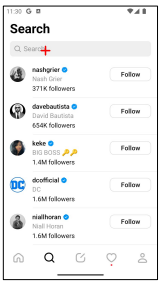
intention: I am choosing to click on the search icon to explore more content related to travel ideas and recommendations for New York City. This action will help me gather information and inspiration for planning my travel itinerary.

Context: So far, the user has navigated to the main screen of their phone and accessed the Threads app.

CLICK: [162, 353]

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 3



low_level_instruction: Tap on the search bar.

description: This is a screenshot of the search page in the Threads app, displaying a list of verified profiles with follower counts and follow buttons.

intention: I am selecting the search bar to input keywords related to New York City travel ideas. This action will help me find relevant content and recommendations to enhance my travel itinerary.

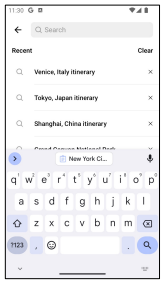
Context: So far, the user has navigated to the home screen of their phone and accessed the Threads app. They have also utilized the search function within the app to explore content related to travel ideas and inspiration for their trip.

CLICK: [274, 163]

Figure 8. Example of an annotation for an unsuccessful task, ending with the IMPOSSIBLE action.

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 4



TYPE: New York City, USA itinerary

low_level_instruction: Enter 'New York City, USA itinerary' in the search bar.

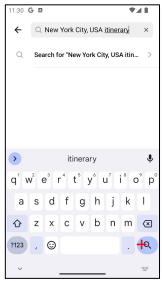
description: This is a screenshot of a search interface on a mobile app showing recent searches for travel itineraries such as Venice, Italy and Tokyo, Japan. The user is currently typing 'New York City, USA itinerary' in the search bar.

intention: I am entering 'New York City, USA itinerary' in the search bar to gather ideas and recommendations for planning my trip. This action is appropriate as it allows me to find relevant information and suggestions to enhance my travel itinerary for New York City.

Context: So far, the user has navigated to the home screen of their mobile phone to efficiently manage travel planning tasks. They have accessed the Threads app and selected the search bar.

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 5



CLICK: [162, 353]

low_level_instruction: Initiate the search query.

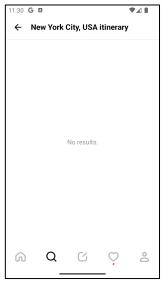
description: This is a screenshot of a mobile search interface where the user has entered 'New York City, USA itinerary' into the search bar. The keyboard is visible, and there is an option to search for the entered query.

intention: I am choosing to tap the search button to initiate a search for 'New York City, USA itinerary' within the Threads app. This action will help me find relevant content and recommendations to enhance my travel planning for the city. By executing this search, I expect to gather useful ideas and suggestions for my itinerary.

Context: So far, the user has navigated to the home screen of their mobile phone and accessed the Threads app to gather ideas for a travel itinerary to New York City. They have utilized the search feature within the app to input relevant keywords, aiming to find content and recommendations to enhance their travel planning for the city.

High-Level Instruction: Utilizing Threads and Booking.com, devise a travel itinerary for a trip to New York City, USA, and secure your lodging arrangements.

step: 6



IMPOSSIBLE

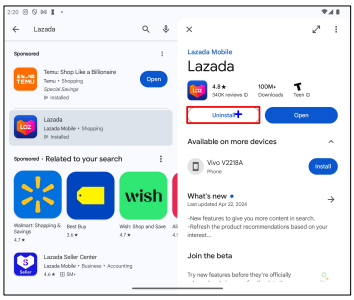
low_level_instruction: Task could not be completed.

description: This is a screenshot of a search result page in the Threads app displaying the message 'No results' for the query 'New York City, USA itinerary'. The screen shows that no content was found matching the search criteria.

intention: The task could not be completed because the search for 'New York City, USA itinerary' in the Threads app yielded no results. This lack of content prevented the gathering of ideas and recommendations necessary for planning the travel itinerary.

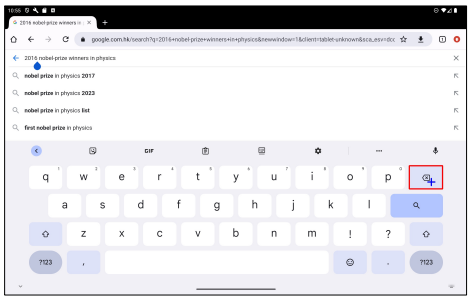
Context: So far, the user has navigated to the Threads app on their mobile phone to gather travel ideas and recommendations for a trip to New York City. They have utilized the search function within the app to find relevant content by entering 'New York City, USA itinerary' as the search term.

Figure 9. Example of an annotation for an unsuccessful task, ending with the IMPOSSIBLE action.



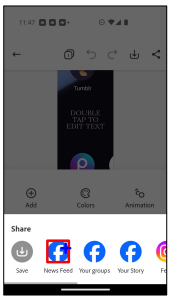
CLICK: [675, 374]

SAM2_BBOX: [528, 349, 736, 405]



CLICK: [923, 609]

SAM2_BBOX: [875, 553, 947, 633]



CLICK: [379, 838]

SAM2_BBOX: [266, 812, 398, 886]

Figure 10. Examples of bounding boxes for UI elements segmented by SAM2. The actual click locations are indicated by blue '+' symbols, while the red rectangles outline the bounding boxes obtained from the SAM2.

