

Supplementary Material of “Scalable Video-to-Dataset Generation for Cross-Platform Mobile Agents”

	iOS	Android	Total
Train	9,755	9,970	19,725
Val	246	249	495
Test	50	50	100
Total	10,051	10,269	20,320

Table A. Distribution of videos across different splits in MONDAY. Validation and test sets are manually balanced between platforms, while training set maintains natural distribution from collection process.

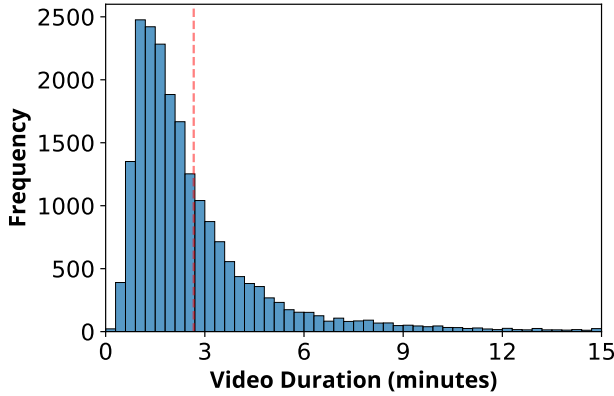


Figure A. Distribution of video duration in minutes. Red vertical dotted line stands for the average duration of 2.66 minutes. The majority of videos (77.8%) fall between 1-5.5 minutes, with a peak at 1.05 minutes.

A. More Statistics about MONDAY Dataset

A.1. Dataset Distribution

Our dataset is split into 19,725 training videos, 495 validation videos, and 100 test videos, as shown in Table A. The validation set contains an equal distribution of 246 iOS and 249 Android videos, while the test set maintains the same balanced 50/50 split between platforms. The training set includes 9,755 iOS and 9,970 Android videos, reflecting the natural distribution from our collection process.

As shown in Figure A, our dataset primarily consists of concise, focused instructional videos with an average duration of 2.66 minutes. The duration distribution shows a clear peak at 1.05 minutes, with 77.8% of videos falling between 1-5.5 minutes. This distribution reflects the typical length of mobile OS instructional content, which tends to focus on specific, well-defined tasks.

The distribution of actions in our dataset reflects real-

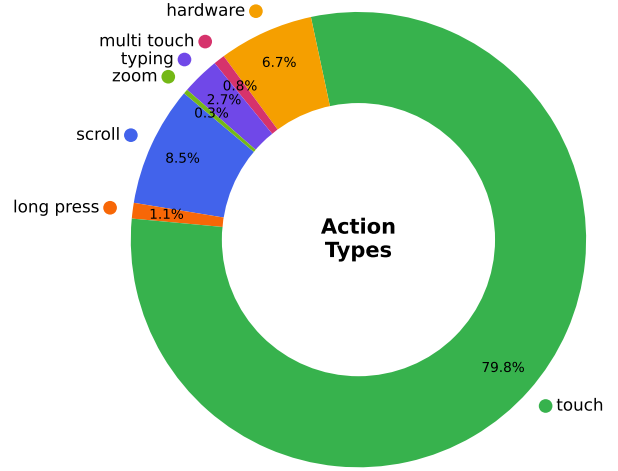


Figure B. Action type distribution in our dataset shows touch actions dominate at 79.83%, followed by scroll (8.53%) and other actions.

world usage patterns, as illustrated in Figure B. Touch actions are the most frequent (79.83%), followed by scroll (8.53%), hardware interactions (6.73%), typing (2.68%), long press (1.11%), multi touch (0.80%) and zoom (0.32%).

In terms of app coverage, we checked which mobile app each video of MONDAY is for: it includes 2,479 unique apps across 20,337 videos. The distribution between OS native and third-party apps (37.6% : 62.4%) demonstrates balanced representation of mobile device usage. Third-party app usage aligns with real-world scenarios, as shown by top applications: Instagram (3.72%), Facebook (2.60%), YouTube (2.08%), Twitter (1.86%), WhatsApp (1.75%), and so on.

A.2. Computational Cost Analysis

Our framework’s processing time is proportional to the inference time of its core components: one Paddle OCR inference and two GroundingDINO inferences (for phone screen and icon detection) per frame, plus three GPT-4o queries per action identification. For a typical three minute video, the total processing time prior to the GPT-4o is approximately 9.7 minutes on a single NVIDIA Titan Xp GPU. The total cost for identifying actions for the 20,320 videos with GPT-4o was \$6976, approximately \$0.34 per video.

To better compare its effectiveness, we measure the cost when we ask the annotator to annotate the scene detection

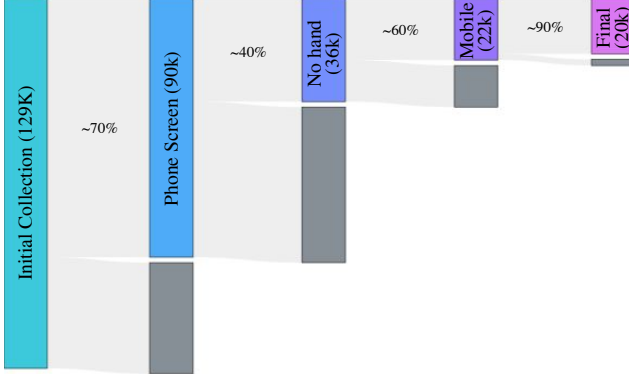


Figure C. Filtering stages in video collection process. Starting from 129K YouTube videos with English transcripts and duration under 15 minutes, each stage progressively filters videos to ensure quality and relevance.

and action identification for 100 test videos used in Section 4.1.1. Scene transition and action annotation takes 12 minutes and costs \$5.76 per video on average from an expert annotator. If there are good open-source models that perform reasonably well for action identification, then we can further reduce the cost by replacing GPT-4o with these alternatives.

