2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE) | 979-8-3503-1594-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISSRE59848.2023.00056

ReSPlay: Improving Cross-Platform Record-and-Replay with GUI Sequence Matching

Shaokun Zhang[a], Linna Wu[b,c], Yuanchun Li[d], Ziqi Zhang[a], Hanwen Lei[a], Ding Li[a†], Yao Guo[a†], Xiangqun Chen[a]
[a] Key Lab of High-Confidence Software Tech (MOE), School of Computer Science, Peking University, Beijing, China
[b] Key Laboratory of Mobile Application Innovation and Governance Technology, Beijing, China
[c] CTTL Terminals Labs, China Academy of Information and Communications Technology, Beijing, China
[d] Institute for AI Industry Research, Tsinghua University, Beijing, China
{skzhang, ziqi_zhang, ding_li, yaoguo, cherry}@pku.edu.cn, lei_hanwen@stu.pku.edu.cn,
wulinna@caict.ac.cn, liyuanchun@air.tsinghua.edu.cn

***Abstract*—Record-and-replay is an important testing technique to ensure the quality of mobile applications (apps in short). State-of-the-art record-and-replay approaches are typically based on widget matching, which has shown limited effectiveness, especially on devices with different platforms and resolutions, due to the difficulty in matching widgets with subtle visual differences. Our key observation is that, even if two widgets look similar, the resulting screenshot sequences can still be very different during execution. Thus, instead of matching GUI widgets directly, we are able to find the correct replay actions by comparing the resulting GUI screenshot sequences, which can be better distinguished across different platforms, thus potentially improving the record-and-replay efficiency through GUI exploration and comparison.**

This paper proposes a general record-and-replay framework called ReSPlay, which leverages a more robust visual feature, GUI sequences, to guide replaying more accurately. ReSPlay pre-trains a deep reinforcement learning model, SDP-Net, offline from random app traces. Specifically, SDP-Net is trained to search a particular path from GUI transition graphs to learn an optimal policy to locate the target operation positions by maximizing the possibilities to reach the target GUI sequence. Finally, the trained SDP-Net is used to search for potential event traces with high rewards and replicate them on the target device for replay. We evaluate our proposed framework on multiple real devices. Experimental results show that the overall average replay accuracy of ReSPlay on devices across different OSes, GUI styles, and resolutions is 28.12% higher than the state-of-the-art baselines.

I. Introduction

Record-and-replay techniques, which automatically rerun user actions on different mobile devices, can help save developers from repetitive manual testing [1]. Conventional attribute-based and pixel-based techniques are focused on replaying user actions on similar devices, which have limited application scope [2]–[4]. Recently, researchers have proposed image-based record-and-replay techniques [3], [5]–[12] to support cross-platform record-and-replay. Although these techniques look promising, they still suffer from *low replay accuracy* issues for cross-platform testing, as their replaying accuracy rarely tops 60% [3], [5]–[12].

The state-of-the-art image-based cross-platform record-and-replay techniques have sub-optimal accuracy because they rely on features that are not robust when dealing with *similar GUI widgets*. Existing image-based techniques [3], [5]–[12] directly match user actions according to visual features of the GUI widgets (e.g., matching widgets that look similar). However, since different GUI widgets may look visually similar, current techniques have a high chance of introducing mismatches while correlating widget execution, leading to sub-optimal replay accuracy [2], [8]. A feature that is robust to visually similar GUI widgets is desired to improve the accuracy of cross-platform record-and-replay.

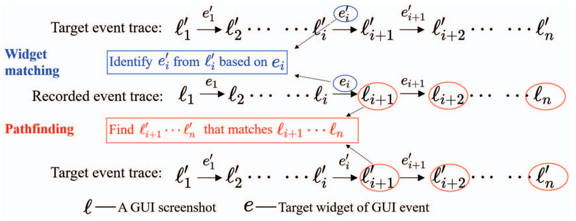
Target event trace: $\ell'_1 \xrightarrow{e'_1} \ell'_2 \cdots \cdots \ell'_i \xrightarrow{e'_i} \ell'_{i+1} \xrightarrow{e'_{i+1}} \ell'_{i+2} \cdots \cdots \ell'_n$
Widget matching
Identify e'_i from ℓ'_i based on e_i
Recorded event trace: $\ell_1 \xrightarrow{e_1} \ell_2 \cdots \cdots \ell_i \xrightarrow{e_i} \ell_{i+1} \xrightarrow{e_{i+1}} \ell_{i+2} \cdots \cdots \ell_n$
Pathfinding
Find $\ell'_{i+1} \cdots \ell'_n$ that matches $\ell_{i+1} \cdots \ell_n$
Target event trace: $\ell'_1 \xrightarrow{e'_1} \ell'_2 \cdots \cdots \ell'_i \xrightarrow{e'_i} \ell'_{i+1} \xrightarrow{e'_{i+1}} \ell'_{i+2} \cdots \cdots \ell'_n$
ℓ — A GUI screenshot
e — Target widget of GUI event

Fig. 1: Comparison between the core insight of ReSPlay and existing image-based solutions.

In this paper, to improve the accuracy of image-based cross-platform record-and-replay, we propose ReSPlay, a framework that leverages image features that are robust to similar GUI widgets. ReSPlay is fundamentally different from existing record-and-replay techniques in its design principle: *instead of matching GUI widgets across platforms, ReSPlay attempts to match the resulting GUI screenshot sequences after executing a widget on different devices*. In other words, unlike existing techniques that search for the same widget on different platforms, ReSPlay aims to find widgets that can lead to the same GUI sequences across devices. The general idea of ReSPlay is shown in Fig. 1. The upper part of Fig. 1 shows the high-level ideas of existing techniques which match the recorded GUI widgets e_i to their counterpart e'_i on the target device. In comparison, in the lower part of Fig. 1, ReSPlay searches for an action that leads to the sequence of GUI screenshots (ℓ'_{i+1} to ℓ'_n) on the target device that matches the coming recorded GUI screenshots (ℓ_{i+1} to ℓ_n) on the source device.

The intuition behind our design is as follows. Matching GUI widgets is challenging because there may exist multiple widgets with the same or similar looks, thus making it difficult to distinguish between them. However, GUI screenshots are

†Corresponding authors.

robust to visually similar GUI widgets because they contain multiple GUI widgets that are well organized [13]. The probability for two GUI screenshots to have the same set of GUI widgets that are organized similarly is much lower than the probability of two single widgets that look similar. Therefore, if we consider the resulting GUI screenshots after executing the intended widget, the chance of making mismatches will be lower. Furthermore, if we consider a sequence of screenshots, the chance of making mismatches will decrease further.

Although our intuition of matching the sequence of GUI screenshots is straightforward, realizing it is conceptually challenging. The key challenge is how to consider the screenshots that cannot be seen at the time when a widget execution is taken. One widget execution can only produce one screenshot, which is the next GUI screenshot. Subsequent GUI screenshots cannot be seen before future widget executions are taken. However, since our approach aims to find the execution that can lead to the same sequence of GUI screenshots, it is important to know all subsequent GUI screenshots that a widget execution may produce, which is computationally expensive.

To address this challenge, we leverage reinforcement learning [14], [15], which enables our approach to make decisions based on unseen screenshots. Specifically, we deploy a DQN framework that uses a deep neural network to predict a cumulative reward, which measures the similarity between a sequence of GUI screenshots generated by an action on the target device and the sequence of GUI screenshots generated on the source device. Using the DQN-based RL learning reduces the computational complexity of searching by compressing all future GUI screenshots into a neural network, which we can solve with back-propagation [16], [17].

Besides the research challenges, realizing ReSPlay also faces two technical challenges. First, the number of training samples is limited. It is difficult to train the neural network directly due to the limited number of training samples. ReSPlay leverages pre-trained models to address this problem [18]–[20]. Second, the dimension of the output space of the neural network is too high. ReSPlay leverages the neural network to predict the coordinates of the next GUI action on the screen. However, since modern high-resolution screens have millions of pixels [21], a neural network cannot accurately pinpoint the GUI action's coordinate with such a high dimensional output space. To address this challenge, we first set a fixed output size greater than or equal to the maximum possible number of actions in a GUI state and then create a mask to filter out invalid actions.

We evaluate ReSPlay on six different experimental devices with various platforms, as well as 12 real-world apps from different categories. Specifically, we achieve a 17.89% improvement in accuracy for cross-OS replay. For cross GUI styles replay, the average accuracy of ReSPlay is 88.63%, which improves the accuracy of the baseline methods by 37.47%. For cross-resolution replay, ReSPlay increases the average accuracy by 28.98% compared to the baseline. Overall, ReSPlay consistently improves accuracy over the SOTA baselines in all cases.

