Published: 10 September 2025

RESEARCH-ARTICLE

On-Device Interaction Mining

DENIZ ARSAN, Siebel School of Computing and Data Science, Urbana, IL, United States

CARL GUO, Siebel School of Computing and Data Science, Urbana, IL, United States

MUHAMMAD RIZKY WELLYANTO, Siebel School of Computing and Data Science, Urbana, IL, United States

ERIK R JI, Siebel School of Computing and Data Science, Urbana, IL, United States

JERRY O. TALTON, Siebel School of Computing and Data Science, Urbana, IL, United States

RANJITHA S KUMAR, Siebel School of Computing and Data Science, Urbana, IL, United States

Proceedings of the ACM on Human-Computer Interaction, Volume 9, Issue 5 (September 2025)
https://doi.org/10.1145/3743726
EISSN: 2573-0142

On-Device Interaction Mining

DENIZ ARSAN*, University of Illinois at Urbana-Champaign, USA
CARL GUO*, University of Illinois at Urbana-Champaign, USA
MUHAMMAD RIZKY WELLYANTO, University of Illinois at Urbana-Champaign, USA
ERIK R JI, University of Illinois, Urbana-Champaign, USA
JERRY O. TALTON III, University of Illinois at Urbana-Champaign, USA
RANJITHA KUMAR, University of Illinois at Urbana-Champaign, USA

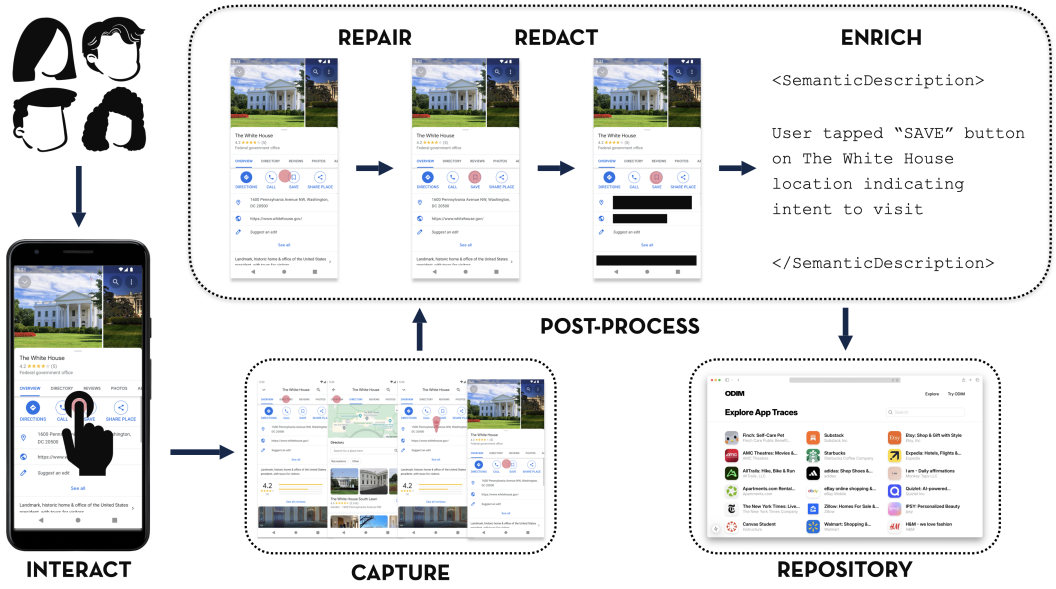

Fig. 1. ODIM is an on-device framework for Android interaction mining that produces detailed trace information without requiring complex infrastructure. Users can interact with apps on their own mobile devices while high-fidelity interaction data is captured in the background. After acquisition, traces can be post-processed to redact personally-identifiable information, repair errors in the capture, and add useful metadata like semantic descriptions. Traces are collected in an online design repository for UX and machine learning applications. The ODIM implementation and public repository are freely available at interactionmining.org, for anyone to use and contribute to.

*Both authors contributed equally to this paper.

Authors' Contact Information: Deniz Arsan, Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, deniz.arsan@gmail.com; Carl Guo, Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Champaign, Illinois, USA, carlguo2@illinois.edu; Muhammad Rizky Wellyanto, Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, wellyan2@illinois.edu; Erik R Ji, Siebel School of Computing and Data Science, University of Illinois, Urbana-Champaign, Champaign, Illinois, USA, erikrj2@illinois.edu; Jerry O. Talton III, Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, talton@illinois.edu; Ranjitha Kumar, Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, ranjitha@illinois.edu.

Interaction mining is a popular technique for capturing design and interaction data while a mobile app is being used. Over the years, researchers have leveraged interaction mining systems to build large repositories of interaction data, enabling novel, ML-based tools for UX practitioners, designers, and programmers. Existing interaction mining systems range from simple screen recorders — which are easy to use but capture sparse, unstructured data — to complex installations requiring dedicated hardware and custom OS forks — which yield rich, high-fidelity traces but are difficult to deploy outside of a lab environment. This paper presents ODIM, an on-device framework for mobile interaction mining that produces detailed trace metadata. The framework is reified in an Android implementation based on a simple APK that users can install on their personal devices. The paper outlines ODIM's design principles, describes its implementation, evaluates the system on traces collected from 100 popular apps on the Google Play Store, and discusses future avenues for scaling the utility and impact of interaction mining systems. The ODIM software, source code, and online trace repository are all freely available at interactionmining.org, for anyone to use and contribute to.

CCS Concepts: • **Human-centered computing** → *Ubiquitous and mobile computing design and evaluation methods*; **Systems and tools for interaction design**; • **Information systems** → Data mining.

Additional Key Words and Phrases: interaction mining, mobile apps, UX design repositories

ACM Reference Format:
Deniz Arsan, Carl Guo, Muhammad Rizky Wellyanto, Erik R Ji, Jerry O. Talton III, and Ranjitha Kumar. 2025. On-Device Interaction Mining. *Proc. ACM Hum.-Comput. Interact.* 9, 5, Article MHCI024 (September 2025), 18 pages. https://doi.org/10.1145/3743726

1 Introduction

Interaction mining is a popular technique for capturing design and interaction data while a mobile app is being used. As a user interacts with an app, interaction mining systems capture the user's journey in the form of an *interaction trace*, comprising sequences of UI screenshots augmented with their structural composition and render-time properties (i.e. view hierarchies) along with the user actions (i.e. gestures) performed on these screens. Since the technique was first introduced [10], interaction mining systems have led to the emergence of a trove of useful mobile UX datasets [4, 5, 8, 22, 38], which researchers have harnessed for purposes as diverse as studying UI patterns at scale [6, 12, 36, 43, 47, 53], training generative models of design [3, 21, 24, 27, 31, 33, 54, 55], and automating user actions [2, 26, 28–30, 45].

