MOBILEVIEWS: SCALABLE AND HIGH-FIDELITY DATA COLLECTION FOR MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training capable mobile GUI agents requires diverse interaction data, yet collecting such data across a broad range of apps remains challenging. We introduce MobileViews, a mobile GUI dataset built through a scalable hardware–software pipeline combining native SoC clusters with VLM-assisted traversal and selective human fallback. It contains over 1.2M unique screenshot–view hierarchy pairs and associated interaction traces from more than 30K Android apps on Google Play. To assess the dataset's utility, we construct MobileViewsIDM, comprising over 81K inverse dynamics samples from 26K apps for training models to understand GUI transitions. Across the three evaluated VLMs, training on MobileViewsIDM generally outperforms training on prior mobile GUI datasets that are limited in app diversity and scale. Our hardware–software co-design also improves native mobile app compatibility from 56% to 96% and increases the traversal success rate from 16.7% to 89.6%. MobileViews will be fully open-sourced.

1 INTRODUCTION

Vision-language models (VLMs) enable mobile GUI agents to interpret app interfaces and execute tasks specified in natural language (Hong et al., 2024; Zhang et al., 2025; Lin et al., 2025). Recent general-purpose VLMs explicitly incorporate GUI-specific data and training objectives to strengthen GUI understanding and interaction capabilities (Bai et al., 2025; Xiaomi MiMo Team, 2026). Mobile GUI interaction involves understanding dense layouts, text-rich components, and the relationships between GUI states and user actions. Learning and evaluating these capabilities across diverse apps calls for interaction data with broad coverage of applications, in-app states, and state transitions. This motivates large-scale, diverse mobile GUI datasets and scalable collection methods that can continually expand such coverage.

Existing datasets and limitations. As shown in Table 1, existing open-source mobile GUI datasets face a critical trade-off between scale and comprehensiveness. Rico (Deka et al., 2017), proposed in 2017, includes only 63K unique screens at a single screen resolution. While AITW (Rawles et al., 2023) contains 2.28M unique screens, it suffers from two key limitations: (1) limited app diversity results in substantial visual repetition (as we will show in §4), and (2) the lack of view hierarchies (VHs) hinders fine-grained and accurate screen interpretation in downstream training tasks. GUIOdyssey (Lu et al., 2025a) and AndroidControl (Li et al., 2024) provide rich task annotations, but their reliance on human task demonstrations introduces additional manual effort when extending coverage to new apps and interactions. ScaleCUA (Liu et al., 2026) collects over 2M screenshots across platforms, but its reported app list includes only 103 Android apps and 39 iOS apps. These limitations call for a large-scale and comprehensive mobile GUI dataset built upon a scalable collection pipeline.

Challenges and key techniques. Collecting mobile GUI data requires scalable exploration across diverse apps and faithful capture of their states and interactions. We address these requirements through complementary software and hardware designs.

• At the software level, efficient and scalable traversal across diverse mobile apps is essential, yet current solutions do not fully address this requirement. (1) Static rule-based GUI navigation tools (Li et al., 2017; Android Developers, 2023) often yield low-quality screen data, as they struggle to adapt to complex GUI contexts and diverse navigation patterns. For instance, they frequently get stuck on login pages or intrusive pop-ups. (2) Most existing large mobile GUI datasets (Deka et al., 2017;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Comparison of open-source mobile GUI datasets. For large datasets (≥50K screens), we count the number of unique screens using ImageHash (Buchner, 2024) to eliminate duplicates.

Mobile GUI Dataset	Scale		Data Comprehensiveness				Collection Environment (Automated Collection)
	Apps	Unique Screens	App Metadata	Screenshot-VH Pairs	GUI Trajectories	Multiple Resolutions	
Rico (Deka et al., 2017)	9,772	63,370	✓	✓	✓	✗	Physical Devices (✗)
PixelHelp (Li et al., 2020)	4	187	✗	✓	✓	✗	Emulators (✗)
Screen2words (Wang et al., 2021)	6,269	22,417	✓	✓	✗	✗	N/A (✗)
ScreenQA (Baechler et al., 2024)	N/A	35,352	✓	✓	✗	✗	N/A (✗)
META-GUI (Sun et al., 2022)	11	24,825	✗	✓	✓	✗	Physical Devices (✗)
DroidTask (Wen et al., 2024)	13	362	✗	✓	✓	✗	Emulators (✗)
AITW (Rawles et al., 2023)	357	2,282,533	✗	✗	✓	✓	Emulators (✗)
LlamaTouch (Zhang et al., 2024b)	57	3,281	✗	✓	✓	✗	Emulators (✗)
GUIOdyssey (Lu et al., 2025a)	212	77,535	✗	✗	✓	✓	Emulators (✗)
AndroidControl (Li et al., 2024)	833	81,459	✗	✓	✓	✗	Physical Devices (✗)
GUI-Net-1M (Zhang et al., 2026)	280	720,374	✗	✗	✓	✓	N/A (✗)
MobileViews	**30,352**	**1,213,866**	✓	✓	✓	✓	**SoC clusters (✓)**

Rawles et al., 2023; Li et al., 2024; Lu et al., 2025a) are manually collected to ensure high quality, as shown in Table 1. However, such a manual collection process is not scalable due to the substantial human labor required.

Design#1: VLM-enhanced semi-automated app crawler. To ensure traversal efficiency and data quality, we design MobileViews Crawler, a VLM-enhanced semi-automated traversal framework. The crawler captures and simplifies VH metadata to identify interactable GUI components. It traverses these components sequentially using predefined interaction rules. The VLM assists traversal when predefined triggers occur, such as prolonged idleness, login pages, or pop-ups. Human operations handle unresolved app states in batches, intervening only when both the rules and the VLM fail to proceed. This hybrid strategy improves GUI coverage while limiting manual intervention.

• At the hardware level, large-scale app traversal requires running numerous Android instances in parallel with high environment fidelity. Most mobile GUI datasets rely on emulators (Toyama et al., 2021; Sun et al., 2022; Rawles et al., 2023; Zhang et al., 2024b) for cost-effective scaling. However, emulators typically run on x86 machines, causing compatibility issues with ARM-based apps and limiting app diversity. For instance, AITW (Rawles et al., 2023) collects about 5M screens from only 357 apps on Android emulators, but fewer than 2.3M are left after image de-duplication. In contrast, Rico (Deka et al., 2017) uses physical device farms to run apps natively, enabling collection from over 9K apps. Although physical devices eliminate compatibility constraints, they face scalability challenges due to high operational costs and space constraints.

Design#2: Scalable, high-fidelity app traversal on SoC clusters. To improve hardware-level traversal efficiency, we deploy two SoC clusters (Zhang et al., 2024a; Shahrad & Wentzlaff, 2017) with 120 Snapdragon processors (Qualcomm Technologies, Inc., 2019), which provide native Android environments and high app compatibility. On top of this, we leverage Android OS virtualization to launch over 200 instances across the two SoC clusters. Together with fine-grained task scheduling, this setup achieves high parallelism and significantly reduces data collection time, which gathers over 1.2M unique screens in approximately 151,600 device-hours.