Contributions. Our contributions are summarized as follows.

- To overcome the challenges in widget matching, we propose to use the sequence of future GUI screenshots as a new feature that is robust to distinguish visually similar widgets.
- We develop and implement a cross-platform record-and-replay framework called ReSPlay that leverages our new features with deep reinforcement learning.
- We perform extensive experiments to evaluate the performance of ReSPlay in replaying events on devices of different platforms. The results demonstrate that our approach can significantly improve the replay accuracy of the state-of-the-art baselines for cross-OS, resolution, and GUI style record-and-replay.

Data Availability: Our tool has been released on GitHub: https://github.com/skzhangPKU/ReSPlay.

II. Background and Related Work

Record-and-replay has been explored in automated testing for many years. A variety of techniques have been developed to replay test scripts on devices with various platforms and OS versions [2]–[4], [10], [11], [22], [23]. According to the type of entity used for matching, these techniques can be broadly categorized into three distinct groups: pixel-based, attributed-based, and image-based approaches.

A. Pixel-based Mobile Test Record and Replay

Early attempts have been made to replay event traces based on matching image pixels to the same coordinate on the same device [1], [4], [24]–[28]. For instance, Gomez *et al.* [4] proposed a non-intrusive record-and-replay tool named RERAN for the Android platform. The limitation of pixel-based techniques is that they are not robust to changes in resolutions, screen sizes, GUI styles, and OSes. Thus, they cannot support cross-platform record-and-replay [2], [3].

B. Attribute-based Mobile Test Record and Replay

People develop attribute-based methods to replay event traces on devices with the same OS but different resolutions [3], [5]–[7], [28]–[33]. These techniques match the GUI operation logs to layout files to support record-and-replay. The problem with attribute-based techniques is that they cannot support cross-OS record-and-replay because different OSes have distinct GUI action logs and layout files [2], [34].

C. Image-based Mobile Test Record and Replay

To address the limitations of Pixel-based and Attribute-based techniques, researchers also propose image understanding techniques to locate target widgets [8]–[12], [35]–[41]. A recent study [2] has shown that image-based techniques achieve higher accuracy than the other two types. Existing image-based techniques are either purely visual or hybrid. For the former, the technique with the highest accuracy in this type is Sikuli [9], which leverages image recognition methods

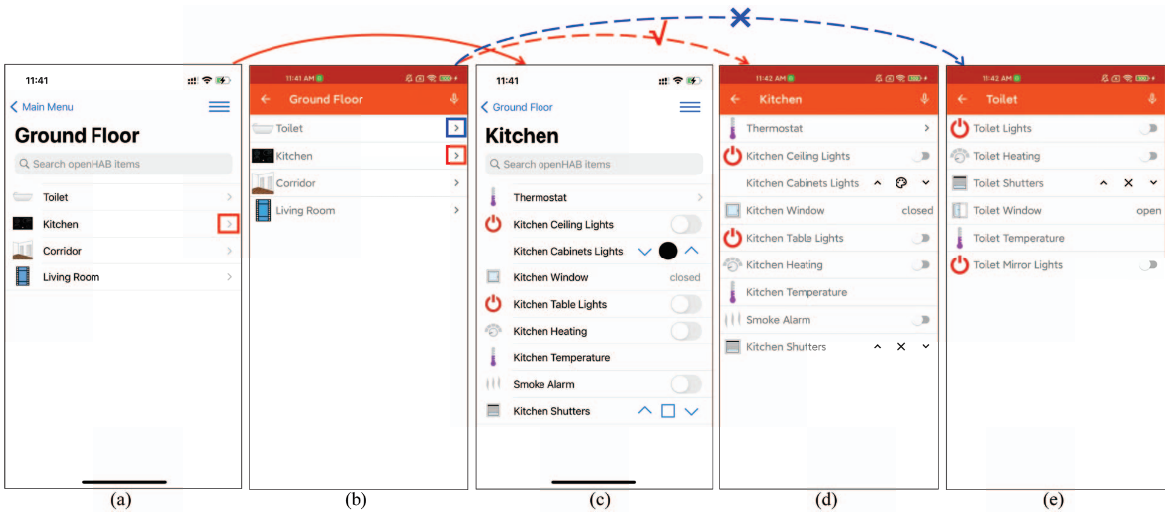

Fig. 2: A motivating example based on the OpenHAB app, including screenshots on iPhone 11 (a and c) and Honor 10 (b, d, and e). Here, the red box in (a) indicates the recorded widget. The blue and red boxes in (b) indicate the operated widget and the actual target widget on the replaying device, respectively. Figures (c), (e), and (d) indicate GUI screenshots after operating the three different widgets, respectively.

to match widgets. For the latter, representative techniques in this group are LIRAT [2] and MAPIT [39], which combine widget image features with layout features or meta-attributes. MAPIT assumes that most GUI widgets have meta-attributes, which is not true for about 80% of apps revealed by recent studies on large-scale analysis [34], [42]. Thus, compared with MAPIT, LIRAT is more generalizable and more suitable for cross-platform record-and-replay.

The limitation of current image-based approaches is that they directly match GUI widgets across devices. They may mismatch widget executions due to similar-looking widgets, causing errors during replaying [8], [9], [11], [43]. Unlike these techniques, ReSPlay leverages the future GUI screenshots sequences, which are robust to similar GUI widgets, for cross-platform replay. Therefore, ReSPlay has fewer mismatches and higher accuracy.

III. A Motivating Example

For illustration, we choose a smart home app with a community of more than 40K active members and developers, *openHAB* [44]. The example is illustrated in Fig. 2, which shows the scenario of replaying an iOS (iPhone 11) test case on an Android phone (Honor 10). Fig. 2(a) shows the GUI screenshot on the source device, in which the red box points to a widget that leads to the status of all connected devices in the kitchen (Fig. 2(c)). Fig. 2(b) shows the corresponding GUI screenshot of Fig. 2(a) on the Android phone, which contains two widgets (which look the same) marked by blue and red boxes, which lead to the device status of the toilet (Fig. 2(e)) and the kitchen (Fig. 2(d)), respectively.

a) Challenges: Ideally, to replay an operation on the widget in the red box in Fig. 2(a), the record-and-replay tool should execute the widget in the red box in Fig. 2(b). However, it is challenging for current solutions to identify the correct widget accurately. Pixel-based techniques fail because the coordinates of the widget in the red box in Fig. 2(a) and the one in the red box in Fig. 2(b) are completely different. Attribute-based techniques cannot match the red box in Fig. 2(b) to the red box in Fig. 2(a) because it is hard to match the system attributes of Android to the attributes of iOS. Finally, image-based methods may mistake the widget in the blue box in Fig. 2(b) for the one in the red box in Fig. 2(a) because the two widgets look identical.

b) ReSPlay: Unlike these existing techniques, ReSPlay does not search for the corresponding widgets in the red box in Fig. 2(a). Instead, it searches for the correct action that can lead to Fig. 2(d), which is the corresponding matching GUI screenshot of Fig. 2(c). Matching the resulting GUI screenshots yields a higher accuracy because GUI screenshots contain multiple widgets, so they are more likely to show detectable visual differences compared to single widgets. For instance, although the widgets in the blue and red boxes in Fig. 2(b) are identical, the GUIs that they may lead to are different (i.e., Fig. 2(e) and Fig. 2(d)). Therefore, ReSPlay can accurately distinguish between the widgets in the blue and red boxes in Fig. 2(b) by checking the resulting GUI screenshots in Fig. 2(e) and Fig. 2(d)[1].

IV. Overview

We first define the necessary terms used in this paper. We define a *GUI state* as a screenshot of the mobile device and a *GUI trace* as a sequence of GUI states.

The workflow of ReSPlay is shown in Fig. 3. It consists of two phases: synchronous recording and fine-tuned replaying. In the recording phase, developers run the record module of ReSPlay and then operate the target app on the source device

[1]We only use the next GUI screenshot in Fig. 2 for the clearness of the presentation. In practice, ReSPlay considers a sequence of future GUI states.

