

Read It, Don't Watch It: Captioning Bug Recordings Automatically

Sidong Feng[†], Mulong Xie[‡], Yinxing Xue[§], Chunyang Chen^{†*}

[†] Monash University

[‡] Australian National University

[§] University of Science and Technology of China

Email: [†]{sidong.feng,chunyang.chen}@monash.edu, [‡]mulong.xie@anu.edu.au, [§]yxxue@ustc.edu.cn

Abstract—Screen recordings of mobile applications are easy to capture and include a wealth of information, making them a popular mechanism for users to inform developers of the problems encountered in the bug reports. However, watching the bug recordings and efficiently understanding the semantics of user actions can be time-consuming and tedious for developers. Inspired by the conception of the video subtitle in movie industry, we present a lightweight approach CAPdroid to caption bug recordings automatically. CAPdroid is a purely image-based and non-intrusive approach by using image processing and convolutional deep learning models to segment bug recordings, infer user action attributes, and generate subtitle descriptions. The automated experiments demonstrate the good performance of CAPdroid in inferring user actions from the recordings, and a user study confirms the usefulness of our generated step descriptions in assisting developers with bug replay.

Index Terms—bug recording, video captioning, android app

I. INTRODUCTION

Software maintenance activities are known to be generally expensive, and challenging [1] and one of the most important maintenance tasks is to handle bug reports [2]. A good bug report is detailed with clear information about what happened and the steps to reproduce the bug. However, writing such clear and concise bug reports takes time, especially for non-developer or non-tester users who do not have that expertise and are not willing to spend that much effort [3], [4]. The emergence of screen recording significantly lowers the bar for bug documenting. First, it is easy to record the screen as there are many tools available, some of which are even embedded in the operating system by default, like iOS [5] and Android [6]. Second, video recording can include more detail and context such as configurations, and parameters, bridging the understanding gap between users and developers.

Unfortunately, in many cases, watching the bug recordings and understanding the user behaviors can be time-consuming and tedious for developers [7], [8]. First, the recording may play too fast to watch, and the developers have to pause the recording, or even replay it multiple times to recognize the bug. Second, the watching experience can be further deteriorated by blurred video resolution, poor video quality, etc. Third, the recording usually contains a visual indicator (in

Fig. 1) to help developers identify the user actions performed on the screen. However, those indicators sometimes are too small to be conspicuously realized, and developers have to the recording back and forth to guess each action to repeat it in their testing environment.

Besides bug recordings, those issues also apply to general videos (e.g., movies, drama, etc). To address those issues in normal video watching, captions or subtitles are provided to add clarity of details, better engage users, maintain concentration for longer periods, and translate the different languages [9], [10]. Inspired by the conception of video subtitles in the movie industry, we intend to generate the caption of an app recording to add analogous benefits to developers. Given a caption accompanying the recording, developers, especially novices can more easily identify the user behaviors in the recording and shift their focus toward bug fixing. Specifically, we segment recordings into clips to characterize the “scenes” in the movie and add action descriptions of each clip to guide developers.

Existing work has investigated methods to generate a textual description for GUI screenshot [11], [12], [13], [14], [15], which has been shown useful for various downstream tasks such as GUI retrieval, accessibility enhancement, code indexing, etc. Chen et al. [11] propose an image captioning model to apply semantic labels to GUI elements to improve the accessibility of mobile apps. Clarity [12] further consider multi-modal GUI sources to generate high-level descriptions for the entire GUI screen. However, none of them can generate descriptions for video recording, which is a more challenging task, translating spatial and temporal information into a semantic natural language.

To create good video subtitles, there are several standards [16], including caption synchronization with the videos, accurate content comprehension, compact and consistent word usage, etc. Similarly, we propose an image-based approach CAPdroid in this paper to non-intrusively caption each action step for a bug recording, including three phases: 1) *action segmentation*, 2) *action attribute inference*, and 3) *description generation*. Inspired by the previous work GIFdroid [4], [17] to localize keyframes in bug recording, we first develop a heuristic method to segment the recording into a sequence of

* Corresponding author



Fig. 1: Examples of touch indicators.

action clips (i.e., TAP, SCROLL, INPUT). Then, we adopt image-processing and deep-learning methods to model the spatial and temporal features across frames in the clips to infer action attributes, such as touch location, moving offset, and input text. A simple description based on the action attribute, e.g. tap on (x,y) coordinate, cannot express the action intuitively. Therefore, we first utilize off-the-shelf GUI models to non-intrusively gather the elements information in the GUI. As the GUI elements of interest may not have enough context to be uniquely identified, we propose a novel algorithm using global information of GUI elements to generate high-level semantic descriptions.

We first evaluate the performance of the CAPdroid in obtaining user actions by *action segmentation* and *action attribute inference*, through an automated method. We collect 439 Android apps from Google Play and leverage an automated app explore tool to simulate user actions on the screen, meanwhile capturing a 10-min screen recording for each app. Results show that our tool achieves the best performance (0.84 Video F1-score and 0.93 accuracy) in action segmentation from the recordings compared with five commonly-used baselines. CAPdroid also achieves on average 91.46% in inferring action attributes, outperforming two state-of-the-art baselines. We further carry out a user study to evaluate the usefulness of *description generation* of CAPdroid in assisting bug replay, with 10 real-world bug recordings from GitHub. Results show that participants save 59.8% time reproducing the bug with the help of the steps we described, compared with the steps written by users. Through questionnaires with participants, they also confirm the clearness, conciseness, and usefulness of our generated action descriptions.

The contributions of this paper are as follows:

- This is the first work to generate the caption of bug recordings to support developers in reproducing bugs.
- The first systematic approach CAPdroid, to non-intrusively segment recordings into clips, infer fine-grained user actions, and create action descriptions as subtitles, with examples in online appendix¹.
- A comprehensive evaluation including automated experiments and a user study to demonstrate the accuracy and usefulness of our approach.

II. CAPDROID APPROACH

Given an input GUI recording, we propose an automated approach to segment the recording into a sequence of clips

based on user actions and subsequently localize the action positions to generate natural language descriptions. The overview of our approach is shown in Fig. 2, which is divided into three main phases: (i) the *Action Segmentation* phase, which segments user actions from GUI recording into a sequence of clips, (ii) the *Action Attribute Inference* phase that infers touch location, moving offset, and input text from action clips, and (iii) the *Description Generation* phase that utilizes the off-the-shelf GUI understanding models to generate high-level semantic descriptions. Before discussing each phase in detail, we discuss some preliminary understanding of user actions in GUI recording.

A. Preliminary Study

To understand the recordings from the end-users, we conducted a small pilot study of the GUI recordings from GitHub [18]. In detail, we built a crawler to automatically crawl the bug reports from GitHub issue repositories that contain GUI recordings with suffix names like .gif, .mp4, etc. To study more recent GUI recordings, we obtained the recordings from 2021. Overall, we obtained 5,231 GUI recordings from 1,274 apps. We randomly sampled 1,000 (11.5%) GUI recordings, and we recruited two annotators online to manually check the user actions from the recordings.

Two students were recruited by the university’s internal slack channel and they were compensated with \$12 USD per hour. They have annotating experience on GUI-related (e.g., GUI element bounding box) and video-related (e.g., video classification) datasets. To ensure accurate annotations, the process started with initial training. First, we gave them an introduction to our study and also an example set of annotated screen recordings where the labels have been annotated by the authors. Then, we asked them to pass an assessment test. Two annotators were assigned the experimental set of screen recordings to label the user actions independently without any discussion. After the initial labeling, the annotators met and sanity corrected the subtle discrepancies. Any disagreement was handed over to the first author for the final decision.