The new dataset: MobileViews. Based on the proposed system, we construct MobileViews, an open-source dataset with over 1.2M unique screenshot-VH pairs from more than 30K Android apps. Compared to AndroidControl (Li et al., 2024), MobileViews includes 36× more apps and 15× more screenshot-VH pairs, with a scalable pipeline for flexible expansion. Detailed dataset statistics are presented in §4. To evaluate MobileViews for GUI transition understanding beyond single-screen grounding and referring, we apply a multi-stage filtering pipeline to construct MobileViewsIDM, comprising over 81K samples from more than 26K apps. MobileViewsIDM uses inverse dynamics modeling (IDM) to assess transition understanding by predicting the intervening action and its arguments from a before–after GUI transition pair (Gao et al., 2025; Wu et al., 2024; Li et al., 2026). This task enables us to compare the effectiveness of MobileViews and prior GUI datasets as training sources for GUI transition understanding.

Experiments. We train three VLMs: UI-TARS-1.5-7B (Qin et al., 2025), Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct (Bai et al., 2025) on MobileViewsIDM, Rico, and AndroidControl on IDM tasks. Training on MobileViewsIDM achieves the highest IDM accuracy among the evaluated datasets in seven of nine model–agent configurations, trailing the best by less than 0.5 percentage points in the remaining two. With only 30K samples, it outperforms both full baseline datasets in

most configurations. Our hardware–software co-design supports scalable dataset expansion through semi-automated collection without requiring task-instruction annotations. The SoC cluster supports 96 of the 100 evaluated top-ranked apps on the Google Play Store, compared with 56 on an x86 Android emulator, while VLM assistance and human fallback increase traversal success from 16.7% to 89.6% among the 96 operational apps.

Contributions of this study are as follows:

- We develop a scalable, high-fidelity mobile GUI data collection pipeline combining native mobile SoC clusters with VLM-enhanced semi-automated app traversal.
- We collect MobileViews, a diverse mobile GUI dataset containing over 1.2M unique screenshot–VH pairs from more than 30K Android apps.
- Experiments demonstrate gains over prior datasets in GUI transition understanding and over collection baselines in app compatibility and traversal success.

2 Related Work

Table 1 compares existing open-source mobile GUI datasets in terms of scale, comprehensiveness, and collection methods.

Data scale. Rico (Deka et al., 2017) is a widely-used dataset released in 2017, containing about 63K unique screens collected from over 9K Android apps. Screen2Words (Wang et al., 2021) and ScreenQA (Hsiao et al., 2022) enrich its screens with textual annotations, without expanding the underlying app set. Other datasets (Li et al., 2020; Rawles et al., 2023; Zhang et al., 2024b; Li et al., 2024; Lu et al., 2025a; Wu et al., 2024) support training and evaluating GUI grounding or navigation models. However, most of these datasets are collected from a limited number of apps and contain relatively few screens, yet recent studies in GUI understanding and robotic manipulation suggest that larger and more diverse training datasets can improve generalization (Fan et al., 2025; Lu et al., 2025b; Khazatsky et al., 2024). These differences motivate expanding both screen collections and app coverage.

Data comprehensiveness. From an informational perspective, both pixel-level screenshots and textual VHs are essential for mobile GUI understanding. Screenshots capture visual appearance, while VHs provide component hierarchies, bounding boxes, and element attributes for constructing structured supervision. Although AITW (Rawles et al., 2023) reaches the largest scale to date, it lacks VH metadata, preventing flexible sample construction by combining both visual and structural information. Similarly, GUI-Net-1M (Zhang et al., 2026), which is collected from online videos instead of direct app traversal, also lacks VHs. Beyond these modalities, resolution and language coverage are important dimensions of dataset comprehensiveness. MobileViews provides aligned screenshot–VH pairs and interaction trajectories across multiple resolutions.

Data collection methods. The data collection platforms and strategies are both crucial for ensuring the fidelity and scalability of mobile GUI datasets. Datasets such as Rico (Deka et al., 2017) rely on physical devices, which provide high-fidelity environments but face scalability challenges due to cost and space constraints. AITW (Rawles et al., 2023) and GUIOdyssey (Lu et al., 2025a) use emulators to collect data, which offers cost-effective scaling but suffers from compatibility issues since most mobile apps are designed for native ARM processors. This incompatibility restricts app diversity and limits the coverage of modern apps. Beyond environment constraints, human demonstrations also require substantial collection effort: AndroidControl (Li et al., 2024) collected 14,548 episodes across 833 apps over one year. These limitations motivate the construction of MobileViews, which combines VLM-enhanced semi-automated app traversal with native SoC clusters to enable scalable and high-fidelity data collection across diverse modern mobile applications.

3 MobileViews Dataset Construction

Figure 1 outlines our hardware–software collection pipeline. MobileViews Crawler combines rule-based interaction, VLM assistance, and selective human intervention to traverse individual apps, while two SoC clusters provide Android instances for parallel collection.

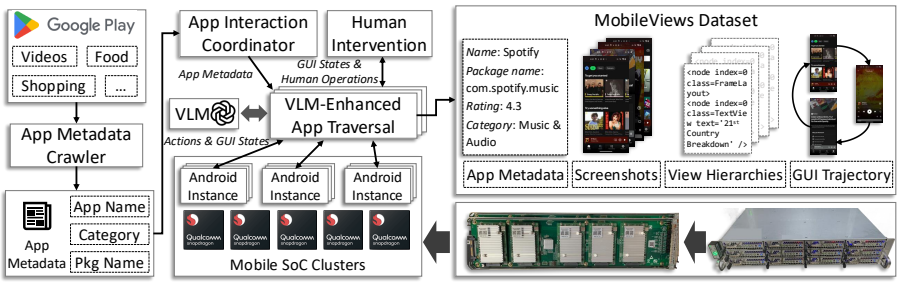


Figure 1: Left: Workflow of the MobileViews dataset construction. Right: Demonstration of the MobileViews dataset, and a physical SoC Cluster to empower high-parallelism app traversal.

3.1 App Traversal Workflow

App metadata collection. The data collection process begins with gathering app metadata, such as app package names. MobileViews covers apps from 33 categories on the Google Play Store, including many trending ones, providing a more accurate reflection of recent mobile interface designs.

App interaction coordination. An *App Interaction Coordinator* uses the collected package names to assign apps to available Android instances on the SoC clusters. On each instance, the crawler installs and launches the assigned app, then interacts with it to collect screenshots and VHs. Traversal stops when the predefined action budget is reached or an unexpected interruption occurs. The coordinator records the current system and app states for subsequent analysis.