B. Details about Video Collection

Our dataset collection process starts by mining mobile OS-related content from CommonCrawl web posts in the C4 [40] and Dolma [46] datasets. To ensure the mobile OS navigation topic, we first filter these posts using an expanded version of AndroidHowTo’s domain whitelist [23], which we augmented to include iOS-related websites alongside the original Android domains. We then employ GPT-3.5 Turbo Instruct [36] to analyze the main body text of each filtered post, identifying titles that describe iOS/Android phone navigation tasks (responding with “N/A” for irrelevant content). These extracted titles are then used as search keywords for collecting relevant YouTube videos.

From our initial collection of 129K videos that have English transcripts and are shorter than 15 minutes, we implement a multi-stage filtering process, as shown in Figure C:

- Process videos at 2 FPS using GroundingDINO, requiring successful detection in at least for 30 seconds (retaining 70% of videos)
- Process videos using Google MediaPipe hand landmark detection and filter out videos where hands appear, ensuring clean views of the interface (keeping 40% of remaining videos)
- Sample 5 frames in equidistance and ask GPT-4o to determine OS type (‘iOS’, ‘Android’, ‘Windows Mobile’, ‘BlackBerry OS’, ‘Multiple OS’, or ‘None’) and device type (‘Phone’, ‘Tablet/Pad’, ‘Watch’, ‘Laptop’,

‘Multi-device’, or ‘None’), preserving 60% of videos

- Remove videos with more than 55 detected scenes to ensure focused, single-task demonstrations
- After sampling the evaluation dataset, remove contaminated videos identified by an n-character overlap [34] (n=30) in video titles

This multi-stage filtering process results in our final dataset of 20K videos capturing clear, unobstructed mobile OS navigation procedures while retaining narrative context through transcripts.

C. Details about MONDAY Framework

Our framework, illustrated in Figure 2, consists of three main components working together to extract mobile OS navigation procedures from instructional videos. The framework begins with scene transition detection (Section 3.2), which identifies meaningful state changes in the mobile interface using OCR-based analysis. This is followed by UI element detection (Section 3.3.1), which combines icon detection and text recognition to identify interactive elements. Finally, our three-step action identification process (Section 3.3.2) leverages these detected components along with temporal context and to determine precise user actions. We will release our complete framework implementation upon acceptance to facilitate future research in mobile OS navigation.

C.1. Scene Transition Detection

For phone screen detection, we use GroundingDINO [26] for all frames in 2 FPS with the following parameters:

- Box confidence threshold: 0.25
- Text confidence threshold: 0.25
- Caption prompt: “phone screen”

During this process, GroundingDINO may occasionally fail to detect the phone screen in some frames, particularly during in-video animations and camera adjustments. To handle such cases, we apply linear interpolation between successfully detected frames within a 3-second window, ensuring continuous phone screen tracking throughout the video.

After detecting the phone screens, our OCR-based scene transition detection algorithm operates as follows:

- Extract text from consecutive frames in 4 FPS using Paddle OCR [21]
- Compute the Levenshtein distance [20] between the text in an identical location but in adjacent frames
- Mark as transition if the distance exceeds 20% of the number of original text characters

We apply several refinements to ensure robust transition detection:

- Filter OCR results by confidence score (> 0.9) to focus on reliable text detections
- Ignore text detected in top 5% and bottom 10% of the screen to avoid system-specific UI elements

- Merge transitions occurring within 0.4 seconds to handle animation effects
- Consider temporal context up to 2 seconds before and after each potential transition for verification
- Apply text normalization using regular expressions to handle minor rendering variations

When multiple transitions are detected in close proximity, we select the most representative frame for each transition segment, typically choosing the frame closest to the temporal midpoint between transitions. This approach helps capture stable states while filtering out intermediate animation frames.

C.2. UI Element Detection

Our UI element detection combines icon detection using GroundingDINO and text detection using OCR, followed by careful filtering to identify genuine interactive elements. The system employs a two-stage approach.

First, we detect potential UI elements using GroundingDINO with relaxed thresholds:

- Box confidence threshold: 0.04
- Text confidence threshold: 0.25
- Caption prompt: “icon”

We deliberately use a lower box confidence threshold here to maximize UI element detection coverage, relying on our subsequent filtering steps to remove false positives.