We observed that 89% of the recordings included a touch indicator, indicating it as a mechanism for the end-user to depict their actions on the screen. We further classified those touch indicators into three categories, following the Card Sorting [19] method:

- **default (68%)**. As shown in Fig. 1(a), the touch indicator renders a small semi-transparent circle, that gives visual feedback when the user presses his finger on the device screen. This is the default touch indicator on Android.
- **cursor (27%)**. As shown in Fig. 1(b), users/developers may test the apps in the emulator and directly record the desktop, so that the user actions are captured by the desktop cursor.
- **custom (5%)**. As shown in Fig. 1(c), the touch indicator is customized by third-party screen recorders, such as DU Recorder [20], etc.

Those findings motivated us to develop a tailored approach, exploiting touch indicators to capture end-user intent, so to

¹<https://github.com/sidongfeng/CAPdroid>

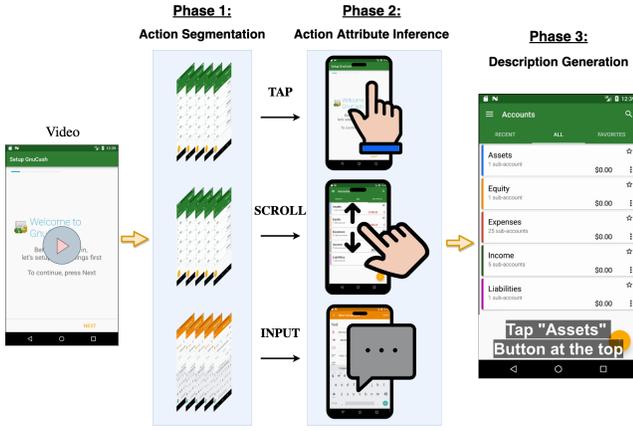


Fig. 2: The overview of CAPdroid.

generate semantic captions for GUI recording. Considering the diversity of touch indicators in the general GUI recordings, a more advanced approach to detect and infer user actions is required.

B. Phase 1: Action Segmentation

A video consists of a sequence of frames to deliver the visual detail of the story for particular scenes. Different from the recognition of discontinuities in the visual-content flow of natural-scene videos, detecting clips in the GUI recording is to infer scenes of user actions that generally display significant changes in the GUIs. To that end, we leverage the similarity of consecutive frames to segment user actions (i.e., *TAP*, *SCROLL*, *INPUT*) from GUI recording.

1) *Consecutive Frame Comparison*: Inspired by signal processing [4], [17], we leverage the image processing techniques to build a perceptual similarity score for consecutive frame comparisons based on Y-Difference (or Y-Diff). YUV is a color space usually used in video encoding, enabling transmission errors or compression artifacts to be more efficiently masked by the human perception than using a RGB-representation [21], [22]. Y-Diff is the difference in Y (luminance) values of two images in the YUV color space, used as a major input for the human perception of motion [23].

Consider a visual recording $\{f_0, f_1, \dots, f_{N-1}, f_N\}$, where f_N is the current frame and f_{N-1} is the previous frame. To calculate the Y-Diff of the current frame f_N with the previous f_{N-1} , we first obtain the luminance mask Y_{N-1}, Y_N by splitting the YUV color space converted by the RGB color space. Then, we apply the perceptual comparison metric, SSIM (Structural Similarity Index) [24], to produce a per-pixel similarity value related to the local difference in the average value, the variance, and the correlation of luminances. A SSIM score is a number between 0 and 1, and a higher value indicates a strong level of similarity.

2) *Action Classification*: To identify the user actions in the GUI recording, we look into the similarity scores of consecutive frames as shown in Fig. 3. The first step is to group frames belonging to the same atomic activity according to

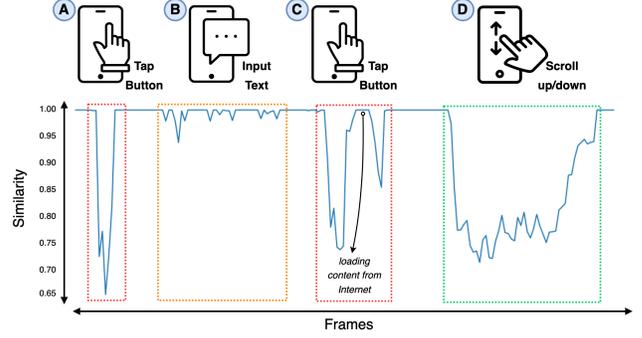
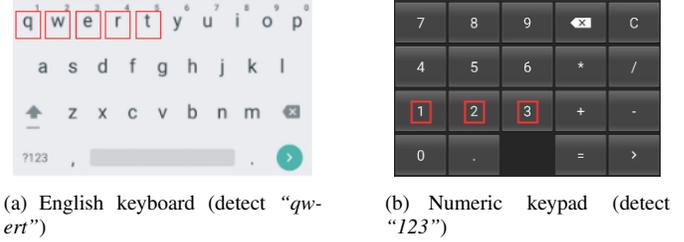


Fig. 3: An illustration of consecutive frame similarity.



(a) English keyboard (detect “qw-ert”)

(b) Numeric keypad (detect “123”)

Fig. 4: Examples of keyboard detection.

tailored pattern analysis. This procedure is necessary because discrete activities performed on the screen will persist across several frames, and thus, need to be grouped and segmented accordingly. Consequently, we observe three patterns of user actions, i.e., *TAP*, *SCROLL*, and *INPUT*. Note that we focus on the most commonly-used actions for brevity in this paper, other actions could be extended by comparing the consecutive frame similarity.

(a) *TAP*: As shown in Fig. 3A (user taps a button), the similarity score starts to drop drastically which reveals an instantaneous transition from one screen to another. In addition, one common case is that the similarity score becomes steady for a small period of time t_s between two drastically droppings as shown in Fig. 3C. The occurrence of this short steady duration t_s is because GUI has not finished loading. While the GUI layout of GUI rendering is fast, resource loading may take time. For example, rendering images from the web depends on device bandwidth, image loading efficiency, etc.

(b) *SCROLL*: As shown in Fig. 3D (user scrolls up/down the screen), the similarity score starts with a drastic drop and then continues to increase slightly over a period of time, which implicates a continuous transition from one GUI to another.

(c) *INPUT*: As shown in Fig. 3B (user inputs text), the similarity score starts to drop and rise multiple times, revealing typing characters and digits. However, the similarity score cannot reliably detect *INPUT* actions, as it may coincide with the *TAP* actions. To address this, we further supplement with Optical Character Recognition (OCR) technique [25] (a detailed description is demonstrated in Section II-C3) to detect whether there is a virtual keyboard in the GUI. Note that we focus on English apps, and it may take additional efforts to

extend our approach to other languages. In detail, we first extract the characters from the frames, and concatenate them into text-based (ocr_{text}) and number-based (ocr_{num}) string. As the OCR may not infer the text perfectly, we discern the keyboard frame by keyboard-specific substrings. For example, Fig. 4(a) is a frame of English keyboard that contains “*qwert*” in ocr_{text} , and Fig. 4(b) is a frame of numeric keypad that contains “*123*” in ocr_{num} . Therefore, the frame of a keyboard is discriminated by

$$frame = \begin{cases} \exists\{qwert, asdfg, zxcvb\} \in lowercase(ocr_{text}) \\ \exists\{123, 456, 789\} \in ocr_{num} \end{cases} \quad (1)$$

where *lowercase* is to convert the uppercase characters into lowercase, in order to detect capital English keyboard. Note that we do not adopt keyboard template matching, as keyboards vary in appearance, such as customized background, different device layouts, etc.