VLM-enhanced semi-automated app traversal. By default, the crawler follows predefined rules to interact sequentially with interactable GUI components on each screen. When traversal gets stuck, such as on login pages, the VLM interprets the current GUI state and guides subsequent interactions to reach deeper interfaces. Human operators intervene in twice-daily batches to resolve complex GUI states, such as login verification or complex app-specific setup, before resuming automatic traversal.

3.2 App Interaction with MobileViews Crawler

Rule-based app interaction. We employ a rule-based app traversal tool, Droidbot (Li et al., 2017), to automate the traversal process. As illustrated in Figure 2(a), the crawler simplifies each screen's VH to identify interactable components using attributes such as *clickable*, *editable*, and *focusable*. It then executes corresponding actions (e.g., clicks, scrolls, or text inputs) on these GUI components in sequence. When Droidbot navigates to a new screen, it repeats this procedure until reaching the maximum number of steps, which is set to 1,000 actions per app in our setup.

VLM integration. While fixed interaction rules can reach many common GUIs effectively, they often fail to handle complex ones, such as accurately filling in specific input fields or closing intrusive pop-ups. To address this, we integrate a VLM to handle complex GUI states under two primary conditions: (1) the traversal tool encounters GUI states matching predefined triggers (e.g., keywords like "sign up" or "login" are detected in the VH), and (2) the current screen remains idle beyond a configured duration. In these cases, the VLM takes over the traversal process until it reports success or failure. This VLM-enhanced framework can also be extended to other scenarios by defining suitable triggers to handle various complex GUI states.

Human intervention. Although the VLM integration can handle complex GUI states effectively, certain challenging cases can still cause the traversal process to stall. This typically occurs when the VLM reaches its maximum retry limit or returns a "task failed" response. Under our *lazy processing* strategy, human operators review these cases in twice-daily batches, resolve the blockers, and resume automatic traversal. This confines manual intervention to states that remain unresolved after rule-based traversal and VLM assistance.

3.3 SoC Cluster-Powered Scalable App Interaction

By combining rule-based interaction, VLM, and human intervention, the traversal tool can efficiently navigate each app and collect diverse GUIs. The next challenge lies in scaling this process to handle

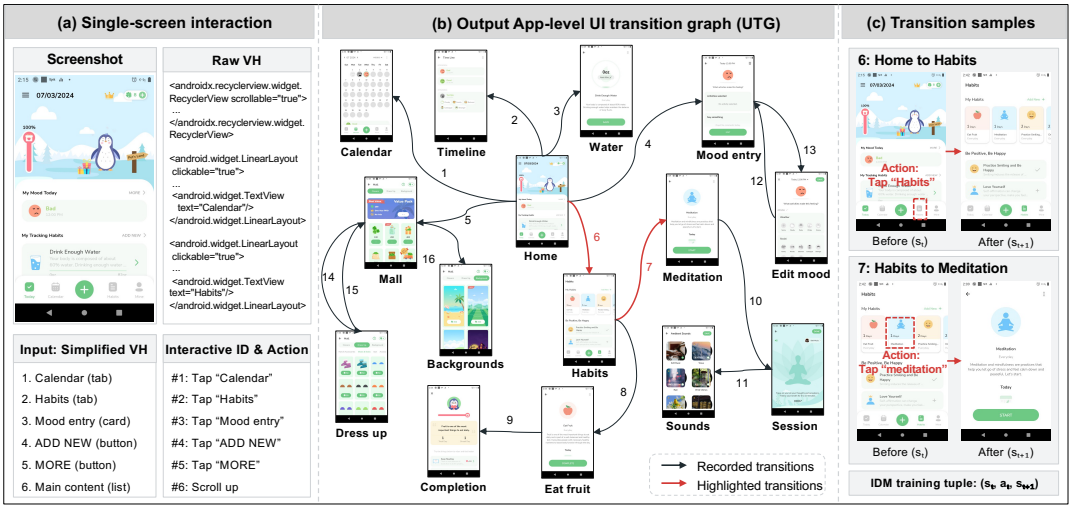


Figure 2: MobileViews Crawler workflow. **(a)** VH simplification and action mapping. **(b)** App-level UI transition graph (UTG), with an example trajectory highlighted in red. **(c)** Before–action–after tuples extracted for IDM training.

a large number of apps in native mobile environments. Existing approaches for mobile GUI dataset construction, such as running apps on physical device farms or Android emulators, both have notable limitations. While physical device farms are costly to scale, Android emulators running on x86-based machines often face compatibility issues with apps built for ARM-based processors. To ensure both scalability and compatibility, we deploy two SoC clusters (Zhang et al., 2024a; Shahrad & Wentzlaff, 2017) equipped with 120 Snapdragon (Qualcomm Technologies, Inc., 2019) processors, which provide native ARM-based execution environments for large-scale mobile app interaction. Figure 1 shows the physical setup of an SoC cluster and its network-interconnected mobile processors inside the server. To maximize resource utilization, we further implement SoC-level virtualization to run multiple Android instances on each processor. As a result, more than 200 Android instances run in parallel across the two SoC clusters, enabling large-scale app interaction in a more efficient and cost-effective manner than traditional device farms. To be noted, our SoC-level virtualization is complementary to recent virtual-display-based approaches, which could further increase interaction throughput by enabling concurrent agent sessions within each Android instance (Zhou et al., 2026).

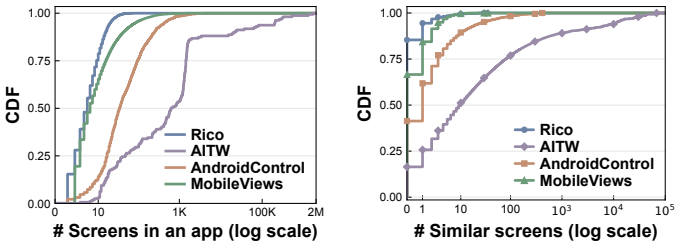
Software-hardware coordination. To achieve better parallelism, the *App Interaction Coordinator* monitors the status of each Android instance at regular intervals. When an instance becomes available, the coordinator connects to it and initiates the app traversal process. The traversal tool then starts its execution. Once the traversal is completed or the process exits unexpectedly, the instance is marked as available again. This fine-grained scheduling strategy enables efficient utilization of hardware resources, allowing us to complete 151,600 device-hours of app traversal within two months.

4 MobileViews Dataset

4.1 Dataset Statistics and Diversity Analysis

As shown in Table 1, MobileViews contains 1,213,866 unique screenshot–VH pairs from 30,352 Android apps. The dataset spans all 32 non-game categories on Google Play, with games grouped into an additional category encompassing genres such as puzzle and strategy. Screenshots capture visual appearance, while VHs provide component hierarchies, bounding boxes, component types, and interaction attributes. The dataset includes more than 60K samples at each of five resolutions (720×1280, 720×1520, 1080×2340, 1440×2560, and 1920×1080), with the remaining screens captured at 1080×1920. Additionally, automated traversal records GUI trajectories comprising sequences of screens and actions, as described in §3.

