SeeAction: Towards Reverse Engineering How-What-Where of HCI Actions from Screencasts for UI Automation

1st Dehai Zhao
CSIRO's Data61
Sydney, Australia
dehai.zhao@data61.csiro.au

2nd Zhenchang Xing
CSIRO's Data61 & School of Computing, ANU
Sydney, Australia
zhenchang.xing@data61.csiro.au

3rd Qinghua Lu
CSIRO's Data61
Sydney, Australia
qinghua.lu@data61.csiro.au

4th Xiwei Xu
CSIRO's Data61
Sydney, Australia
xiwei.xu@data61.csiro.au

5th Liming Zhu
CSIRO's Data61 & School of CSE, UNSW
Sydney, Australia
liming.zhu@data61.csiro.au

***Abstract*—UI automation is an useful technique for UI testing, bug reproduction and robotic process automation. Recording the user actions with an application assists rapid development of UI automation scripts, but existing recording techniques are intrusive, rely on OS or GUI framework accessibility support or assume specific app implementations. Reverse engineering user actions from screencasts is non-intrusive, but a key reverse-engineering step is currently missing - recognize human-understandable structured user actions ([command] [widget] [location]) from action screencasts. To fill the gap, we propose a deep learning based computer vision model which can recognize 11 commands and 11 widgets, and generate location phrases from action screencasts, through joint learning and multi-task learning. We label a large dataset with 7260 video-action pairs, which record the user interactions with Word, Zoom, Firefox, Photoshop and Windows 10 Settings. Through extensive experiments, we confirm the effectiveness and generality of our model, and demonstrate the usefulness of a screencast-to-action-script tool built upon our model for bug reproduction.**

***Index Terms*—UI Automation, Multi-task Learning, Action Recognition, UI Testing**

I. Introduction

UI automation reduces or assists the repetitive, manual digital tasks of a human worker by a "digital worker". In the context of software engineering, it is used to simulate an end-user's interaction with an application for UI testing [1], [2], [3], [4] or bug reproduction [5], [6], [7], [8]. Beyond software engineering, UI automation is a core capability in robotic process automation [9] which aims to automate business operations such as invoice processing, inventory or human resources management. UI automation can be achieved through textual-script or visual-script based techniques, such as Selenium [10], Appium [11], UIAutomator [12], UiPath [13] and AirTest [14]. UI Automation scripts can be coded from scratch, which involves repetitive and error prone manual work. In practice, UI automation tools often support record-and-replay for recording human-computer interaction (HCI) actions from which automation scripts can be produced. The recorded scripts can be automatically replayed or be used as a boilerplate script to expand from.

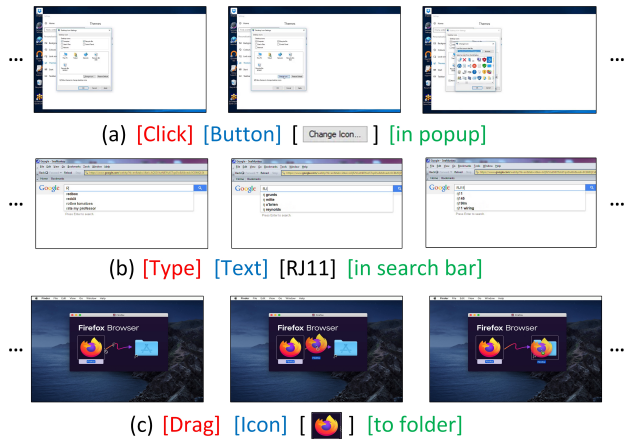

Fig. 1. Examples of structured HCI actions

HCI actions can be recorded at three levels: operating system (OS) [15], [16], [17], [18], GUI framework [19] or application [3], [20], [11], [10], [1]. Recording at application level is preferable because it outputs human understandable scripts, rather than system or API calls. Figure 1 shows some examples of application-level HCI actions which we refer to as *structured HCI actions* in this work. Structured refers to three integral parts of an action: *how* (command, e.g., click, type, drag); *what* (widgets, if any, that the action operates, e.g., button, text, icon); and *where* (location in the UI where the action occurs, e.g., popup, search bar, folder). Such structured HCI actions are in a human understandable format that we see in application usage tutorials [21] and steps-to-reproduce descriptions in bug reports [6], [7], [8]. Furthermore, they are

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00144

robust to changes of screen resolutions and sizes, and changes in spatial arrangements of GUI widgets, compared with low-level screen coordinates.