Before the advent of interaction mining techniques, granular interaction data could only be collected on a per-app basis, by instrumenting an application's source code to capture clickstream data and perform action reporting. While such instrumentation allowed researchers to conduct in-the-wild user experience studies of their *own* apps, it provided no support for collecting data about apps for which the source code and deployment process were inaccessible. Early interaction mining systems overcame this hurdle by transferring the responsibility of trace capture from the app itself to the operating system (OS), releasing custom mobile OS versions tailored for global interaction capture [10]. In this way, interaction mining enabled — for the first time — the development of centralized repositories of UX data collected from tens of thousands of *disparate* mobile applications [5, 8, 22].

Despite the proven utility of these systems, the difficulty of developing and maintaining the infrastructure required to accurately capture system-level interactions has limited their widespread adoption. As the upfront cost of launching an interaction mining effort remains high, most large UX repositories result from one-off, supervised, in-the-lab collection activities, and quickly become

ACM 2573-0142/2025/9-ARTMHCI024
https://doi.org/10.1145/3743726

outdated as popular apps are updated [39]. In response, some recent efforts have abandoned structural capture entirely, focusing instead on the use of purpose-built LLMs to predict segmentations and labels from unstructured screen recordings [54]. While these early models are promising — particularly for their ability to enable new, useful interactions like grounded conversation — their performance does not yet rival human annotation, a gap mostly attributable to the dearth of high-quality training data [34].

This paper presents ODIM, the first **O**n-**D**evice **I**nteraction **M**ining framework designed for in-the-wild capture of mobile apps, at scale, directly on personal devices, without any special hardware or operating system customization (Figure 1). To demonstrate the efficacy of the framework, we present an Android implementation, which enables users to collect interaction data simply by downloading the software and toggling its capture feature. As they interact with other apps, ODIM records data transparently in the background, capturing screenshot and view hierarchy information for each interaction as well as all the granular event data generated by the Android OS. Once a raw trace has been captured, it can be post-processed to repair missing or errant interaction data, redact personally-identifying information (PII), and enrich its constituent screens and interactions with useful semantic metadata.

The paper outlines ODIM's design principles, describes our implementation, evaluates the system on traces collected from 100 popular apps on the Google Play Store, and presents results from a small usability study. Traces are stored in a new, public interaction mining repository — available at interactionmining.org — which anyone can contribute to by downloading the ODIM Android Package (APK). Over time, ODIM aims to dramatically increase the availability of useful UX interaction data by lowering the barrier to capturing, cleaning, and sharing it.

2 Related Work

ODIM builds on previous work in interaction mining, and draws on concepts from the behavioral analytics and UX testing literature. Data collected from the system can be used to power a variety of novel data-driven and ML-backed applications.

2.1 Behavioral Analytics & UX Testing

To evaluate how digital products perform, companies use a number of systems to examine two primary data sources: behavioral analytics and UX assessments. Behavioral analytics provides large-scale, coarse-grained data which is often used to identify *how much* a certain behavior occurs within an experience. UX assessments yield smaller-scale affective data focusing on *why* a behavior occurs.

Many services provide behavioral analytics data — information like which screens users navigated to, which buttons they clicked, what kind of device they were using, how much time they spent. Platforms like Google Analytics, Adobe Experience Cloud, Flurry, Heap Analytics, Mixpanel, Amplitude, Qualtrics, and Pendo Analytics provide ways to track user activity, identify drop-off points, and aggregate user data to track key performance indicators. Human insight platforms like UserTesting and Maze facilitate the collection of affective data by curating panels of human testers for studies.

ODIM advances the state of the art in analytics and UX testing by creating a centralized, public repository for interaction data, where both quantitative and qualitative UX data can be stored and made widely available. The system accomplishes this by drastically lowering the barrier to deploying interaction mining techniques at scale and aggregating their results.

2.2 Interaction Mining Systems

Researchers have explored a number of ways to mine interaction data from the Web, mobile, and desktop devices [37, 41]. *Webzeitgeist* mined website design information and enabled data-driven web design tools [20]. *ERICA* applied design mining principles to mobile apps for the first time [10]. Interaction mining techniques have been used to find patterns across web and mobile application UIs [9, 11, 23, 35, 46, 51, 56], and led to the development of design and interaction repositories such as *Rico* [8], *Enrico* [22], *VINS* [4], *ReDraw* [38], *MoTIF* [5], and *AMP* [56].

Interaction mining systems all face challenges. *ERICA* required custom OS modifications for data capture, which are difficult to author, maintain, and distribute. The largest and most widely-cited interaction mining effort, *Rico*, facilitated data collection through a web interface by pixel-streaming the screen from a dedicated Android device. The resultant dataset — compiled in 2016, and comprising 70k UI screens from 10k Android apps — has been used by hundreds of authors for diverse purposes, but was never updated after its initial release almost a decade ago. Apple's *AMP* dataset, compiled in 2019 and comprising 77k UI screens from 4k iOS apps, was captured using proprietary iOS APIs only available to company employees; the company's terms of service prevent it from releasing the dataset, the software, or any of the models it was used to train [15, 56]. Additionally, most systems — including *Rico*, *VINS*, *ReDraw*, and the *Never-ending UI Learner* [50] — capture traces resulting from inorganic or automatic execution of mobile app flows, raising questions about the ecological validity of the resulting data.

ODIM seeks to alleviate the key limitations of all these prior systems. It requires no OS modifications, pixel streaming, or dedicated hardware: the capture app can be downloaded and installed on a user's own Android device in mere minutes. Traces are uploaded to a public repository by default, facilitating collaboration and allowing researchers to easily track new apps and emerging or evolving interaction patterns. While ODIM can be used for in-the-lab or automated UX capture, it also facilitates trace collection from end-users interacting with their own devices and accounts, and provides thoughtful tools for preserving user privacy in these settings.

2.3 Interaction Mining Applications

With the advent of large-scale design and interaction datasets, researchers leveraged them in many creative ways. Design and interaction data have been used to automatically generate semantic annotations for mobile app UIs [33], for UI design generation [3, 24], and for search and retrieval of UI designs [19, 38]. Prior work has also explored translation of UI Screens into natural language [27, 48], and how design and interaction data could be used to automate UI testing and evaluation [9, 32].

A number of interactive ML systems [1, 14] have leveraged interaction mining data. Task automation, in particular, is one area where the training data provided by interaction mining has proved useful, powering several programming-by-demonstration systems [26, 28, 29, 45] as well as tools for automating app navigation and autofilling task-relevant screens [2, 30]. More broadly, researchers have leveraged the data collected by interaction mining for purposes as diverse as aiding musical instrument creation [16], automating data science projects [49], and enabling interactive correction for grounding [25, 54].

ODIM aims to increase the availability of training data for interaction mining applications by an order of magnitude, and commensurately improve its quality, recency, and ecological validity. As part of the ODIM workflow, traces are post-processed for accuracy, checked for PII, and enriched with semantic metadata. All told, the system produces more accurate interaction data, more quickly, at lower cost.

3 ODIM Workflows

To motivate ODIM's design and implementation, we describe three representative interaction mining workflows, spanning small and large collection activities across machine learning, UX research, and personalized automation. By considering the parts of these workflows that are underserved by prior work, we can derive practical desiderata for the ODIM architecture.