Screen diversity analysis. Agents trained on diverse screens are more likely to generalize across apps, interface layouts, and visual styles. However, dataset size alone does not necessarily imply broad visual coverage. For example, before image de-duplication, AITW contains over 5M screens collected



(a) Screens per app (b) Visual redundancy

Figure 3: Screen diversity across Rico, AITW, AndroidControl, and MobileViews: (a) CDF of screens per app; (b) CDF of the number of visually similar counterparts per screen.

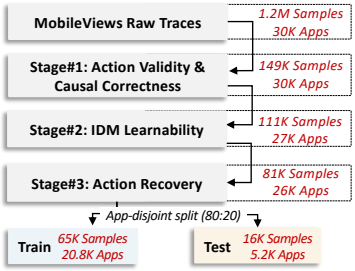


Figure 4: Construction pipeline of MobileViewsIDM and #samples/#apps after each stage.

from only 357 apps, suggesting substantial visual redundancy (Rawles et al., 2023). Accordingly, MobileViews aims to provide both broad app coverage and diverse screen content. We evaluate the screen diversity of MobileViews from two complementary perspectives from four representative datasets, including Rico (Deka et al., 2017), AITW (Rawles et al., 2023), AndroidControl (Li et al., 2024), and MobileViews: (1) the distribution of recorded screens per app across datasets, which characterizes per-app sampling intensity and its variation across apps; and (2) the distribution of the number of visually similar screens per screen within MobileViews and the comparison datasets, which characterizes visual redundancy. For the second analysis, we use *ImageHash* (Buchner, 2024) to compute a 64-bit perceptual hash for each screen. We regard two screens as visually similar when the Hamming distance between their hashes is at most 5.

• *MobileViews provides broad app coverage while capturing a non-trivial number of GUI states within each app.* Figure 3a compares the distributions of recorded screens per app. AndroidControl has a median of 35 screens per app, while AITW generally contains even more screens per app. In contrast, MobileViews and Rico cover larger sets of apps, as shown in Table 1. Within this broad coverage, MobileViews includes 1,968 apps with more than 100 recorded screens each, compared with 212 in AndroidControl and 112 in AITW[1]. Screen counts alone, however, do not distinguish between distinct GUI states and repeated observations of similar states. We therefore examine visual redundancy in the following analysis.

• *MobileViews exhibits limited visual repetition across its collected screens.* Figure 3b compares the distributions of the number of visually similar counterparts per screen. Our core observation is, over 95% of screens in MobileViews have fewer than five visually similar counterparts, close to Rico's 98.2%, compared with 80.7% in AndroidControl and 39.5% in AITW. Moreover, in AITW, over 1.2M screens (23% of the whole dataset) have more than 100 similar counterparts, while the largest group contains 67K screens (e.g., the Android home screen), accounting for approximately 1.2% of the dataset. Together with the broad app coverage of MobileViews, the relatively low similarity counts suggest its potential to provide training examples spanning many apps with limited repetition of visually similar GUI states.

4.2 MobileViewsIDM

The scale and app diversity of MobileViews support learning screen-level capabilities such as GUI grounding and referring, which have been widely studied in prior work. However, strong performance on these tasks does not necessarily imply that models understand GUI state transitions, an important capability for long-horizon tasks. To evaluate the value of MobileViews as training data for GUI models, we focus on GUI transition understanding through inverse dynamics prediction (Agrawal et al., 2016; Torabi et al., 2018; Gao et al., 2026). This task is well suited to MobileViews because its app traversal naturally records consecutive GUI states together with the executed actions, allowing us to derive training samples without additional manual annotation. We leverage MobileViews to construct MobileViewsIDM, in which models predict the intervening action and its arguments from a before–after screen pair. By comparing models trained on MobileViews with those trained on existing

[1]For AITW, we count Android application packages using the analyzed metadata index. Web-based GUIs are attributed to their browser package; individual websites are not counted as separate apps.

datasets, we investigate whether the larger scale and broader app diversity of MobileViews translate into better action inference and generalization.

Dataset construction. As shown in Figure 4, we apply three filtering stages: (1) *action validity and causal correctness*, which checks whether a recorded interaction plausibly explains the observed transition; (2) *IDM learnability*, which removes visually uninformative or ambiguous samples; and (3) *action recoverability*, which retains samples whose recorded actions can be recovered from the screenshot pairs under the matching criteria in §5.1. We use Qwen3.5-397B-A17B for all VLM-based checks. To preserve broad app coverage while controlling dataset size, the first stage retains up to five accepted samples per app. Detailed sampling procedures, filtering rules, and prompts are provided in Appendix A.

Dataset size and split. The final MobileViewsIDM dataset contains 81,790 samples from 26,081 apps. We partition apps into training and test sets at an 80%/20% ratio, with no shared apps. The training set contains 65,305 samples from 20,864 apps, and the test set contains 16,485 samples from 5,217 apps. Figure 2(c) illustrates example transition samples.

5 EVALUATION

In this section, we evaluate whether MobileViews provides better training data for GUI transition understanding than prior datasets, and whether its collection pipeline improves app compatibility and traversal success over existing approaches.

5.1 EXPERIMENTAL SETUP

Models, Datasets, and Training. We evaluate three open-source VLMs on the IDM benchmark: UI-TARS-1.5-7B (Qin et al., 2025) and Qwen3-VL-4B/8B-Instruct (Bai et al., 2025). For the open-source models, we compare zero-shot performance with performance after fine-tuning on MobileViewsIDM, AndroidControl (Li et al., 2024), and Rico (Deka et al., 2017), using varying numbers of training samples. From the annotated interaction traces in AndroidControl and Rico, we construct 31,011 and 70,308 IDM training samples, respectively, representing the maximum number obtainable from each dataset. We map dataset-specific action labels to a shared set of action types following AndroidControl. We follow the training procedure of GUI-Shift (Gao et al., 2026) for fine-tuning across all three datasets. We use verl (Sheng et al., 2024) as the training framework and conduct experiments on a cluster equipped with 32 NVIDIA H100 GPUs.

Agent configurations. We designed three agent configurations. (1) The *None-CoT agent* takes raw before-and-after screenshots and directly outputs the action without explicit reasoning; (2) The *CoT agent* takes the same inputs but explains the visual changes before outputting the action; (3) The *SoM agent* (Yang et al., 2023) additionally receives the before screenshot overlaid with bounding boxes and numeric labels, together with GUI component details derived from VH. It selects action targets by index without explicit reasoning. Both None-CoT and CoT agents use coordinates for clicks and long presses. The full prompts for all three configurations are provided in Appendix B.