Some HCI recording tools require app source code (e.g., Espresso [20]) or customized OS (e.g., VALERA [19]). Due to their intrusiveness, such recoding tools have limited applicability [22]. Most recording techniques are less intrusive, but they still require either accessibility support from OS or GUI framework [11], [12] or dynamic instrumentation of software applications [1], [2]. As such, user actions involving non-accessible-friendly widgets, for example custom widgets like non-Android webview in hybrid apps, cannot be effectively recorded. Dynamic instrumentation hooks to specific APIs (e.g., event dispatcher, Android TextView for rich text editor) and is sensitive to system and implementation changes. It may also introduce the incompatibility with the instrumented apps [22].

It is desirable to develop non-intrusive record-and-replay techniques, independent of OS, GUI frameworks and application implementations. This non-intrusiveness is important to support user experience testing [23] and to support closed embedded systems that are increasingly used in entertainment, education and industry [9]. Screen recording non-intrusively produces a video recording of HCI actions. Structured user actions can be non-intrusively reverse-engineered from screen recordings using computer vision techniques. This reverse-engineering process involves three steps: first, *video segmentation* to segment the video into video fragments of individual actions; second, *structured action recognition* to generate how-what-where information of HCI actions from video fragments; third, *widget identification* to identify GUI widget details involved the actions. Techniques have been developed for video segmentation [24], [25], simple HCI action prediction [26], [27], and widget detection [28]. However, effective techniques for structured action recognition from action videos are missing, without which we will not have a working reverse-engineering pipeline.

Recently, large language models (LLMs) have gained popularity, with some advancing to support vision features[29]. However, supporting video action recognition remains a significant challenge. While some work has developed LLMs for action recognition in videos[30], [31], these models primarily support videos in natural scenes. It has been demonstrated that screencasts present different challenges compared to natural scenes[26]. Moreover, LLMs require substantial computational resources. It is not necessary to employ large and heavy models for all problems. A specialized and compact AI model is a more efficient and effective choice for addressing domain-specific problems.

This work fills the gap. We propose a novel model to recognize user actions in screencast fragments and generate structured natural language descriptions of user actions such as those in Figure 1. Given an action screencast, we first compute the structural similarity [32] of adjacent frames to obtain change regions and similarity maps. Our model takes three data streams as input: the sequences of original video frames, cropped change regions and similarity maps. Then the three data streams pass through three 3D Covolutional Neural Networks (CNNs) [33] respectively which extract spatio-temporal features. The features from three channels are concatenated for three tasks: command classification, widget classification and location phrase generation. We define 11 classes of commands and 11 classes of widgets to be recognized from screencast fragments, based on previous literature [17], [34], [26] and the actions and widgets supported by visual script based automation tools such as Airtest [14] and sikulix [35]. Location phrase is free form that covers common expressions of locations on UI from application usage tutorials [21]. The model is trained end-to-end by joint learning and multi-task learning.

To train and evaluate our model, we build the first dataset of screencasts labeled with structured user action descriptions, which contains a total of 7,260 video-action pairs. The videos are application demonstration videos from YouTube and bug reproduction screencasts from Firefox. We select five desktop applications with distinct functionalities: Word, Zoom, Firefox, Photoshop, Windows 10 Settings. The applications run on Windows, Linux and macOS. The videos are segmented and labeled with structured user-action descriptions manually by the two authors. Our experiments on this dataset show that our model achieves high recognition accuracy (0.82 F1-score for command classification, 0.85 F1-score for widget classification, and 0.77 BLEU [36], 0.51 METEOR [37], 0.76 ROUGE [38] and 2.85 CIDEr [39] for location phrase generation). Furthermore, the joint learning of the three data streams can significantly improve the model performance, and the multi-task learning can also enhance the model performance, especially for location phrase generation.