Then, we apply mobile-specific filtering heuristics:

- Integrate OCR-detected text element boxes
- Remove oversized elements (box area > 0.4 of screen)
- Merge overlapping boxes with significant intersection ($\text{IoU} > 0.5$)
- Filter by aspect ratio and relative positioning

For text elements, we perform additional processing to identify interactive text components like context menu options (e.g., ‘more’ button in text posts) or actionable labels (e.g., ‘unsubscribe’ button in emails):

- Split text by natural spaces
- Compute box for each text segment, split by a white space, based on character count
- Set dominant color as background
- Select next dominant color as text color
- Add box if color difference in LAB space > 50 (with step-wise reduction by 5 until text box detection succeeds)

C.3. Action Identification

Our action identification process follows a three-step approach to ensure accurate action prediction:

1. **Scene Summary:** First, we analyze each frame independently to understand the overall UI layout and component relationships, creating a comprehensive scene description without any preconceptions about actions.

2. **Initial Action Identification:** Using the scene summaries and temporal context from adjacent frames, we identify potential actions that could lead to the observed state changes, considering both visible UI elements and narrative guidance.

3. **Refined Action Identification:** Finally, we employ a zone-based system for precise spatial localization of the predicted action, dividing the screen into five vertical zones based on UI element positions. Zones are calculated as follows:

- Zone 1: 0.0 - 45.0% of screen height (top)
- Zone 2: 12.5 - 57.5% of screen height
- Zone 3: 25.0 - 70.0% of screen height
- Zone 4: 37.5 - 82.5% of screen height
- Zone 5: 55.0 - 100.0% of screen height (bottom)

As a result of three step identification, MONDAY captures the following categories of mobile OS device control:

- **Single-point actions:**
 - touch: Single tap at specific coordinates
 - long press: Extended press at specific coordinates
- **Motion-based actions:**
 - scroll: [up, down, left, right]
 - zoom: [in, out]
 - multi touch: swipe (up/left/right), four-finger pinch, double tap, rotate content (clockwise/counterclockwise), multi taps
- **Hardware interactions:**
 - Navigation: home, recent apps (Android-only), back double/triple taps
 - Device controls: volume up/down, power, authentication
 - Physical actions: shake, orientation change (clockwise/counterclockwise), silent mode change on/off
- **Text input:** Typing actions with corresponding text content

D. Annotation of the Evaluation Dataset

We employed two experienced annotators familiar with both iOS and Android platforms for evaluation dataset. The annotation process consisted of two main tasks:

Scene Transition detection. Annotators identified transition points in videos, with timestamps aligned between annotators using minimum distance matching. When transition counts differed between annotators, a third annotator reviewed the unmatched timestamps to determine the correct transitions.

Action identification.

Using our scene transition detection output, annotators labeled actions between consecutive scenes using Label Studio with a custom interface. The annotation interface supported the layout in Listings 1.

```

<View>
  <Header value="File: $image"/>
  <RectangleLabels name="label" toName="image" fillOpacity="0.7" strokeWidth="3">
    <Label value="click" background="blue"/>
    <Label value="long_press" background="red"/>
  </RectangleLabels>
  <TextArea name="typing" toName="image" editable="true" required="false"
    maxSubmissions="1" placeholder="typed_text"/>
  <Image name="image" value="$image"/>
  <Choices name="other_actions" toName="image" choice="multiple">
    <Choice alias="end_of_video" value="End of the video"/>
    <Choice alias="ambiguous" value="Ambiguous"/>
    <Choice alias="hardware_recentapps" value="Hardware - Recent Apps (Android left key)"/>
    <Choice alias="hardware_home" value="Hardware - Home"/>
    <Choice alias="hardware_back" value="Hardware - Back (Android right key)"/>
    <Choice alias="hardware_authentication" value="Hardware - Authentication"/>
    [Additional action choices...]
  </Choices>
</View>

```

Listing 1. Label Studio interface configuration for action annotation.

When cases were ambiguous (no clear single action between scenes), annotators marked them as ‘ambiguous’ and these were excluded from evaluation. For any disagreements between annotators, a third annotator made the final decision.

Annotation was conducted at a rate of \$16/hour, with each annotator spending approximately 6 hours on scene transition detection and 7 hours on action identification. The presence of ground-truth video and annotator expertise in both platforms contributed to high initial agreement rates. We followed this exact same annotation protocol and quality control process when creating our Windows Mobile test set of 50 videos, ensuring consistent evaluation criteria across all platforms.

E. More Examples from Dataset Collection Method Evaluation

In this section, we provide additional examples demonstrating the effectiveness of our framework components. Figure D shows extended cases where our OCR-based scene transition detection successfully handles challenging scenarios. Figure E illustrates our UI element detection system’s ability to handle complex interface layouts. Figure F presents comparisons between our multi-step action identification approach and simpler variants.

F. Human Evaluation of the MONDAY Dataset

We conducted a human evaluation involving 10 workers examining 100 randomly sampled sequences in MONDAY training set, with each sequence reviewed by two people.