Large-Scale Capture For ML Applications. Nick is a machine learning engineer who works at a company that builds digital accessibility software. Nick's current project is developing a deep vision-language model that can detect accessibility issues in client applications and generate possible solutions for them. Nick starts with an open source foundational model, but wants to fine-tune its weights on a large dataset of interaction data, labeled with accessibility problems. Nick recruits a set of one hundred participants with disabilities on a crowdworking platform, and asks them each to crawl a set of popular apps and document the issues they encounter. Once the crawls are completed, Nick downloads the resultant dataset, and uses the multi-modal traces for training. Nick needs an interaction mining platform that is easy for end-users to install on their own devices; collects semantic annotations alongside screenshots, structured element data, and gestures; and allows for the redaction of PII before traces are uploaded.

Small-Scale Capture For UX Research. Olivia is a UX researcher at an e-commerce firm, tasked with redesigning the checkout process on her company's mobile app. To evaluate an alternate design she has prototyped, she decides to run a comparative UX test across competitor apps. She recruits five customers who also have accounts with other e-commerce platforms via an internal marketing campaign, and assigns each one to test her redesign against a competitor's checkout flow. Because her design is not yet widely-deployed and must be distributed in a development build, Olivia does not wish to upload the traces to a public repository; since her design is new, she knows there are some UX bugs. Once the test is complete, Olivia will analyze the collected data to identify user-friendly UI features, potential pain points, and preferred interaction elements. Olivia needs an interaction mining system that can collect traces over apps she cannot instrument; allows her to host her own, private repository; and provides an easy way for users to correct errors in their traces.

Personal Capture for End-User Applications. Pat downloads SAVANT, a virtual assistant that maps natural language task descriptions to app UI states as task shortcuts, using data collected by interaction mining systems [2]. Pat regularly has food delivered from GrubGallop, and always orders the same set of things from the same set of restaurants. Because these tasks are so repetitive, Pat would like to automate them. Since these flows show Pat's address and credit card information, Pat doesn't want to upload them: just use the local trace as input data for SAVANT. Pat needs an interaction mining system that accurately captures complete, reproducible interaction traces; can be run entirely on a local device; and is simple enough for a layperson to install and use.

4 The ODIM System

ODIM is based on five design principles that are necessary to support the critical mass of interaction mining workflows: zero integration, instantiability, completeness, ecological validity, and privacy preservation. The ODIM system comprises two components: an Android capture app, and a web repository. The trace acquisition workflow consists of four stages: capture, repair, redact, and enrich. The first stage occurs on the app, while the remaining can be implemented on-device or in the web application. While we have released and maintain a public version of the repository — including APIs for efficient data access — private versions can be hosted as desired.

4.1 Design Principles

To ensure safe, useful, accurate data capture at scale, ODIM's design is based on five key principles. We note that no prior interaction mining system in the literature simultaneously satisfies all five.

- **Zero integration.** Interaction mining systems should support data capture over the widest possible spectrum of mobile applications, without requiring access to an app's source code or deployment process.
- **Instantiability.** Interaction mining systems must not be unduly difficult for companies and researchers to provision, operate, and maintain. This means that special-purpose hardware, custom OS forks, and resource-intensive backend services should be avoided.
- **Completeness.** Interaction mining systems should capture all the available static and dynamic elements of user experience design: UI screenshots, view hierarchies exposing the elements comprising each UI, and the user interactions performed on each screen.
- **Ecological validity.** Interaction mining systems should capture in-situ behavior wherever possible to maximize the ecological validity of their data. Systems should not mandate the use of unfamiliar devices, emulators, or sockpuppet accounts.
- **Privacy preservation.** Interaction mining systems must provide functionality to prevent personally-identifiable information from being inadvertently shared.

4.2 Capture

The first step in the ODIM framework is to capture design and interaction data on-device while an application is in use. When a user interacts with an interface, the system should capture all representations of the screen, including the interaction if possible. This captured data forms an **event**, the fundamental unit of an interaction trace, encapsulating the visual and structural properties of a UI at a particular time. Depending on the capture system's capabilities, each event may also include a user interaction (or gesture) corresponding to the captured UI. A series of sequential events forms a **trace**; events are compiled into a trace after the capture session is complete. Once a trace has been captured, it undergoes post-processing to repair incomplete or errant capture data, redact personal information, and add semantic metadata.

Our implementation leverages the Android platform's `AccessibilityService` API [17] to dynamically capture screens and user interactions while a mobile app is in use. Once enabled, the accessibility service monitors user interface actions and sends `AccessibilityEvents` to the OS in response to interactions detected by UI components. These events provide valuable information about the interaction, including details about its type, time, and a reference to the interacted element in the form of an `AccessibilityNodeInfo` object. This object is the key component that allows the accessibility service to programmatically access information about the UI and extract the structural and visual details of elements on the screen. A screen's structural composition and render-time properties are represented in the `AccessibilityService` as a tree of `AccessibilityNodeInfo` nodes, with the root node providing the starting point for extracting screen representations.

Captured events are formed via combining two data streams: system detected user interface actions, and a full-screen touch listener. When the accessibility service is activated, the app instantiates a transparent overlay spanning the entire device screen to listen for user touches. Upon detecting a touch, the app spawns two parallel threads to take a screenshot and retrieve the root `AccessibilityNodeInfo` of the active screen. The app then collects the visual properties of the UI elements on the screen via a depth-first traversal from this node, and structures this information

into a JSON blob called a **view hierarchy**. To efficiently transform the `AccessibilityNodeInfo` tree into a JSON view hierarchy, we stream node properties token by token during traversal. We save the latest screenshot and extracted view hierarchy in local storage as "incomplete" events.

While the overlay monitors touches, the accessibility service listens for `AccessibilityEvents`. To optimize performance, we listen only for the subset of `AccessibilityEvents` that correspond to view interactions (i.e. click, scroll) [42]. Upon detecting an accessibility event, the app examines its properties to determine the interaction type. `AccessibilityService` captures three basic gesture types: tap, long tap, and scroll. From the accessibility event, we extract the center `(x,y)` coordinates of the interacted element, along with (dx, dy) displacement offsets that are provided when scrolls are captured. These coordinates and interaction type form a gesture, which we combine with the most recent screenshot and view hierarchy to complete the event.

Trace collection is initiated when users launch a new app, and finalized when they return to the home screen. In the app's local storage, traces are organized by app and sorted by timestamp. The trace view presents a sequence of event screenshot thumbnails; clicking on a thumbnail displays the full screen view of the event.

4.3 Repair

After a trace has been captured, a user may wish to edit it for completeness or correctness, for instance by deleting unimportant interactions or screens (e.g., from errant key presses or clicks). In some instances, the capture system might register an erroneous gesture for a particular screen, or be unable to capture a gesture at all. In these cases, ODIM's repair step can be used to correct or add interactions to complete an event.