To demonstrate the usefulness of our model, we integrate it with a simple heuristic-based video fragmentation method and the GUI widget detection method [28] into a screencast-to-actionscript tool. We apply this tool to 100 bug reproduction screencasts from Firefox and obtain the action scripts for reproducing these bugs. In a comparative study, the Firefox users reproduce 45% more bugs following the action scripts generated by our tool, compared with those following the original steps-to-reproduce (S2R) descriptions in the bug reports due to many S2R quality issues such as ambiguous steps, vocabulary mismatch, missing steps, or even absent S2R text [5]. The results show the potentials of our model for supporting UI automation tasks such as bug reproduction.

We make the following contributions in this paper:

- To the best of our knowledge, this is the first work to recognize structured user actions from screencasts. It is an essential step towards non-intuitive UI automation.
- To recognize structured user actions from screencasts, we propose a novel computer vision model, which is trained end-to-end by joint learning and multi-task learning.
- We label and contribute the first screencast dataset with 7260 video-action pairs, which record the user interactions with five popular desktop applications.

- Through extensive experiments, we not only confirm the effectiveness and generality of our model, but also demonstrate its usefulness for generating high-quality, human-understandable action scripts for bug reproduction.

II. Approach

Given a screencast fragment recording of a user action with an application, our approach outputs a structured natural language description of this action. The screencast fragment is first processed by computer vision techniques to obtain three data streams, i.e., the sequence of original frames, cropped change regions and similarity maps. Next, the three image sequences pass through three 3D CNNs [33] respectively to extract spatio-temporal features (see Figure 2), which are finally used to predict structured user actions including command category, widget category and location phrase. We formulate the spatio-temporal feature extraction as joint learning and structured user action prediction as multi-task learning.

A. Structured HCI Actions

We summarize 11 command classes and 11 widget classes based on our personal experiences with software applications, previous work [26], [17], [18], and the commands and widgets supported by popular visual-script automation tools (e.g., AirTest [14], sikuli [35]). The 11 command categories include Click, Drag, Hover, Scroll down, Scroll up, Select, Type, Zoom in, Zoom out, Appear and Disappear. The first nine commands are user operations, while Appear and Disappear are app responses (i.e., the outcomes of user operations). We include app responses because they represent critical information in the screencasts for downstream applications, for example, assertions for UI testing [19]. The 11 widget classes are Button, Checkbox (including Radio Button), Dropdown, Icon, Image, Text (including Text Field), Window, Page, Tab, Popup and Others. The first six widget classes are atomic widgets for presenting information and collecting user inputs, while Window, Page, Tab and Popup are container widgets for grouping visual content. We include others to represent miscellaneous widgets that are not widely present in applications (e.g., geometry graph). Different from commands and widgets, the location information requires a free-form short phrase to express, which will be assembled from a vocabulary. The vocabulary can be built from application usage tutorials (e.g., WikiHow [21]) or from the labeled screencast dataset (see Section III-A). Note that command, widget and location have latent associations. For example, user actions can be [click] [button] or [type] [text] [in search bar], but not [type] [button] or [click] [text] [in toolbar].

B. Model Input

A user interaction with application is recorded as a screencast. The raw screencast will be processed into a fixed-length sequence of adjacently-distinct screenshots. From this sequence of screenshots (frames), a grayscale similarity map is computed between adjacent screenshots and change regions are subsequently detected from the similarity map. The sequence of the original screenshots, together with the obtained sequences of similarity maps and change regions, will be fed into the deep learning model.

1) Input Screencast Processing: We decode a raw screencast video into a sequence of frames at a sample rate of 5 frames per second (fps). According to our empirical observation, 5fps sample rate keeps sufficient frames to represent HCI actions and screen changes, without too many redundant frames. We further scan the frame sequence from the beginning and remove the adjacently-identical frames (i.e., no screen changes). User actions vary in duration (0.2 to 10 seconds in our labeled data), leading to different frame sequence lengths. But the deep learning model requires a fixed length sequence of screenshots. We normalize an input frame sequence to a fixed length S by down- or up-sampling. Given a frame sequence $F = \{f_1, f_2, \ldots, f_M\}$, we pick the first frame f_1 and the last frame f_M to a sampled frame sequence $F' = \{f_1, f_M\}$, as these two frames contain the most important information to represent how a user action starts and ends. Next, if $S \leq M$, we randomly down-sample $S-2$ frames $\{f_i, \ldots, f_j\}$ from the remaining frame sequence $F = \{f_2, \ldots, f_{M-1}\}$ and insert them into the sampled frame sequence $F' = \{f_1, f_i, \ldots, f_j, f_M\}$ by the original frame order. If $S > M$, we randomly duplicate the frames from the original frame sequence $S - M$ times and insert the obtained frames into the sampled frame sequence by the original frame order. In this work, we set $S = 8$ as this length is long enough to cover the duration of most user actions and contains the least duplicate frames.