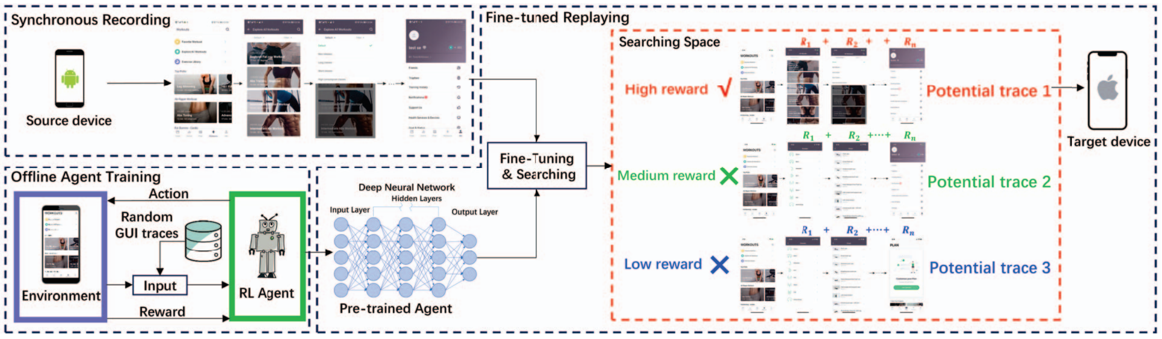

Fig. 3: The workflow of our proposed framework.

manually. This step generates the GUI screenshots that would be used for the replaying phase. In the fine-tuned replaying phase, ReSPlay leverages an offline trained model to search for potential event traces and replay them on the target device.

The core part of ReSPlay is the Fine-tuned Replaying phase, which searches for the matched screenshot sequences on the target device. The general process of the Fine-tuned Replaying phase is modeled as a reinforcement-learning process: an RL agent is designed to explore the GUI space by optimizing a reward. The reward is larger when the explored GUI states match the given sequence of screenshots. Otherwise, the reward is lower. We model the process as RL because it considers future state sequences, leading to more robust feature matching, and does not require labeled data.

V. Design of ReSPlay

We discuss the detailed design of the components in ReSPlay.

A. *Synchronous Recording*

The goal of this phase is to record the GUI traces from the source device. Specifically, ReSPlay records the following information: GUI screenshots and widget screenshots. A widget screenshot is the smallest image region that wraps the widget in a GUI screenshot. GUI screenshots are used for two different purposes: one is to check whether test cases are successfully replayed, and the other is as part of the model input. ReSPlay records GUI screenshots using an Appium API [45]. For convenience, ReSPlay also extracts widget screenshots cropped from GUI screenshots.

B. *RL Agent*

The RL agent is used to explore the GUI space on the target device and find the GUI states that match the given input. It interacts with the environment, which is defined as an entity that can return a reward and a new GUI state and automatically updates its search policy for more efficient searching. Our RL agent takes the DQN architecture. Besides this architecture, there are DDPG [46], A3C [47], and SAC [48]. However, these agent architectures are unsuitable for our approach because they require a longer training time to reach the same accuracy due to the lack of a cooperation mechanism [49]–[51]. Our RL agent consists of three components: Q network, action space, and reward. The RL agent requires an action space containing all possible actions to transfer between GUI states. For ReSPlay, the action space is all the possible GUI widgets a user may execute. The goal of the Q network is to learn an optimal policy to choose the action with the highest reward from the action space. The learning process is guided by the reward, which measures the similarity between the input screenshot sequence and the current screenshot sequence on the replay device.

1) Q Network: The structure of the Q network is called SDP-Net. Its backbone architecture is a pre-trained ResNet model [52]. The input of SDP-Net is a combined GUI state, which involves the current GUI state on the replaying device and the next GUI state on the recording device. The output is the likelihood of each action that leads to the target GUI sequence. This likelihood is termed a cumulative future reward, which includes the immediate reward at each step in the future. The immediate reward refers to the similarity score between the GUI states on the recording device and the replaying device.

A major challenge is the lack of data. Training an accurate model requires millions of images [53]. However, it is difficult to get this many images in the context of record-and-replay. Therefore, ReSPlay leverages a pre-trained model based on a publicly available mobile GUI dataset [54]. Although this model is not trained from GUI traces, it is useful for replaying GUI traces because latent semantic features learned are capable of distinguishing GUI states [42], [55], [56]. The benefit of using pre-trained models is that it allows us to achieve an efficient and robust approach in data-scarce settings. SDP-Net modifies the pre-trained model by replacing the original output layer with the output layer of SDP-Net, which is the index of the next GUI action.

2) Reward Function: Intuitively, the reward is used to measure the similarity between GUI state sequences. Specifically, we consider not only the next state but also a sequence of future GUI states in the reward to improve the robustness to similar GUI widgets. The reward is defined as follows:

$$R_t = \sum_{i=t}^{\infty} \gamma^i R_x^i \tag{1}$$

where γ^i and R_x^i denote the discount factor and immediate reward at time i, respectively. The discount factor indicates how much the RL agent cares about rewards in the distant future. *In particular, it is a cumulative future reward upon which ReSPlay models subsequent possible GUI sequences.* The key challenge for implementing the reward function is that we can only obtain a transition tuple, including the current state and next state, at each step while subsequent GUIs are unavailable. Since the sequence of future states is unknown at each step, it must be estimated recursively using the maximum future reward for the next state as an approximation. Specifically, the maximum future reward of the current state is the sum of the immediate reward and that of the next state. By repeatedly expanding the recursive formula, one can obtain the sum of the immediate rewards for all future states. This way, the agent can estimate the value of each state-action pair by looking ahead and evaluating the expected return. In other words, ReSPlay also considers potentially the matching of subsequent GUI states. Even when encountering similar GUIs, their subsequent GUIs may differ. ReSPlay can distinguish those GUIs similar to the target GUI based on the long-term reward that considers future GUI sequences.

The immediate reward R_x consists of the vision-level reward and the word-level reward. The vision-level reward is necessary but insufficient because it ignores local changes introduced by text in the GUI state. Fig. 4 illustrates an example of local changes. In this case, Fig. 4(a) and 4(b) cannot be distinguished solely based on visual similarity because existing techniques cannot detect subtle differences between the texts in blue boxes. To address this issue, we propose a word-level reward that considers the textual differences between GUI states. Specifically, when considering an operation, the GUI after taking an action on the recording device is denoted as $\mathcal{X}$. Correspondingly, the one on the replaying device is denoted as $\widehat{\mathcal{X}}$. The immediate reward is defined as follows:

$$R_x(\mathcal{X}, \widehat{\mathcal{X}}) = \sum_{k \in \{w,v\}} \Gamma^k R_k(\mathcal{X}, \widehat{\mathcal{X}}) \tag{2}$$

where Γ^* is a penalty factor that regulates the balance of the effects of visual semantics and text semantics, and R_w and R_v represent the word-level reward and the vision-level reward, respectively. The Γ^* is used to improve the robustness to noise [57]. The specific details are described as follows.

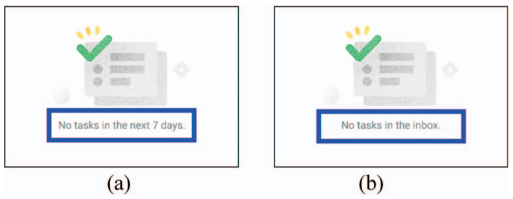

Fig. 4: Local changes for the Evernote app on one device.

1) Vision-level reward. Vision-level reward measures the visual differences between GUI states by calculating the Euclidean distance between their compressed vector representations. Calculating the Euclidean distance directly on GUI screenshots is not feasible because it loses dependencies between pixels, which is important to measure image similarities [58], [59]. To capture the dependencies, we extract meaningful features from images into a vectorized form using an embedded model. In our framework, we take an unsupervised visual representation model [52], denoted as VBN, as the embedded model. We use the output of the last hidden layer of the model as the embedding for two images. This method is motivated by recent CV techniques [60], [61]. Given two GUI images $\mathcal{X}$ and $\widehat{\mathcal{X}}$ dumped from the recording device and the replaying device, we define the vision-level reward as follows:

$$R_v(\mathcal{X}, \widehat{\mathcal{X}}) = \mathcal{I}(sim(VBN(\mathcal{X}), VBN(\widehat{\mathcal{X}})) < \delta) \tag{3}$$

where $\mathcal{I}$ represents the indicator function, δ stands for the image similarity threshold, VBN is the embedding function that compresses a two-dimensional image into a one-dimensional vector, and $sim(\cdot,\cdot)$ denotes a similarity metric based on Euclidean distance. We choose to use the indicator function since prior studies have shown that a discrete reward function can help improve accuracy [62], [63].