<image>current_screenshot.png</image>
<image>current_screenshot_w_labels.png</image>
Based on the original and marked screenshots of an Android mobile phone, where the marked screenshot is the original screenshot marked with action location, please follow the instructions below:

Low-Level Instruction Identification
- Identify the low-level instruction that the current action represents, such as "Go to the alarm section."

P.S.:
- When following these instructions, use natural language, avoid mentioning technical details (such as action coordinates) or direct use of action tags (such as "PRESS_HOME").

Output Format:
The output should be in JSON format as follows:
{{"instruction": "low-level instruction that the current action represents in the desired format"}}

Please return the result in pure JSON format, without any json tags like ```json ```.

Figure 11. Prompts for generating low-level instruction.

You are completing the task: {task} on a mobile phone, and the actions you have performed along with their respective intentions are listed in chronological order as:
{intentions}

Please follow the instructions below:
- Summarize the completed progress of the task based on the actions performed and their intentions in 2-3 sentences.
- Ensure your summary is purely focused on completed actions, and avoid provide further insight.
- Create a logically connected summary, rather than simply listing each action in detail.
- Avoid using any time-sequence phrases, such as 'after completing,' 'upon finishing,' or similar expressions.
- Use a completed-action tone, describing the progress as if each step has already occurred.
- Use an objective tone and describe concisely from an impersonal perspective.
- Format the context as follows: "So far, [summary of what has been accomplished]."

P.S.:
- When following these instructions, use natural language, avoid mentioning technical details (such as action coordinates) or direct use of action tags (such as "PRESS_HOME").

Output Format:
The output should be in JSON format as follows:
{{
"context": "a 2-3 sentence summary of task progress up to this point in the desired format"
}}
Please return the result in pure JSON format, without any json tags like ```json ```.

Figure 12. Prompts for generating contextual information.

<img>current_screenshot.png</img>
<img>current_screenshot_w_labels.png</img>
Based on the original and marked screenshots of an Android mobile phone, where the marked screenshot is the original screenshot marked with action location, please follow the instructions below:

1. Screenshot Description:
- Analyze and describe the overall content of the current screenshot.
- Provide a concise 2-3 sentence summary of the screenshot content.
- Format the description as follows: "This is a screenshot of [summary of the screenshot content]."

2. Intention Recognition:
- You are viewing the current screenshot while completing the task: {task} on a mobile phone, and you have chosen to perform the action: {action}. Analyze the reasoning behind this action choice.
- The progress made on the task before this action is: {context}.
- Focus on why this action is appropriate within the current context, using present tense as if actively solving the problem.
- Explain your intention in the first person.
- Format the intention in 2-3 sentences as follows: "To [goal or purpose], I choose to [action to take]. This allows me to [result or benefit]."

P.S.:
- When following these instructions, use natural language, avoid mentioning technical details (such as action coordinates) or direct use of action tags (such as "PRESS_HOME").

Output Format:
The output should be in JSON format as follows:
{{
"description": "2-3 sentences summarizing the current screenshot content in the desired format",
"intention": "2-3 sentences explaining reasoning for choosing this action in the desired format"
}}

Please return the result in pure JSON format, without any json tags like ```json ```.

Figure 13. Prompts for generating screen description and decision rationale.

Prompt for evaluating closed-source proprietary LVLMs

<img>current_screenshot.png</img>
Given a device screenshot and an instruction, please provide the corresponding action.

Available Actions:
CLICK: <coordinate>
LONG_PRESS: <coordinate>
TYPE: <text>
SCROLL: UP
SCROLL: DOWN
SCROLL: LEFT
SCROLL: RIGHT
PRESS_BACK
PRESS_HOME
PRESS_RECENT
IMPOSSIBLE
COMPLETE

All <coordinates> are in the form (x, y), representing the coordinates to click or long press. The coordinate of the top-left corner is (0, 0), and the coordinate of the bottom-right corner is (1000, 1000).

The instuction is: {instuction}
The historical actions are: {history_actions}

Based on the screenshots and the available actions, provide the next step directly.

Prompt for evaluating closed-source proprietary LVLMs with OmniParser

<img>current_screenshot.png</img>
<img>current_screenshot_w_labels.png</img>
Given two device screenshots and an instruction, provide the corresponding action.
The first image is the original screenshot, and the second is the same screenshot with numeric tags on different interface elements. If the action requires clicking or pressing, choose the closest numeric tag that aligns with your intended location.

Here are the Available Actions:
CLICK: <element_idx chosen from the second screen>
LONG_PRESS: <element_idx chosen from the second screen>
TYPE: <text>
SCROLL: UP
SCROLL: DOWN
SCROLL: LEFT
SCROLL: RIGHT
PRESS_BACK
PRESS_HOME
PRESS_RECENT
IMPOSSIBLE
COMPLETE

The instuction is: {instuction}
The historical actions are: {history_actions}

Based on the screenshots and the available actions, provide the next step directly.

Figure 14. The prompt for evaluating closed-source proprietary Large Vision Language Models (LVLMs).