SARA: Self-Replay Augmented Record and Replay for Android in Industrial Cases*

Jiaqi Guo†
Xi'an Jiaotong University
Xi'an, China
jasperguo2013@stu.xjtu.edu.cn

Shuyue Li†
Xi'an Jiaotong University
Xi'an, China
lishuyue1221@stu.xjtu.edu.cn

Jian-Guang Lou
Microsoft Research Asia
Beijing, China
jlou@microsoft.com

Zijiang Yang
Western Michigan University
Kalamazoo, MI, USA
zijiang.yang@wmich.edu

Ting Liu
Xi'an Jiaotong University
Xi'an, China
tingliu@mail.xjtu.edu.cn

ABSTRACT

Record-and-replay tools are indispensable for quality assurance of mobile applications. Due to its importance, an increasing number of tools are being developed to record and replay user interactions for Android. However, by conducting an empirical study of various existing tools in industrial settings, researchers have revealed a gap between the characteristics requested from industry and the performance of publicly available record-and-replay tools. The study concludes that no existing tools under evaluation are sufficient for industrial applications. In this paper, we present a record-and-replay tool called SARA towards bridging the gap and targeting a wide adoption. Specifically, a dynamic instrumentation technique is used to accommodate rich sources of inputs in the application layer satisfying various constraints requested from industry. A self-replay mechanism is proposed to record more information of user inputs for accurate replaying without degrading user experience. In addition, an adaptive replay method is designed to enable replaying events on different devices with diverse screen sizes and OS versions. Through an evaluation on 53 highly popular industrial Android applications and 265 common usage scenarios, we demonstrate the effectiveness of SARA in recording and replaying rich sources of inputs on the same or different devices.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; **Software maintenance tools**.

KEYWORDS

Android, Testing, Record-and-Replay

*We would like to thank the anonymous reviewers for their helpful comments. Ting Liu is the corresponding author.
†Work done during an internship at Microsoft Research Asia.

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3330557

ACM Reference Format:
Jiaqi Guo, Shuyue Li, Jian-Guang Lou, Zijiang Yang, and Ting Liu. 2019. SARA: Self-Replay Augmented Record and Replay for Android in Industrial Cases. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3293882.3330557

1 INTRODUCTION

Mobile devices have quickly become the most accessible and popular computing devices in the world. Record-and-replay tools play an important role in the process of quality assurance for mobile applications, or apps for short, due to its capability in recording user's interactions with apps and replaying on various devices at a later time, allowing developers to test apps on different devices at once [27, 31]. For example, the process of regression testing can be automated in collaboration with advanced test selection techniques [7, 19, 22]. To this end, the burden of developers and testers is greatly eased since manual testing is still preferred today in apps development [23, 26, 29]. Much effort from academia and industry has been dedicated in building record-and-replay tools [1, 3, 10, 13, 15, 18, 20, 21, 30, 33–35].

However, a recent study [28] on most existing tools shows that there is a significant gap between the capability of currently available tools and the record-and-replay needs of Android apps in industry. Developers from WeChat [38], one of the most popular social media apps with over 1 billion monthly active users, request that a desirable record-and-replay tool should provide four features and satisfy four constraints. The four features include (F1) Recorded motion events should be based on screen coordinates (touch points of testers on the screen); (F2) Recorded motion events should be based on widgets (e.g., buttons, text fields); (F3) Recorded events should be insensitive to the state of the app; and (F4) Timing between events should be recorded. The four constraints are (C1) No custom OS is required; (C2) No instrumentation on app is required; (C3) No root access is required; and (C4) Source code of app is not required. Meanwhile, there are some other common requirements, namely, source code of the tool is available (also requested by the developers from WeChat); recorded data should be human-readable [33]; recorded data should be able to be replayed on different devices with diverse screen sizes and OS versions [18].

Table 1: Characteristics comparison of appetizer, monkeyrunner, RERAN and SARA.

Tool	Coordinate Sensitive	Widget Sensitive	Timing Sensitive	State Insensitive	No Custom OS	No Instrumentation	No Root Access	No Access to Source Code	Open Source	Different Screen Sizes	Readable
appetizer	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓	✗
monkeyrunner	✓	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓
RERAN	✓	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
SARA	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓

Based on the study, no publicly available tools satisfy all the desired characteristics [28]. Table 1 shows the characteristics exhibited by three state-of-the-practice tools: appetizer [3], monkeyrunner [15], and RERAN [10]. There are three challenges in providing all features under constraints requested from industry:

Recording and Replaying Rich Sources of Inputs. Mobile devices provide rich sources of inputs to an app, including motion, key, and sensor inputs. Recording and replaying inputs from sensors like GPS usually requires a custom OS [20] or the source code of the app [33]. However, as requested from industry, a custom OS is not preferable (C1) and the source code of the app is usually unavailable to testers (C4). Hence, it is challenging to record and replay inputs with rich sources under various constraints.

Efficiently Recording Motion Events Based on Widgets. To achieve robustness during replay, it is necessary to record both coordinates and widgets (F1, F2) of motion events. There are various ways to record the widgets under interactions, but they usually introduce large time overhead, making them impractical to use in industrial settings. For example, it takes about 10 seconds for Culebra [1] to perform a tap on a widget. In fact, most of the proposed tools record motion events based solely on screen coordinates because it is non-trivial to efficiently record motion events based on both coordinates and widgets while preserving good user experience for testers in practice.

Replaying Events on Different Devices. Android OS searches for appropriate layout configurations declared in an app based on the device screen resolution and size when the app is launched. Developers are recommended but not strictly required to create alternate UI layouts and UI resources for different screen resolutions and sizes [17]. Thus, it is very desirable to replay events on different devices with diverse screen sizes and OS versions. Unfortunately, how to achieve the goal remains an open question.

In this work, we design and implement a record-and-replay tool called SARA (Self-Replay Augmented Record and Replay for Android) to address all aforementioned challenges and target a wide adoption. To achieve the goal, SARA integrates three major techniques:

(1) Dynamic instrumentation is applied to record and replay diverse kinds of inputs on mobile devices, including inputs from sensors. It works even under various aforementioned constraints such as no custom OS and no source code.
(2) Self-Replay mechanism is proposed to address the problem of efficiently recording motion events based on widgets. Specifically, SARA first records motion events based solely on screen coordinates during user interactions. It then automatically replays recorded events on the same device to identify the widgets under interactions. The recording before the self-replay is a low cost operation. Thus SARA can record motion events based on both screen coordinates and widgets while providing good user experience.
(3) An adaptive replay method is designed to replay events on different devices. SARA is able to replay events by heuristically searching for widgets and transforming coordinates to adapt to diverse screen sizes and OS versions.

We evaluate SARA on three different Android devices, with 53 highly popular industrial Android apps and 265 common usage scenarios. As shown in Table 1, SARA satisfies most desired characteristics desired by industry. The evaluation results show that

(1) SARA manages to record and replay 228 out of 265 common usage scenarios on the same device, which is much more than 161 and 29 successful replays by appetizer [3] and RERAN [10], respectively. We do not compare SARA with monkeyrunner [15] because it is insensitive to the timing between events. The timing between events is crucial during a reply.
(2) SARA manages to replay 41 out of 42 common usage scenarios on two devices with the same display aspect ratio but different screen sizes, which is more than 31 successful replays by appetizer. SARA replays 34 out of 42 on two devices that vary in both display aspect ratios and screen sizes. Note that appetizer fails to replay any events on devices with different display aspect ratios. We do not compare SARA with RERAN as it does not support replaying events on different devices.
(3) Although SARA applies the instrumentation technique and records motion events based on widgets, compared with the original runtime, the overheads of recording and replaying are only 4.29% and 7.07%, respectively. The space overhead is just 1.78 KB per second.