Android's `AccessibilityService` always correctly identifies when a user interaction has taken place via the touch listener and accurately captures the corresponding screenshot and view hierarchy. However, the data exposed through the accessibility service is not always sufficient to precisely identify the specific element interacted with, or the type of that interaction. In these cases, captured traces require repair.

For instance, if an app developer implements a custom component and fails to follow Android's accessibility guidelines it is possible for interactions with that component to fail to emit accessibility events. Precise interaction metadata may also be unavailable for interactions which drastically modify screen state: even if the event is properly emitted, it may reference a node object which was destroyed during the state change. In both of these cases, our implementation marks the captured event as *incomplete*, and flags it for repair in post-processing.

It is also possible for the ODIM app to associate an incorrect element or gesture with a captured interaction. If the user interaction fails to emit an accessibility event but quickly triggers a state change that does (e.g. if a poorly-implemented button component closes a pop-over and then scrolls the view), the app will errantly associate the subsequent gesture with the captured event. In these cases, we rely on the user to notice the mistake in the trace and correct it during post-processing.

We create a simple manual interface for repair (Figure 2, middle), allowing a user to review a trace and manually correct errant interaction data from inside the ODIM app. We also create a web implementation where users can upload their captured data from their phones to our ODIM website and correct incorrect interaction locations and types with a web interface. Both repair methods will be made available to users. In addition, some large language models demonstrate UI understanding capabilities [13]. By providing these models with the multimodal context captured by interaction mining, some parts of the repair process could be automated (Figure 2, right).

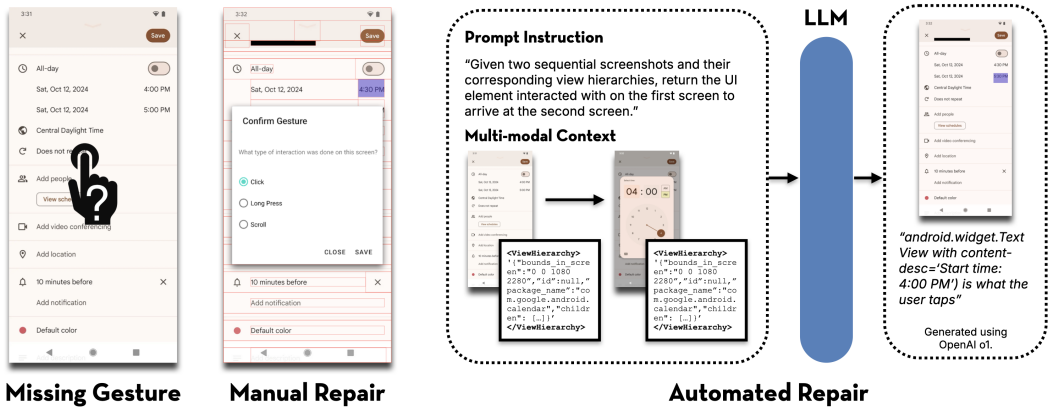

Fig. 2. For events with undetermined gestures, ODIM allows for multiple approaches for repair. The repair process takes a screen with a missing or incorrect interaction and adds the correct interaction to it (left). Our implementation uses a manual approach; users add the correct interaction to the affected screen through an interface made available on-device (middle). An alternate approach could be to automate the process by leveraging multi-modal LLMs to determine the correct gesture through providing both the affected screen and screen following the interaction (right). The prompts and outputs displayed for "automated repair" are real results using OpenAI's o1 model.

4.4 Redact

Many mobile apps display or collect personally-identifiable information (PII) and other types of sensitive data. This data may be captured by interaction mining systems either on-screen or in the associated metadata, and must be removed from traces before they can be used for research or shared.

Our ODIM implementation provides two affordances for protecting user privacy: an easy start/stop toggle to pause capture before entering sensitive information, and a scrubbing interface to review and redact PII (Figure 3, middle).

Redaction can be performed on device, before traces are sent to the cloud (for maximum privacy preservation), or via a web interface within a trusted capture environment. Redaction, too, can be automated, at least in part, using LLMs or special-purpose privacy models (Figure 3, right), though cloud-based models pose additional privacy and security concerns. We implement both on-device and web-based redaction flows in ODIM.

The on-device app interface displays redactable UI elements as bounding boxes over the screen. To redact part of a screen, a user taps the element containing the PII to toggle it, and clicks "save" when they are finished identifying elements. Redacted elements are deleted from the captured screenshot, and all corresponding text and content elements in the view hierarchy are given a special redaction tag. To gather training data for future data-driven UI privacy applications, the app prompts a user to provide a short textual justification for each redaction.

For screens with multiple elements containing similar PII, the on-device redaction interface also offers a keyword-based mode. When a keyword is entered, the system scans the view hierarchy text and content description fields and covers matched elements with a redaction marker. Users can manually adjust these selections before finalizing.

The web-based redaction interface allows users to review their screens after a trace has been transmitted to the ODIM web app. The web-based redaction method is pixel-based rather than bounding-box-based. Users redact screens by clicking and dragging over the sensitive area, which

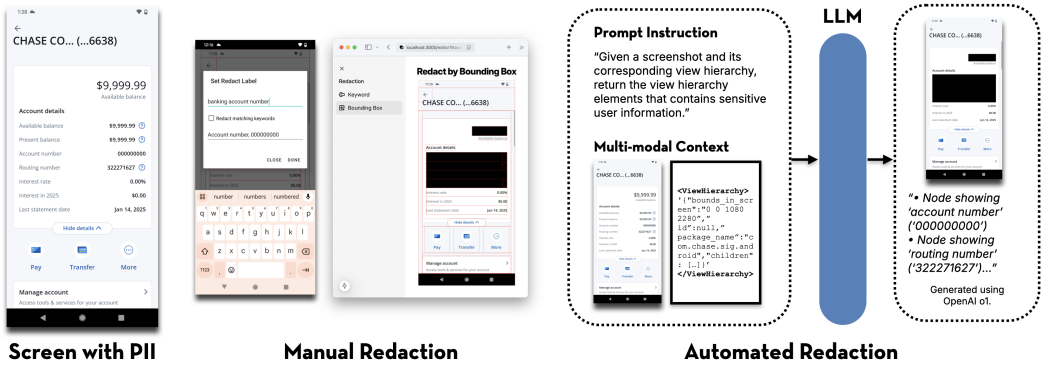

Fig. 3. ODIM allows for multiple approaches for redaction. Our implementation uses manual redaction, where users interact with an interface on the mobile device to select the element on the screen displaying PII and labeling the redaction (middle left). An alternate possible manual approach could use a web interface to redact (middle right). An automated approach to redact could utilize LLMs to detect elements displaying PII for redaction (right). The prompts and outputs displayed for "automated redaction" are real results using OpenAI's o1 model.

draws a black box over the screen region. When a user saves their redactions, the ODIM web app edits the captured screenshot and removes metadata from covered view elements with sufficiently high intersection over union values.