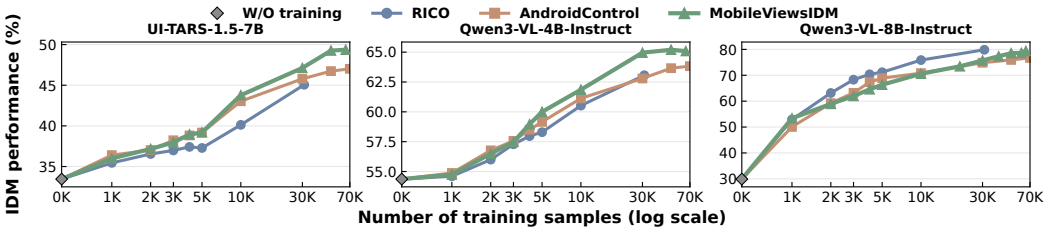
Metrics. We measure IDM performance using action prediction accuracy, defined as the percentage of screenshot pairs for which the predicted action matches the ground truth. Following the evaluation principles of AndroidControl (Li et al., 2024), we evaluate actions using their type and action-specific arguments. For None-CoT and CoT agents, clicks and long presses require the correct action type and predicted coordinates within the ground-truth target's bounding box. Text input requires an exact match with the ground-truth text, while scrolling requires the correct direction. A predicted back action is also accepted when the annotated target is a back or navigation-up control. For SoM agents, component-targeted actions require the correct action type and target index, with exact text or direction matching where applicable. Invalid predictions are counted as incorrect.

5.2 IDM PERFORMANCE

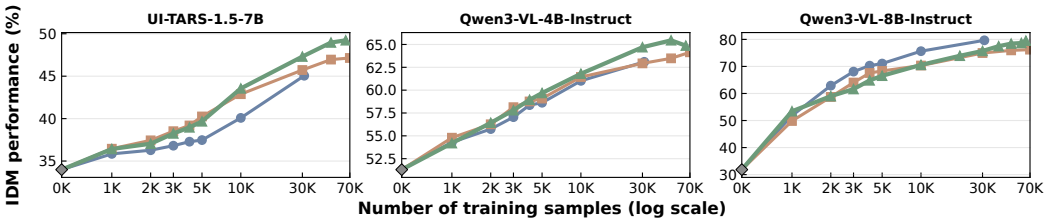
Table 2 shows the performance of small VLMs trained on Rico, AndroidControl, and MobileViewsIDM. Our main observation is that VLMs trained on MobileViewsIDM generally achieve better performance across agent configurations. Specifically, UI-TARS-1.5-7B and Qwen3-VL-4B trained on MobileViewsIDM improve over their zero-shot counterparts by an average of 10.87% across the

Table 2: IDM performance of different VLMs trained with None-CoT configuration on diverse datasets.

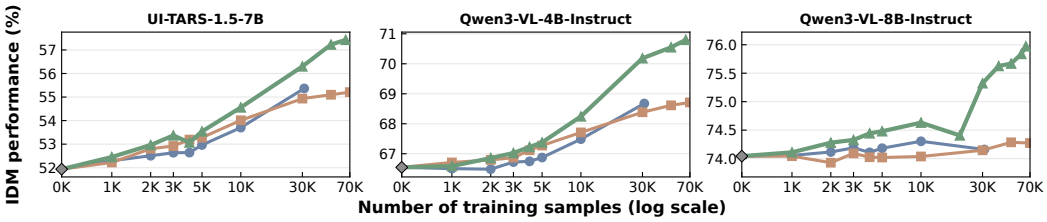
Model	Agent	IDM Performance (%)			
		W/O Training	RICO	AndroidControl	MobileViewsIDM
UI-TARS-1.5-7B	None-CoT	33.45	45.06 (+11.60)	47.02 (+13.56)	**49.36 (+15.91)**
	CoT	33.99	45.05 (+11.06)	47.15 (+13.16)	**49.23 (+15.24)**
	SoM	51.94	55.37 (+3.43)	55.21 (+3.27)	**57.43 (+5.49)**
Qwen3-VL-4B-Instruct	None-CoT	54.36	63.06 (+8.70)	63.83 (+9.46)	**65.09 (+10.72)**
	CoT	51.30	63.10 (+11.80)	64.14 (+12.84)	**64.87 (+13.57)**
	SoM	66.55	68.67 (+2.13)	68.71 (+2.17)	**70.80 (+4.25)**
Qwen3-VL-8B-Instruct	None-CoT	29.90	**79.85 (+49.95)**	76.72 (+46.82)	79.38 (+49.48)
	CoT	31.84	**79.65 (+47.81)**	76.21 (+44.37)	79.51 (+47.67)
	SoM	74.04	74.16 (+0.12)	74.27 (+0.23)	**75.97 (+1.93)**



(a) Performance of None-CoT Agent



(b) Performance of CoT Agent



(c) Performance of SoM Agent

Figure 5: IDM performance with the increased number of training samples on different datasets.

six model–configuration combinations, compared with 8.12% and 9.08% when trained on Rico and AndroidControl, respectively. The maximum gains over Rico and AndroidControl are 4.30% and 2.34%, corresponding to a non-trivial number of approximately 709 and 386 additional samples correctly inferred actions on the test set, respectively. Qwen3-VL-8B trained on MobileViewsIDM performs comparably to its counterpart trained on Rico under None-CoT and CoT setups, with differences below 0.5%. Under SoM, it outperforms its counterparts trained on Rico and AndroidControl by 1.81% and 1.70%, respectively. Together, these results support the value of MobileViewsIDM for GUI transition understanding and suggest that its broad app coverage may help models learn transferable state–action relationships.

Table 3: App compatibility comparison between Android emulator and SoC cluster.

Metrics (#Apps = 100)	Android Emulator	SoC Cluster
Operational Compatibility	56/100	96/100
Platform-exclusive Compatible Apps	2/100	42/100

Table 4: App traversal success. #Apps counts apps traversed without getting stuck.

Metrics	Fully Rule-based	W/ VLM	W/ Human Intervention
#Apps (Total = 96)	16	38	86
Success Rate	16.7%	39.6%	89.6%

Figure 5 examines how IDM performance changes with the number of training samples across the three datasets. Training on the full MobileViewsIDM generally yields higher accuracy than training on Rico or AndroidControl. With only 30K training samples, MobileViewsIDM outperforms both full baseline datasets in most configurations, including all three models under SoM. Notably, Qwen3-VL-8B requires only 3K samples to reach 74.33% accuracy, comparable to the 74.16% and 74.27% obtained using the full Rico and AndroidControl training sets, respectively. These results suggest that MobileViewsIDM offers benefits beyond training-set size, particularly when combined with a suitable agent design, as demonstrated by the SoM configuration. Furthermore, our semi-automated pipeline reduces manual collection effort and requires no well-labelled task instructions for IDM supervision, enabling expansion to more apps and interactions. The observed scaling trends suggest that such expansion could further improve GUI transition understanding.

5.3 Effectiveness of MobileViews Data Collection

We evaluate whether our hardware–software co-design addresses two successive barriers to mobile GUI data collection: whether an app can run on the hardware platform and whether traversal can proceed without unresolved blocking states.