2 RELATED WORK

There has been a lot of progress made from academia and industry in developing record-and-replay tools for desktop applications [9, 24, 25, 32, 36] and Android applications [1, 3, 10, 13, 15, 18, 20, 21, 30, 33–35]. The tools for Android applications can be grouped into the following three categories according to the layer in which they record inputs.

The Linux Kernel Layer. RERAN [10] is one of the very first record-and-replay tools proposed by researchers. It works in the kernel layer. Specifically, it captures low-level events with ADB command *getevent* by reading logs in */dev/input/event** files, and utilizes command *sendevent* to replay events. The low-level events are tightly coupled to the hardware, making it hard to reconstitute into high-level gestures and be replayed on other devices. Following tools like Mosiac [18] and appetizer [3] capture events through the

same avenue. These tool have limitations in recording and replaying some sensor inputs, because inputs like GPS are not written in logs.

The Android Framework Layer. VALERA [20] modifies the Android framework to capture sensor and network input, event schedules, and inter-app communication, making it achieve precise record and replay. However, VALERA requires a custom OS to record and replay, which violates the constraints from industry.

The Application Layer. There are a bunch of tools recording input data in the application layer. Specifically, Mobiplay [33] adopts a client-server architecture, involving a client app running on a mobile device and a target app running on the server to intercept inputs to the app. However, it fails to replay sensor inputs like GPS in absence of the source code of the app. Espresso [13] records motion events based on widgets and key events by attaching a debugger to the app under record, but it cannot record sensor input and complicated gestures like zoom, pinch, and it also requires source code of the app. Robotium [35] is derived from the Selenium web browser automation tool [5]. It is only able to intercept those widgets that are controlled by the main process of the app. During evaluations, we find that most popular industrial apps in fact launch more than one process. Culebra [1] provides a graphical user interface on desktop for users to interact with the app under record. Before sending the user action to the app, Culebra tries to identify the widget that is going to be interacted within the view hierarchy, which is very similar to MobiPlay in spirit. But it introduces large time overhead and cannot record sensor inputs. Ranorex [34] is a cross-platform commercial test automation tool. It supports recording events based on both coordinates and widgets through instrumentation, but it cannot record sensor inputs and fails to instrument large-size apps. SARA also falls into this category. It is designed to record and replay diverse kind of inputs under the constraints requested from industry.

3 DESIGN OF SARA

In this section, we elaborate on the design decisions behind SARA and how SARA addresses the challenges in providing features under various constraints requested from industry.

3.1 Recording and Replaying Rich Sources of Input through Dynamic Instrumentation

A custom OS or the source code of an app is usually required by existing tools to record and replay rich sources of inputs on mobile devices. However, industry developers do not favor tools requiring a custom OS. Installing a custom OS is time-consuming and error-prone in practice because the custom OS may be incompatible with devices. Another major concern is that events recorded in the custom OS may be not transferable to the official Android OS. Developers also do not like tools that require the source code of the app under record since they sometimes outsource the testing to other companies. However, recording and replaying rich sources of inputs in absence of a custom OS and source code is non-trivial [33], especially for those with inputs from sensors such as GPS.

To tackle the challenge, SARA applies dynamic instrumentation to record rich sources of input data in the application layer. Firstly, rich sources of inputs are delivered to the application layer, allowing developers to process them. The input data in this layer are high-level by design compared to those in the Linux kernel and the Android framework layer. Hence, one of the most prominent advantages of recording input data in the application layer is that the recorded data is human-readable and easy to be reconstituted into high-level gestures like zoom, making it fairly easy for testers to analyze, revise and re-assemble these data for further testing purpose. Secondly, dynamic instrumentation, which works at runtime of the app, does not need a custom OS or the source code. It is also hardly influenced by packing techniques that are increasingly popular on the Android platform for hiding code [8]. This is because the hidden dex-code will be unpacked in order to be executed at runtime. The dynamic instrumentation technique in SARA also plays an important role in replaying sensor inputs such as GPS (discussed in Section 4).

Note that developers from WeChat also do not favor instrumentation as they are concerned about the compatibility between instrumentation tools and the app. But as we will show in Section 5, SARA is compatible with WeChat and other highly popular industrial apps, which relieves their concerns to some extend.

3.2 Efficiently Recording Motion Events Based on Widget through Self-Replay

Industry developers request a record-and-replay tool that records events based on both coordinates and widgets. Intuitively, events that are recorded based on widgets are more robust than those that are recorded solely based on coordinates. There are various ways to record the widgets under interactions, but they usually introduce large overhead since information about widgets is far more complicated than coordinates. For example, Espresso [13] is able to intercept the widgets by attaching a debugger to an app during record. But it inevitably results in slow response of the app and consequently poor user experience.

To efficiently record events based on both coordinates and widgets while providing solid usability, SARA introduces a novel self-replay mechanism, breaking a typical recording phase into two sub-phases. The goal of the first phase is to record events based on coordinates, which does not introduce much overhead and hence provides good user experience. The goal of the second phase is to augment the events recorded in the previous phase with relevant widgets information. Specifically, SARA replays the events automatically on the recording device, and efficiently identifies the widgets under interactions. The extra overhead introduced in the second phase is transparent for testers. To this end, SARA is able to record events based on both coordinates and widgets while preserving good user experience.

It is worth noting that the self-replay mechanism can be easily incorporated in existing record-and-replay tools. For example, it can be added at the end of recording phase of Mobiplay [33] so that the events recorded by MobiPlay come to be both coordinate- and widget-sensitive.

3.3 Adaptive Replay

Although all events are recorded and the motion events are even augmented with widget information, it still remains a challenging task to replay motion events on different devices. The main reasons are as follows. First, the view hierarchy in the same context of

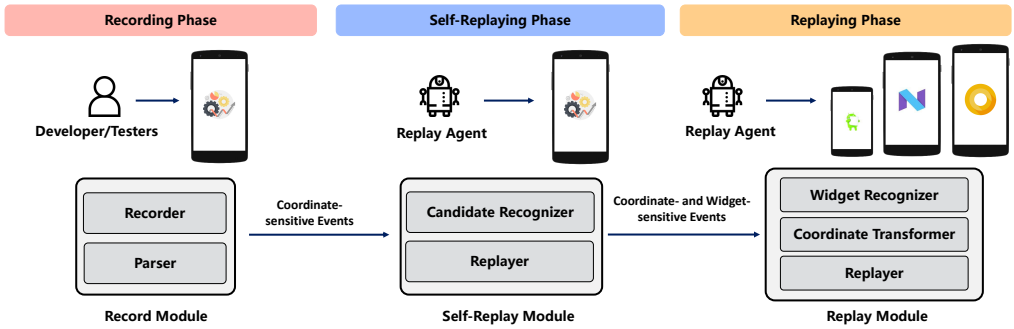

Figure 1: The workflow of SARA

an app usually varies on different devices especially when there are scrollable widgets in the hierarchy. The number of items in a scrollable widget is closely related to screen size. For example, there can be seven visible items in a scrollable widget on the recording device, but only the first five of them are visible on a device with a smaller screen size. Second, for some high-level motion gestures like swipe, the widget information is still not enough for an accurate replay as the trajectory of the swipe also matters, especially for those drawing scenarios. Finally, breaking changes in UI layout are occasionally observed on different devices.