In rare instances, the captured UI may contain PII that is not visible on the screen, but is present in the view hierarchy metadata: for example, the view hierarchy may include content descriptions, which are set by developers to describe interface elements for accessibility purposes. ODIM therefore provides a card list of content descriptions and other text fields extracted from the view hierarchy, so that users can quickly scan suspect screens and select elements to redact.

4.5 Enrich

Interaction mining datasets can be further enriched by attaching meaningful semantic descriptions of traces and screens to the captured trace data These annotations serve as useful training data for downstream machine learning applications, such as learning UI semantics [31, 50] and automated design assistance [47].

We create a mobile and web-based implementation that enables basic annotation of traces and screens. When a user is ready to upload a trace in the mobile app, they can provide a description before upload (Figure 4, left). In the web interface, users will be prompted to provide the same trace annotation, but will also have the ability to annotate their screens as well by adding descriptions to their labelled interactions. These annotations — along with the labels collected during redaction — provide a useful starting point for further trace or screen descriptions, and can be augmented by descriptions produced automatically by language models (Figure 4, center).

4.6 Web Repository

Once a user uploads a trace, it becomes accessible on a public web application called the ODIM Explorer. The explore page of the web app presents a grid view of apps icons from which traces have been captured (Figure 5); clicking on an app icon takes users to a page with a list of recorded traces (Figure 6). Each trace is displayed as a horizontal sequence of screens, with user gestures shown as green circles and icons representing various gesture interactions. Trace metadata is shown, and a download option is provided for the raw interaction data.

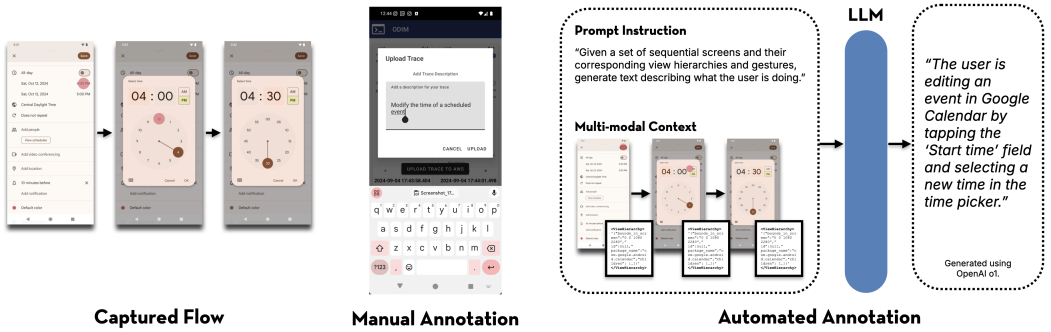

Fig. 4. Captured traces can contain rich but latent semantic data, which annotations can draw out and enable important applications (left). There are multiple approaches with ODIM to add annotations. Our implementation asks users to manually add a description for the trace (middle). Alternatively, annotations could be added by prompting a multi-modal LLM with trace data (right). The prompts and outputs displayed for "automated annotation" are real results using OpenAI's o1 model.

New trace uploads are associated with apps based on their package id. If a package id is not found in the ODIM database, the Explorer API scrapes the Google Play Store using the package name to retrieve app metadata like the description, ratings, and promotional screens. Then, a new trace collection is created for the app, and the trace is added. Users can filter and search for traces via app name; in the future, more advanced search capabilities can be added.

4.7 Public Release

The ODIM trace repository — along with the capture APK — are both publicly available at interactionmining.org. Traces uploaded to the repository are associated with individual user accounts, and periodically made public (at the platform's sole discretion) after undergoing light moderation for quality and errant PII.

ODIM is open source software. Source code for the Android capture application and web trace repository are both available on GitHub. Active development continues as of 2025, and interested contributors are encouraged to report bugs, request features, and submit pull requests.

5 Evaluation

We evaluate the ODIM framework through three lenses: capture accuracy across 100 Android apps, usability in a small-scale user study, and potential automation using large language models (LLMs). Overall results show that the ODIM app captures 100% of screens from interaction flows and 60.8% of gestures correctly. In a user study with five participants, users found our system easy to use ($\mu = 1.67$, $\sigma = 0.8$), expressed low privacy concerns about uploading their traces to a public repository ($\mu = 1.73$, $\sigma = 1.01$), and found the final trace representative of their actions ($\mu = 4.80$, $\sigma = 0.56$). All scores use a 5-point Likert scale (1 = low and 5 = high). Finally, LLMs show promising performance for automating annotation post-processing tasks like Enrich, while having mixed results for more complex tasks like Repair and Redact.

5.1 ODIM Capture Evaluation

To evaluate the quality of the ODIM app's capture, three researchers from our team recorded one to three traces on 100 popular apps from the Google Play Store, using two physical Android mobile devices and an emulator. From these traces, we measured how well ODIM captures screen (screenshot and view hierarchy) and interaction data. We selected apps by manually sampling

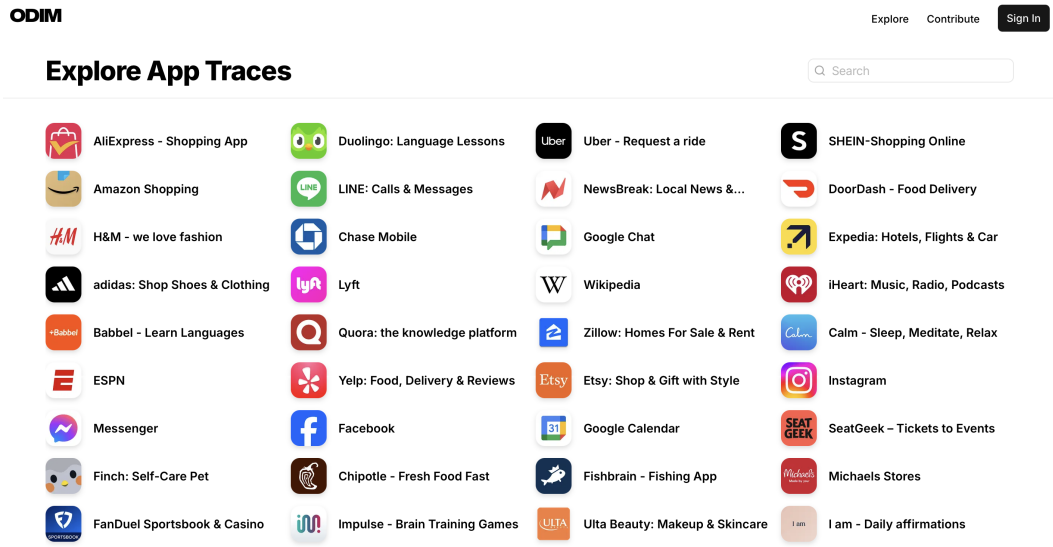

Fig. 5. The ODIM Explorer page provides a searchable, grid-based interface for browsing interaction traces from a wide range of mobile applications. Each app is represented by a clickable icon and name, allowing users to navigate to detailed traces associated with that app.