2) Model Input Representation: Many user actions (e.g., click button, type text) result in only small-scale screen changes. Image features in such small-scale screen changes would be too weak to recognize user actions if the whole frame is taken as the only input. Therefore, we detect change regions which will be used as an additional input to the model. First, we compute the structural similarity [32] which produces a similarity score for each pair of pixels at the same location in the two frames. Based on these similarity scores, we produce a grayscale image (i.e., a similarity map) in which brighter pixels indicate lower similarities between the corresponding pixels in the two frames. On this similarity map, we compute the regions of connected non-black pixels and determine their bounding boxes. The corresponding areas in original frames are cropped by these bounding boxes as change regions. There can be more than one change region between the two frames and we keep the largest change region. We prepare three input data streams: 1) the sequence of original frames (each frame is a RGB image) which provide the detailed context of user actions; 2) the sequence of cropped change-regions (each cropped change-region is a RGB image) which provide screen-change details resulting from user actions; and 3) the sequence of similarity maps (each map is a grayscale image) which provide a simplified overview of screen changes. All input images are resized to the same size (e.g., 224×224).

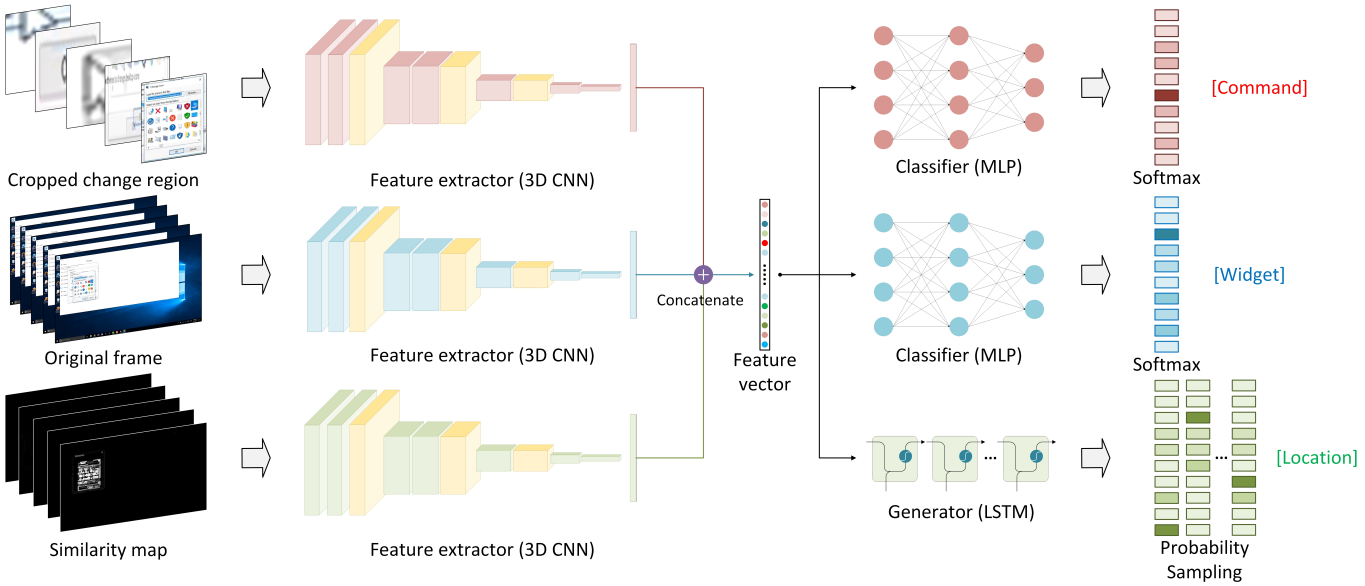

Fig. 2. Overall model architecture of SeeAction

C. Model Architecture

Our model takes as input three image sequences (original frames, change regions and similarity maps) obtained from a user-action screencast, and outputs a structured natural language description of the user action performed in the input screencast, in the form of [command] [widget] [location], such as [click] [button] [in popup].