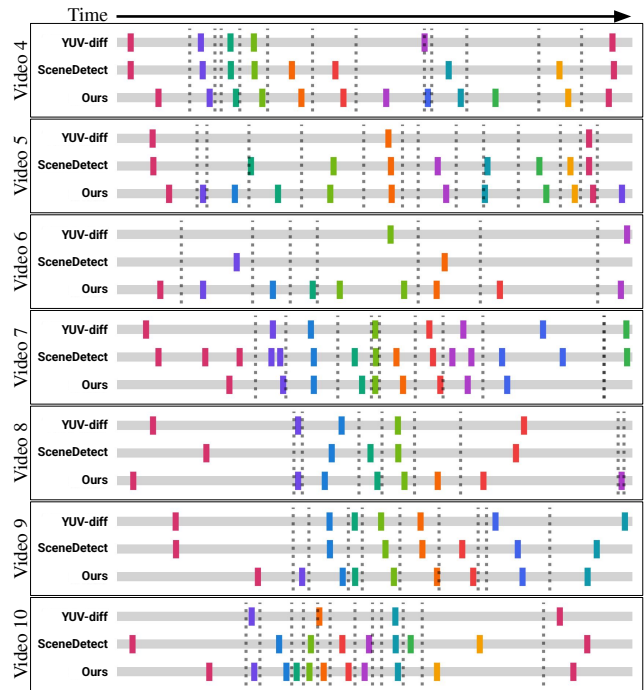


Figure D. Additional examples of scene transition detection results across different interface configurations. Our OCR-based method successfully handles most transitions, though it missed one segment in Video 5 and detected two segments in Video 6. Even with these edge cases, our approach achieves more accurate transition detection compared to baseline methods. See Section 4.1.2 for detailed experimental settings.



Figure E. Additional comparisons between (a) OmniParser [28] and (b) our UI element detection module. While OmniParser detects more boxes in the first column, many are not interactable elements. In the next three columns, OmniParser fails to detect important actionable UI elements (e.g., back button, delete button). The last two columns show OmniParser’s consistent failure to detect the lower portions of home screen icons.

The evaluators assessed whether the identified action is accurate, inaccurate, or not enough information to answer, based on the current, two previous and two next scenes with the title. Workers found that 80.40% of 250 sampled actions were accurate, while ‘not enough information (8.60%)’ primarily stemmed from either insufficient context window coverage or incomplete user configuration details. This human evaluation, along with our model’s test accuracy of 80.90%, indicates inherent task complexity due to incomplete context and interface ambiguity.

G. Details about Model Training Experiment

G.1. Training Details

We apply LoRA finetuning [15] for all models using the PEFT library [30] with its default configuration on their public repository: $\text{lora}_\alpha = 16$, $\text{lora}_r = 64$, $\text{lora}_{\text{dropout}} = 0.05$ for SeeClick, and $\text{lora}_\alpha = 32$, $\text{lora}_r = 8$, $\text{lora}_{\text{dropout}} = 0.05$ for Llama-3.2.

We first create MONDAY-induced variants of SeeClick [8] and Llama3.2 [31], named SeeClick-MONDAY and Llama3.2-MONDAY, by fine-tuning them on MONDAY. For SeeClick-MONDAY, we fine-tune SeeClick for 10 epochs using the AdamW optimizer (learning rate: $1e-5$, cosine decay, batch size: 16). The checkpoint from epoch 7 is selected. For Llama3.2-MONDAY, we fine-tune Llama3.2 for 10 epochs using AdamW (learning rate: $1e-4$, StepLR with gamma: 0.85, batch size: 24). The checkpoint from epoch 10 is selected.

Next, both the original and MONDAY-induced models are trained for 10 epochs on either of AitW and AMEX datasets using the AdamW (learning rate: $3e-5$, batch size:

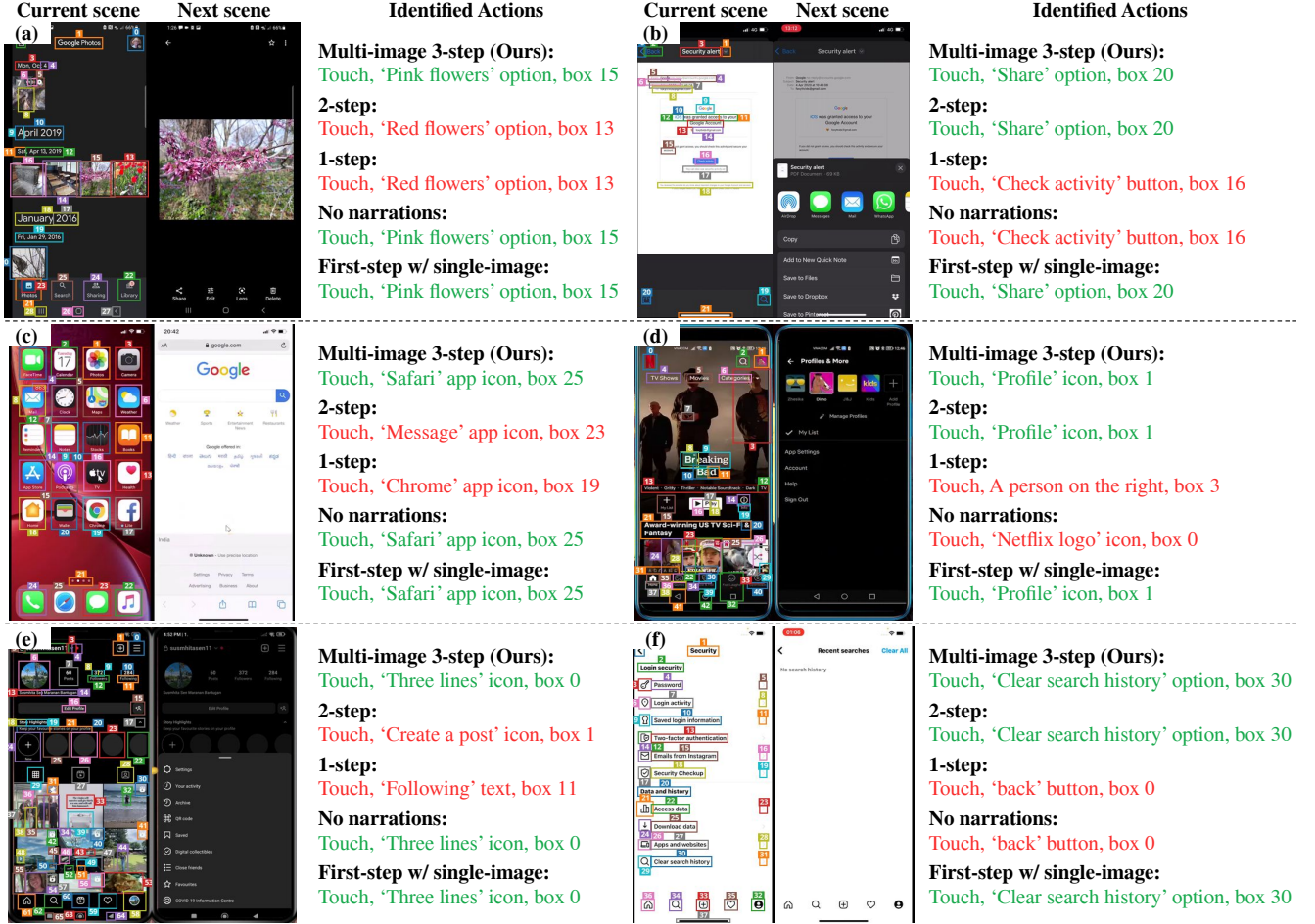


Figure F. Additional comparisons of action identification results between different approaches. The examples highlight two common types of errors: (a,c,e) selecting nearby but incorrect UI elements, as shown in the left column examples, and (b,d,f) cases requiring complex reasoning with audio transcription (ASR) for correct identification, as demonstrated in the right column examples.

16). We select the checkpoint with the lowest validation loss for evaluation. Learning rate schedulers follow the settings in their public repositories: cosine decay for SeeClick and StepLR (gamma: 0.85) for Llama-3.2. Each training sample consists of:

- Current screen image
- Task description
- Previous 4 actions as context (list of action types, coordinates, and typed texts)

G.2. Unifying the action space for comparison

To evaluate the finetuned models on the AitW, AMEX, MONDAY, and Windows Mobile test sets simultaneously, we focus on touch operations along with long press and typing actions. These actions have clear one-to-one mappings between the datasets and represent fundamental mobile OS interactions, covering 78.51% of the AitW test set, 82.60% of the MONDAY test set and 94.39% of the Windows Mo-

bile test set. For touch actions, we evaluate coordinate predictions against ground truth interaction regions. Typing actions are validated using flexible text matching, considering a prediction correct if the predicted text exactly matches the reference text or if either contains the other. We believe this focused evaluation approach allows for meaningful comparisons while acknowledging the diverse interaction patterns across mobile platforms.

G.3. Expanded Results

On the AitW dataset, Table B expands on the summary results in Table 5 by providing task-specific performance across five categories: General, Google (short for GoogleApps), Install, Shopping (short for WebShopping), and Single. The results show that the MONDAY-finetuned models consistently outperform the AitW-finetuned baselines in all evaluation categories, demonstrating their robustness in handling diverse tasks.

Finetuned Models	Test set								
	AitW						AMEX	MONDAY	Windows Mobile
	General	Google	Install	Shopping	Single	Avg			
<i>AitW-finetuned from:</i>									
SeeClick	63.19	68.67	64.26	79.09	60.63	67.17	47.23	37.45	35.61
SeeClick-MONDAY	62.83	69.48	64.50	79.56	68.13	68.90	47.76	60.71	47.56
<i>AMEX-finetuned from:</i>									
SeeClick	33.45	46.99	32.25	35.31	40.94	37.79	68.19	40.88	37.56
SeeClick-MONDAY	35.04	41.37	34.52	43.32	47.19	40.29	80.00	59.98	51.95
<i>AitW-finetuned from:</i>									
Llama-3.2	55.93	63.45	58.08	68.87	48.44	58.96	43.74	35.62	23.90
Llama-3.2-MONDAY	62.83	71.49	66.69	77.58	62.19	68.16	56.62	53.98	45.85
<i>AMEX-finetuned from:</i>									
Llama-3.2	28.67	29.32	27.54	31.94	31.56	29.81	61.30	37.33	23.17
Llama-3.2-MONDAY	37.88	42.57	38.83	49.59	45.94	42.96	72.36	57.04	47.32

Table B. Comparison of navigation action accuracies with the original pre-trained models (SeeClick, Llama-3.2) vs. the corresponding MONDAY-induced variants (SeeClick-MONDAY, Llama-3.2-MONDAY). Results on AitW [41] test set are broken down by their original evaluation categories alongside overall averages. The MONDAY-induced variants achieve higher performance across different mobile platforms, including significantly better adaptation to the previously unseen mobile platform (Windows Mobile).