2)Word-level reward. Word-level reward measures the textual differences in GUI states by computing the sequence similarity between the corresponding strings. We design this reward to capture subtle local changes introduced by texts. Specifically, we apply optical character recognition (OCR) technology [64] to extract text from images and then compare the sequence similarity between them. The word-level reward can be formulated as follows:

$$R_w(\mathcal{X}, \widehat{\mathcal{X}}) = \mathcal{I}(sim_{seq}(\mathcal{X}, \widehat{\mathcal{X}}) > \gamma) \tag{4}$$

$$sim_{seq}(\mathcal{X}, \widehat{\mathcal{X}}) = 2 * \frac{f'_T(X) \cap f'_T(\widehat{\mathcal{X}})}{f'_T(X) \cup f'_T(\widehat{\mathcal{X}})} \tag{5}$$

where $f'_T(\cdot)$ denotes the OCR procedure, γ represents the text similarity threshold, and sim_{seq} stands for the sequence similarity [65]. We choose not to directly extract text from layout files because multiple GUI states may share a layout file.

3) Action Space Design: We use all clickable GUI widgets in a GUI state as action space. Several prior studies [66]–[68] employ grid division methods, resulting in numerous invalid regions, such as unresponsive regions post-click. In contrast, our approach operates directly on all clickable widgets, ensuring higher efficiency. Nonetheless, one issue arises as each GUI state may contain a different number of widgets, whereas the Q network requires a fixed-sized output. To address this concern, we find the largest number of widgets in an individual GUI state from the RICO dataset and set it as the action space size. Specifically, given a GUI state, we create a mask vector to prevent the Q network from predicting invalid actions.

C. Offline Agent Training

We train the RL agent offline. This is achieved by running a monkey runner to generate random traces on a set of apps. To improve GUI coverage, we use manually recorded event traces from the recording phase as seeds for random mutations [69], [70]. Notably, we only need to train the RL agent once.

D. Fine-tuned Replaying Phase

During the replaying phase, our approach leverages the trained RL agent to predict the best possible action leading to the target GUI screenshot. Note that our RL agent is fine-tuned during the replaying. It updates the deep learning models while replaying new traces. In other words, the accuracy of ReSPlay increases when it replays more traces. The model learned by one trace can be reused by new traces.

Fine-tuning is achieved by directly deploying the trained agent to replay GUI traces on apps under test. Specifically, the trained agent is used to explore the GUI space on the apps and search for potential event traces with high rewards. This experience is stored as a transition tuple and put into the replay buffer for fine-tuning. After fine-tuning, the trained agent is used to replay the recorded events accurately on target devices with different platforms, GUI styles, and resolutions.

VI. Evaluation

In this section, we conduct extensive experiments to compare our proposed framework with other approaches on multiple devices with different platforms. The goal of this evaluation is to investigate the following research questions:

- **RQ1:** How effective is ReSPlay in replaying events on mobile devices across diverse OSes?
- **RQ2:** How effective is ReSPlay in replaying events on mobile devices across different GUI styles?
- **RQ3:** How effective is ReSPlay in replaying events on mobile devices across different resolutions?
- **RQ4:** How does the new feature help in improving replay accuracy?

A. Framework Implementation

Our framework is implemented to interact with mobile devices through Appium, an open-source test automation tool [45]. In the offline training phase, our method is based on the Deep Q-Network, which is implemented in the Pytorch framework. The average time overhead of this phase is 3.45 hours. This is a one-time fixed cost to train the agent, which can be used multiple times on various apps. In particular, we extract a low-dimensional latent representation of states with AugNet [52], which is an unsupervised visual representation model based on ResNet-50 as the backbone. AugNet is fine-tuned on the mobile GUI dataset Rico [54]. The embedded text in GUI states is extracted by the Tesseract OCR engine [71]. In the implementation, widget matching is optional but can accelerate the consecutive replay. In addition, according to Mohammad *et al.* [72] and Xiao *et al.* [73], both δ and γ in the reward function are set to 0.8 since it achieves the best results. In cases without established references, we construct small-scale experiments to determine hyper-parameters based on best practices. The baselines are established by following previously reported protocols [2], [9].

We conduct dynamic monitoring at different phases to evaluate our framework's time overhead. In the recording and replaying phases, the time overheads are 2.48 and 7.34 minutes, respectively. For baselines, the time overhead of Sikuli and LIRAT in the recording phase is 23.75 minutes and 3.29 minutes, respectively. In the replaying phase, their time overheads are 2.62 and 3.57 minutes, respectively.

TABLE I: Devices used in the experiments.

Device	Model	Resolution	System
D0	Honor 10	1080×2280	EMUI 10.0.0
D1	Xiaomi 10S	1080×2340	MIUI 12.5.7
D2	Google Pixel 5	1080×2340	Android 11
D3	iPhone 11	828×1792	iOS 14
D4	iPhone 12	1170×2532	iOS 15
D5	Honor 10 (low resolution)	720×1280	EMUI 10.0.0

B. Experimental Setup

We conduct experiments on six real devices with different versions of iOS and Android. Detailed settings for each device are listed in Table I. Our experiments cover most mainstream smartphone brands, which include Apple, Google, Xiaomi, and Honor. In the United States, they account for about 62% of the market between August 2021 and August 2022 [74]. The devices are with different platforms and screen resolutions. Here, MIUI and EMUI are both customized third-party OSes with various GUI styles based on Android.

We developed our own benchmarks because the existing ones [2] are unsuitable for our evaluation. Some apps in the baseline benchmarks require mandatory updates, which change their GUI sequences. Furthermore, a few apps have crashed outright, rendering them unusable for benchmarking purposes. To determine the experimental subjects, we first identified 10 of the most popular categories in the Google Play store, according to the market report [75]. Then we selected the most installed app that has cross-platform versions in each category as the representative of the category. In addition, we also selected two open-source apps according to previous research [2]. We recruited three senior students majoring in software engineering to design five test scenarios for each app, with about 15 steps per scenario. Thus, there are 60 testing scenarios and approximately 900 steps for an individual device. Considering the combination of different devices, our evaluation has 420 testing scenarios and around 6,300 steps.

To evaluate the effectiveness of our tool, we compare ReSPlay with two image-based tools for cross-platform replay: LIRAT [2] and Sikuli [9]. To select appropriate baselines, we studied 12 papers on image-based methods [2], [7]–[12], [39], [76]–[79]. There are two categories, pure visual and hybrid approaches. For the former, the one that achieves the highest accuracy is Sikuli [9], while for the latter, representative approaches are LIRAT [2] and MAPIT [39]. We did not evaluate pixel-based and attribute-based methods because they have lower replaying accuracy than image-based methods [2]. Besides, MAPTI is not included since the implicit assumptions are far from the truth reported by recent studies [34], [42].

To measure the effectiveness of ReSPlay, we use step-level and scenario-level accuracy [2], [3]. These two metrics are defined in Equation 6 and Equation 7. The step accuracy is the number of progress steps divided by the actual total number of steps. However, the step accuracy cannot evaluate how many times ReSPlay perfectly replays the whole scenario. To mitigate the side effect, we also use scenario accuracy as an evaluation criterion. The scenario accuracy is computed as

TABLE II: Overview of the selected applications. 'N' and 'Y' represent apps from commercial and open-sourced communities, respectively. '#Activity' refers to the number of activities. '#Install' shows the number of installs of an app according to Google Play ('m' indicates million). Note that the data of '#Install' is unavailable in the iOS store. '-' indicates apps downloaded from Github with an unknown number of installs.

App name	Category	Open	#Install	#Activity
Keep	Health & Fitness	N	5m+	112
Booking	Travel	N	100m+	246
Amazon Shopping	Shopping	N	500m+	121
Evernote	Productivity	N	10m+	286
App Music	Music	N	50m+	65
Kindle	Books	N	100m+	126
AdGuard	Personalization	N	5m+	54
HERE WeGo	Tools	N	10m+	9
Tricount	Finance	N	1m+	68
WikiPedia	News	N	50m+	32
Monkey	Development	Y	-	7
openHAB	Lifestyle	Y	-	14

the fraction of scenarios that have been correctly replayed. A successful scenario replay means that all steps in the scenario are replayed correctly. For the correctness of the replay, we manually check the consistency between GUI screenshots dumped from the recording device and the replaying device.

$$step\_acc = \frac{|Successfully\ replayed\ steps|}{|Steps\ needed\ to\ be\ replayed|} \tag{6}$$

$$scenario\_acc = \frac{|Successfully\ replayed\ scenarios|}{|Scenarios\ needed\ to\ be\ replayed|} \tag{7}$$

C. RQ1: Replaying results on devices across different OSes

We first show the comparison between ReSPlay and the baselines in cross-platform testing. Here, Honor 10 (D0) and iPhone 12 (D4) are used to record events while they are replaying devices of each other. We record 60 scenarios and about 900 steps on D0 and replay them on D4 and then vice-versa. We choose iPhone 12 because it is the latest iOS device at the time of the experiment. We choose Honor 10 because it uses a different OS and has a similar resolution. Besides, Honor 10 is among the main-stream models across the world [2]. Using other devices, such as Xiaomi 10S and Google Pixel 5, also generates similar results, adding little information to the analysis. Moreover, due to space limitations, we only present the case of Honor 10. The same holds for iPhone 12. For the convenience of subsequent description, we refer to 'X-to-Y' as recording event traces on X and replaying them on Y.