1) 3D-CNN based Spatio-Temporal Feature Extraction: User actions result in changes on screen and last for a short time period. 3D-CNNs has been successfully applied to video data for both natural scenes [40] and computer rendered screencasts [41] to encode video data into low-dimensional abstract spatio-temporal features for action recognition.

As shown in Figure 2, we use three different 3D-CNN feature extractors to encode the spatio-temporal features in the three input image sequences respectively. The three 3D-CNN encoders share the same model structure but do not share weights. In our model, we aggregate three 3D-CNNs as one spatio-temporal feature extractor and optimize them jointly. Joint learning makes the whole model smaller and improves the efficiency of training process. The three 3D-CNN encoders extracts a low-dimensional feature vector E_{chreg}, E_{orig}, E_{simmap} for cropped change regions, original frames and similarity maps, respectively. The model concatenates the three vectors $E = E_{chreg} \bigoplus E_{orig} \bigoplus E_{simmap}$ as an overall representation of the user action occurred in the action screencast. We set the length of three feature vectors as 512, resulting in the length of fused feature vector E as 1536. The fused feature vector is fed into the subsequent multi-task action prediction.

2) Multi-task User Action Prediction: Considering the latent associations among [command] [widget] [location], we formulate the model prediction as three sub-tasks in a multi-task learning setting. As command and widget comes from a fixed set of labels, we formulate command and widget prediction as two classification tasks, i.e., video tagging [42]. As the location is a phrase assembled from a vocabulary, we formulate location prediction as a sequence-to-sequence generation task, i.e. video captioning [43].

The two classifiers and one generator produce three outputs, i.e., command class, widget class and location phrase, respectively. We compute three loss functions for the three tasks. The command classification and widget classification task use cross-entropy loss, which is denoted as $L_{act} = -\sum_{c=1}^{N_{act}} y_c \log(p_c)$ and $L_{obj} = -\sum_{c=1}^{N_{obj}} y_c \log(p_c)$ respectively, where y is the ground-truth label and p is the predicted class. The location phrase generation task applies cross-entropy loss to each word output by LSTM, and it is computed as $L_{loc} = -\sum_{i=1}^{maxL} \sum_{c=1}^{N_{voc}} y_c^i \log(p_c^i)$, where $maxL$ is the maximum length of the generated text sequence, y^i is the ground-truth label and p^i is the predicted word at i-th position. We formulate the three tasks as multi-task learning by computing the sum of three losses and get the total loss, i.e., $L = L_{act} + L_{obj} + L_{loc}$. The training objective is to minimize the total loss L and make the model reach global optimization.

III. Experiment Design

This section describes our experiment dataset for model training and testing, and the metrics used for evaluation.

A. Dataset

In this study, we manually label 288 application usage screencasts (in total 12.8 hours) for five desktop applications (see Table I). This creates a dataset of 7,260 screencast fragments labeled with corresponding structured user action descriptions, which to the best of our knowledge is the first large-scale dataset of its kind.

TABLE I
DATASET OF USE-ACTION SCREENCASTS

Application	#Video	Dur.(h)	OS	Resolution
Photoshop	6	0.75	macOS	1280 × 720
Win10 Settings	21	3.83	Windows	1280 × 720
Firefox	158	1.32	Linux & macOS	Multiple
Zoom	55	4.70	Windows	1280 × 720
Word	48	2.21	Windows	1280 × 720
Total	**288**	**12.82**	**-**	**-**

TABLE II
OVERALL STATISTICS OF LABELED ACTION DATASET

App	#Action	Duration (s) (mean ± std)	#Words (mean ± std)	VocSize
PS	1629	0.93 ± 0.75	4.43 ± 0.65	42
WinSet	1524	1.42 ± 1.13	4.40 ± 0.60	28
Firefox	1425	0.86 ± 1.06	4.16 ± 0.43	42
Zoom	1352	0.98 ± 1.01	4.10 ± 0.31	33
Word	1330	1.29 ± 1.32	4.03 ± 0.18	40
Total	**7260**	**0.99 ± 1.10**	**4.16 ± 0.42**	**71**