C. Phase 2: Action Attribute Inference

Given a recording clip of user action segmented by the previous phase, we then infer its detailed attributes, including touch location of *TAP* action, its moving offset of *SCROLL* action, and its input text of *INPUT* action, to reveal where the user interacts with on the screen. The overview of our methods is shown in Fig. 5. The prediction of *TAP* location requires a semantic understanding of the GUI transition captured in the clip, such as touch indicators (in Section II-A), transition animation, GUI semantic relation, etc. Therefore, we propose a deep-learning-based method that models the spatial and temporal features across frames to infer the *TAP* location. To infer the moving offset of *SCROLL*, we adopt an off-the-shelf image-processing method to detect the continuous motion trajectory of GUIs, thus, measuring the user’s scrolling direction and distance. To infer the input text of *INPUT*, we leverage the OCR technique to identify the text difference between the frames of keyboard opening (i.e., where the user starts entering text) and keyboard closing (i.e., where the user ends entering).

1) *Inferring TAP location*: Convolutional Neural Networks of 2D (Conv2ds) [26], [27] have demonstrated remarkable success in efficiently capturing the hypothesis of spatial locality in two-dimensional images. A video that is encoded by a sequence of 2d images, aggregates another dimension: spacetime. To predict the touch location from a GUI recording clip, we adopt a Conv3d-based model X3D [28], that simultaneously models spatial features of single-frame GUIs and temporal features of multi-frames optical flow. The architecture of our X3D model is shown in Fig. 5(a).

Given a video clip $V^{T \times H \times W \times C}$ where T is the time length of the clip, W , H , and C are the width, height, and channel of the frame, usually $C = 3$ for RGB frame. We first apply 3d convolution layers, consisting of a set of learnable filters to extract the spatio-temporal features of the video. Specifically, the convolution is to use a 3d kernel, i.e. $t \times d \times d$ where t and d denote the temporal and spatial kernel size, to slide

around the video and calculate kernel-wise features by matrix dot multiply. After the convolutional layers, the video V will be abstracted as a 3d feature map, preserving features along both the spatial and the temporal information. We then apply a 3d pooling layer to eliminate unimportant features and enhance spatial variance of rotation and distortion. After blocks of convolutional and pooling layers, we flatten the feature map and apply a fully connected layer to infer the logits of *TAP* location.

For the detailed implementation, we adopt the convolutional layers from ResNet-50 [29] and borrow the idea of residual connection to improve the model performance and stability between layers. We use MaxPooling [30] as the pooling layer, where the highest value from the kernel is taken, for noise suppressant during abstraction. The output of the fully connected layer is 2 neurons, representing (x, y) coordinates. To accelerate the training process [31], we standardize the coordinate relative to the width and height of the frame. Although the frames are densely recorded (i.e. 30fps), the GUI renders slowly. To extract discriminative features from the recording, we uniformly sample 16 frames at 5 frame intervals ($T = 16$) as suggested in [28]. Note that if the length of the recording clip is smaller than the sample rate 16×5 , we will sample the frames based on nearest neighbor interpolation. To make our training more stable, we adopt Adam as the optimizer [32] and MSELoss as the loss function [33]. Moreover, to optimize the model, we apply an adaptive learning scheduler, with an initial rate of 0.01 and decay to half after 10 iterations. The hyperparameter settings are determined empirically by a small-scale experiment.

2) *Inferring SCROLL offset*: To infer the scrolling direction (i.e., upward, downward) and distance (i.e., amount of movement) from the GUI recording clip, we measure the motion trajectory of GUI elements. Since the elements may scroll off-screen [34], we adopt the K-folds template matching method as shown in Fig. 5(b).

Given a GUI recording clip $\{f_0, f_1, \dots, f_{N-1}, f_N\}$, where f_N is the current frame and f_{N-1} is the previous frame. We first divide the previous GUI f_{N-1} into K pieces vertically. We set K to 10 by a small pilot study to mitigate the off-screen issue and preserve sufficient features for template matching. And then, we match the template of each fold in the current frame f_N to compute the scrolling offset between consecutive frames. At the end, we derive the scrolling distance by summing the offsets ($\sum_{n=0}^N offset_n^{n-1}$), and infer the scrolling direction by the sign of the distance, e.g., positive for downward, otherwise upward.

3) *Inferring INPUT text*: Detecting input text based on user actions on the keyboard can be error-prone, as the user may edit text from the middle of the text, switch to capital, delete text, etc. Therefore, we leverage a practical OCR technique PP-OCRv2 [25] to detect the text difference between the first frame (opening keyboard) and the last frame (closing keyboard) from the *INPUT* recording clip, as shown in Fig. 5(c). Given a GUI frame, PP-OCRv2 detects the text areas in the GUI by using an image segmentation network and

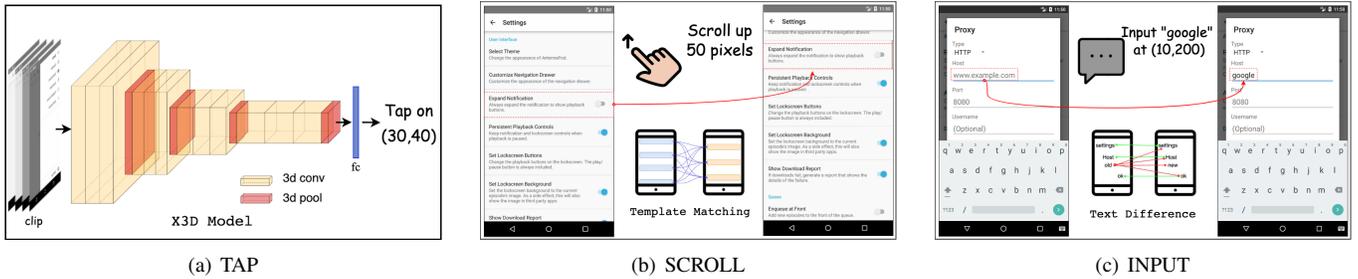


Fig. 5: Approaches of Action Attribute Inference.

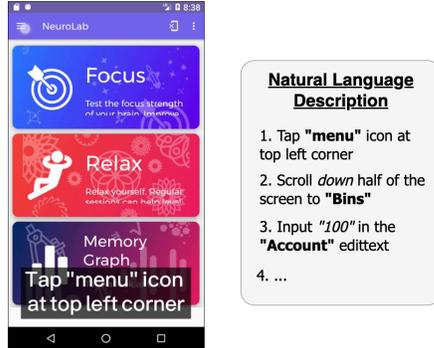


Fig. 6: Subtitle and textual steps in the GUI recording.

then applies a sequence and classification model to recognize the text. As the GUI text is similar to scene text [35], we directly use the pre-trained PP-OCRv2 without any fine-tuning on GUI text, that the overall performance reaches 84.3% state-of-the-art accuracy.

After deriving the text from the frames of keyboard opening and keyboard closing, we first remove all the text on the keyboard to keep the text concise. Then, we detect the text difference between the frames using SequenceMatcher [36]. Albeit good performance of PP-OCRv2, it may still make wrong text recognition, e.g., missing space. To address this, SequenceMatcher measures text similarity by computing the longest contiguous matching subsequence (LCS). Finally, we extract the text that appears only in the frame where the keyboard is closed, as input text.

D. Phase 3: Description Generation

Once the attributes of the action are derived from the previous phases, we proceed by generating in-depth and easy-to-understand natural language descriptions. To accomplish this, we first leverage mature GUI understanding models to obtain GUI information non-intrusively. Then, we propose a novel algorithm to phrase actions into descriptions and embed them as subtitles in the recording as shown in Fig. 6.

1) *GUI understanding*: To understand the GUI, we adopt non-intrusive approaches to obtain GUI information, to avoid the complexity of app instrumentation or handling of the diverse software stack, especially for closed-source systems where no underlying instrumentation support is accessible [37]. An example of GUI understanding is shown in Fig. 7.