Hardware: expanded operational app coverage on SoC clusters. Table 3 compares our SoC cluster with an x86 Android emulator on the same top-100 apps within Google Play Store. We define *operational compatibility* as the absence of environment-related installation, launch, or execution failures; access restrictions and traversal or GUI-extraction failures are assessed separately. The SoC cluster supports 96 apps, compared with 56 on the Android emulator, a net gain of 40 apps. Of the 44 apps that are non-operational on the emulator, 42 (95.5%) operate on the SoC cluster. Thus, the SoC-based environment makes nearly all emulator-incompatible apps available for subsequent traversal under the evaluated conditions.

Software: improved app traversal success through the semi-automated pipeline. Table 4 compares three traversal configurations on the 96 apps operational on the SoC cluster. We define *traversal success* as reaching the predefined action budget without an unresolved blocking state. Rule-based traversal (i.e., the Droidbot's intrinsic traversal approach) succeeds on 16 apps (16.7%). Adding VLM assistance increases success to 38 apps (39.6%), while further adding human fallback raises it to 86 apps (89.6%), a net gain of 70 apps over rule-based traversal. The ten apps without a successful traversal remain in the denominator, including those affected by access restrictions or unavailable GUI-state representations (i.e., VH in WebView-based apps). Thus, the app traversal design enables the collection pipeline to overcome blocking states and reach its action budget on substantially more operational apps.

To conclude, across the original 100-app benchmark, the full pipeline successfully traverses 86 apps. Since only 54 apps are operational on both platforms, at least 32 of these successful traversals involve apps that are non-operational on Android emulator. Thus, the SoC cluster's expanded app compatibility translates into successful collection with the staged traversal pipeline.

6 Conclusion

We identify key barriers to efficient mobile GUI data collection and introduce MobileViews, containing over 1.2M unique screens from more than 30K Android apps. Our hardware–software co-design combines SoC clusters with VLM-enhanced semi-automated traversal to support broad app coverage. Experiments show that MobileViews generally improves GUI transition understanding compared with prior mobile GUI datasets. We hope our dataset and collection approach enable continued expansion of diverse GUI interaction data and advance more capable mobile GUI agents.

AI USE STATEMENT

Generative AI was used for app traversal and dataset filtering, as explicitly described in the main paper. For paper writing and editing, generative AI tools were used only to correct grammatical errors and improve the readability of content written by the authors. All AI-assisted edits were later reviewed and verified by the authors.

REFERENCES

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. Advances in neural information processing systems, 29, 2016.

Android Developers. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey, 2023.

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, 2024.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. arXiv preprint arXiv:2511.21631, 2025.

Johannes Buchner. Imagehash: An image hashing library written in python. https://github.com/JohannesBuchner/imagehash, 2024.

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

Yue Fan, Handong Zhao, Ruiyi Zhang, Yu Shen, Xin Eric Wang, and Gang Wu. Gui-bee: Align gui action grounding to novel environments via autonomous exploration. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 33249–33266, 2025.

Longxi Gao, Li Zhang, and Mengwei Xu. Uishift: Enhancing vlm-based gui agents through self-supervised reinforcement learning. arXiv preprint arXiv:2505.12493, 2025.

Longxi Gao, Li Zhang, Pengzhi Gao, Wei Liu, Jian Luan, and Mengwei Xu. Gui-shift: enhancing vlm-based gui agents through self-supervised reinforcement learning. In International Conference on Learning Representations, volume 2026, pp. 39236–39255, 2026.

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. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14281–14290, 2024.

Yu-Chung Hsiao, Fedir Zubach, Maria Wang, et al. Screenqa: Large-scale question-answer pairs over mobile app screenshots. arXiv preprint arXiv:2209.08199, 2022.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. arXiv preprint arXiv:2403.12945, 2024.

Wei Li, William E. Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In NeurIPS, 2024.

Weiwei Li, Junzhuo Liu, Tong Chu, Hengfu Yu, and Wen Li. The next screenshot knows: Gated hindsight distillation for mobile gui agents. arXiv preprint arXiv:2608.06065, 2026.

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.

Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 23–26. IEEE, 2017.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In Proceedings of the Computer Vision and Pattern Recognition Conference, pp. 19498–19508, 2025.

Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. In International Conference on Learning Representations, volume 2026, pp. 62707–62766, 2026.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 22404–22414, 2025a.

Yuheng Lu, Qian Yu, Hongru Wang, Zeming Liu, Wei Su, Yanping Liu, Yuhang Guo, Maocheng Liang, Yunhong Wang, and Haifeng Wang. Transbench: Breaking barriers for transferable graphical user interface agents in dynamic digital environments. In Findings of the Association for Computational Linguistics: ACL 2025, pp. 12464–12478, 2025b.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025.

Qualcomm Technologies, Inc. Qualcomm snapdragon 865 5g mobile platform. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/prod_brief_qcom_sd865_5g.pdf, 2019.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. Advances in Neural Information Processing Systems, 36:59708–59728, 2023.

Mohammad Shahrad and David Wentzlaff. Towards deploying decommissioned mobile devices as cheap energy-efficient compute nodes. In 9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17), 2017.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. arXiv preprint arXiv: 2409.19256, 2024.

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.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, pp. 4950–4957, 2018.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. arXiv preprint arXiv:2105.13231, 2021.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. Screen2words: Automatic mobile ui summarization with multimodal learning. In The 34th Annual ACM Symposium on User Interface Software and Technology, pp. 498–510, 2021.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, pp. 543–557, 2024.

Qinzhuo Wu, Weikai Xu, Wei Liu, Tao Tan, Liujian Liujianfeng, Ang Li, Jian Luan, Bin Wang, and Shuo Shang. Mobilevlm: A vision-language model for better intra-and inter-ui understanding. In Findings of the Association for Computational Linguistics: EMNLP 2024, pp. 10231–10251, 2024.

Xiaomi MiMo Team. Mimo-v2.6-pro-rl. `https://huggingface.co/XiaomiMiMo/MiMo-V2.6-Pro-RL`, 2026.

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.

Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pp. 12367–12375, 2026.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, pp. 1–20, 2025.

Li Zhang, Zhe Fu, Boqing Shi, Xiang Li, Rujin Lai, Chenyang Yang, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. More is different: Prototyping and analyzing a new form of edge server with massive mobile socs. In 2024 USENIX Annual Technical Conference (USENIX ATC 24), pp. 285–302, 2024a.

Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. Llamatouch: A faithful and scalable testbed for mobile ui task automation. In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology, pp. 1–13, 2024b.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, Jianan Zhang, Long Li, Long Chen, et al. Qwen-ui-agent technical report: Toward next-generation real-world centric foundation gui agents. arXiv preprint arXiv:2607.28227, 2026.

APPENDIX

A MOBILEVIEWSIDM CONSTRUCTION DETAILS

This section details candidate selection and the three-stage filtering procedure used to construct MobileViewsIDM. We use Qwen3.5-397B-A17B for all VLM-based checks. The filtering procedure was developed iteratively: Stage 2 re-examines the samples retained by Stage 1, and Stage 3 tests whether their recorded actions can be recovered from the visual evidence.