To address these problems, an adaptive replay method is proposed. In particular, SARA replays events by heuristically searching for widgets, e.g., automatically swiping scrollable widgets, and transforming coordinates based on the screen parameters of the replaying device to adapt to layouts on different devices.

4 IMPLEMENTATION

In this section, we first illustrate the overall three-phase workflow of SARA. Then we demonstrate the implementation of the three phases in detail.

4.1 SARA Overview

SARA operates in a three-phase process in order to record and replay. Figure 1 depicts the workflow of SARA. In the recording phase, SARA records all input data including motion, key, and sensor inputs, and parses them into events. Then in the self-replaying phase, SARA augments the recorded events with relevant widgets data through self-replay. That is, SARA automatically replays the events on the recording device and identifies the widgets under interactions. Finally in the replaying phase, SARA heuristically searches for widgets and conducts coordinate transformations based on the screen parameters of the replaying device during replay.

4.2 Recording Phase

The goal of this phase is to precisely record inputs and the timing between two consecutive inputs. SARA accomplishes the goal with its Record Module that consists of Recorder and Parser.

Recorder. Recorder focuses on capturing rich sources of inputs. As discussed in Section 3.1, SARA records input data in the application layer by applying dynamic instrumentation technique. Typically, inputs of Android apps can be grouped into three categories, namely, motion, key, and sensor. In the following, we describe how Recorder captures each of the three categories of input in the application layer.

4.2.1 Motion Inputs. The services provided by the Android framework deliver motion inputs to the active app. The motion inputs are then dispatched starting from the top of the view hierarchy of the app, and then down, until it reaches the target widget. Hence, to intercept all motion inputs, Recorder dynamically instruments *dispatchTouchEvent* method of Activity, Dialog, and View in Popup Window, recording the input data (e.g., the coordinate, the action code) and the time since this phase starts. Note that Recorder records the motion inputs based on coordinates in this phase. SARA is able to recognize all sorts of motion gestures in principle as developers also recognize them in *dispatchTouchEvent* method.

4.2.2 Key Inputs. Key inputs can be further classified into physical key inputs (e.g., back, volume up) and soft keyboard inputs. Similar to motion inputs, Recorder intercepts physical key inputs by dynamically instrumenting *dispatchKeyEvent* method of Activity and Dialog, recording the input data (e.g., the key code) and the time since this phase starts. Instead, soft keyboard inputs are usually delivered to the active input method. The input method then delivers inputs to the active app via Inter-Process Communication. Specifically, the *InputConnection* interface provides a communication channel from the input method to the app that is receiving the inputs. Hence, Recorder instruments *beginBatchEdit* and *performEditorAction* of *InputConnection* to record soft keyboard input data (e.g., the input content, the text view that is receiving input). Note that for WebView, Recorder captures key input data by registering an input listener in web pages.

4.2.3 Sensor Inputs. Smartphones provide a richer set of sensors than desktop/server machines. They can be classified into low-level sensors and high-level sensors [20]. Low-level sensors include accelerometer, gravity, gyroscope, etc. Developers are able to retrieve real-time low-level sensors inputs by overwriting *onSensorChanged* method of *SensorEventListener*. Hence, Recorder intercepts the method *onSensorChanged* to record low-level sensor data. As for high-level sensors such as GPS, Android provides rich APIs to handle each kind of input. For example, when the GPS/location is changed, the Android framework delivers a message to *LocationListener* to notify the app. Developers are able to handle the change of location by overwriting the *onLocationChanged* method of *LocationListener*. Therefore, Recorder also intercepts *onLocationChanged* to record location data.

Note that the methods to be instrumented may not be loaded in memory when the phase starts. Recorder, therefore, also instruments the *loadClass* method of *ClassLoader* so that it can perform instrumentation once the method is loaded.

Parser. Input data in the application layer is high-level, making it easy to be parsed into human-readable events. For example, two consecutive motion inputs that share the same coordinates and the same time when the user pressed down, or down time for short, can be grouped together. Based on the time elapsed since down time, the grouped inputs can be parsed into either a tap or a long tap. In particular, Parser follows the official Android development guideline [14] to parse motion inputs, key inputs, and sensor inputs. We define the parsing results of Parser as events, including motion events, input events, and sensor events. In addition, the timing between events can be precisely calculated by Parser as Recorder records the time since the recording phase starts when the input is captured.

4.3 Self-Replaying Phase

The goal of this phase is to augment the motion events with relevant widgets data through self-replay. SARA achieves the goal with a Self-Replay Module that consists of Candidate Recognizer and Replayer. Algorithm 1 outlines the process of this phase. It takes the recorded events in the recording phase as input and outputs the augmented events.

Candidate Recognizer. Candidate Recognizer is responsible for recognizing the widgets under interactions and generating a unique identifier for each of the widgets so that they can be uniquely recognized in the following replaying phase. Before replaying a motion event, Candidate Recognizer recognizes candidate widgets in the view hierarchy of an app (line 5). If the coordinate of the event lies in the region of a widget (determined by the position and size of the widget), then this widget is regarded as a candidate. The number of candidates is usually larger than one because overlaps among widgets usually occur and the view hierarchy lacks of the order of overlapped widgets. Hence, to deterministically recognize the widget under interaction, Candidate Recognizer performs an instrumentation on each candidate to record whether it receives an input or not. The last widget that receives the motion input is regarded as the target widget (line 7). Once the target widget is recognized, Candidate Recognizer generates an unique identifier for it (line 8). In particular, Candidate Recognizer searches a shortest xpath in the view hierarchy to uniquely identify the target widget in a bottom-up manner (lines 16-29). This xpath eventually serves as the identifier for the widget.

Replayer. Replayer is responsible for replaying events on the recording device. In terms of motion events and key events, Replayer utilizes UIAutomator [11] to replay these events as it provides a set of APIs to perform interactions on apps, such as swipe, press key, and set text. As for sensor events, Android does not provide any tools to inject sensor events to an app. But we find that *onSensorChanged* method of *SensorEventListener* will be called when there is a reading from a sensor even if the sensor data is not changed [16]. In addition, the frequency of reading is configured in the app. Hence, to replay low-level sensor inputs, Replayer makes an assumption that the frequency remains the same during replay. Based on this assumption, Replayer dynamically instruments the *onSensorChanged* method and replaces the read sensor data with the recorded data in real-time. As for high-level sensor inputs such as GPS, similarly, Replayer dynamically instruments *onLocationChanged* of *LocationListener* and replaces the location data with the recorded data. Therefore, the dynamic instrumentation technique enables SARA to replay sensor inputs in absence of a custom OS and the source code which is usually a must in existing record-and-replay tools [20, 33].