Fig. 6. An example of a recorded interaction trace in the ODIM Explorer, showing a sequence of user actions during a French language lesson in the Duolingo app. The trace captures UI transitions and user inputs such as tapping buttons and long-presses, providing insights into real-world app usage behavior.

from the top 100 charts (by downloads and ratings) across 23 categories, excluding non-mobile categories such as "Watch Apps" and "Android Auto" as well as those with sparse view hierarchies such as "Gaming." Researchers used their personal accounts to record common tasks that could be performed on each app. If a researcher was unfamiliar with an app, they would spend around five minutes exploring the app to learn how to use it before recording.

The resulting dataset comprises 159 interaction traces and 1740 individual UI screens from 100 apps spanning 23 app categories. On average, they created 1.59 traces per app ($\sigma = .50$), composed of 10.6 screens per trace ($\sigma = 6.04$). After using each app, researchers manually reviewed the recorded data on ODIM to compute overall screen and interaction accuracy.

Interaction Type	#	Success	Misclassified	Missed
Tap	1460	57%	18%	25%
Long Tap	32	63%	3%	34%
Scroll	248	83%	4%	13%
All	1740	61%	16%	23%

Table 1. Screen Gesture Capture Statistics (n = 1740). Screen capture statistics from 100 apps recorded by ODIM, split by captured interaction type. "Success" means both screen UI and gesture data was captured for event. "Misclassified" means both screen and gesture were captured but gesture was incorrect (see 4.3). "Missed" means that the gesture was not captured. Screen UIs were successfully captured for all events.

During capture, researchers documented all app UI screens and user gestures encountered, then calculated accuracy as coverage ratio of screens and gestures captured by ODIM over the total interactions explored. For this data collection, ODIM accurately captured *all* screens — screenshots and view hierarchies — corresponding to interactions (n = 1740) performed by the user. ODIM accurately recorded 60.8% of the interactions on those screens. Specifically, 57% of taps, 63% of long taps, and 83% of scrolls were correctly identified. Sometimes ODIM misclassifies the user interaction (15.8%), which can occur when an `AccessibilityEvent` is sent for a system event triggered by a user action instead of for the user action itself. We observed this misclassification for 18% of taps, 3% of long taps, and 4% of scrolls. Other times, ODIM is unable to identify the UI element on the screen which is the interaction's source (23%); this occurs for 25% of taps, 34% of long taps, and 13% of scrolls (Table 1). Fortunately, both of these error cases can be addressed by the *repair* operation.

After computing these metrics over the traces, the researchers *repaired* any screens with missing/incorrect interactions and *redacted* any privacy-sensitive information. Researchers were able to use the ODIM app's interface to manually repair *all* of the missing and misclassified interactions (n = 682). Finally, longer traces were segmented into smaller ones, annotated with semantic task descriptions, and uploaded to the ODIM Explorer repository.

During evaluation, the highest latency and most resource intensive task during capture was extracting the view hierarchy and serializing it into JSON. This operation took 100ms on average and around 300ms maximum. We also conducted rudimentary measurements for power consumption while the ODIM app was recording in the background. Using a Google Pixel 4, we measured the battery drain differential of normal use and when the ODIM app was running. For an hour of usage equally split across 10 apps (Gmail, Reddit, Instagram, Amazon, Spotify, NYT Games, LinkedIn, Slack, Etsy, and Crossy Road), we noticed a 4% (16mAh) increase in battery drain when ODIM was running.

5.2 Usability Study

To evaluate the usability of the ODIM system, we conducted a user study with five university students recruited through the research team's professional and personal networks. All participants owned Android devices, and we instructed them to record and post-process traces for two to three apps installed on their phones. We asked participants to install the ODIM APK and showed them how to record interaction traces on their device. After recording tasks on their apps, participants were asked to upload their captured traces to the Web post-processing interface to repair, redact, enrich, and publish them. Each session took approximately one hour.

Participant	Interactions Captured	Screens Removed	Gestures Repaired	Elements Redacted	Creation Time (*Minutes*)	Trace Quality	Cognitive Load	Privacy Concern
						(*5-point Likert scale, 1 = low*)		
P1	16.0	4.3	3.7	14.3	7.5	5.0	1.2	1.2
P2	10.0	4.0	4.0	15	9.1	5.0	2.7	2.7
P3	16.7	1.3	4.7	0	8.7	5.0	1.0	1.0
P4	20.3	3.0	3.3	0.3	7.5	4.7	2.1	2.5
P5	9.0	2.7	3.3	15	6.4	4.3	1.3	1.3
All	14.4	3.1	3.8	8.9	7.8	4.8	1.7	1.7

Table 2. This table shows the results of the usability study for each participant, averaged across their three tasks. The repair phase is split by screen and gesture.

During each session, we measured the time and effort to complete the capture, repair, and redact phases. After creating each trace, participants completed a post-task survey with questions selected from the NASA-TLX workload index [18] to measure the cognitive load of using ODIM. The survey also measured if they had any privacy concerns with uploading their traces to the public ODIM repository, and their assessment of how well the traces matched their interactions with the apps. All survey questions used a 5-point Likert scale (1 = low, 5 = high).

The study yielded 15 traces from 10 apps: YouTube, Google Maps, Discord, ChatGPT, Alarmy, Transit, Instagram, Steam, LinkedIn, and Duolingo. On average, it took participants 7.8 minutes (σ = 3.5) to create a trace. For each trace, participants spent on average 48 seconds (σ = 40) recording 14.4 screens (σ = 10.4); 49 seconds (σ = 38) removing 3.1 (σ = 2.8) screens; 215 seconds (σ = 140) repairing 3.8 (σ = 2.0) gestures; and 119 seconds (σ = 124) redacting 8.9 (σ = 9.6) elements. Participants were able to use the post-processing interfaces to manually fix *all* screens that needed repair (n = 47) and redact *all* UI elements containing sensitive information (n = 134). Survey responses indicate users experienced low cognitive load (μ = 1.67, σ = 0.80), had minimal privacy concerns uploading their traces to the public repository (μ = 1.73, σ = 1.01), and thought the traces they created were highly representative of their app sessions (μ = 4.80, σ = 0.56) (Table 2).

5.3 LLMs for Post-processing

Finally, we offer an LLM baseline for the repair, redact, and enrich phases of ODIM. For each post-processing phase, we created a zero-shot prompt for GPT-4.1 mini that leverages multimodal inputs including screenshots, view hierarchies, and gestures. With the permission of the participants, we run the prompts using the original screens captured during the usability study and compare against the ground truth of the participants' post-processing actions.

For gesture repair, the prompt comprises a pair of consecutive screenshots (before and after a gesture), a view hierarchy of the initial screen, the currently-captured gesture (if any), and a taxonomy of common mobile interaction types (e.g., tap, swipe) based on Apple's documentation.[1] The prompt instructions ask the LLM to identify whether the current gesture is correct and, if it is not, suggest a new gesture type and the corresponding UI element from the view hierarchy. For redaction, the prompt comprises an image and view hierarchy of a screen and instructs the LLM to identify elements containing PII, labeling each with its data type (e.g., name, email). For trace annotation, the prompt provides the complete sequence of screenshots from a trace and asks the LLM to generate a brief natural language description of the interaction sequence.