Specifically, we first implement the state-of-the-art object detection model Faster-RCNN with ResNet-101 [29] and Feature Pyramid Networks [38] to detect 11 GUI element classes on the screen: button, checkbox, icon, imageview, textview, radio button, spinner, switch, toggle button, edittext, and chronometer. We train the model on the Rico dataset [39] contains 66k GUIs from 9.7k apps. Following the previous work [40], we split the GUIs in the training:validation:testing dataset by apps in the ratio of 8:1:1. As a result, the model achieves an overall Mean Average Precision (MAP) of 51.45% on the test set. For each GUI element, we adopt the OCR technique (the detailed implementation is elaborated in Section II-C3) to detect the text (if any). For the icon, annotation based on common human understanding can enhance the GUI understanding. For example, in Fig. 7, the icon of a group of people informs the semantic of “Friend”. To achieve this, we adopt a transformer-based model from the existing work [11] to caption the icon image. We follow the implementation in their original paper to train the model and achieve 60.7% accuracy on the test set.

Besides from understanding the information of GUI elements, we also attempt to obtain their global information relative to the GUI, including absolute positioning and element relationship. Absolute positioning describes the element as a spatial position in the GUI, which is particularly useful to represent an element in an image [41]. To accomplish this, we uniformly segment the GUI into 3×3 grids, delineating horizontal position (i.e, *left*, *right*), vertical position (i.e, *top*, *bottom*), and *center* position. For example, in Fig. 7, the “100m” spinner is at *the top right* corner. GUI element relationship aims to transform the “flat” structure of GUI elements into connected relationships. A natural way of representing the relationship is using a graph structure, where elements are linked to the nearest elements. To accomplish this, we first compute the horizontal and vertical distance between GUI elements by euclidean pixel measurement. And then, we construct the graph of the GUI elements by finding the nearest elements (neighbors) in four directions, including *left*, *right*, *top*, and *bottom*. Note that we set up a threshold to prevent the neighbors from being too far apart. Ultimately, it will generate a graph representing the relationships between the elements in the GUI. For example, in Fig. 7, the “100” spinner has two neighbors: the “Advanced” element at the *top*,

Action	Id	Condition	Template	Example
TAP	1	$(obj_{text/caption} \neq NULL) \wedge (obj_{confid} > \alpha)$	Tap $[obj_{text}] [obj_{class}]$	Tap “OK” button
	2	$(obj_{text/caption} \neq NULL) \wedge (\beta < obj_{confid} < \alpha)$	Tap $[obj_{text}] [obj_{class}]$ at $[obj_{position}]$	Tap “menu” icon at top left corner
	3	$(obj_{text/caption} == NULL) \vee (obj_{confid} < \beta)$	Tap the $[obj_{class}] [nbr_{relation}] [nbr_{text}]$	Tap the checkbox next to “Dark Mode”
SCROLL	4	$obj_{text} \neq NULL$	Scroll $[direction] [offset]$ of the screen to $[obj_{text}]$	Scroll down half of the screen to “Advanced Setting”
	5	$obj_{text} == NULL$	Scroll $[direction] [offset]$ of the screen	Scroll up a quarter of the screen
INPUT	6	$(obj_{text/caption} \neq NULL) \wedge (obj_{confid} > \alpha)$	Input $[text]$ in the $[obj_{text}]$ edittext	Input “100” in the “Amount” edittext
	7	$(obj_{text} == NULL) \vee (obj_{confid} < \alpha)$	Input $[text]$ in the edittext $[nbr_{relation}] [nbr_{text}]$	Input “John” in the edittext below “Name”

TABLE I: Description template, where “obj” and “nbr” denote the GUI element and its neighbor, α and β denote high- and low-confidence element.

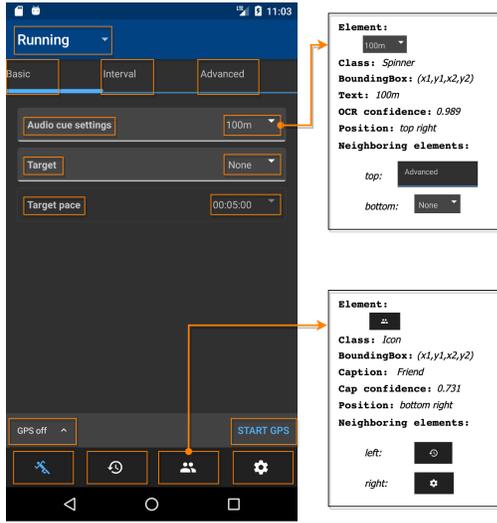


Fig. 7: Example of GUI understanding.

and the “None” element at the *bottom*. Note that the “Audio cue settings” element is omitted due to large spacing, which is consistent with human viewing.

2) *Subtitle Creation*: The main instruction of interest is to create a clear and concise subtitle description based on $\{action, object\}$. The global GUI information is further used to complement the description by $\{position, relationship\}$. Based on the *action* obtained in Section II-B, the attribute of *object* inferred in Section II-C, and the corresponding GUI element information retrieved in Section II-D1, we propose description templates for *TAP*, *SCROLL*, *INPUT*, respectively. A summary of description templates can be seen in Table I.

For *TAP* action, the goal of the description should be clear and concise, e.g., tap “OK” button. However, we find that this simple description may not articulate all *TAP* actions due to two reasons. First, the text and caption of *object* are prone to errors or undetected, as the OCR-obtained text and the caption-obtained annotation are not 100% accurate. Second, there may be multiple *objects* with the same text on the GUI. To resolve

this, we set up an *object* confidence value obj_{confid} as:

$$obj_{confid} = \begin{cases} OCR_{confid} & \text{if } obj_{text} \text{ is unique in GUI} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where OCR_{confid} denotes the confidence predicted by OCR. Note that the confidence value of icon *object* is calculated likewise by captioning. The smaller the confidence value, the less intuitive the *object* is. Therefore, only the *object* with the highest confidence value ($obj_{confid} > \alpha$) will apply the simplest and most straightforward description (Template 1), otherwise, we add the context of absolute position to help locate the *object* (Template 2). For the *object* whose text is not detected or recognized with low confidence, we leverage the context of its *neighbor* to help locate the target *object* (Template 3), e.g., tap the checkbox next to “Dark Mode”.

It is easy to describe a *SCROLL* action by its scrolling direction and offset (Template 5), e.g., scroll up a quarter of the screen. However, such an offset description is not precise and intuitive. To address this, if a new element with text appears by scrolling, we add this context to help describe where to scroll to (Template 4), e.g., scroll down half of the screen to “Advanced Setting”.

The description of *INPUT* is similar to *TAP*. For the high-confidence *object* with text (Template 6), it generates: Input $[text]$ in the $[obj_{text}]$ edittext. Different from the *TAP* descriptions, we do not apply the context of absolute position to help locate the low-confidence *object*. This is because the *objects* are gathering at the top when the keyboard pops up, so the absolute positioning may not help. Instead, we use the relative position of *neighbor* to describe the input *object* of which text is not detected or recognized with low confidence (Template 7), e.g., Input “John” in the edittext below “Name”.

After generating the natural language description for each action clip, we embed the description into the recording as subtitles as shown in Fig. 6. In detail, we create the subtitles by using the Wand image annotation library [42] and synchronize the subtitle display at the beginning of each action clip.

III. AUTOMATED EVALUATION

In this section, we described the procedure we used to evaluate CAPdroid in terms of its performance automatically. Since our approach consists of two main automated steps to obtain the actions from the recordings, we evaluate these phases accordingly, including Action Segmentation (Section II-B), and Action Attribute Inference (Section II-C). Consequently, we formulated the following two research questions:

- **RQ1:** How accurate is our approach in segmenting action clips from GUI recordings?
- **RQ2:** How accurate is our approach in inferring action attributes from clips?