Table III presents detailed evaluation results of selected apps. Due to space limitations, details of scenario-level accuracy for each application are not reported. Instead, we summarize the results in Fig. 5. In our evaluation, ReSPlay gives a 17.89% higher accuracy than baselines, according to Fig. 5. For instance, when replaying event traces on D4, the average accuracy of ReSPlay for the step-level replay is 17.25% higher than that of LIRAT and 22.5% higher than that of Sikuli, respectively. Regarding scenario-level replay on D4, there are 16.67% and 20% improvements in accuracy compared with

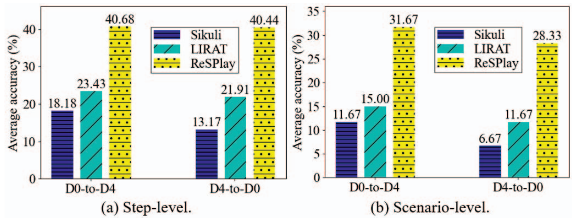

Fig. 5: The average replay accuracy of ReSPlay, LIRAT, and Sikuli on devices across different OSes.

TABLE III: Step-level accuracy of replaying events on devices across different OSes.

App name	D0-to-D4			D4-to-D0		
	Sikuli	LIRAT	ReSPlay	Sikuli	LIRAT	ReSPlay
Keep	36.00%	52.00%	97.33%	8.00%	28.00%	74.67%
Booking	10.96%	12.33%	21.92%	38.36%	38.36%	45.21%
Amazon Shopping	13.51%	14.86%	39.19%	32.43%	33.78%	37.84%
Evernote	11.59%	13.04%	43.48%	0%	11.59%	30.43%
Apple music	4.05%	6.76%	12.16%	12.16%	14.86%	21.62%
Kindle	11.27%	18.31%	19.72%	7.04%	12.68%	19.72%
AdGuard	1.33%	2.67%	2.67%	2.67%	2.67%	2.67%
HERE WeGo	100%	100%	100%	24.00%	45.33%	92.00%
Monkey	0%	0%	0%	0%	0%	0%
openHAB	12.50%	26.39%	77.78%	12.50%	27.78%	80.56%
Tricount	7.81%	26.56%	67.19%	17.19%	45.31%	76.56%
Wikipedia	2.98%	2.98%	2.98%	1.49%	1.49%	1.49%
Average	18.18%	23.43%	40.68%	13.17%	21.91%	40.44%

LIRAT and Sikuli, respectively. For D4-to-D0, ReSPlay has the highest accuracy on both step-level and scenario-level. Specifically, regarding step-level replay, ReSPlay increases the average accuracy on different devices by 18.53% and 27.27% for LIRAT and Sikuli, respectively. Regarding scenario-level replay, the average accuracy of ReSPlay is 16.66% higher than LIRAT and 21.66% higher than Sikuli.

ReSPlay achieves the best performance because it leverages GUI screenshot sequences, which contain extra features to discriminate visually similar GUI widgets that the baseline techniques cannot differentiate. A representative example is illustrated in Fig. 2. ReSPlay increases its accuracy by correctly distinguishing the widget in the red box from the one in the blue box in Fig. 2(b), while baseline methods cannot.

From Table III, we observe that the cross-OS replay accuracy number of all three approaches is less than 5% on AdGuard, Monkey, and Wikipedia. The main reason is that the same apps on different OSes have different images, GUI designs, and functionalities. For example, in Fig. 6, the Android version of Wiki displays different images from the iOS version, causing ReSPlay to fail to reach the non-existent target GUI state due to the missing target widget. Consistent with our expectations, baselines cannot accurately replay on Wiki either. Besides, there are discrepancies between D0-to-D4 and D4-to-D0 because the workloads on the source device are generated differently by different players.

D. RQ2: Replaying results across different GUI styles

The goal of ReSPlay is to accurately replay recorded traces for apps that look different on different devices. In practice, since the Android ecosystem is highly fragmented [80], the

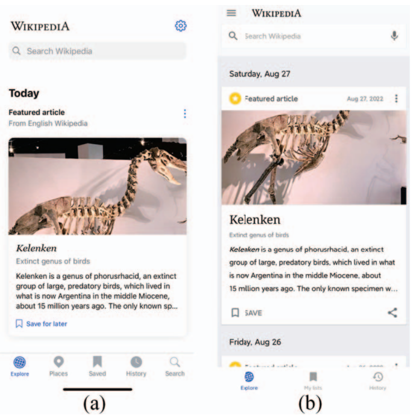

Fig. 6: The same app displays different images on different OSes. Figures (a) and (b) are dumped from D4 and D0, respectively.

same app may look different on different Android devices due to different GUI styles of vendors. For example, the texts in the three colored boxes in Fig. 8 have different fonts on different Android devices. Therefore, in this research question, we evaluate the performance of ReSPlay on Android systems with different GUI styles. Specifically, we record test cases on Honor 10 (D0) and replay them on Xiaomi 10S (D1) and Google Pixel 5 (D2). D1 and D2 are chosen because they share the same resolutions with different GUI styles. This is performed to eliminate any potential influence from business logic and resolution through a variable control method. Moreover, they represent the latest and flagship devices from their respective manufacturers at the time of the experiment.

Table IV shows the evaluation results of ReSPlay, LIRAT, and Sikuli. When replaying scenarios on D1, ReSPlay achieves an average accuracy of 90.27% on the step-level replay (see Fig. 7). Correspondingly, the average accuracy of LIRAT is 65.65%, while that of Sikuli is 50.41%. On the scenario-level replay, the accuracy of ReSPlay is 86.67%, which is 61.67% and 50% higher than Sikuli and LIRAT, respectively. When replaying scenarios on D2, the step-level accuracy of ReSPlay is 86.99%, which improves the accuracy of baselines by 25.79%. On the scenario level, the accuracy of ReSPlay is 58.33% higher than Sikuli. Compared with LIRAT, the scenario-level accuracy of ReSPlay is 50% higher.

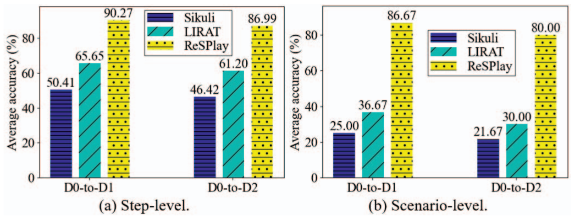

Fig. 7: The average replay accuracies of ReSPlay, LIRAT, and Sikuli on devices with different GUI styles.

We notice that ReSPlay achieves the highest performance in both cases. The results demonstrate the effectiveness of ReSPlay on devices across different GUI styles. For example, in Fig. 8(a) and Fig. 8(b), 'M' in 'Manage', 'l' in 'loyalty', and 'Q' in 'Questions' can be different. In these cases, baselines cannot accurately match widgets due to font changes. Instead, ReSPlay can find the correct action due to the consideration of possible future GUI sequences.

TABLE IV: Step-level accuracy of replaying events across different GUI styles.

App name	D0-to-D1			D0-to-D2		
	Sikuli	LIRAT	ReSPlay	Sikuli	LIRAT	ReSPlay
Keep	56.00%	86.67%	89.33%	50.67%	84.00%	86.67%
Booking	78.08%	78.08%	82.19%	75.34%	75.34%	79.45%
Amazon Shopping	8.11%	37.84%	94.59%	2.70%	28.38%	90.54%
Evernote	30.43%	53.62%	78.26%	18.84%	36.23%	63.77%
Apple music	54.05%	68.92%	87.84%	37.84%	66.22%	90.54%
Kindle	39.44%	57.75%	100%	16.90%	54.93%	100%
AdGuard	49.33%	76.00%	100%	48.00%	74.67%	100%
HERE WeGo	25.33%	33.33%	88.00%	14.67%	28.00%	82.67%
Monkey	65.22%	66.67%	76.81%	63.77%	68.12%	78.26%
openHAB	58.33%	77.78%	100%	59.72%	75.00%	90.28%
Tricount	82.81%	82.81%	85.94%	79.69%	79.69%	82.81%
Wikipedia	62.68%	71.65%	100%	50.75%	68.66%	100%
Average	50.41%	65.65%	90.27%	46.42%	61.20%	86.99%

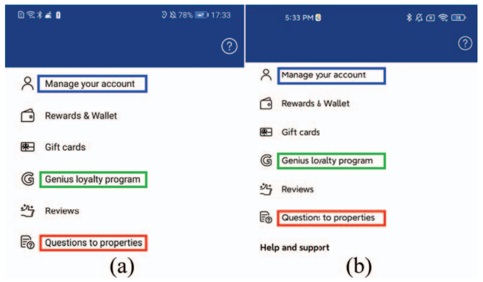

Fig. 8: Example of font changes. Figures (a) and (b) are dumped from D0 and D1, respectively.