H. Expanded Related Work

H.1. Cross-Domain GUI Datasets and Approaches

Early GUI agent benchmarks often focused on simple, single-step tasks or were confined to a single platform, limiting cross-environment generalization [44]. Recently, there have been efforts to scale up data collection for web and GUI-based environments to support the training of agents on a wider range of computer interaction tasks. Agent-Trek [50] simulates actions in a virtual environment based on tutorial text, with step-by-step instructions from GPT-4o. WebArena [56] offers a high-fidelity browser simulation with complex, long-horizon web tasks. Mind2Web [10] collects crowdsourced demonstrations on real-world websites, though data collection is expensive and limited to web domains. GUI-World [1] spans multiple platforms (web, mobile, desktop), but is restricted to question-answering rather than full action-based tasks.

While simulator-based approaches in web and computer OS domains can extend to Android via emulators, iOS’s closed APIs hinder automated interaction extraction, limiting multi-platform coverage. MONDAY avoids direct GUI access by leveraging YouTube videos and automatically detecting scenes and actions. Unlike simulators, which provide built-in interaction logs, MONDAY tackles data extraction using OCR-based scene segmentation, UI detection via GroundingDINO, and a three-step action identification pipeline. This approach is also adaptable to web and desktop GUIs, although higher resolutions and complex interactions may introduce new challenges.

I. Episode Examples

We present example episodes from our dataset to demonstrate the effectiveness of our action identification framework. The examples are organized into three categories: perfectly identified sequences, near-miss cases with multiple valid action paths, and challenging cases involving platform-specific operations. Figures G and H showcase successful action identification sequences on Android and iOS platforms, respectively. In these examples, our framework correctly identifies all user interactions, demonstrating its robustness across different mobile operating systems. Figures I and J illustrate cases where multiple valid interaction paths exist. Our framework typically selects the most direct path to accomplish the task, though this may occasionally differ from human demonstrations. Figures K and L present challenging scenarios involving platform-specific operations or security features. These examples highlight current limitations in handling specialized interactions like secure input or complex scrolling patterns.

Video Title: “How to Delete A Direct Message on Twitter”

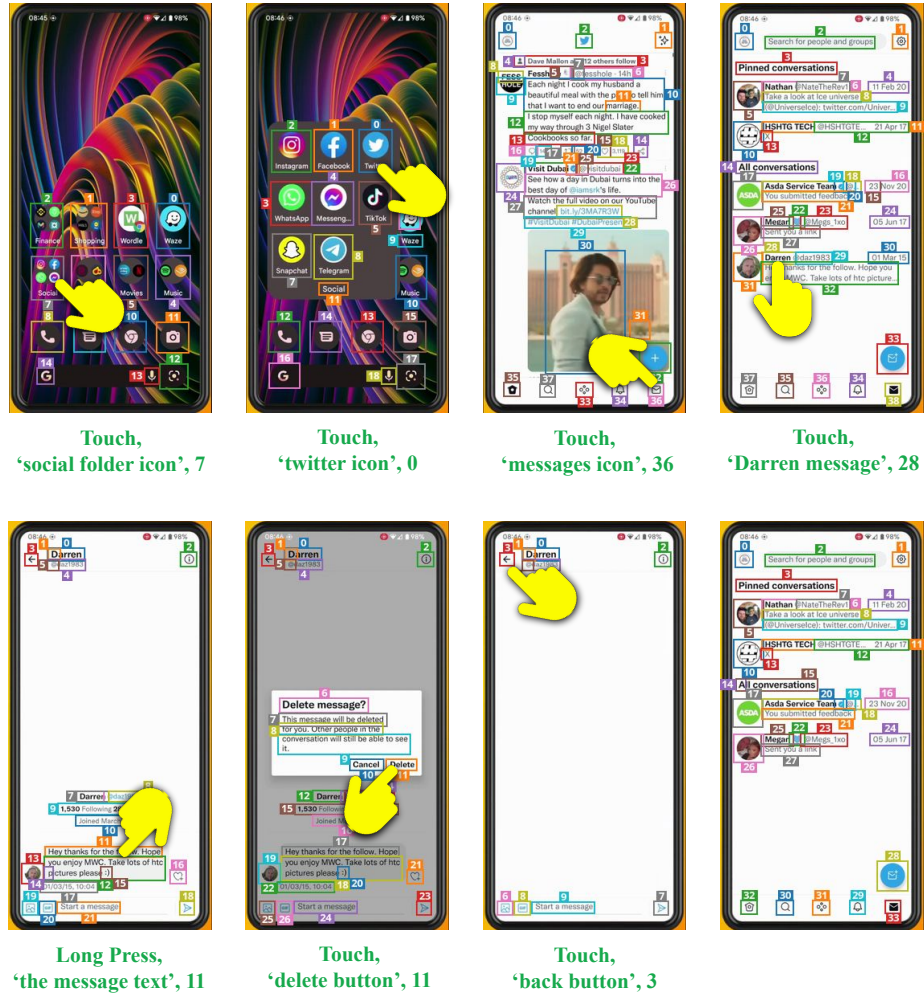


Figure G. Example of perfect action identification for deleting a direct message on Twitter in Android. Each touch and long press action is annotated with the corresponding box ID and visual indicator.