1) Data collection: We consider five popular desktop applications that most people use daily for diverse tasks, including Mozilla Firefox (web browsing), Microsoft Word (document processing), Adobe Photoshop (graphics editing), Zoom (online conference) and Windows 10 Settings (system configuration). We collect application usage screencasts from two sources. First, we collect application demonstration videos from YouTube by searching keywords "How to use applicationname" and download the most viewed ones at high definition resolution (1280×720). We keep the videos in which users are actually using the applications but exclude those recorded in slide presentation. Second, we crawl steps-to-reproduce (S2R) screencasts from the Firefox's bug reports [44]. These steps-to-reproduce screencasts have diverse screen sizes and resolutions. As shown in Table I, we finally collect a total of 288 videos with 12.8 hours duration. The collected videos involve three main stream operating systems (i.e. Windows, Linux and macOS). Because of different usage characteristics of applications (e.g., the density of user actions), video numbers and duration vary among the five applications. Zoom window may contain other application views, for example, screen shared by online conference participants, which poses an interesting challenge to predicting user actions with Zoom.

2) Manual labeling: Given an action screencast, the manual labeling has two main steps: video segmentation and structured action description generation. The screencast is decoded to a sequence of frames at 5fps sampling rate. We discard frames with no screen changes by computing image similarity because they contain no action information, which removes about 49% of frames in the raw screencasts. For efficient and consistent labeling, we develop a web application by which the annotator can view and navigate a screencast by frames, with the change regions highlighted. The annotators use this tool to annotate the start and end frame of an action fragment and to select command/widget class (as defined in Section II-A) and enter location phrase for the action fragment. Location is a free-form phrase and should be described specifically as "in toolbar", "in popup menu", etc., rather than general window areas such as "at top" or "on the right screen", because specific location information would be more human understandable.

Two authors participate in the label processing. The two annotators use the five studied applications regularly in their work and are very familiar with the GUIs and user operations of these applications. Following the common practice for action video labeling [26], two annotators label the whole dataset independently. When there are disagreements, two annotators discuss to decide the final labels. The labeling process took about 7 person-weeks. Table II summarizes the statistics of labeled action fragments. The five applications have similar numbers of action fragments. We obtain in total 7260 action fragments. The action fragment duration is about 1 second (5 frames) on average. Each user action is described by about 4 words including command name, widget name and location phrase. The location vocabulary contains 71 distinct words.

B. Evaluation metrics

We evaluate command and widget classification by Precision, Recall, F1-score and Accuracy. The correctness of a prediction is determined against the human label (i.e. ground truth) for an input action fragment. Accuracy is an overall performance of 11 commands (or widgets), which is computed by the number of action fragments predicted correctly over all action fragments in the test dataset. As location phrase generation is a video captioning task, we evaluate it by BLEU [36], ROUGE [38], METEOR [37] and CIDEr [39], which are widely used in image captioning tasks [43]. BLEU, ROUGE and METEOR roughly correspond to precision, recall and F1 in the classification task. CIDEr uses TF-IDF to weight the words in the phrase. In this work, considering the short length of location phrases (2.16±0.42), we compute 1-gram BLEU, ROUGE, METEOR and CIDEr.

IV. EVALUATION RESULTS AND FINDINGS

We conduct extensive experiments on our labeled action dataset to investigate the following three research questions:

- **RQ1.** What is the overall performance of the proposed model for command classification, widget classification and location phrase generation?
- **RQ2.** How do the three input data streams and the joint learning affect the model performance?
- **RQ3.** How does the multi-task learning affect the model performance?

A. Overall model performance (RQ1)

1) Motivation: Our model generates structured user action descriptions which include 11 command classes, 11 widget classes, and free-form location phrases for user action screencasts. Different commands, widgets and locations have diverse