A.1 CANDIDATE EXTRACTION AND SAMPLING

We extract before–action–after tuples (s_t, a_t, s_{t+1}) from the app-level UI transition graphs produced by MobileViews Crawler. We remove incomplete records, transitions with missing screenshots, and duplicate transitions. We also exclude app-start, app-termination, and login-control events that do not represent direct GUI interactions.

Within each app, candidates are evaluated in center-out trajectory order until five samples pass Stage 1 or the candidates are exhausted. This cap applies to accepted samples rather than evaluated candidates, allowing additional candidates to be examined after rejection. The cap controls dataset size while limiting the contribution of individual apps and can be increased for future expansion. Stages 2 and 3 further filter the accepted samples without acquiring replacements.

A.2 STAGE 1: ACTION VALIDITY AND CAUSAL CORRECTNESS

The VLM receives the before and after screenshots together with the recorded action. When available, a bounding-box overlay identifies the action region on the before screenshot. The model checks whether the action is a single, identifiable interaction and whether the resulting visual change is a plausible consequence of that interaction.

We reject missing, ambiguous, or multi-step actions, as well as actions targeting invisible elements or blank regions. We also reject transitions involving the operating-system home screen, actions without a visible result, and changes inconsistent with the recorded action. Unrelated changes caused by advertisements, background updates, or notifications are excluded. Action-triggered permission dialogs are retained because they represent a plausible response to the interaction. This stage retains 149,315 samples from 30,352 apps.

A.3 STAGE 2: IDM LEARNABILITY

Stage 2 assesses whether the retained samples provide clear and informative supervision for inverse dynamics learning. We reject samples exhibiting any of the following issues:

- **Insufficient visual change:** nearly identical screens, changes limited to system bars or minor animations, or scrolls without coherent directional content movement.
- **Unreliable visual evidence:** loading states, incomplete rendering, severe clipping, rotation-related defects, or blank and corrupted screenshots.
- **Ambiguous action targets:** unclear interactive elements, misaligned bounding boxes, or degenerate target geometry.
- **Uninformative text inputs:** synthetic placeholder inputs in credential fields or masked inputs whose contents cannot be recovered visually.
- **External handoffs:** transitions dominated by system share menus, app choosers, or external sign-in pages.

This stage retains 111,356 samples from 27,836 apps.

A.4 STAGE 3: ACTION RECOVERABILITY

The filtering VLM uses the CoT agent configuration B.2 to predict the intervening action and its arguments from each retained before–after screenshot pair, with the detailed prompt provided in

Appendix B. We retain samples only when the prediction matches the recorded action under the action-matching criteria in §5.1. This stage yields 81,790 samples from 26,081 apps.

This check selects samples recoverable by the filtering VLM; it does not establish that the recorded action is the only possible explanation of the transition.

A.5 Filtering Prompts

In this study, we developed the filtering procedure iteratively. We initially applied only the Stage 1 rules. After identifying remaining quality issues, we re-filtered the retained samples using the additional Stage 2 rules. In future studies, the rules from both stages can be combined into a single filtering pass. In the following subsections, we demonstrate our prompt design for each stage.

A.5.1 Shared input and instructions.

The placeholder `[Rules]` indicates where the stage-specific rules listed below are inserted.

```
You are an expert AI data filter specializing in mobile GUI analysis.
Your task is to evaluate GUI transition pairs and determine whether they
    are high-quality samples for training/evaluating an Inverse Dynamics
    Model (IDM).
An IDM infers the action that causes the GUI state transition, given two
    consecutive screens as the input.

INPUT
You will receive:
[Screen 1]: The GUI state before the action
[Screen 2]: The GUI state after the action
[Action]: The user operation performed, including
(touch,intent,scroll,long_touch,set_text,kill_app,select,unselect,
    wait_user_login).
If [Action] includes a bounding box, [Screen 1] may contain a red overlay
     box highlighting that action region.

GLOBAL VISUAL RULES
- Ignore system status/navigation bar changes (time, battery, signal, etc
    .)
- Ignore dynamic updates incurred by ads refresh, timestamps, background
    feeds, etc.
- Focus ONLY on meaningful GUI changes within the app that are caused by
    the performed action
- Do NOT assume hidden steps or intermediate actions
- Do NOT infer invisible interactions; rely ONLY on visible UI evidence

EVALUATION PROCEDURE & RULES
Rely ONLY on visible UI evidence.
Do not assume hidden steps.
You MUST evaluate the sample against the following criteria and their
    associated violation rules.
Check ALL rules; do not skip any.

[Rules]

DECISION LOGIC & STRICT CONSISTENCY
If NO rules are triggered: valid = true, violations = [], and all
    criteria fields = true.
If ANY rule is triggered: valid = false, list ALL triggered rule IDs in
    violations.
Consistency: Criteria fields MUST be false if their corresponding rules
    are triggered:
- If Rule 1, 9, 12, or 13 triggered -> action_valid = false
- If Rule 2, 3, 4, or 11 triggered -> causal_correct = false
- If Rule 5, 6, 7, 8, 10, 14, or 15 triggered -> idm_learnable = false
```

```
OUTPUT FORMAT
Return a valid JSON object ONLY.
Do NOT include markdown formatting (like ```json), explanations, or extra
     text.
{
    "valid": true | false,
    "criteria": {
        "action_valid": true | false,
        "causal_correct": true | false,
        "idm_learnable": true | false
    },
    "violations": [rule_ids],
    "reason": "1-2 concise sentences grounded in visible UI evidence."
}
```

A.5.2 Stage 1: Action Validity and Causal Correctness

```
1. Action Validity (action_valid)
The action must be a single, atomic interaction.
Spatial Tolerance: It is VALID if the action area significantly overlaps
    with an interactable element, even if it includes surrounding empty
    space.

[Rule 1] Invalid Action: Action is missing, multi-step, ambiguous,
    targets elements not visible in Screen 1, or the action area is
    entirely on a blank background with no nearby elements.

2. Causal Correctness (causal_correct)
There must be a clear UI change that is the direct, logical result of the
     action.

[Rule 2] Home Screen Transition: Screen 1 is the mobile OS home screen (
    app launch), or Screen 2 is the home screen (app exit/crash).

[Rule 3] Illogical Mapping / No Result: The UI does not change, or the
    result violates standard mobile UI conventions (e.g., tap causes
    scroll, non-destructive action yields destructive warning, "three
    dots" doesn't open a menu).

[Rule 4] Unrelated Interference: The UI change is caused by ads, dynamic
    background feeds, low battery, or unrelated notifications.
(Exception: OS-level permission dialogs directly triggered by the action
    are VALID).
```