E. RQ3: Replaying results across different resolutions

Resolution can also affect the size of GUI widgets on the same screen. It can also affect the accuracy of ReSPlay. Therefore, in RQ3, we evaluate the effectiveness of ReSPlay in replaying events on devices across different resolutions. There are three types of device combinations: (i) iPhone 11 (D3) to iPhone 12 (D4), indicating iOS to iOS replay; (ii) Honor 10 (D0) to iPhone 11 (D3), representing Android to iOS replay; (iii) Honor 10 with low resolution (D5) to Honor 10 with high resolution (D0), representing Android to Android replay. Table V presents the step-level replay accuracy of ReSPlay, LIRAT, and Sikuli. For D0-to-D3, the average accuracy of ReSPlay on the step-level replay is 15.48% higher than that of LIRAT and 21.31% higher than that of Sikuli. Regarding the scenario-level replay, there are 13.33% and 15.33% improvements in performance compared with LIRAT and Sikuli, respectively (see Fig. 9). For D3-to-D4, ReSPlay achieves an average accuracy of 91.10% on the step-level replay. Correspondingly, the average accuracy of LIRAT is 52.24%, while that of Sikuli is 40.08%. On the scenario-level replay, the accuracy of ReSPlay is 81.67%, which is 65% and 55% higher than Sikuli and LIRAT, respectively. For D5-to-D0, the step-level accuracies of ReSPlay, LIRAT, and Sikuli are 93.94%, 61.05%, and 26.95%, while the scenario-level accuracies are 88.33%, 38.33%, and 10%, respectively.

During evaluations, we find that the accuracy of ReSPlay exceeds 90% for D3-to-D4 and D5-to-D0 (same OS). This

TABLE V: Step-level accuracy of replaying events on devices across different resolutions.

App name	D0-to-D3			D3-to-D4			D5-to-D0		
	Sikuli	LIRAT	ReSPlay	Sikuli	LIRAT	ReSPlay	Sikuli	LIRAT	ReSPlay
Keep	44.00%	57.33%	97.33%	74.32%	81.08%	95.95%	37.33%	77.33%	89.33%
Booking	12.33%	15.07%	19.18%	23.94%	35.21%	100%	31.82%	75.76%	89.39%
Amazon Shopping	16.22%	17.57%	36.49%	14.86%	28.38%	90.54%	4.05%	27.03%	90.54%
Evernote	17.39%	18.84%	40.58%	30.99%	42.25%	100%	23.08%	52.31%	100%
Apple music	4.05%	4.05%	14.86%	38.46%	44.62%	56.92%	27.27%	63.64%	100%
Kindle	14.08%	19.72%	19.72%	56.16%	72.60%	100%	22.22%	76.38%	91.67%
AdGuard	1.33%	4.00%	5.33%	86.67%	86.67%	93.33%	43.66%	74.65%	100%
HERE WeGo	100%	100%	100%	43.66%	43.66%	100%	10.14%	20.29%	100%
Monkey	0%	0%	0%	16.00%	52.00%	94.67%	18.30%	60.60%	87.32%
openHAB	12.50%	33.33%	86.11%	32.84%	37.31%	100%	48.68%	76.32%	100%
Tricount	7.81%	29.69%	76.56%	32.84%	53.73%	85.07%	29.85%	62.69%	92.54%
Wikipedia	2.99%	2.99%	2.99%	30.14%	49.32%	76.71%	26.87%	65.67%	86.57%
Average	19.39%	25.22%	40.70%	40.08%	52.24%	91.10%	26.95%	61.05%	93.94%

result shows the advantage of ReSPlay over baselines. Different resolutions on the same OS mainly affect the size of widgets displayed. Variability in widget sizes could lead to a matching failure of baselines because they are sensitive to any changes of widgets (font, color, size, etc.) [81]. In contrast, our model achieves better results, which benefits from robust GUI screenshot sequences. Besides, for D0-to-D3, the accuracy of ReSPlay is 50.4% and 53.24% lower than that in the other two cases, respectively. The main reason for the lower accuracy is the GUI discrepancy between OSes, as discussed in Section VI-C. Nevertheless, the accuracy of ReSPlay is still higher than the baselines.

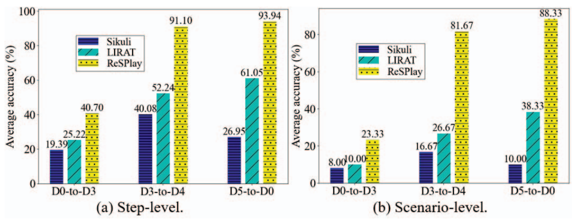

Fig. 9: The average replay accuracies of ReSPlay, LIRAT, and Sikuli on devices across different resolutions.

F. RQ4: Insight validation

To validate the insights of ReSPlay and understand why ReSPlay outperforms the baselines, we evaluate whether future GUI sequences are more robust than conventional widget screenshots. Specifically, we answer two sub-questions: (1) do GUI sequences reduce the mismatches during GUI action matching? (2) can we find no fewer corresponding GUI states than corresponding GUI widgets across different platforms, particularly different OSes? The first question evaluates the effectiveness of our insights. Reducing mismatch in GUI action matching is the core reason for the high accuracy of ReSPlay. The second question evaluates the availability of our insights. ReSPlay only works when it can find corresponding future GUI sequences across OSes. Fewer corresponding GUI screenshots than corresponding GUI widgets means ReSPlay can only be applied to fewer apps than baselines.

To answer the first question, we calculate the proportion of false matches on each app for ReSPlay, LIRAT, and Sikuli. Statistics are given by considering approximately 6,300 steps from different combinations of source and target devices. We get the ground truth by manual visual inspection. Particularly, we also measure the probability of mismatch happening in different steps of a replayed trace. An early step is more critical than a later step. For instance, a mismatch in the first step of a replayed trace can lead to mismatches in all future steps. On the contrary, the mismatch in the last step of a trace only affects itself. The result is shown in Table VII. In general, ReSPlay has a magnitude lower average mismatch rate than baselines. This proves the effectiveness of the GUI sequences as features for record-and-replay. Specifically, as shown in Table VI, ReSPlay has a probability of 9.05%, 15.95%, and 18.81% mismatch actions in the first, second, and third steps of a trace. The baselines have more than 2$\times$ of mismatch rate. This is the main reason why ReSPlay achieves 2$\times$ higher accuracy in apps Evernote and openHAB. In these apps, LIRAT and Sikuli fail to match the correct GUI widgets in the first three steps, causing the whole trace to fail in the early stages. Thus, the baselines have lower replay accuracy.

To answer the second question, we count the fraction that a GUI state or a widget of the source device has its counterpart on the target device. The counterpart of a GUI state is defined as the screenshot on the target device that has similar functionality and GUI style. Similarly, the counterpart of a GUI widget is the widget on the target device that has similar functionality. To calculate the counterpart discovery rate, we recruited 11 undergraduate students to vote on whether a counterpart on the target device could be found for a GUI state or a widget of the source device. The conclusion depends on the majority voting. The numerical results are in the last column of Table VII. In general, we conclude that the average probability of finding a counterpart on the target device corresponding to a GUI state of the source device is around 85%. For most apps, ReSPlay can find 92.64% of corresponding GUI states in a different OS, except AdGuard, Monkey, and Wikipedia.

The counterpart discovery rate for widgets (not shown in Table VII) is the **same** as for GUI states. This is because our undergraduates consider that two widgets are counterparts if they lead to equivalent GUI screenshots. This result indicates that for apps where ReSPlay cannot work, conventional GUI matching also does not work. This result proves that the availability of ReSPlay is not lower than existing techniques.