To perform the evaluation automatically, we leveraged the existing automated app exploration tool Droidbot [43] to collect GUI recordings with ground-truth actions. In detail, we first collected 439 top-rated Android apps from Google Play covering 14 app categories (e.g., news, tools, finance, etc.). Each app was run for 10 minutes by Droidbot to automatically explore app functionalities by simulating user actions on the GUI. The simulated actions, including operation time, types, locations, etc, were dumped as metadata, representing the ground truth. Meanwhile, we captured a screen recording to record the actions for each app at 30 fps. As discussed in Section II-A, users may use different indicators to depict their touches. To make our recordings as similar to real-world recordings as possible, we adopted different touch indicators to record actions, including 181 default, 152 cursor, and 106 custom. In total, we obtained 439 10-min screen recordings as the experimental dataset for the evaluation.

A. RQ1: Accuracy of Action Segmentation

Experimental Setup. To answer RQ1, we evaluated the ability of our CAPdroid to precisely segment the recordings into action clips and accurately classify the actions. To accomplish this, we utilized the metadata of action operation time as the ground-truth. During preliminary observation with many recordings, we found that, due to the delay between commands and operations on the device, it may have small time-frame differences between the ground-truth and the recorded actions. To avoid these small differences, we broadened the ground-truth of the actions by 5 frames. In total, we obtained 12k *TAP*, 4k *SCROLL*, and 1k *INPUT* clips from 439 screen recordings.

Metrics. We employed two widely-used evaluation metrics, e.g., video segmentation F1-score, and accuracy. To evaluate the precision of segmenting the action clips from recordings, we adopted video segmentation F1-score [44], which is a standard video segmentation metric to measure the difference between two sequences of clips that properly accounts for the relative amount of overlap between corresponding clips. Consider the clips segmented by our method (c_{our}) and ground truth (c_{gt}), vs-score is computed as $\frac{2|c_{our} \cap c_{gt}|}{|c_{our}| + |c_{gt}|}$, where $|c|$ denotes the duration of the clip. The higher the score value, the more precise the method can segment the video. We further adopted accuracy to evaluate the performance of our

Method	TAP		SCROLL		INPUT		Overall	
	VS	Acc	VS	Acc	VS	Acc	VS	Acc
ABS	0.56	0.69	0.59	0.69	0.67	0.73	0.61	0.71
HIST	0.71	0.80	0.62	0.71	0.75	0.84	0.70	0.79
SIFT	0.61	0.71	0.60	0.73	0.63	0.79	0.62	0.75
SURF	0.55	0.71	0.59	0.72	0.60	0.77	0.58	0.74
EDGE	0.61	0.75	0.55	0.70	0.66	0.78	0.61	0.75
Ours	0.81	0.89	0.83	0.92	0.90	0.97	0.84	0.93

TABLE II: Performance comparison of action segmentation. “VS” denotes the video segmentation F1-score, and “Acc” denotes the accuracy of action classification.

approach to discriminate action types from clips. The higher the accuracy score, the better the approach can classify the actions.

Baselines. To demonstrate the advantage of using SSIM as the image similarity metric to segment actions from GUI recordings, we compared it with 5 image-processing baselines, including pixel level (e.g, absolute differences ABS [45], color histogram HIST [46]), structural level (e.g., SIFT [47], SURF [48]), and motion-estimation level (e.g., edge detection EDGE [49]). Due to the page limit, we omitted the details of these well-known methods.

Results. Table II shows the overall performance of all baselines. The performance of our method is much better than that of other baselines, i.e., 20%, 17% boost in video segmentation F1-score and accuracy compared with the best baseline (HIST). Although HIST achieves the best performance in the baselines, it does not perform well as it is sensitive to the pixel value. This is because the recordings can often have image noise due to fluctuations of color or luminance. The image similarity metrics based on structural level (i.e., SIFT, SURF) are not sensitive to image pixel, however, they are not robust to compare GUIs. This is because, unlike images of natural scenes, features in the GUIs may not distinct. For example, a GUI contains multiple identical checkboxes, and the duplicate features of checkboxes can significantly affect similarity computation. Besides, motion-estimation baseline (EDGE) cannot work well in segmenting actions from GUI recordings, as GUI recordings are artificial artifacts with different rendering processes. In contrast, our method using SSIM achieves better performance as it takes similarity measurements in many aspects from spatial and pixel, which allows for a more robust comparison.

Our method also makes mistakes in action segmentation due to two reasons. First, we wrongly segment one action clip into multiple ones due to the unexpected slow resource loading, e.g., one clip for the GUI transition of a user action, and the other clip for the GUI’s resource loading. Second, some GUIs may contain animated app elements such as advertisements or movie playing, which will change dynamically, resulting in mistake action segmentation and classification.

B. RQ2: Accuracy of Action Attribute Inference

Experimental Setup. To answer RQ2, we evaluated the ability of our CAPdroid to accurately infer the action attributes from the segmented clips. To accomplish this, we

Methods	TAP			SCROLL			INPUT			Overall
	default	cursor	custom	default	cursor	custom	default	cursor	custom	
V2S [8]	84.19%	69.66%	36.10%	85.19%	63.31%	29.00%	-	-	-	61.24%
GIFdroid [4]	85.78%	88.01%	87.16%	72.84%	71.01%	69.77%	35.13%	32.11%	28.39%	63.35%
CAPdroid	91.06%	90.28%	92.67%	94.87%	94.63%	95.12%	87.86%	88.62%	88.11%	91.46%

TABLE III: Performance comparison of action attribute inference.



Fig. 8: Examples of bad cases in action localization.

leveraged the metadata of action attributes as the ground-truth. Since our approach employs a deep-learning-based model (Section II-C1) to infer *TAP* location, we trained and tested our model based on the metadata of *TAP* actions. Note that a simple random split cannot evaluate the model’s generalizability, as tapping on the screens in the same app may have very similar visual appearances. To avoid this data leakage problem [50], we split the *TAP* actions in the dataset by apps, with the 8:1:1 app split for the training, validation, and testing sets, respectively. We also ensure a similar number of three types of touch indicators (i.e. default, cursor, custom) in the split dataset. The resulting split has 9k actions in the training dataset, 1.5k in the validation dataset, and 1.5k in the testing dataset. The model was trained in an NVIDIA GeForce RTX 2080Ti GPU (16G memory) with 30 epochs. In total, we obtained 1.5k *TAP* locations, 4k *SCROLL* offsets, and 1k *INPUT* text as the attributes of testing data.

Metrics. We employed accuracy as the evaluation metric to measure the performance of our approach in inferring *TAP*, *SCROLL*, and *INPUT* action attributes, respectively. As one element occupies a certain area, tapping any specific point within that area can successfully trigger the action. So, we measured whether our predictions are within the ground-truth element. For *SCROLL* actions, we measured whether our inferred scroll offset is the same as the ground-truth. For *INPUT* actions, we measured whether our approach can infer the correct input text. The higher the accuracy score, the better the approach to infer action attributes.

Baselines. We set up 2 state-of-the-art methods as our baselines to compare with our CAPdroid. V2S [8] proposed the first GUI video analysis technique, that utilizes deep-learning models to detect the touch indicator for each frame in a video and then classify them to user actions. As V2S only detects the default touch indicator, we followed their procedure to train corresponding deep-learning models to detect cursor and custom indicators. GIFdroid [4] developed a novel lightweight tool to detect the user actions by first extracting the keyframes from the GUI recording and then mapping it to the GUI transition graph (UTG) to extract the execution actions. We also followed the details in their paper to obtain the UTG graph.