Algorithm 1: Self-Replay

```
Input     : Recorded events E in recording phase
Output    : Augmented events M
Component : Candidate Recognizer CRecognizer,
            Replayer Replayer
1  let M be a list of events (initialized as empty);
2  Replayer.performSensorEvent(E);
3  foreach event e ∈ E do
4      if e is motion event then
5          CRecognizer.recognizeCandidates(e);
6          Replayer.performMotionEvent(e);
7          w = CRecognizer.targetWidget();
8          i = CRecognizer.generateIdentifier(w);
9          n = new MotionEvent(w, e, i);
10         M ← M ∪ {n};
11     else if e is key event then
12         Replayer.performKeyEvent(e, v);
13         M ← M ∪ {e};
14     sleep(e);
15 return M;
16 Procedure generateIdentifier(v, w)
17     let i be an identifier for w (initialized as empty string);
18     P = findParents(v, w);
19     c = w;
20     foreach parent p ∈ P do
21         s = findChildren(p, c.text, c.description, c.package,
             c.resourceId);
22         if s.length > 0 then
23             i ← i+ generateXpath(c.text, c.description,
               c.package, c.resourceId, indexOf(s, c));
24         else
25             i ← i+ generateXpath(c.text, c.description,
               c.package, c.resourceId);
26         if isUnique(v, i) then
27             break;
28         c ← p;
29     return i;
```

4.4 Replaying Phase

The goal of this phase is to replay events on different devices with diverse screen sizes and OS versions. SARA achieves the goal with

Algorithm 2: Replay motion events in replaying phase

Input : Widget unique identifier i,
Motion event e,
dpi of the recording device sdpi,
dpi of the replaying device tdpi
Component : Widget Recognizer WRecognizer,
Coordinate Transformer Transformer,
Replayer Replayer

```
1  w = WRecognizer.recognize(i);
2  if w is null then
3      t = Transformer.transform(e, sdpi, tdpi);
4  else
5      t = Transformer.widgetTransform(w, e, sdpi, tdpi);
6  Replayer.performMotionEvent(t, e);
7  Procedure Transformer.transform(e, sdpi, tdpi)
8      let T be a list of coordinates (initialized as empty) ;
9      foreach coordinate c ∈ e.abs_coordinates do
10         x_dp, y_dp = px2dp(c.x, sdpi), px2dp(c.y, sdpi);
11         x_px, y_px = dp2px(x_dp, tdpi), dp2px(y_dp, tdpi);
12         T ← T ∪ {(x_px, y_px)}
13     return T;
```

a Replay Module, which consists of Widget Recognizer, Coordinate Transformer and Replayer. Replayer in this module is the same as that in the Self-Replay Module. What sets the Replay Module apart is that in this replaying phase motion events are replayed based on widgets and transformed coordinates. Algorithm 2 outlines the replay process of a motion event in this phase. Widget Recognizer first recognizes the target widget (line 1). If the widget is managed to be recognized, Coordinate Transformer transforms the coordinates relative to the widget based on the screen size and the pixel density of the replaying device. Otherwise, Coordinate Transformer simply transforms the coordinates relative to the screen (lines 2-5). Finally, Replayer performs the events based on the transformed coordinates (line 6).

Widget Recognizer. Widget Recognizer is responsible for recognizing the widget. Specifically, Widget Recognizer first recognizes the target widget in the view hierarchy with its unique identifier generated in the self-replaying phase. When Widget Recognizer fails to recognize the target widget, it tries to search for the widget by automatically swiping scrollable widgets in the view hierarchy.

Coordinate Transformer. The basic idea behind Coordinate Transformer is that it tries its best to preserve the trajectory and the motion distance on the replaying device (lines 7-13). Specifically, dpi (dots per inch) refers to the physical density of the pixel on the screen. Procedure *px2dp* converts pixel (px) to density-independent pixel (dp) which is a commonly used distance unit in Android as it preserves the visible size or distance on devices with different pixel densities. Procedure *dp2px* instead performs the reverse conversion.

4.5 Tool Implementation

SARA is implemented as a desktop application that can run on both Windows and Linux operating systems. Specifically, the Recorder in the Record Module of SARA is mainly implemented with Frida [2], a cross-platform dynamic instrumentation toolkit , which is capable of instrumenting applications without root access. Replayer in the Self-Replay Module and Replay Module is mainly implemented with the python wrapper of Android uiautomator2 [6] and Frida. Both Candidate Recognizer and Widget Recognizer heavily rely on the view hierarchy, which is extracted by both uiatuomator2 and ADB command *adb shell dumpsys activity top*.

Table 2: Overview of phones used in evaluations.

Phone	Resolution	Dpi	Size (inch)	OS Version
RedMi 1s	720 × 1280	320	4.7	7.0.1
Samsung Galaxy A8	1080 × 1920	480	5.7	6.0.1
Samsung Galaxy A9 Star Lite	1080 × 2220	420	6.0	8.0.0

5 EVALUATION

In this section, we perform evaluations of SARA in terms of its effectiveness in recording and replaying events, and the overhead of it. Following previous works [10, 33], we evaluate SARA on recording and replaying events on the same device. In addition, we evaluate SARA on replaying events on different devices, since it allows developers to test apps on various devices at once, especially in regression testing settings. To this end, we investigate the following three research questions:

RQ1. How effective is SARA in recording and replaying events on the same device?

RQ2. How effective is SARA in replaying events on different devices with diverse screen sizes and OS versions?

RQ3. What is the time and space overhead of SARA?

5.1 Evaluation Setup

We conduct evaluations on 3 real phones as listed in Table 2. We sample 53 top-recommended apps with the highest numbers of downloads from each category in Google Play, the official application store for Android OS, as listed in Table 3. Sampled apps are required to be compatible with at least two of our phones. As shown in the table, each app has at least 1 million installs, and these apps span across 24 different categories, demonstrating the representation of these apps. For each sampled app, we further identify 5 common usage scenarios from its description in Google Play and our hands-on experiences in using the app. For example, we regard as the 5 common scenarios for app Twitter, "Create a post", "Search people", "Like and comment", "Edit profile", and "Send message". The 265 scenarios cover each source of input to Android. Detailed descriptions of scenario and screenshots along with the source code of SARA are available on SARA's website [4].

For **RQ1**, SARA is used to record and replay each common usage scenario on the same device. To establish baselines, we also evaluate appetizer [3] and RERAN [10], which are state-of-the-practice record-and-replay tools for Android as reported in the study [28]. We do not evaluate other recently proposed and publicly available tools like Espresso [13] and Culebra [1] because they either require the source code of an app or introduce large time overhead, which prevents us from effective evaluation on industrial apps in large scale. Both appetizer and RERAN record inputs by reading logs in */dev/input/event** files. What sets appetizer apart is that it does not

Table 3: Overview of selected applications and evaluation results of recording and replaying events on the same device. '#Install' shows the number of installs of an app according to Google Play ('m' and 'b' indicates million and billion). 'Size' shows the size of an app (MB).