Across all 15 traces, the LLM identified 63 screens to be repaired; 39 (61.9%) of those screens were also repaired by participants. However, the LLM missed 8 of the 47 screens (17.0%) repaired by participants. Of the 39 screens that were correctly identified for repair, the LLM accurately identified the UI element interaction on 24 screens (61.5%). Across all traces, the LLM identified 151 elements to be redacted; 76 (50.3%) of those elements had also been redacted by the participants. However,

[1] https://developer.apple.com/design/human-interface-guidelines/gestures#Standard-gestures

the LLM missed 58 of the 134 elements (43.3%) redacted by participants. We asked participants from the usability study to review their captured traces and rate the quality of the LLM-generated task descriptions on a 5-point Likert scale (1 = low, 5 = high). Participants generally agreed that the LLM generated annotations were high quality semantic descriptions of the tasks they performed ($\mu = 4.6$, $\sigma = 0.63$). These baseline results indicate promising avenues of future work. We hope researchers can leverage ODIM to create repositories to train more sophisticated multi-modal models, as LLMs are a promising method to automate post-processing.

6 Discussion & Future Work

As with natural language and image data, building large-scale repositories of UX data has the potential to unlock many transformative applications. Unlike language and images, however, interaction data has a *short* shelf life: digital interfaces and experiences evolve quickly and design patterns become outdated. For a dataset to stay relevant for UX researchers and ML practitioners, traces must be captured continuously, as new apps are added and old ones updated. The true value of the ODIM platform will be determined by whether or not this future comes to pass.

The ODIM implementation leverages the Android accessibility service to capture interaction data; the Android platform terms of service state that "accessibility services should only be used to assist users with disabilities in using Android devices and apps." While a strict reading of these terms may suggest that ODIM does not directly comply (and provide some impediment to publishing the APK in the Google Play Store), we posit that it is difficult to imagine a research tool that could prove more valuable for accessibility research and applications. Indeed, prior interaction mining systems have already seen extensive use for exactly this purpose [7, 40, 43, 44, 52, 56, 57]. In the near term, it is easy to see how larger repositories of interaction data could power more advanced versions of Anthropic's Claude Computer Use or Amazon's Nova Act, which can perform actions within a web browser through natural language input.

Beyond the privacy issues related to PII, interaction mining also has implications for copyright. Traces can inadvertently infringe on content ownership rights, and it is likely that some artfully-collected repositories would not constitute fair use. Our public ODIM trace repository will remove traces reported for copyright infringement and those caught obviously infringing during moderation. As we further develop the platform, we intend to create more useful tools and guidelines for content redaction, including those required under regulations like GDPR and CCPA.

Perhaps the most obvious avenue for future work is to extend ODIM to capture traces from non-Android apps. Expanding support to iOS would make available a whole new UI ecosystem of ecologically-valid interaction mining data. Unfortunately, iOS does not provide a publicly-available service to supply UI hierarchies or structured interactions [56]. Vision-based approaches may suffice, however: recent work has explored MLLM-based solutions for mobile UI element detection and post-hoc semantic annotations of screenshots [54]. It seems likely that the highly-structured and annotated data produced by ODIM's Android platform could be used to train even more sophisticated and accurate models in the future. Augmenting this data with additional modalities — such as the think-aloud audio or facial expression information captured in many affective UX tests — may eventually provide even deeper UX understanding.

We hope that ODIM serves as a catalyst for establishing an open, *ongoing*, sustainable interaction mining ecosystem that thrives on collective contributions and mutual benefit. We intend ODIM to further democratize the collection and dissemination of interaction data: helping the UX community share information, coalesce around best-practices in a data-driven way, call out dark patterns, and power the next generation of ML-backed UIs. Given the pervasiveness of digital UX in daily life, even small improvements in accessibility, personalization, and efficiency have the potential to deliver inordinate value in aggregate.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments and suggestions. This work was supported in part by NSF Grant IIS-1750563. We would like to thank Lily Ge, Ethar Hussein, Savannah Lau, and Zora Zhang for their contributions to the redaction interfaces and dataset creation; Zhilin Zhang for prototyping the first Android APK for capture; and Ayush Prasad for helping build a post-processing pipeline for the Web.

References

[1] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine* 35, 4 (2014), 105–120.

[2] Deniz Arsan, Ali Zaidi, Aravind Sagar, and Ranjitha Kumar. 2021. App-Based Task Shortcuts for Virtual Assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 1089–1099.

[3] Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*.

[4] Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual Search for Mobile User Interface Design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 423, 14 pages. https://doi.org/10.1145/3411764.3445762

[5] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. 2022. A Dataset for Interactive Vision-Language Navigation with Unknown Command Feasibility. In *Proceedings of the 16th European Conference on Computer Vision*. Springer, 312–328.

[6] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xin Xia, Liming Zhu, John Grundy, and Jinshui Wang. 2020. Wireframe-based UI design search through image autoencoder. *ACM Transactions on Software Engineering and Methodology* 29, 3 (2020).

[7] Jieshan Chen, Amanda Swearngin, Jason Wu, Titus Barik, Jeffrey Nichols, and Xiaoyi Zhang. 2022. Towards Complete Icon Labeling in Mobile Applications. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*.

[8] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. RICO: a mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 845–854.

[9] Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 2017. ZIPT: Zero-integration performance testing of mobile app designs. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 727–736.

[10] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction mining mobile apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 767–776.

[11] Jing Dong, Yajing Zhao, and Tu Peng. 2009. A review of design pattern mining techniques. *International Journal of Software Engineering and Knowledge Engineering* 19, 06 (2009), 823–855.

[12] Bardia Doosti, Tao Dong, Biplab Deka, and Jeffrey Nichols. 2018. A computational method for evaluating UI patterns. *arXiv preprint arXiv:1807.04191* (2018).

[13] Peitong Duan, Jeremy Warner, Yang Li, and Bjoern Hartmann. 2024. Generating Automatic Feedback on UI Mockups with Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, Article 6, 20 pages. https://doi.org/10.1145/3613904.3642782

[14] Jerry Alan Fails and Dan R. Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*. 39–45.

[15] Shirin Feiz, Jason Wu, Xiaoyi Zhang, Amanda Swearngin, Titus Barik, and Jeffrey Nichols. 2022. Understanding Screen Relationships from Screenshots of Smartphone Applications. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*. Association for Computing Machinery, New York, NY, USA, 447–458. https://doi.org/10.1145/3490099.3511109

[16] Rebecca Fiebrink. 2017. Machine learning as meta-instrument: Human-machine partnerships shaping expressive instrumental creation. *Musical Instruments in the 21st Century: Identities, Configurations, Practices* (2017), 137–151.

[17] Google for Developers. 2025. Android Accessibility Service API Reference. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService.

[18] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*. Advances in Psychology, Vol. 52. North-Holland, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[19] Forrest Huang, John F Canny, and Jeffrey Nichols. 2019. Swire: Sketch-based user interface retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*.