Results. Table III shows the overall performance of all

methods. Our method outperforms in all actions, e.g., on average 91.33%, 94.87%, 88.19% for *TAP*, *SCROLL*, and *INPUT*, respectively. Our method is on average 30.2% more accurate compared with V2S in action attribute inference, due to three main reasons. First, our method models the features from both the spatial (i.e., touch indicator) and temporal (i.e., GUI animation) across the frames to enhance the performance of the model in inferring *TAP* actions, i.e., on average 91.33% vs 63.32% for CAPdroid and V2S respectively. Second, our method achieves better performance in inferring action attributes even for the recordings with different touch indicators. This is because, CAPdroid proposes a novel touch indicator-independent method by leveraging the similarity of consecutive frames to identify actions, while V2S leverages the opacity of the indicator, e.g., a fully solid touch indicator represents the user first touches the screen, and it fades to less opaque when a finger is lifted off the screen. The opacity of the indicator works well for the default touch indicator (on average 84.69%), but not for the others (on average 66.48%, 32.55% for cursor and custom). Third, CAPdroid can accurately (on average 88.19%) infer the input text from the clips, while V2S cannot detect semantic actions.

CAPdroid is on average 28% (91.46% vs 63.35%) more accurate even compared with the best baseline (GIFdroid). This is because, the content in GUIs of some apps (e.g., financial, social, music apps) are dynamic, causing the keyframes wrongly map to the states in the UTG. This issue further exacerbates input text inference, as the input text from the recording is specific but the input text in UTG is randomly generated.

Albeit the good performance of our approach, we still make wrong inferences about some actions. We manually check those wrong cases and find two common causes. First, as shown in Fig. 8, the overlap of similar colors between the touch indicators and icons leads to less distinct features of the indicators, causing false-positive action localization. Second, although the good performance of our OCR method, it still makes wrong text recognition, especially missing spaces. We believe the emergence of advanced OCR methods can further improve the accuracy of our approach.

IV. USEFULNESS EVALUATION

In this section, we conducted a user study to evaluate the usefulness of our generated descriptions (reproduction steps) for replaying bug recordings in real-world development environments.

Procedure: We recruited another 8 participants including 6 graduate students (4 Master, 2 Ph.D) and 2 software developers

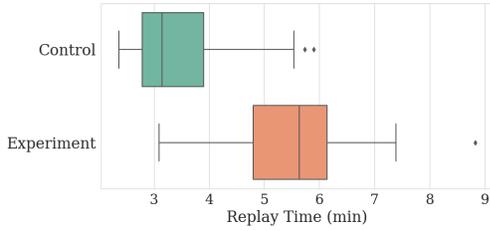


Fig. 9: Bug replay time.

to participate in the experiment. All students have at least one-year experience in developing Android apps and have worked on at least one Android apps project as interns in the company. Two software developers are more professional and have two-year working experience in a large company in Android development. Given that they all have experience in Android app development and bug replay, they are recognized as substitutes for developers in software engineering research experiments [51].

To mitigate the threat of user distraction, we conducted the experiment in a quiet room individually without mutual discussion. We first gave them an introduction to our study and also a real example to try. Each participant was then asked to reproduce the same set of 10 randomly selected bug recordings from real-world issue reports in GitHub, on average, 3.6 TAP, 1.2 SCROLL, and 1.0 INPUT per recording. The experimental bug recordings can be seen in our online appendix². The study involved two groups of four participants: the control group P_1, P_2, P_3, P_4 who gets help with the reproduction steps written by reporters from GitHub, and the experimental group P_5, P_6, P_7, P_8 who gets help with the natural language description generated by our tool. Each pair of participants $\langle P_x, P_{x+4} \rangle$ has comparable development experience, so the experimental group has similar capability to the control group in total. Note that we did not ask participants to finish half of the tasks with our tool while the other half without assisting tool to avoid potential tool bias. We recorded the time used to reproduce the bug recordings in Android. Participants had up to 10 minutes for each bug replay. To minimize the impact of stress, we gave a few minutes break between each bug replay. At the end of the tasks, we provided 5-point Likert-scale questions to collect their feedback, in terms of clearness, conciseness, and usefulness. We further collected participants’ feedback through a few open-ended questions, which can help us bring more insight into our tool, including how could the subtitles be improved, are there any software engineering tasks that would benefit from subtitles, etc.

Results: Overall, participants appreciate the usefulness of our approach for providing them with clear and concise step descriptions to describe the actions performed on the bug recordings, so that they can easily replay them. Box plot in Fig. 9 shows that, given our generated reproduction steps, the experimental group reproduces the bug recording much

Measures	Control	Experiment
Clearness	2.50	4.25*
Conciseness	1.75	4.50*
Usefulness	-	4.75

TABLE IV: Performance comparison between the experimental and control group. * denotes $p < 0.01$.

faster than that of the control group (with an average of 3.46min versus 5.53min, saving 59.8% of the time). This brings a preliminary insight of the usefulness of our generated reproduction steps to help participants locate and replay the actions.

Table IV shows the overall results received from participants. All participants admit that our approach can provide more easy-to-understand step descriptions for them, in terms of 4.25 vs 2.50 in clearness, and 4.50 vs 1.75 in conciseness, compared with the control group. In addition, they demonstrate several advantages of our reproduction steps, such as complete steps, region/text of interest, technical language, etc. Since the steps we generate are matched with each action one-to-one, participants can easily track each step, while the missing steps in the control group may confound participants: whether the step description corresponds to the current GUI. P_5 also finds the absolute positioning and element relationship description particularly useful to him, because such description can narrow down the spatial regions in GUI and easily locate the GUI element in which a bug occurs. P_3 reports that some users may use inconsistent words to describe the steps. For example, users may use “play the film” to describe the button with the text “movie”, making the developers hard to reproduce in practice. In contrast, the descriptions we generate are entirely based on GUI content, so it is easy to find the GUI elements.

The participants strongly agree (4.75) with the usefulness of our approach due to two reasons. One is the potential of our structured text to benefit short- and long-term downstream tasks, such as bug triaging, test migration, etc. The potential downstream is discussed in Section V. The other is the usefulness of the subtitle in the recording, revealing the action segmentation of our approach. P_2 in the control group finds the touch indicator to be inconspicuous and sometimes GUI transitions are too abrupt to realize. In contrast, with the help of our approach, P_6 praises the subtitle in the recording as it informs the timing of each action.

To understand the significance of the differences, we further carry out the Mann-Whitney U test [52] (specifically designed for small samples) on the replaying time, clearness, conciseness, and usefulness between the experimental and the control group respectively. The test results suggest that our approach does significantly help the participants reproduce bug recordings more efficiently ($p < 0.01$). There is also some valuable feedback provided by the participants to help improve the CAPdroid. For example, participants want higher-level semantic step descriptions, e.g., tap the first item in the list group, which can lead to more insights into the bugs. We will

²<https://github.com/sidongfeng/CAPdroid>

investigate the possible solution as our future work.

V. DISCUSSION

We have discussed the limitations of our approach at the end of each subsection of the evaluation in Section III, such as errors due to slow rendering in action segmentation (Section III-A), low contrast between touch indicators and icons in action attribute inference (Section III-B), etc. In this section, we discuss the implication of our approach and future work.

Downstream tasks supported by video captioning. There are many downstream tasks based on the textual bug reports, such as automated bug replay [53], [54], test migration [55], [56], duplicate bug detection [57], [58], [59], etc. Few of them can be applied to visual bug recordings. Our approach to automatically caption bug recording provides a semantic bridge between textual and visual bug reports. In detail, CAPdroid complement the existing methods, as the first process of these downstream tasks is usually to employ natural language processing (NLP) techniques to extract the representations of bug steps into a structural grammar, such as action, object, and position, which can be automatically extracted by our approach in visual bug recording.

Generality across platforms. Results in the usefulness evaluation in Section IV have demonstrated the usefulness of our approach in generating high-quality descriptions for Android bug recordings to help developers with bug replay in real-world practice. Supporting bug recordings of different platforms (e.g., iOS, Web) can bring analogous benefits to developers [60]. As the actions from different platforms exert almost no difference, and our approach is purely image-based and non-intrusive, it can be generalized to caption bug recordings for other platforms with reasonable customization efforts to our approach. In the future, we will conduct thorough experiments to evaluate the performance of CAPdroid in supporting those platforms.