App name	Category	#Install	Size	SA.	ap.	RE.
Accuweather	Weather	50m+	34	5	1	0
Adobe Acrobat Reader	Productivity	10m+	19	4	1	0
Alipay	Finance	1m+	62	2	4	0
BBC News	News & Magzine	10m+	15	5	3	0
Booking	Travel & Local	100m+	31	5	1	0
Citymapper	Maps & Navigation	5m+	13	5	3	0
CNN	News & Magzine	10m+	31	5	2	0
Daylio	Lifestyle	5m+	9	5	5	1
Drugs.com	Medical	1m+	23	2	5	0
eBay	Shopping	10m+	20	5	3	0
ESPN	Sports	10m+	24	4	2	1
Fackbook	Social	1b+	70	0	3	1
File Commander	Business	100m+	16	4	1	0
Flow Flee	Game	100m+	11	5	2	0
Gmail	Communication	1b+	22	3	5	0
GoodRx	Medical	1m+	12	4	4	0
Google Calculator	Tools	10m+	1	5	2	1
Google Chrome	Communication	1b+	42	5	5	2
Google Photos	Photography	1b+	31	5	4	2
Google Translate	Tools	50m+	14	4	2	3
Google Wallpapers	Personalization	50m+	9	5	2	0
Instagram	Social	1b+	27	5	5	3
Go Music	Music & Audio	50m+	23	5	0	2
Investing.com	Finance	5m+	20	3	5	0
Keep Trainer	Health & Fitness	1m+	21	4	3	1
Kindle	Books & Reference	1b+	52	4	1	1
KFC	Food & Drink	10m+	58	5	4	0
LinkedIn	Social	100m+	33	5	5	1
McDonald	Food & Drink	10m+	49	4	4	0
MSN Weather	Weather	1m+	8	5	3	0
NBA App	Sports	10m+	23	5	4	0
OneDrive	Productivity	50m+	48	4	3	1
Onenote	Productivity	100m+	71	4	2	2
Period Calendar	Health & Fitness	1b+	13	5	3	0
QQ Music	Music & Audio	5m+	49	4	5	0
Realtor.com Real Estate	House & Home	10m+	12	5	4	0
Redfin Real Estate	House & Home	1m+	20	4	4	0
SBB Mobile	Maps & Navigation	1m+	24	4	1	0
Sketch	Art & Design	50m+	21	5	1	0
Snapseed	Photography	50m+	28	5	2	0
Steam	Entertainment	10m+	5	5	5	0
Taobao	Shopping	10m+	75	4	5	1
TED	Education	10m+	18	3	5	0
Textgram	Art & Design	10m+	20	5	2	0
Trivago Hotel Search	Travel & Local	50m+	10	4	4	0
Twitter	News & Magazines	5b+	30	4	5	3
VLC for Android	Video	10m+	16	4	1	1
Wallpapers HD	Personalization	1m+	3	5	0	0
Walmart	Shopping	10m+	51	4	2	0
Wechat	Communication	100m+	59	4	5	1
WhatsApp	Communication	10b+	30	5	3	3
Wikipedia	Books & Reference	10m+	19	5	3	0
Youtube	Video	1b+	21	4	2	0
Total				**228**	**161**	**29**

require root access and it supports replaying events on different devices. We regard the replay as correct if and only if the Externally Visible State is the same, where Externally Visible State is the subset of app state that is viewed by users [20]. We manually check the External Visible States in replaying phase to assess the correctness of a replay. Evaluations are conducted on the Galaxy A8 device.

For **RQ2**, SARA is used to record scenarios on one device and replay them on the remaining two devices. We compare SARA with appetizer [3] for its capability in recording and replaying events on devices that share the same display aspect ratio. To systematically study the effectiveness of SARA and appetizer, we randomly sample 42 scenarios that are replayable with both of them on the same device. Then, we record the sampled scenarios on RedMi 1s and Samsung Galaxy A8, respectively, and replay events on the remaining two devices. The Externally Visible State usually varies on devices with different screen sizes. Hence, in this study, we regard the replay as correct if and only if all recorded events are triggered on corresponding widgets in order.

Table 4: Overview of the main causes of failures of SARA in recording and replaying events on the same devices.

Phase	Main Cause	#Scenarios	Total
Recording	Incorrect soft keyboard input	7	**20**
	Instrumentation failure	5	
	Implementation error	5	
	WebView failure	3	
Self-Replaying	Imprecise timing between events	7	**11**
	Unsupported screen orientation	2	
	Swipe gesture failure	2	
Replaying	Non-deterministic states of app	6	**6**

For **RQ3**, we measure the time and space overhead of SARA by running SARA on 36 randomly sampled scenarios that are replayable with SARA. Specifically, in order to measure the time overhead, we first record the original runtime of a scenario. Then, we compare it with the runtime of the scenario in the recording phase and replaying phase of SARA, respectively. As for the space overhead, we measure the size of event logs generated in the recording phase and the log rate. Evaluations are conducted on the Samsung Galaxy A8 device.

5.2 Effectiveness in Recording and Replaying Events on the Same Device

Table 3 shows the evaluation results of SARA ("SA."), appetizer ("ap.") and RERAN ("RE.") in recording and replaying scenarios on the Samsung Galaxy A8. Digit in a table cell of the last three columns indicates the number of scenarios of an app that a tool manages to record and replay. Table cells with gray background indicate the highest value compared with other tools. SARA, appetizer, and RERAN manage to record and replay 86.0%, 60.7%, and 10.9% of 265 common usage scenarios on the same device, respectively. It is clear that SARA achieves the best performance. RERAN achieves much lower performance than the other tools because it usually halts during a replay and produces error messages such as *"Could not open /dev/input/event0, Too many open files"*. This is a well-known problem of RERAN but no patches in its repository[1] has fixed the problem yet. As for appetizer, it fails to record the physical key inputs, e.g., tap a back button, and it usually omits several events during a replay.

Developers from WeChat do not favor instrumentation as they are concerned about the compatibility between instrumentation tools and the app. But as we show in the table, SARA is compatible with WeChat and all other sampled industrial apps except Facebook, which relieves their concerns to some extent.

To understand the source of failures of SARA, we analyze all the fail cases and summarize the main causes of failures as shown in Table 4.

[1] https://github.com/lorenzogomez/RERAN

Failures in Recording Phase. There are 20 cases where SARA fails to record correct inputs. Specifically, SARA fails to instrument Facebook, accounting for 5 fail cases. There are 3 cases where SARA fails to record special soft keyboard inputs in WebView, e.g., tap a search key in the keyboard, because such inputs are not delivered to the listener registered in a web page. In another 7 cases, SARA fails to record the correct soft keyboard inputs. The reasons are two-folds. Firstly, the instrumentation tool, Frida, fails to process classes that implement the Interface *android.text.Editable* which provides rich methods to process input string. We have opened an issue for this problem in the issue tracker of Frida but have not yet received a reply. We try to infer input with several heuristic strategies to bypass the problem, but in some cases we still miss inputs from the soft keyboard, resulting in 5 failures. Secondly, apps like OneNote support rich text editing by integrating a rich text editor. There are in fact many ways to implement a rich text editor. SARA currently only supports those based on *android.widget.TextView*, consequently resulting in 2 failures. The rest 5 cases are due to our implementation in recording the motion inputs inside a Popup Window.

Failures in Self-Replaying Phase. There are 11 cases where SARA fails to replay events in the self-replaying phase. In 7 cases, SARA fails to capture the precise timing to replay a motion event during a self-replay because in the self-replaying phase SARA instruments candidate widgets before performing a motion event. This inevitably introduces time overhead. One possible way to solve the problem is to deduct the average time overhead introduced by the instrumentation during self-replay between events. We leave it as our future work. Current implementation of SARA does not support replaying motion events on a rotated screen, which causes another 2 failures. The rest 2 fail cases are caused by the problem of uiautomator2 in performing swipe gestures.