We also notice that the apps with fewer GUI state matches across OSes also generate lower accuracy in Table III. This result explains why ReSPlay performs lower accuracies in

TABLE VI: The average mismatch rate of ReSPlay, LIRAT, and Sikuli in the first, second, and third steps.

App name	Sikuli			LIRAT			ReSPlay		
	first	second	third	first	second	third	first	second	third
Keep	0%	0%	11.43%	0%	0%	0%	0%	0%	0%
Booking	34.29%	34.29%	42.86%	34.29%	34.29%	34.29%	0%	5.72%	20.00%
Amazon Shopping	0%	57.14%	74.29%	0%	0%	17.14%	0%	0%	11.43%
Evernote	20.00%	30.00%	53.33%	17.14%	28.57%	37.14%	0%	8.57%	8.57%
Apple music	25.71%	42.86%	54.29%	25.71%	34.29%	34.29%	5.71%	22.86%	22.86%
Kindle	0%	68.57%	80.00%	0%	25.71%	28.57%	0%	20.00%	20.00%
AdGuard	42.86%	45.72%	54.29%	25.71%	42.86%	42.86%	20.00%	28.57%	37.14%
HERE WeGo	11.43%	11.43%	22.86%	0%	0%	11.43%	0%	0%	0%
Monkey	42.86%	42.86%	42.86%	42.86%	42.86%	42.86%	42.86%	42.86%	42.86%
openHAB	42.86%	45.71%	45.71%	31.43%	37.14%	40.00%	14.29%	20.00%	20.00%
Tricount	0%	48.57%	48.57%	0.00%	11.43%	11.43%	0%	0%	0%
Wikipedia	25.71%	54.29%	54.29%	25.71%	42.86%	42.86%	25.71%	42.86%	42.86%
Average	20.48%	40.12%	48.73%	16.90%	25.00%	28.57%	9.05%	15.95%	18.81%

AdGuard, Monkey, and Wikipedia. In general, if the Android version and the iOS version of an app are completely different, ReSPlay does not work for this app. However, if the two versions are completely different, widget matching also does not work. This result can be inferred from the lower accuracy of baselines in Table III. We notice this limitation of ReSPlay and leave it to future work. Nevertheless, our evaluation still shows that ReSPlay works for 83.33% of apps with a GUI state matching rate higher than 70%. And it also shows that GUI sequences can be applied to no fewer apps than conventional widget matching.

TABLE VII: Mismatch rate and counterpart discovery rate for GUI screenshot sequence.

App name	Average mismatch rate			Counterpart discovery rate
	Sikuli	LIRAT	ReSPlay	
Keep	56.24%	33.37%	9.91%	99.11%
Booking	61.61%	52.84%	37.52%	98.58%
Amazon Shopping	86.87%	73.17%	31.47%	83.33%
Evernote	81.10%	67.45%	34.78%	97.20%
Apple music	74.59%	61.56%	45.15%	87.69%
Kindle	76.13%	55.38%	35.60%	72.22%
AdGuard	66.72%	54.10%	42.29%	58.34%
HERE WeGo	54.60%	47.06%	5.33%	100%
Monkey	76.67%	64.66%	51.85%	52.50%
openHAB	66.13%	49.44%	9.32%	99.02%
Tricount	63.14%	45.65%	19.05%	99.63%
Wikipedia	74.59%	62.46%	47.04%	70.32%
Average	64.60%	55.60%	30.78%	84.83%

VII. DISCUSSIONS

Threats to Validity. Threats to the external validity of our approach are mainly related to (i) the number of devices used, (ii) the number of apps in the experiments, and (iii) the representativeness of test scenarios. We use four Android devices and two iOS devices in the evaluation. However, there are many kinds of smartphone models on the market, each with diverse platforms and resolutions. There may exist a performance bias under different configurations. To mitigate the first threat, the mobile devices we select are the latest models and with a large market share. To mitigate the second threat, we selected apps from the 12 most popular categories based on the number of installations and multi-platform compatibility. While it would be great to conduct additional experiments with more apps, our experimental subjects involve a considerably large number of apps with different categories, suggesting the relatively broad applicability of ReSPlay. Finally, a potential threat is recruiting senior students majoring in software engineering to design test scenarios. However, Salman *et al.* [82] suggest that senior students are sufficient to act as a reasonable proxy for developers in a well-controlled scenario.

Limitations. The current implementation has several limitations. It does not support advanced actions like drag, zoom, and swipe. It also assumes a deterministic environment for record and replay of events, which is unrealistic. Moreover, it cannot handle GUI changes across different platforms, such as vertical vs. horizontal screens. These may affect the scalability and robustness of our tool. We plan to improve the tool to address these issues in the future.

VIII. CONCLUDING REMARKS

In this paper, we have proposed a general framework called ReSPlay to record and replay test scripts for mobile apps on different platforms. The key idea of our approach is that we aim to match the resulting GUI screenshot sequences instead of searching for the GUI widgets directly during replay, which can significantly improve replay accuracy because we can avoid the widely existing ambiguities caused by similar widgets. We introduce SDP-Net, a cross-platform tool based on deep reinforcement learning, which is used to identify the best possible path to the target GUI screenshot in the GUI transition graphs. SDP-Net learns an optimal decision policy, which can be used to replay the recorded events accurately on target devices with different platforms. Extensive experiments demonstrate that our approach achieves substantially better performance than state-of-the-art approaches. In the future, we will consider using the source code to improve record-and-replay for apps developed by cross-platform solutions like Flutter or ReactNative [83].

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was partly supported by the National Natural Science Foundation of China (62141208, 62172009) and an Open Project of the Key Laboratory of Mobile Application Innovation and Governance Technology, MIIT.

References

[1] Z. Qin, Y. Tang, E. Novak, and Q. Li, "Mobiplay: A remote execution based record-and-replay tool for mobile applications," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 571–582.

[2] S. Yu, C. Fang, Y. Yun, and Y. Feng, "Layout and image recognition driving cross-platform automated mobile testing," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1561–1571.

[3] J. Guo, S. Li, J.-G. Lou, Z. Yang, and T. Liu, "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*, 2019, pp. 90–100.

[4] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for android," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 72–81.

[5] R. Developer, "Robotium," 2009. [Online]. Available: https://github.com/RobotiumTech/robotium.

[6] E. Developer, "Espresso," 2019. [Online]. Available: https://developer.android.com/training/testing/espresso.

[7] A. Developer, "Airtest," 2019. [Online]. Available: http://airtest.netease.com.

[8] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using gui screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 183–192.

[9] T.-H. Chang, T. Yeh, and R. C. Miller, "Gui testing using computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1535–1544.

[10] J. Qian, Z. Shang, S. Yan, Y. Wang, and L. Chen, "Roscript: a visual script driven truly non-intrusive robotic testing system for touch screen applications," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 297–308.

[11] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk, "Translating video recordings of mobile app usages into replayable scenarios," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 309–321.

[12] E. Developer, "Eyeautomate," 2018. [Online]. Available: http://eyeautomate.com/eyestudio/

[13] Z. Wang, S. Chang, Y. Yang, D. Liu, and T. S. Huang, "Studying very low resolution recognition using deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4792–4800.

[14] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[16] M. Cilimkovic, "Neural networks and back propagation algorithm," *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, vol. 15, no. 1, 2015.

[17] A. Van Ooyen and B. Nienhuis, "Improving the convergence of the back-propagation algorithm," *Neural networks*, vol. 5, no. 3, pp. 465–471, 1992.

[18] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China National Conference on Chinese Computational Linguistics*. Springer, 2019, pp. 194–206.

[19] Y. Liu, "Fine-tune bert for extractive summarization," *arXiv preprint arXiv:1903.10318*, 2019.

[20] S. Kumar *et al.*, "Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things," *Future Generation Computer Systems*, vol. 125, pp. 334–351, 2021.

[21] "The complete guide to iphone screen resolutions and sizes," 2022. [Online]. Available: https://www.appmysite.com/blog/the-complete-guide-to-iphone-screen-resolutions-and-sizes/

[22] J. Jeon and J. S. Foster, "Troyd: Integration testing for android," Tech. Rep., 2012.

[23] W. Lam, Z. Wu, D. Li, W. Wang, H. Zheng, H. Luo, P. Yan, Y. Deng, and T. Xie, "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*, 2017, pp. 854–859.

[24] Y. Hu, T. Azim, and I. Neamtiu, "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*, 2015, pp. 349–366.

[25] M. Halpern, Y. Zhu, R. Peri, and V. J. Reddi, "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 (ISPASS)*. IEEE, 2015, pp. 215–224.

[26] appetizer toolkit, "appetizer," 2017. [Online]. Available: https://github.com/appetizerio/appetizer-toolkit.