Video Title: “How to Clear Cache in Telegram App to Save Space on iPhone”

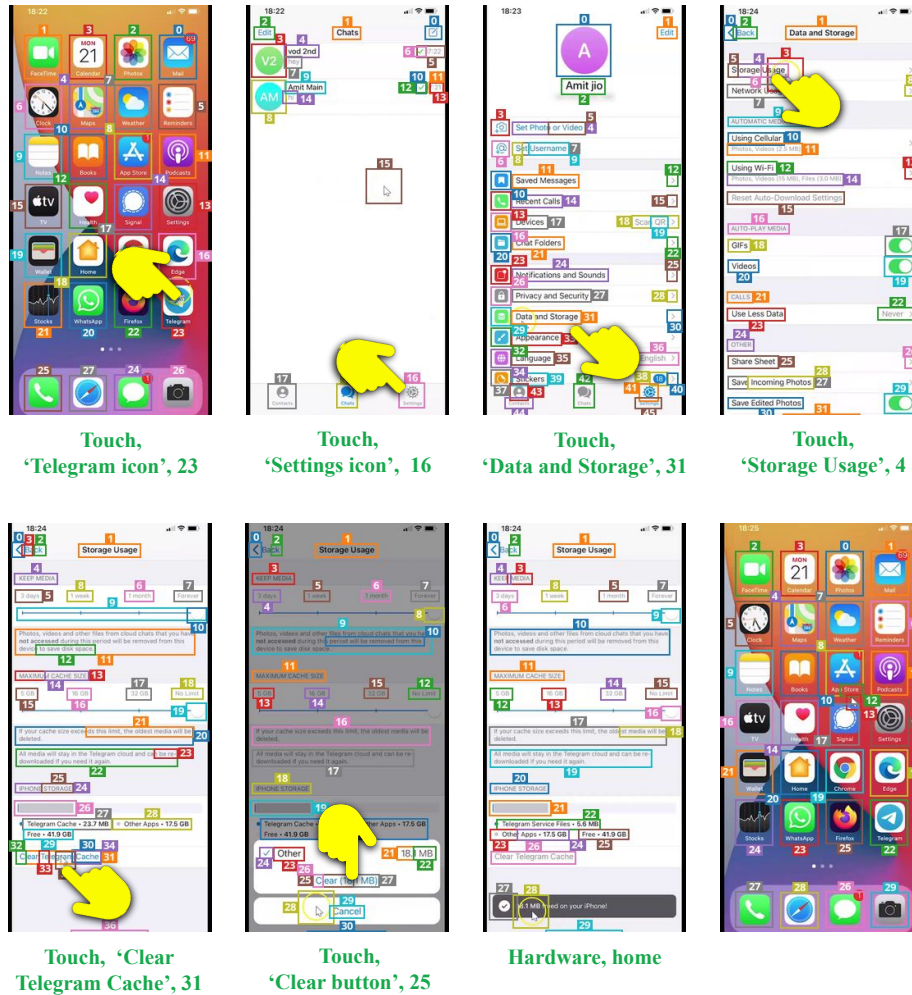


Figure H. Example of perfect action identification for clearing Telegram cache on iOS. Each touch action is labeled with the corresponding box ID and highlighted with a visual indicator.

Video Title: “How to Sign Out of All Devices on Netflix”