Failures in Replaying Phase. The 6 fail cases in the replaying phase are mainly caused by the nondeterministic states of an app. For example, in SBB Mobile which shows available bus tickets in real time, SARA fails to recognize the target widget as the state of it changes continuously. How to deal with the nondeterministic states of an app under test remains an open question in community.

5.3 Effectiveness in Replaying Events on Devices with Different Screen Sizes and OS Versions

Table 5 shows the sampled 42 scenarios that are replayable with both SARA and appetizer. When recording scenarios on the RedMi 1s, SARA manages to replay 41 and 34 of them on the Galaxy A8 and Galaxy A9, respectively. appetizer manages to replay 31 scenarios on the Galaxy A8. appetizer, however, does not support replaying scenarios on the Galaxy A9 since A9 has different display aspect ratio with RedMi 1s. When recording scenarios on the Galaxy A8, SARA manages to replay 38 and 32 of them on the RedMi 1s and Galaxy A9, respectively, while appetizer manages to replay 30 scenarios on the RedMi 1s. SARA is more effective than appetizer in replaying scenarios on different devices in both evaluation settings.

During evaluations, we find that replaying scenarios on the RedMi 1s and Galaxy A8 is much simpler than on the Galaxy A9. Even though RedMi 1s and Galaxy A8 vary in dpi and screen sizes,

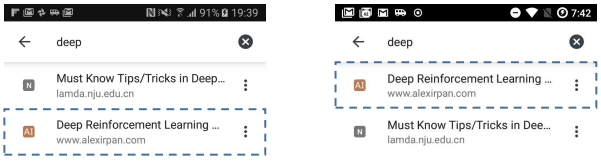

Figure 2: For the app Google Chrome, we select the second item during a record on the RedMi 1s (left). During a replay on the Galaxy A8 (right), we find that the item is prioritized to the first one. appetizer fails to replay the selection as the event it recorded is insensitive to widget. SARA instead successfully replays the selection.

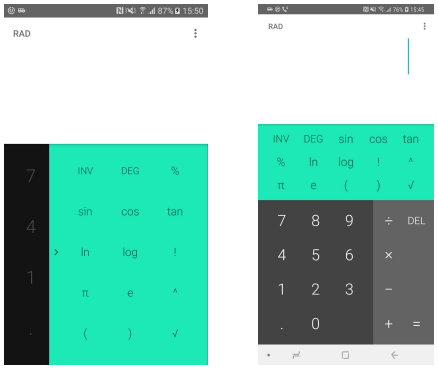

Figure 3: The layouts of Google Calculator on the Galaxy A8 (left) and Galaxy A9 (right) varies greatly.

the layouts of an app on them are roughly the same. Hence, scaling recorded coordinates with respect to screen resolution works well on these two devices. In fact, appetizer heavily relies on this scaling technique to support replaying events on multiple devices. In contrast to appetizer, SARA first tries to search for the widgets and then transforms the coordinates relative to the target widget before performing a motion event, making it more robust in dealing with the state differences of an app in replaying phase. Figures 2 demonstrates the robustness of SARA through an illustrative example.

When replaying events on the Galaxy A9, we find that the layouts of an app are usually different from that on the RedMi 1s and Galaxy A8, indicating that purely scaling recorded coordinates is error-prone during a replay. Identifying the widgets under interactions and conducting coordinate transformation based on them can resolve the problem to some extent. Hence, SARA manages to replay most of the scenarios on the Galaxy A9.

We analyze the 10 scenarios that are recorded on the Galaxy A8 and SARA fails to replay on the Galaxy A9. The main cause of failures is that SARA fails to identify the correct widgets under interactions in the replaying phase. Seven of them are due to inaccurate view hierarchies extracted by uiautomator2 [6]. Since SARA relies heavily on the view hierarchy to find candidate widgets in the self-replaying phase, it can hardly recognize the correct widgets if none of the candidates are correct. Other factors accounting for the failures include breaking change in layout (as shown in Figure 3), nondeterministic state of the app (the target widget is not always

Table 5: Evaluation results of replaying events on devices with different screen sizes and OS versions.

App name	Scenario	RedMi 1s				Galaxy A8			
		Galaxy A8		Galaxy A9		RedMi 1s		Galaxy A9	
		SA.	ap.	SA.	ap.	SA.	ap.	SA.	ap.
Accuweather	Set units	✓	✗	✓	-	✓	✓	✓	-
Adobe Acrobat Reader	Change modes	✓	✓	✓	-	✓	✓	✓	-
BBC	Edit my news	✓	✗	✓	-	✓	✗	✗	-
Booking	Sort hotels	✓	✓	✓	-	✓	✓	✓	-
CNN	Save stories	✓	✓	✓	-	✓	✓	✓	-
Citymapper	Get me somewhere	✓	✓	✗	-	✗	✓	✗	-
eBay	Add to cart	✓	✗	✓	-	✓	✗	✓	-
ESPN	Add game alert	✓	✓	✓	-	✓	✓	✓	-
File Commander	Manage category	✓	✗	✓	-	✓	✗	✓	-
Flow Flee	Play 8 * 8	✓	✓	✗	-	✗	✓	✗	-
GoodRx	Search for condition	✓	✓	✓	-	✓	✓	✓	-
Google Calculator	Change answer format	✓	✓	✗	-	✓	✓	✗	-
Google Chrome	Search bookmarks	✓	✗	✓	-	✓	✗	✓	-
Google Translate	Translate	✓	✓	✓	-	✓	✓	✓	-
Google Wallpapers	Visit art Wallpapers	✓	✗	✓	-	✓	✓	✓	-
Instagram	Search peopole	✓	✗	✗	-	✓	✓	✗	-
Investing.com	Browse market	✓	✓	✓	-	✓	✓	✓	-
Keep Trainer	Add a plan	✓	✓	✓	-	✓	✓	✓	-
KFC	Order meal	✓	✓	✗	-	✓	✓	✗	-
LinkedIn	Edit profile	✓	✓	✓	-	✓	✓	✓	-
MSN Weather	Check weather	✓	✓	✓	-	✓	✓	✓	-
NBA	Change mode	✓	✓	✓	-	✗	✓	✓	-
OneDrive	Manage Folder	✓	✗	✓	-	✓	✗	✓	-
OneNote	New a section	✓	✓	✓	-	✓	✗	✓	-
Period Calendar	Add a note	✓	✓	✓	-	✓	✓	✓	-
QQ Music	Post comments	✓	✓	✓	-	✓	✗	✓	-
Redfin Real Estate	Filter	✓	✓	✓	-	✓	✗	✓	-
Realtor.com Real Estate	Search houses	✓	✓	✓	-	✓	✓	✓	-
Sketch	Browse my feed	✓	✗	✗	-	✓	✗	✗	-
Snapseed	View Edits	✓	✓	✓	-	✓	✗	✓	-
Ted	Search a topic	✓	✓	✓	-	✓	✓	✓	-
Textgram	Change canvas background	✓	✓	✗	-	✓	✓	✗	-
Trivago Hotel Search	Search hotels	✓	✓	✓	-	✓	✓	✓	-
Twitter	Create post	✓	✓	✓	-	✓	✓	✓	-
Twitter	Like and comment	✓	✓	✓	-	✓	✓	✓	-
Walmart	Sort & filter items	✓	✓	✓	-	✓	✓	✓	-
WeChat	Open photos	✓	✗	✓	-	✓	✓	✓	-
WeChat	Post moment	✓	✓	✓	-	✓	✗	✓	-
WhatsApp	Search message	✓	✓	✓	-	✓	✓	✓	-
WhatsApp	Create group chat	✓	✓	✓	-	✓	✓	✓	-
Wikipedia	Search wikipedia	✓	✓	✗	-	✓	✓	✗	-
Youtube	Search videos	✗	✗	✓	-	✗	✗	✗	-
Total		**41**	**31**	**34**	-	**38**	**30**	**32**	-

shown in the app), and a problem of uiautomator2 in performing swipe gesture on the device.