Accessibility of GUI recording. Tutorial videos (e.g., app usage recordings) are widely used to guide users to access unfamiliar functionalities in mobile apps. However, it is hard for people with vision impairments (e.g., the aged or blind) to understand those videos unless asking for caregivers to describe the action steps from the tutorial videos to help them access the video content [61]. Our approach might be applied to enhance the accessibility of tutorial videos by generating clear and concise subtitles for reproduction steps, enabling people with vision impairments to easily access information and service of the mobile apps for convenience.

VI. RELATED WORK

Vision to Language semantically bridges the gap between visual information and textual information. The most well-known task is image captioning, describing the content of an image in words. Many of the studies proposed novel methods to generate a textual description for GUI image, in order to enhance app accessibility [11], [62], [14], [15], screen navigation [63], [64], GUI design search [65], [66],

[67], automate testing [68], [69], [70], [71], [72], [73], etc. Chen et al. [13] designed an approach that uses a machine translator to translate a GUI screenshot into a GUI skeleton, a functional natural language description of GUI structure. Moran et al. [12] proposed image captioning methods Clarity to describe the GUI functionalities in varying granularity. In contrast, we focused on a more difficult task - video captioning, generating natural language to describe the semantic content of a sequence of images. To the best of our knowledge, this is the first work translating the GUI recording into textual descriptions.

Earlier works [74], [75] proposed sequence-to-sequence video captioning models that extract a sequence of image features to generate a sequence of text. These models showed their advantage in video summarization, but it was hard to achieve the goal of generating multiple concrete captions with their temporal locations from the video (a.k.a dense video captioning). Intuitively, dense video captioning can be decomposed into two phases: event segmentation and event description. Existing methods tackled these two sub-problems using event proposal and captioning modules, and exploited two ways to combine them for dense video captioning. We borrowed the two-phase idea to generate a natural language description for GUI recording, denoting events as user actions.

To segment the events from the videos, Krishna et al. [76] proposed the first segmentation method by using a multi-scale proposal module. Some of the following works [77], [78] aimed to enrich the event representations by context modeling, event-level relationships, or multi-modal feature fusion, enabling more accurate event segmentation. However, these methods were designed for general videos which contain more natural scenes like human, plants, animals, etc. Different from those videos, our GUI recordings belonged to artificial artifacts with different image motions (i.e., GUI rendering). While some previous studies worked on domain-specific GUI recordings, they focused on high-level GUI understanding, such as duplicate bug detection [60], GUI animation linting [79], [80], etc. In contrast, we focused on the fine-grained user actions in the GUI recording. To analyse and segment actions from the GUI recording, many record-and-replay tools were developed based on different types of information, including the runtime information [81] and app artifacts [82], [4], [17]. Nurmuradov et al. [83] introduced an advanced lightweight tool to record user interactions by displaying the device screen in a web browser. Feng et al. [4], [17] proposed an image processing method to extract the keyframes from the recording and mapped them to states in the GUI transitions graph to replay the execution trace. However, they required the installation of underlying frameworks, or instrumenting apps which is too heavy and time-consuming. Bernal et al. [8] implemented a deep learning-based tool named V2S to detect and classify user actions from specific recordings, a high-resolution recording with a default Android touch indicator. But more than 32% of end-users cannot meet that requirement in real-world bug reports according to our analysis in Section II-A. In contrast, considering the diversity of touch indicators in the general

GUI recordings from end-users, we propose a more advanced approach to capture the spatial features of touch indicators and the temporal features of touch effects, to achieve better performance on user action identification.

To generate video captions, many works [78], [84] started using one single unified deep-learning model (one-fit-all). Recent works infused knowledge about objects in the video by using object detectors to generate more informative captions. For example, Zhang et al. [85] adopted an object detector to augment the object feature to yield object-specific video captioning. Different from the natural scenes, generating action-centric descriptions for GUI recording requires a more complex GUI understanding, as there are many aspects to consider, such as the elements in the GUI, their relationships, the semantics of icons, etc. Therefore, we modeled GUI-specific features by using mature methods, and then proposed a tailored algorithm to automatically generate natural language descriptions for GUI recordings.

VII. CONCLUSION

The bug recording is trending in bug reports due to its easy creation and rich information. However, watching the bug recordings and understanding the user actions can be time-consuming. In this paper, we present a lightweight approach CAPdroid to automatically generate semantic descriptions of user actions in the recordings, without requiring additional app instructions, recording tools, or restrictive video requirements. Our approach proposes image-processing and deep-learning models to segment bug recordings, infer user actions, and generate natural language descriptions. The automated evaluation and user study demonstrate the accuracy and usefulness of CAPdroid in boosting developers' productivity.

In the future, we will keep improving our method for better performance in terms of action segmentation and action attribute inference. According to user feedback, we will also improve the understanding of GUI to achieve higher-level semantic descriptions.

REFERENCES

- [1] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, 2002.
- [2] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, 2005, pp. 35–39.
- [3] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 298–308.
- [4] S. Feng and C. Chen, "Gifdroid: Automated replay of visual bug reports for android apps," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022, pp. 1045–1057.
- [5] "Record the screen on your iphone, ipad, or ipod touch," <https://support.apple.com/en-us/HT207935>, 2022.
- [6] "Take a screenshot or record your screen on your android device," <https://support.google.com/android/answer/9075928?hl=en>, 2021.
- [7] S. Feng and C. Chen, "Gifdroid: Automated replay of visual bug reports for android apps," *arXiv preprint arXiv:2112.04128*, 2021.
- [8] 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.
- [9] M. A. Gernsbacher, "Video captions benefit everyone," *Policy insights from the behavioral and brain sciences*, vol. 2, no. 1, pp. 195–202, 2015.

- [10] T. J. Garza, "Evaluating the use of captioned video materials in advanced foreign language learning," *Foreign Language Annals*, vol. 24, no. 3, pp. 239–258, 1991.
- [11] 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.
- [12] K. Moran, A. Yachnes, G. Purnell, J. Mahmud, M. Tufano, C. B. Cardenas, D. Poshyvanyk, and Z. H'Doubler, "An empirical investigation into the use of image captioning for automated software documentation," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 514–525.
- [13] C. Chen, T. Su, G. Meng, Z. Xing, T. Zhou, and Y. Liu, "From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 665–676.
- [14] S. Feng, S. Ma, J. Yu, C. Chen, T. Zhou, and Y. Zhen, "Auto-icon: An automated code generation tool for icon designs assisting in ui development," in *26th International Conference on Intelligent User Interfaces*, 2021, pp. 59–69.
- [15] S. Feng, M. Jiang, T. Zhou, Y. Zhen, and C. Chen, "Auto-icon+: An automated end-to-end code generation tool for icon designs in ui development," *ACM Transactions on Interactive Intelligent Systems*, vol. 12, no. 4, pp. 1–26, 2022.
- [16] J. D. Cintas and A. Remael, *Audiovisual translation: subtitling*. Routledge, 2014.
- [17] S. Feng and C. Chen, "Gifdroid: an automated light-weight tool for replaying visual bug reports," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 95–99.
- [18] "Github," <https://github.com/>, 2022.
- [19] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.
- [20] "Du recorder," <https://www.du-recorder.com/>, 2022.
- [21] H. Chen, M. Sun, and E. Steinbach, "Compression of bayer-pattern video sequences using adjusted chroma subsampling," *IEEE transactions on circuits and systems for video technology*, vol. 19, no. 12, pp. 1891–1896, 2009.
- [22] R. Sudhir and L. D. S. S. Baboo, "An efficient cbir technique with yuv color space and texture features," *Computer Engineering and Intelligent Systems*, vol. 2, no. 6, pp. 78–85, 2011.
- [23] M. Livingstone and D. H. Hubel, *Vision and art: The biology of seeing*. Harry N. Abrams New York, 2002, vol. 2.
- [24] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [25] Y. Du, C. Li, R. Guo, C. Cui, W. Liu, J. Zhou, B. Lu, Y. Yang, Q. Liu, X. Hu et al., "Pp-ocrv2: Bag of tricks for ultra lightweight ocr system," *arXiv preprint arXiv:2109.03144*, 2021.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
- [27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [28] C. Feichtenhofer, "X3d: Expanding architectures for efficient video recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 203–213.
- [29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [30] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.
- [31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.
- [32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [33] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [34] P. Irani, C. Gutwin, and X. D. Yang, "Improving selection of off-screen targets with hopping," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 299–308.