[27] O. Sahin, A. Aliyeva, H. Mathavan, A. Coskun, and M. Egele, "Late breaking results: Towards practical record and replay for mobile applications," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–2.

[28] K. Moran, R. Bonett, C. Bernal-Cárdenas, B. Otten, D. Park, and D. Poshyvanyk, "On-device bug reporting for android applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 215–216.

[29] C. Li, Y. Jiang, and C. Xu, "Cross-device record and replay for android apps," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 395–407.

[30] M. Fazzini, E. N. D. A. Freitas, S. R. Choudhary, and A. Orso, "Barista: A technique for recording, encoding, and running platform independent android tests," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2017, pp. 149–160.

[31] O. Sahin, A. Aliyeva, H. Mathavan, A. Coskun, and M. Egele, "Randr: Record and replay for android applications via targeted runtime instrumentation," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 128–138.

[32] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "Automated replay and failure detection for web applications," in *Proceedings of the 20th IEEE/ACM international conference on automated software engineering*, 2005, pp. 253–262.

[33] J. Zheng, L. Shen, X. Peng, H. Zeng, and W. Zhao, "Mashredroid: enabling end-user creation of android mashups based on record and replay," *Science China Information Sciences*, vol. 63, pp. 1–20, 2020.

[34] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhut, G. Li, and J. Wang, "Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 322–334.

[35] F. Behrang and A. Orso, "Automated test migration for mobile apps," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 384–385.

[36] X. Qin, H. Zhong, and X. Wang, "Testmig: Migrating gui test cases from ios to android," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 284–295.

[37] F. Behrang and A. Orso, "Test migration for efficient large-scale assessment of mobile app coding assignments," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 164–175.

[38] Y. Zhao, J. Chen, A. Sejfia, M. Schmitt Laser, J. Zhang, F. Sarro, M. Harman, and N. Medvidovic, "Fruiter: a framework for evaluating ui test reuse," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1190–1201.

[39] S. Talebipour, Y. Zhao, L. Dojcilović, C. Li, and N. Medvidović, "Ui test migration across mobile platforms," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 756–767.

[40] F. Behrang and A. Orso, "Test migration between mobile apps with similar functionality," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 54–65.

[41] J.-W. Lin, R. Jabbarvand, and S. Malek, "Test transfer across mobile apps through semantic mapping," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 42–53.

[42] S. Zhang, Y. Li, W. Yan, Y. Guo, and X. Chen, "Dependency-aware form understanding," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 139–149.

[43] E. Alegroth, M. Nass, and H. H. Olsson, "Jautomate: A tool for system-and acceptance-test automation," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 439–446.

[44] O. Developer, "Openhub." [Online]. Available: https://www.openhab.org/
[45] A. Developer, "Appium," 2015. [Online]. Available: http://appium.io.
[46] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[47] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.
[48] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
[49] S. Zhang and T. Wong, "Integrated process planning and scheduling: an enhanced ant colony optimization heuristic with parameter tuning," *Journal of Intelligent Manufacturing*, vol. 29, no. 3, pp. 585–601, 2018.
[50] Z. Cheng, M. Min, M. Liwang, L. Huang, and Z. Gao, "Multiagent ddpg-based joint task partitioning and power control in fog computing networks," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 104–116, 2021.
[51] J. Wainer and P. Fonseca, "How to tune the rbf svm hyperparameters? an empirical evaluation of 18 search algorithms," *Artificial Intelligence Review*, vol. 54, no. 6, pp. 4771–4797, 2021.
[52] M. Chen, Z. Chang, H. Lu, B. Yang, Z. Li, L. Guo, and Z. Wang, "Augnet: End-to-end unsupervised visual representation learning with image augmentation," 2021.
[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
[54] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 2017, pp. 845–854.
[55] Z. He, S. Sunkara, X. Zang, Y. Xu, L. Liu, N. Wichers, G. Schubiner, R. Lee, and J. Chen, "Actionbert: Leveraging user actions for semantic understanding of user interfaces," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 7, 2021, pp. 5931–5938.
[56] C. Bai, X. Zang, Y. Xu, S. Sunkara, A. Rastogi, J. Chen *et al.*, "Uibert: Learning generic multimodal representations for ui understanding," *arXiv preprint arXiv:2107.13731*, 2021.
[57] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk, "It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 957–969.
[58] A. Mehdi, "Image similarity: Theory and code," 2021. [Online]. Available: https://towardsdatascience.com/image-similarity-theory-and-code-2b7bcce96d0a
[59] S. Zhao, Y. Wang, Z. Yang, and D. Cai, "Region mutual information loss for semantic segmentation," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[60] X. Xia, C. Xu, and B. Nan, "Inception-v3 for flower classification," in *2017 2nd international conference on image, vision and computing (ICIVC)*. IEEE, 2017, pp. 783–787.
[61] B. Ay, G. Aydın, Z. Koyun, and M. Demir, "A visual similarity recommendation system using generative adversarial networks," in *2019 international conference on deep learning and machine learning in emerging applications (Deep-ML)*. IEEE, 2019, pp. 44–48.
[62] Mathworks, "Define reward signals," 2021. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/define-reward-signals.html
[63] Y. Zhu, G. Zhai, X. Min, and J. Zhou, "Learning a deep agent to predict head movement in 360-degree images," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 4, pp. 1–23, 2020.
[64] S. Mori, H. Nishida, and H. Yamada, *Optical character recognition*. John Wiley & Sons, Inc., 1999.
[65] T. Peters, "difflib," 2001. [Online]. Available: https://docs.python.org/3/library/difflib.html
[66] X. Yang and K.-T. Cheng, "Ldb: An ultra-fast feature for scalable augmented reality on mobile devices," in *2012 IEEE international symposium on mixed and augmented reality (ISMAR)*. IEEE, 2012, pp. 49–57.
[67] T. Liu, H. Wang, L. Li, X. Luo, F. Dong, Y. Guo, L. Wang, T. Bissyandé, and J. Klein, "Maddroid: Characterizing and detecting devious ad contents for android apps," in *Proceedings of The Web Conference 2020*, 2020, pp. 1715–1726.
[68] T. D. White, G. Fraser, and G. J. Brown, "Improving random gui testing with image-based widget detection," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 307–317.
[69] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.
[70] Y. Chen, Y. Jiang, F. Ma, J. Liang, M. Wang, C. Zhou, X. Jiao, and Z. Su, "{EnFuzz}: Ensemble fuzzing with seed synchronization among diverse fuzzers," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1967–1983.
[71] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.
[72] M. Alahmadi, A. Malkadi, and S. Haiduc, "Ui screens identification and extraction from mobile programming screencasts," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 319–330.
[73] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 257–268.
[74] S. GlobalStats, "Mobile vendor market share united states of america," 2022. [Online]. Available: https://gs.statcounter.com/vendor-market-share/mobile/united-states-of-america
[75] Asoworld, "Making the most of the app marketing," 2022. [Online]. Available: https://asoworld.com/blog/making-the-most-of-the-app-marketing-in-2022-trends-opportunity-and-aso-strategies/
[76] J.-l. Sun, S.-w. Zhang, S. Huang, and Z.-w. Hui, "Design and application of a sikuli based capture-replay tool," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 42–44.
[77] V. Garousi, W. Afzal, A. Çağlar, İ. B. Işık, B. Baydan, S. Çaylak, A. Z. Boyraz, B. Yolaçan, and K. Herkiloğlu, "Comparing automated visual gui testing tools: an industrial case study," in *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing*, 2017, pp. 21–28.
[78] T. Kanstrén, P. Aho, A. Lämsä, H. Martin, J. Liikka, and M. Seppänen, "Robot-assisted smartphone performance testing," in *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE, 2015, pp. 1–6.
[79] K. Mao, M. Harman, and Y. Jia, "Robotic testing of mobile apps for truly black-box automation," *Ieee Software*, vol. 34, no. 2, pp. 11–16, 2017.
[80] L. Wei, Y. Liu, and S.-C. Cheung, "Taming android fragmentation: Characterizing and detecting compatibility issues for android apps," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 226–237.
[81] H. Li, P. Wang, M. You, and C. Shen, "Reading car license plates using deep neural networks," *Image and Vision Computing*, vol. 72, pp. 14–23, 2018.
[82] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *2015 IEEE/ACM 37th IEEE international conference on software engineering*, vol. 1. IEEE, 2015, pp. 666–676.
[83] W. Wu, "React native vs flutter, cross-platforms mobile application frameworks," 2018.