5.4 Overhead

Table 6 shows the 36 sampled scenarios replayable with SARA, and the time and space overhead of SARA on these scenarios. 'Original Time' indicates the original runtime of a scenario without applying a record-and-replay tool. 'Time' in 'Recording phase' and 'Replaying phase' headers show the runtime of a scenario in the recording phase and replaying phase of SARA, respectively.

As shown in the table, the average record and replay overheads of SARA are 4.29% and 7.07%, respectively. Although SARA applies the dynamic instrumentation technique, it introduces relatively low overhead in the recording phase, which is critical in industrial settings as testers are involved in this phase. As for the replaying phase, we find that the overhead is mainly introduced in the process of searching for widgets when replaying motion events. Note that several apps, e.g., ESPN, have overhead of around 10.21%. Upon investigation, we find that the search for target widgets is much slower than most cases owing to their deeper depth in view hierarchy. The time overhead in searching for widgets is closely related to the complexity of the layout of an app. On average, SARA achieves low enough time overhead in both the recording phase and replaying phase.

The last two columns give the space overhead of SARA, including the size and rate of recorded logs (SARA does not store logs on phones). As the table illustrates, the average log size of is 77 KB and the average log rate is 1.78 KB per second. The extremely low space overhead meets our expectations, since SARA records input data in the application layer. The size of logs depends on user operations and duration of the scenarios.

6 DISCUSSION

Threats to Validity. As is the case for most empirical evaluations, there are both external and constructive threats to validity associated with the results we present. In terms of external validity, our results might not generalize to other industrial apps and other Android devices. In particular, we only consider 3 real phones, 53

Table 6: The time and space overhead of SARA.

App name	Scenario	Time					Space	
		Original Time (sec.)	Recording phase		Replaying phase		Log size (KB)	Log rate (KB/s)
			Time (sec.)	Overhead (%)	Time (sec.)	Overhead (%)		
Accuweather	Check daily weather	49.74	51.42	3.38	52.27	5.09	70	1.41
Adobe Acrobat Reader	Add a comment	31.16	31.17	0.03	33.59	7.80	57	1.83
Alipay	Start a group chat	38.89	39.30	1.05	39.80	2.34	97	2.49
BBC News	Edit my news	33.67	34.87	3.56	36.65	8.85	58	1.72
Booking	Search hotel	35.20	38.45	9.23	39.31	8.84	35	0.99
Citymapper	Find bus or line	51.13	53.17	3.99	53.30	4.24	73	1.43
CNN	Read top news	34.48	35.61	3.28	35.61	3.28	79	2.29
Daylio	Add notes	39.77	42.35	6.49	43.30	8.88	34	0.85
ESPN	Add a favourite	27.72	29.10	4.98	30.55	10.21	35	1.26
File Commander	Manage Category	17.53	19.25	9.81	19.42	10.78	13	0.74
Flow Flee	Play 5*5	34.70	36.74	5.88	36.62	5.53	98	2.82
Gmail	Search mail	41.72	44.24	6.04	45.11	8.13	124	2.97
GoodRx	Pill identifier	44.54	47.98	7.72	48.40	8.67	117	2.63
Google Chrome	Search bookmarks	22.11	24.77	12.03	24.12	9.09	100	4.52
Google Photos	Search photos	46.71	48.95	4.80	46.81	0.21	77	1.65
Google Translate	Translate	49.16	51.32	4.39	51.42	4.60	95	1.93
Instagram	Create a post	20.37	21.76	6.82	22.90	12.42	63	3.09
Keep Trainer	Begin a workout	51.88	52.77	1.72	55.70	7.36	54	1.04
Kindle	Read book	41.40	45.69	10.36	45.63	10.22	44	1.06
McDonald's	Order meals	100.01	100.77	0.76	109.99	9.98	185	1.85
MSN Weather	Add favorite city	28.21	28.92	2.52	28.69	1.70	86	2.97
Period Calendar	Write a record	53.74	53.88	0.26	58.46	8.78	61	1.14
QQ Music	Search music	48.57	49.65	2.22	51.98	7.02	113	2.33
Realtor.com Real Estate	Search houses	71.81	72.47	0.92	74.20	3.33	172	2.40
Redfin Real Estate	Search houses	28.97	29.62	2.24	30.22	4.31	32	1.10
Sketch	New canvas	44.63	47.21	5.78	47.90	7.32	32	0.68
Snapseed	Edit with tools	27.56	28.86	4.72	30.85	11.94	27	0.98
Taobao	Browse a store	84.29	91.56	8.62	92.26	9.46	186	2.21
TED	Watch a video	73.22	75.29	2.83	77.81	6.27	118	1.61
Textgram	Create a new canvas	24.88	25.58	2.81	25.88	4.02	17	0.68
Trivago Hotel Search	Check hotel	56.73	58.15	2.50	57.70	1.71	112	1.93
VLC for Android	Jump to time	43.78	44.97	2.72	45.21	3.27	27	0.62
Wallpapers HD	Refresh daily wallpaper	46.35	46.40	0.11	52.45	13.16	61	1.32
Walmart	Sort items	48.83	49.24	0.84	52.62	7.76	129	2.64
WeChat	Open photos	22.43	24.18	7.78	24.65	9.86	16	0.71
Wikipedia	Browse reading list	38.80	39.24	1.13	41.94	8.09	84	2.16
Average		**43.19**	**44.86**	**4.29**	**46.18**	**7.07**	**77**	**1.78**

industry apps and 265, 42, and 36 common usage scenarios in RQ1, RQ2 and RQ3, respectively. This limitation is an artifact of complexity involved in figuring out scenarios, setting up environment and recording scenarios on different devices. To mitigate this threat, we sample top-recommended apps with the highest numbers of downloads from each category in Google Play and figure out scenarios from the description in Google Play and our hands-on experiences. The three real phones we select cover the most widely used Android OS versions and screen sizes at the time of writing [12]. We believe that, given the highly popular industrial apps from broad categories and the representative phones we select, SARA should also be applicable to other industrial apps and Android devices. In terms of constructive validity, there might be errors in the implementation of SARA. To mitigate this threat, we extensively inspect the evaluation results.

Limitations. There are still some limitations of SARA. First, hybrid apps (apps that are built with a mix of both native and web technologies) pose steering challenges in recording and replaying events based on widgets, since that some widgets are rendered by WebView as HTMLElements, which cannot be found in the view hierarchy. Second, game apps are usually developed with game engines (e.g., Unity [37]). Most of the widgets in game apps are rendered by the game engine, and also cannot be found in the view hierarchy, making SARA fail to work with these widgets.