A.5.3 Stage 2: IDM Learnability Filtering

```
3. IDM Learnability (idm_learnable)
The UI change must be stable, unambiguous, and meaningful for learning.

[Rule 5] Trivial Change: Screens are nearly identical.
Changes are limited to ignored system bar updates (time/battery), cursor
    blinks, minor animations, or masked text (e.g., passwords as ***).

[Rule 6] Incomplete Rendering: Screen 2 shows loading spinners, skeleton
    screens, or is caught mid-transition.

[Rule 7] Invalid Scroll: A scroll action results in minimal, ambiguous,
    or inconsistent content displacement.

[Rule 8] Partial/Rotated Rendering Defect: Either screen is not fully/
    cleanly rendered (e.g., only part of the UI is visible after rotation,
     severe clipping/cropping, stretched layout, or obvious viewport
    mismatch that prevents reliable action-outcome judgment).
```

```
[Rule 9] Bounding Box Misalignment/Scale Error: For bbox-based actions,
    the highlighted bbox on Screen 1 is clearly wrong (e.g., too small/
    too large, shifted away from the actual target, or enclosing mostly
    irrelevant area), making the action target unreliable.

[Rule 10] Synthetic Auth/Text Placeholder Input: set_text uses synthetic
    placeholder text (e.g., dummy_user_input) on login/register/account-
    recovery/credential fields (email/phone/password/OTP/username),
    producing low-value or non-generalizable supervision.

[Rule 11] External/System Handoff: The transition is dominated by leaving
     the app flow or invoking OS/external surfaces (e.g., app chooser/
    open-with sheet, browser/social auth handoff, share sheet, external
    intent target), so the result is not a stable in-app causal outcome.

[Rule 12] Ambiguous Non-Atomic Target Node: The touched element is a
    generic text/container fragment (e.g., non-clickable child text,
    blank text node, broad informational paragraph) where the true
    interactive target is unclear.

[Rule 13] Degenerate Target Box Geometry: The action bbox is
    geometrically unreliable (e.g., tiny dot/line strip, extreme edge
    strip, or overly large region covering mostly irrelevant area),
    making target localization ambiguous.

[Rule 14] Scroll Without Coherent Displacement: A scroll action does not
    produce clear, directional content movement (or movement is
    negligible/contradictory), so the transition is not learnable as a
    scroll effect.

[Rule 15] Corrupted/Blank Visual Evidence: Screen 1 or Screen 2 is blank/
    black/corrupted/unreadable, preventing trustworthy visual grounding.
```

A.5.4 STAGE#3: ACTION RECOVERABILITY FILTERING.

We use prompt of the CoT agent configuration, which is detailed in the following section (§B.2).

B AGENT DESIGN

All prompts are sent as user messages, followed by the images in the order specified in each prompt.

B.1 NONE-COT AGENT

```
You are an expert mobile UI automation agent. You are given two
    screenshots of a mobile application:
- IMAGE 1: The UI state BEFORE an action is taken.
- IMAGE 2: The UI state AFTER an action is taken.

Your task is to deduce the single user action performed on IMAGE 1 that
    caused the transition to IMAGE 2.

### INSTRUCTIONS:
1. Compare IMAGE 1 and IMAGE 2 and identify what changed.
2. Locate the exact UI element in IMAGE 1 that was interacted with.
3. Determine the absolute pixel coordinates (x, y) of the CENTER of that
    UI element in IMAGE 1. The coordinate (0,0) is the top-left corner.
4. If the screen scrolled, determine the direction. "scroll down" means
    the page content moved up so that lower content became visible.

### ACTION SCHEMA:
Choose exactly ONE action from the following formats:
```

```
1. {"action_type": "click", "x": <integer>, "y": <integer>}
2. {"action_type": "input_text", "text": "<text>"}
3. {"action_type": "scroll", "direction": "up" | "left" | "right" | "down
    "}
4. {"action_type": "navigate_back"}
5. {"action_type": "long_press", "x": <integer>, "y": <integer>}
6. {"action_type": "wait"}

### OUTPUT:
Return exactly one valid JSON object containing the action. Do not
    include reasoning, prose, or Markdown fences.
```

B.2 CoT Agent

```
You are an expert mobile UI automation agent. You are given two
    screenshots of a mobile application:
- IMAGE 1: The UI state BEFORE an action is taken.
- IMAGE 2: The UI state AFTER an action is taken.

Your task is to deduce the single user action performed on IMAGE 1 that
    caused the transition to IMAGE 2.

### INSTRUCTIONS:
1. Compare IMAGE 1 and IMAGE 2. Identify what changed (e.g., a new menu
    opened, the screen scrolled, a button changed color, text was entered)
    .
2. Locate the exact UI element in IMAGE 1 that was interacted with to
    cause this change.
3. Determine the absolute pixel coordinates (x, y) of the CENTER of that
    UI element in IMAGE 1. The coordinate (0,0) is the top-left corner of
     the image.
4. If the screen scrolled, determine the direction. Note: "scroll down"
    means the page content moved up so that lower content became visible.

### ACTION SCHEMA:
Choose exactly ONE action from the following formats:
1. {"action_type": "click", "x": <integer>, "y": <integer>}
2. {"action_type": "input_text", "text": "<text>"}
3. {"action_type": "scroll", "direction": "up" | "left" | "right" | "down
    "}
4. {"action_type": "navigate_back"}
5. {"action_type": "long_press", "x": <integer>, "y": <integer>}
6. {"action_type": "wait"}

### OUTPUT:
First, provide a brief, step-by-step reasoning of the visual changes and
    how you deduced the action.
Then, output the precise action in a standard JSON code block.
```

B.3 SoM Agent

The placeholder {components} is replaced with one JSON record per selected component from the before-state UI hierarchy. Each record contains its index, bounding box, class, interaction flags, and available text or identifiers. The indexes match the numbered boxes in the marked screenshot.

```
You are an expert mobile UI inverse-dynamics agent. You are given three
    images:
- IMAGE 1: The raw UI state BEFORE an action.
- IMAGE 2: The same BEFORE state with Set-of-Mark boxes on GUI components.

- IMAGE 3: The raw UI state AFTER the action.
```

```
Infer the single action on the BEFORE state that caused the transition.
    For component-targeted actions, select an index that is visible in
    IMAGE 2 and listed below. All indexes refer to BEFORE-state GUI
    components.
High-value interaction targets are preferred. If the hierarchy exposes no
     such targets, visible components with valid geometry are used as a
    fallback. Duplicate boxes are omitted.

### BEFORE-STATE GUI COMPONENTS:
{components}

### ACTION SCHEMA:
Choose exactly ONE action:
1. {"action_type": "click", "index": <integer>}
2. {"action_type": "input_text", "index": <integer>, "text": "<text>"}
3. {"action_type": "scroll", "direction": "up" | "left" | "right" | "down
    ", "index": <integer>}
4. {"action_type": "navigate_back"}
5. {"action_type": "long_press", "index": <integer>}
6. {"action_type": "wait"}

For a whole-screen scroll, select the index of the full-screen or
    scrollable container. "scroll down" means content moves up and lower
    content becomes visible.

### OUTPUT:
Return exactly one valid JSON object. Do not include reasoning, prose, or
     Markdown fences.
```