[20] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R Klemmer, and Jerry O Talton. 2013. Webzeitgeist: design mining the web. In *Proceedings of the 2013 CHI Conference on Human Factors in Computing Systems*. 3083–3092.

[21] Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. 2020. Neural design network: Graphic layout generation with constraints. In *Proceedings of the 16th European Conference on Computer Vision*. Springer, 491–506.

[22] Luis A Leiva, Asutosh Hota, and Antti Oulasvirta. 2020. Enrico: A dataset for topic modeling of mobile UI designs. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*.

[23] Florian Lettner, Christian Grossauer, and Clemens Holzmann. 2014. Mobile Interaction Analysis: Towards a Novel Concept for Interaction Sequence Mining. In *Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices & Services*. Association for Computing Machinery, 359–368. https://doi.org/10.1145/2628363.2628384

[24] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *7th International Conference on Learning Representations*. https://openreview.net/forum?id=HJxB5sRcFQ

[25] Tao Li, Gang Li, Jingjie Zheng, Purple Wang, and Yang Li. 2024. MUG: Interactive Multimodal Grounding on User Interfaces. In *Findings of the Association for Computational Linguistics: EACL 2024*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, St. Julian's, Malta, 231–251. https://aclanthology.org/2024.findings-eacl.17/

[26] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[27] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, and Brad A Myers. 2021. Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 578, 15 pages. https://doi.org/10.1145/3411764.3445049

[28] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M. Mitchell, and Brad A. Myers. 2019. PUMICE: A Multi-Modal Agent That Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 577–589. https://doi.org/10.1145/3332165.3347899

[29] Toby Jia-Jun Li and Oriana Riva. 2018. Kite: Building Conversational Bots from Mobile Apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. Association for Computing Machinery, New York, NY, USA, 96–109. https://doi.org/10.1145/3210240.3210339

[30] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 8198–8210. https://doi.org/10.18653/v1/2020.acl-main.729

[31] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 5495–5510. https://doi.org/10.18653/v1/2020.emnlp-main.443

[32] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2019. Humanoid: A deep learning-based approach to automated black-box android app testing. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 1070–1073.

[33] Thomas F Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning design semantics for mobile apps. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 569–579.

[34] Yuxuan Lu, Bingsheng Yao, Shao Zhang, Yun Wang, Peng Zhang, Tun Lu, Toby Jia-Jun Li, and Dakuo Wang. 2023. Human Still Wins over LLM: An Empirical Study of Active Learning on Domain-Specific Annotation Tasks. arXiv:2311.09825 [cs.CL] https://arxiv.org/abs/2311.09825

[35] Arunesh Mathur, Gunes Acar, Michael J. Friedman, Eli Lucherini, Jonathan Mayer, Marshini Chetty, and Arvind Narayanan. 2019. Dark Patterns at Scale: Findings from a Crawl of 11K Shopping Websites. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 81 (Nov. 2019), 32 pages. https://doi.org/10.1145/3359183

[36] Nicholas Micallef, Erwin Adi, and Gaurav Misra. 2018. Investigating login features in smartphone apps. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing*

and Wearable Computers. 842–851.

[37] Microsoft. 2024. Introducing Recall for Copilot+ PCs. https://blogs.microsoft.com/blog/2024/05/20/introducing-recall-for-copilot-pcs/. Accessed: 2025-06-25.

[38] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering* 46, 2 (2018), 196–221.

[39] Ed Novak and Chris Marchini. 2019. Android app update timing: A measurement study. In *2019 20th IEEE International Conference on Mobile Data Management*. IEEE, 551–556.

[40] Sujeath Pareddy, Anhong Guo, and Jeffrey P. Bigham. 2019. X-Ray: Screenshot accessibility via embedded metadata. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 389–395.

[41] Thorsten Prante, Jens Sauer, and Seif Lotfy. 2010. Personal Experience Trace: Orienting Oneself in One's Activities and Experiences. In *Extended Abstracts of the 2010 CHI Conference on Human Factors in Computing Systems*. ACM, Atlanta, GA, USA.

[42] Mishaal Rahman. 2016. "Working As Intended" - An Exploration into Android's Accessibility Lag. https://www.xda-developers.com/working-as-intended-an-exploration-into-androids-accessibility-lag/

[43] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 119–130.

[44] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2020. An epidemiology-inspired large-scale analysis of android app accessibility. *ACM Transactions on Accessible Computing* 13, 1 (2020).

[45] Alborz Rezazadeh Sereshkeh, Gary Leung, Krish Perumal, Caleb Phillips, Minfan Zhang, Afsaneh Fazly, and Iqbal Mohomed. 2020. VASTA: A Vision and Language-Assisted Smartphone Task Automation System. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*. Association for Computing Machinery, New York, NY, USA, 22–32. https://doi.org/10.1145/3377325.3377515

[46] Amanda Swearngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Amy J Ko. 2018. Rewire: Interface design assistance from examples. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*.

[47] Amanda Swearngin and Yang Li. 2019. Modeling mobile interface tappability using crowdsourcing and deep learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*.

[48] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2Words: Automatic Mobile UI Summarization with Multimodal Learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 498–510. https://doi.org/10.1145/3472749.3474765

[49] Dakuo Wang, Josh Andres, Justin D Weisz, Erick Oduor, and Casey Dugan. 2021. Autods: Towards human-centered automation of data science. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.

[50] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P. Bigham, and Jeffrey Nichols. 2023. Never-ending Learning of User Interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, Article 113, 13 pages. https://doi.org/10.1145/3586183.3606824

[51] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P. Bigham. 2023. WebUI: A Dataset for Enhancing Visual UI Understanding with Web Semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

[52] Jason Wu, Xiaoyi Zhang, Jeff Nichols, and Jeffrey P. Bigham. 2021. Screen parsing: Towards reverse engineering of UI models from screenshots. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 470–483.

[53] Ziming Wu, Yulun Jiang, Yiding Liu, and Xiaojuan Ma. 2020. Predicting and Diagnosing User Engagement with Mobile UI Animation via a Data-Driven Approach. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 13 pages. https://doi.org/10.1145/3313831.3376324

[54] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-UI: Grounded Mobile UI Understanding with Multimodal LLMs. In *Proceedings of the 16th European Conference on Computer Vision*. Springer-Verlag, Berlin, Heidelberg, 240–255. https://doi.org/10.1007/978-3-031-73039-9_14

[55] Tao Zhang, Ying Liu, Jerry Gao, Li Peng Gao, and Jing Cheng. 2020. Deep learning-based mobile application isomorphic gui identification for automated robotic testing. *IEEE Software* 37, 4 (2020), 67–74.

[56] Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, Aaron Everitt, and Jeffrey P. Bigham. 2021. Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 275, 15 pages. https://doi.org/10.1145/3411764.3445186

[57] Xiaoyi Zhang, Anne Spencer Ross, and James Fogarty. 2018. Robust annotation of mobile application interfaces in methods for accessibility repair and enhancement. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 609–621.