- [35] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu, and G. Li, "Object detection for graphical user interface: old fashioned or deep learning or a combination?" in *proceedings of the 28th ACM joint meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1202–1214.
- [36] "diffilib in python," <https://docs.python.org/3/library/diffilib.html>, 2022.
- [37] 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.
- [38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.
- [39] B. Deka, Z. Huang, C. Franzen, J. Hibschan, D. Afegan, 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.
- [40] X. Zhang, L. de Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach *et al.*, "Screen recognition: Creating accessibility metadata for mobile applications from pixels," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.
- [41] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4055–4064.
- [42] "Wand - python package," <https://docs.wand-py.org/en/0.6.2/>, 2022.
- [43] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 23–26.
- [44] A. Truong, P. Chi, D. Salesin, I. Essa, and M. Agrawala, "Automatic generation of two-level hierarchical tutorials from instructional make-up videos," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–16.
- [45] C. Watman, D. Austin, N. Barnes, G. Overett, and S. Thompson, "Fast sum of absolute differences visual landmark detector," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4827–4832.
- [46] X.-Y. Wang, J.-F. Wu, and H.-Y. Yang, "Robust image retrieval based on color histogram of local feature regions," *Multimedia Tools and Applications*, vol. 49, no. 2, pp. 323–345, 2010.
- [47] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [48] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.
- [49] C. Zhan, X. Duan, S. Xu, Z. Song, and M. Luo, "An improved moving object detection algorithm based on frame difference and edge detection," in *Fourth international conference on image and graphics (ICIG 2007)*. IEEE, 2007, pp. 519–523.
- [50] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, pp. 1–21, 2012.
- [51] 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.
- [52] M. P. Fay and M. A. Proschan, "Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics surveys*, vol. 4, p. 1, 2010.
- [53] Y. Zhao, T. Yu, T. Su, Y. Liu, W. Zheng, J. Zhang, and W. G. Halfond, "Redroid: automatically reproducing android application crashes from bug reports," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 128–139.
- [54] M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso, "Automatically translating bug reports into test cases for mobile apps," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 141–152.
- [55] S. Talebipour, Y. Zhao, L. Dojčilović, 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.
- [56] 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.
- [57] A. Hindle and C. Onuczko, "Preventing duplicate bug reports by continuously querying bug reports," *Empirical Software Engineering*, vol. 24, no. 2, pp. 902–936, 2019.
- [58] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "Uied: a hybrid tool for gui element detection," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1655–1659.
- [59] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering*. IEEE, 2012, pp. 70–79.
- [60] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshy-vanyk, "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.
- [61] X. Liu, P. Carrington, X. Chen, and A. Pavel, "What makes videos accessible to blind and visually impaired people?" in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.
- [62] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan, "Widget captioning: Generating natural language description for mobile user interface elements," 2020, pp. 5495–5510.
- [63] T. J.-J. Li, M. Radensky, J. Jia, K. Singarajah, T. M. Mitchell, and B. A. Myers, "Pumice: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations," in *Proceedings of the 32nd annual ACM symposium on user interface software and technology*, 2019, pp. 577–589.
- [64] T. J.-J. Li and O. Riva, "Kite: Building conversational bots from mobile apps," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 96–109.
- [65] C. Chen, S. Feng, Z. Xing, L. Liu, S. Zhao, and J. Wang, "Gallery dc: Design search and knowledge discovery through auto-created gui component gallery," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–22, 2019.
- [66] S. Feng, C. Chen, and Z. Xing, "Gallery dc: Auto-created gui component gallery for design search and knowledge discovery," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 80–84.
- [67] C. Chen, S. Feng, Z. Liu, Z. Xing, and S. Zhao, "From lost to found: Discover missing ui design semantics through recovering missing tags," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. CSCW2, pp. 1–22, 2020.
- [68] Z. Liu, C. Chen, J. Wang, X. Che, Y. Huang, J. Hu, and Q. Wang, "Fill in the blank: Context-aware automated text input generation for mobile gui testing," *arXiv preprint arXiv:2212.04732*, 2022.
- [69] M. Xie, Z. Xing, S. Feng, C. Chen, L. Zhu, and X. Xu, "Psychologically-inspired, unsupervised inference of perceptual groups of gui widgets from gui images," *arXiv preprint arXiv:2206.10352*, 2022.
- [70] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Guided bug crush: Assist manual gui testing of android apps via hint moves," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–14.
- [71] —, "Owl eyes: Spotting ui display issues via visual understanding," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 398–409.
- [72] Y. Su, C. Chen, J. Wang, Z. Liu, D. Wang, S. Li, and Q. Wang, "The metamorphosis: Automatic detection of scaling issues for mobile apps," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.
- [73] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Nighthawk: Fully automated localizing ui display issues via visual understanding," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 403–418, 2022.

- [74] Y.-F. Ma, L. Lu, H.-J. Zhang, and M. Li, "A user attention model for video summarization," in *Proceedings of the tenth ACM international conference on Multimedia*, 2002, pp. 533–542.
- [75] Y.-F. Ma, X.-S. Hua, L. Lu, and H.-J. Zhang, "A generic framework of user attention model and its application in video summarization," *IEEE transactions on multimedia*, vol. 7, no. 5, pp. 907–919, 2005.
- [76] R. Krishna, K. Hata, F. Ren, L. Fei-Fei, and J. Carlos Niebles, "Dense-captioning events in videos," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 706–715.
- [77] J. Wang, W. Jiang, L. Ma, W. Liu, and Y. Xu, "Bidirectional attentive fusion with context gating for dense video captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7190–7198.
- [78] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, "Videobert: A joint model for video and language representation learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7464–7473.
- [79] D. Zhao, Z. Xing, C. Chen, X. Xu, L. Zhu, G. Li, and J. Wang, "Seenomaly: Vision-based linting of gui animation effects against design-don't guidelines," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1286–1297.
- [80] S. Feng, M. Xie, and C. Chen, "Efficiency matters: Speeding up automated testing with gui rendering inference," *arXiv preprint arXiv:2212.05203*, 2022.
- [81] 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.
- [82] P. Krieter and A. Breiter, "Analyzing mobile application usage: generating log files from mobile screen recordings," in *Proceedings of the 20th international conference on human-computer interaction with mobile devices and services*, 2018, pp. 1–10.
- [83] D. Nurmuradov and R. Bryce, "Caret-hm: recording and replaying android user sessions with heat map generation using ui state clustering," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 400–403.
- [84] L. Zhu and Y. Yang, "Actbert: Learning global-local video-text representations," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8746–8755.
- [85] Z. Zhang, Y. Shi, C. Yuan, B. Li, P. Wang, W. Hu, and Z.-J. Zha, "Object relational graph with teacher-recommended learning for video captioning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 13 278–13 288.