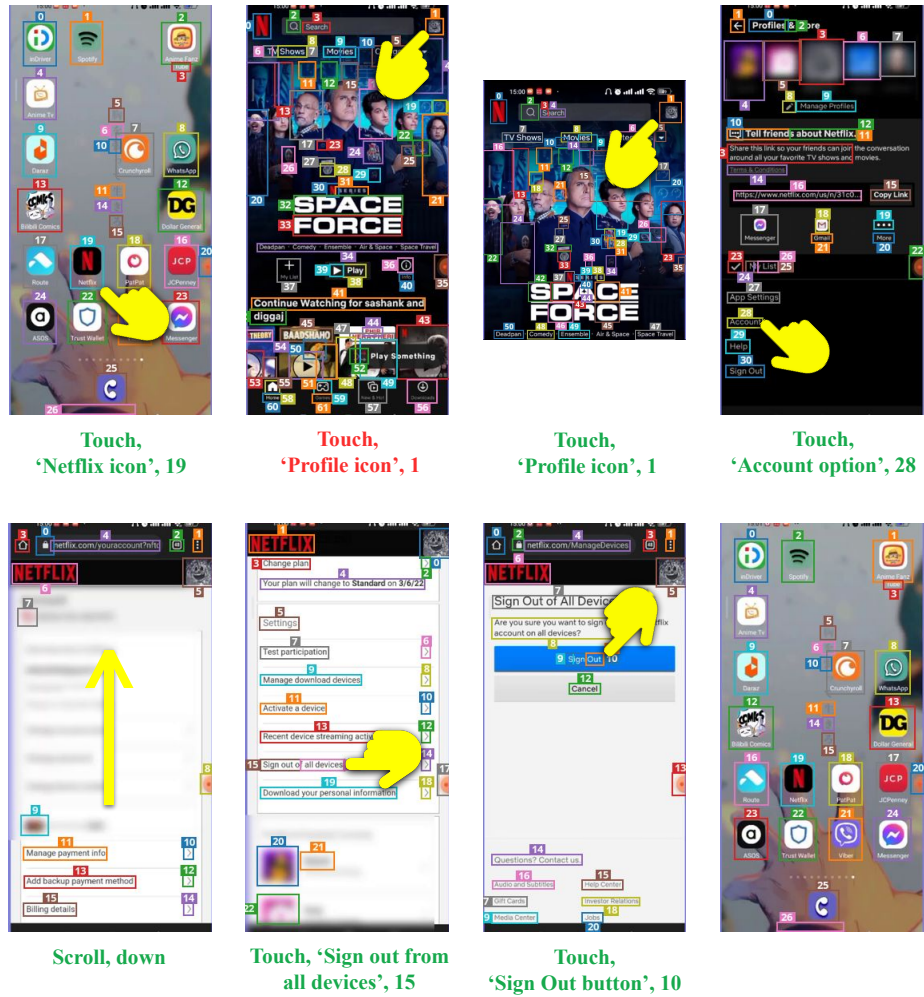


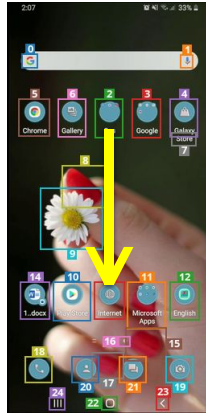
Figure I. Example showing path selection behavior for signing out of all Netflix devices on Android. Green indicates correct actions, red indicates alternate valid actions that could achieve the same goal.

Video Title: “How to Manage and Delete Your Alexa History and Recordings”



Figure J. Example showing path selection behavior for managing Alexa history and recordings on iOS. Green indicates correct actions, red indicates alternate valid approaches that were not selected.

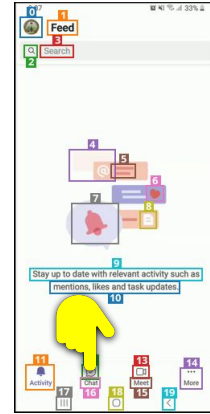
Video Title: “How to Block Someone on Microsoft Teams”



Scroll, up



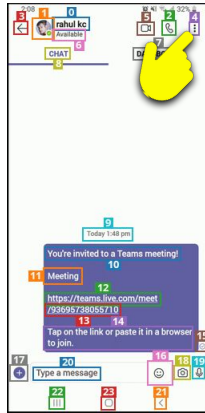
Touch,
'Teams app', 28



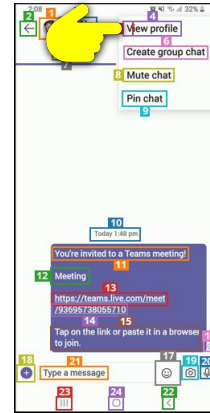
Touch,
'Chat icon', 16



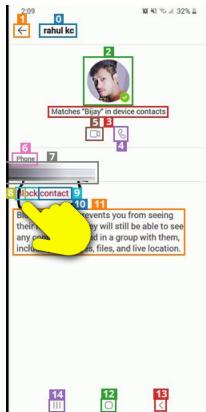
Touch,
'raahul kc chat', 8



Touch,
'Three dots icon', 4



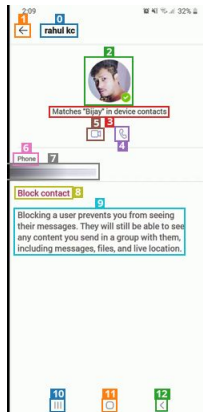
Touch,
'View profile', 4



Touch,
'Block contact', 8



Touch,
'Unblock content', 6



Hardware, home



Figure K. Example identifying scrolling direction in Android. Green indicates correct actions, red shows incorrect scrolling direction prediction.

Video Title: “How to Reset Keyboard Dictionary on iPhone”



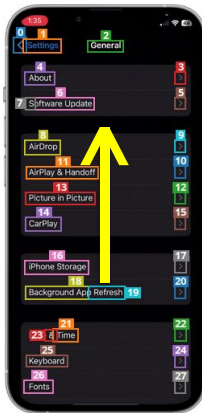
Touch,
'Settings icon', 3



Scroll, down



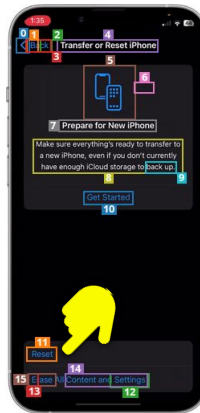
Touch,
'General option', 29



Scroll, down



Touch, 'Transfer or
Reset iPhone', 21



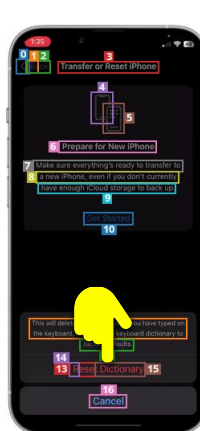
Touch,
'Reset button', 11



Touch, 'Reset Keyboard
Dictionary', 16



Touch,
'Cancel button', 3



Touch,
'Reset Dictionary', 15



Hardware, home



Figure L. Example showing handling of authentication challenges when resetting keyboard dictionary on iOS. Green indicates correct actions, red shows where the system selected cancel instead of handling passcode entry.