Implication. WeChat has been a highly popular industrial-strength Android app with more than 1 billion monthly active users. The requests for a reliable record-and-replay tool with desired features from the WeChat developers are valuable for the research community and the industry, which motivates us to design and implement SARA. Although the motivation of SARA comes from WeChat, the adoption of SARA does not limit to WeChat. We are looking forward to working with industrial partners to improve SARA and integrate SARA into their testing and maintenance process.

REFERENCES

[1] 2019. Culebra. https://github.com/dtmilano/AndroidViewClient/wiki/culebra.
[2] 2019. Frida. https://www.frida.re/.

[3] 2019. Replaykit. https://github.com/appetizerio/replaykit.
[4] 2019. SARA. https://sites.google.com/view/sara-record-and-replay.
[5] 2019. Selenium. https://www.seleniumhq.org/.
[6] 2019. uiautomator2. https://github.com/openatx/uiautomator2.
[7] Quan Do, Guowei Yang, Meiru Che, Darren Hui, and Jefferson Ridgeway. 2016. Regression Test Selection for Android Applications. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*. ACM, New York, NY, USA, 27–28. https://doi.org/10.1145/2897073.2897127
[8] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and X Wang. 2018. Things you may not know about android (un) packers: a systematic study based on whole-system emulation. In *25th Annual Network and Distributed System Security Symposium, NDSS*. 18–21.
[9] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. 2002. ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 211–224. https://doi.org/10.1145/844128.844148
[10] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. 2013. RERAN: Timing- and Touch-sensitive Record and Replay for Android. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, Piscataway, NJ, USA, 72–81. http://dl.acm.org/citation.cfm?id=2486788.2486799
[11] Google. 2019. Android UI Automator. https://developer.android.com/training/testing/ui-automator.
[12] Google. 2019. Distribution dashboard. https://developer.android.com/about/dashboards/.
[13] Google. 2019. Espresso Test Recorder. https://developer.android.com/studio/test/espresso-test-recorder.
[14] Google. 2019. GestureDetector. https://developer.android.com/reference/android/view/GestureDetector.
[15] Google. 2019. monkeyrunner. https://developer.android.com/studio/test/monkeyrunner/.
[16] Google. 2019. SensorEventListener. https://developer.android.com/reference/android/hardware/SensorEventListener.htm.
[17] Google. 2019. Support different screen sizes. https://developer.android.com/training/multiscreen/screensizes.
[18] Matthew Halpern, Yuhao Zhu, Ramesh Peri, and Vijay Janapa Reddi. 2015. Mosaic: cross-platform user-interaction record and replay for the fragmented android ecosystem. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, Philadelphia, PA, USA, 215–224. https://doi.org/10.1109/ISPASS.2015.7095807
[19] Mouna Hammoudi. 2016. Regression Testing of Web Applications Using Record/Replay Tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, USA, 1079–1081. https://doi.org/10.1145/2950290.2983942
[20] Yongjian Hu, Tanzirul Azim, and Iulian Neamtiu. 2015. Versatile Yet Lightweight Record-and-replay for Android. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM, New York, NY, USA, 349–366. https://doi.org/10.1145/2814270.2814320
[21] Imaginea. 2019. Bot-bot. http://imaginea.github.io/bot-bot/.
[22] Bo Jiang, Yu Wu, Yongfei Zhang, Zhenyu Zhang, and Wing-Kwong Chan. 2018. ReTestDroid: Towards Safer Regression Test Selection for Android Application. In *2018 IEEE 42nd Annual Computer Software and Applications Conference*, Vol. 01. 235–244. https://doi.org/10.1109/COMPSAC.2018.00037
[23] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, Baltimore, MD, USA, 15–24. https://doi.org/10.1109/ESEM.2013.9
[24] Milan Jovic, Andrea Adamoli, Dmitrijs Zaparanuks, and Matthias Hauswirth. 2010. Automating Performance Testing of Interactive Java Applications. In *Proceedings of the 5th Workshop on Automation of Software Test*. ACM, New York, NY, USA, 8–15. https://doi.org/10.1145/1808266.1808268
[25] Shahedul Huq Khandkar, S. M. Sohan, Jonathan Sillito, and Frank Maurer. 2010. Tool Support for Testing Complex Multi-touch Gestures. In *ACM International Conference on Interactive Tabletops and Surfaces*. ACM, New York, NY, USA, 59–68. https://doi.org/10.1145/1936652.1936663
[26] Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. 2015. Understanding the Test Automation Culture of App Developers. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation*. IEEE, Graz, Austria, 1–10. https://doi.org/10.1109/ICST.2015.7102609
[27] Pingfan Kong, Li Li, Jun Gao, Kui Liu, TegawendÃĪ F. BissyandÃĪ, and Jacques Klein. 2019. Automated Testing of Android Apps: A Systematic Literature Review. *IEEE Transactions on Reliability* 68, 1 (2019), 45–66. https://doi.org/10.1109/TR.2018.2865733
[28] Wing Lam, Zhengkai Wu, Dengfeng Li, Wenyu Wang, Haibing Zheng, Hui Luo, Peng Yan, Yuetang Deng, and Tao Xie. 2017. Record and Replay for Android: Are We There Yet in Industrial Cases?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, USA, 854–859. https://doi.org/10.1145/3106237.3117769
[29] Mario Linares-VÃąsquez, Carlos Bernal-Cardenas, Kevin Moran, and Denys Poshyvanyk. 2017. How do Developers Test Android Applications?. In *2017 IEEE International Conference on Software Maintenance and Evolution*. IEEE, Shanghai, China, 613–622. https://doi.org/10.1109/ICSME.2017.47
[30] Kevin Moran, Richard Bonett, Carlos Bernal-Cárdenas, Brendan Otten, Daniel Park, and Denys Poshyvanyk. 2017. On-Device Bug Reporting for Android Applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems*. IEEE Press, Piscataway, NJ, USA, 215–216. https://doi.org/10.1109/MOBILESoft.2017.36
[31] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2016. FUSION: A Tool for Facilitating and Augmenting Android Bug Reporting. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, New York, NY, USA, 609–612. https://doi.org/10.1145/2889160.2889177
[32] Satish Narayanasamy, Gilles Pokam, and Brad Calder. 2005. BugNet: continuously recording program execution for deterministic replay debugging. In *32nd International Symposium on Computer Architecture*. IEEE, Washington, DC, USA, 284–295. https://doi.org/10.1109/ISCA.2005.16
[33] Zhengrui Qin, Yutao Tang, Ed Novak, and Qun Li. 2016. MobiPlay: A Remote Execution Based Record-and-replay Tool for Mobile Applications. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, New York, NY, USA, 571–582. https://doi.org/10.1145/2884781.2884854
[34] Ranorex. 2019. ranorex. https://www.ranorex.com/mobile-automation-testing/android-test-automation/.
[35] RobotiumTech. 2019. robotiumrecorder. https://github.com/RobotiumTech/robotium.
[36] Sudarshan M. Srinivasan, Srikanth Kandula, Christopher R. Andrews, and Yuanyuan Zhou. 2004. Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. USENIX Association, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=1247415.1247418
[37] Unity Technologies. 2019. Unity. https://unity.com.
[38] WeChat. 2019. Wechat. https://play.google.com/store/apps/details?id=com.tencent.mm.