TABLE III
OVERALL MODEL PERFORMANCE

Output	Class	Precision	Recall	F1-score
Command Acc = 0.84	Click	0.89	0.93	0.91
	Type	0.91	0.88	0.90
	Drag	0.93	0.84	0.88
	Hover	0.85	0.89	0.87
	Select	0.93	0.82	0.87
	Scroll down	0.85	0.83	0.84
	Appear	0.79	0.76	0.78
	Zoom in	0.69	0.80	0.74
	Scroll up	0.64	0.83	0.72
	Disappear	0.71	0.73	0.72
	Zoom out	0.78	0.62	0.69
	Average	**0.82**	**0.83**	**0.82**
Widget Acc = 0.87	Image	0.95	0.87	0.91
	Button	0.86	0.91	0.88
	Text	0.89	0.87	0.88
	Icon	0.88	0.85	0.86
	Checkbox	1.00	0.75	0.86
	Window	0.93	0.78	0.85
	Others	0.85	0.83	0.84
	Popup	0.78	0.87	0.83
	Dropdown	0.83	0.81	0.82
	Tab	0.92	0.73	0.81
	Page	0.70	0.92	0.80
	Average	**0.87**	**0.84**	**0.85**
Location	**BLEU**	**ROUGH**	**METEOR**	**CIDEr**
	0.77	0.76	0.51	2.85

visual and temporal characteristics and changes the UIs differently. This RQ aims to investigate the overall performance of our model in this diverse and challenging learning setting.

2) Method: In this experiment, we combine the action data of five applications, and split it into 80% for model training and 20% for model testing. We use the full model for this evaluation, which takes three input data streams (i.e. original frames, cropped change regions and similarity maps) and simultaneously predicts three outputs: command class, widget class and location phrase. The model is trained for 10 epochs until convergence. We perform 5-fold cross validation and report the average performance metrics.

3) Result: Table III[1] shows the full model's performance on the three outputs[2]. For command classification, it achieves an average of 0.84 accuracy and 0.82 F1-score. Both "Click" and "Type" have higher F1-score than other command classes, but the reasons are not the same. The large amount of "Click" instances in the dataset makes the model well-trained for the "Click" class and results in better performance. Although "Type" does not have many instances, it always interacts with text which has similar visual features and is easy to learn. We find that our model is often confused between "Scroll down" and "Scroll up". In fact, most incorrectly-recognized "Scroll down" is classified as "Scroll up". This is because the two actions have very similar visual features, and "Scroll down" has better performance because it has more instances than "Scroll up" in the dataset. A Similar situation is observed

[1]Readers can refer to this GitHub https://github.com/DehaiZhao/SeeAction for all experiment results in this paper.

[2]We also examine the model performance on each application, which is similar to the overall performance. Please see the results in the Github repo

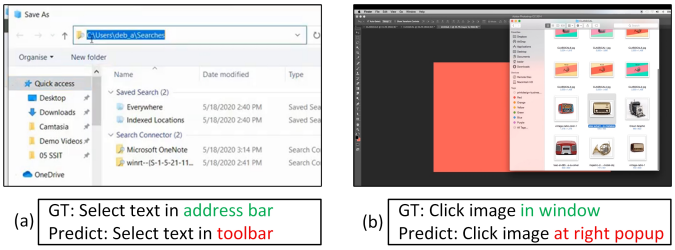

Fig. 3. Failure examples of location prediction

for "Zoom in" versus "Zoom out". "Appear" and "Disappear" have relatively low F1-score, and most incorrectly-classified "Appear" and "Disappear" instances are predicted as "Click". This is because click often triggers appearing or disappearing of certain UI parts.

Widget classification achieves an average of 0.87 accuracy and 0.85 F1-score, and the F1-score for all widget classes is above 80%. "Image" achieves 91% F1-score, because image-related actions usually involve larger change regions than the actions on small widgets such as button and icon. Furthermore, images have more salient features than other large widgets such as window and page. "Button" and "Text" have F1-score of 88%, because our dataset has a large number of "Button" instances, and all "Text" widgets have similar visual features. We find that most incorrectly-classified "Dropdown" instances are predicted as "Button" or "Popup". This is because "Dropdown" has very similar appearance as "Button" before it is expanded, and clicking "Dropdown" usually triggers a menu appearing, which looks like "Popup". Some "Window" instances are incorrectly predicted as "Popup" or "Page" as these widgets share similar visual features. In fact, it might not be necessary to distinguish these window-like widgets involved in user actions, such as [zoom-in] [window] or [zoom-out] [page].

Location phrase generation achieves 0.77 BLEU, 0.76 ROUGH, 0.51 METEOR and 2.85 CIDEr, which means our model can generate location phrases with good precision, recall and F1 and can generate the most important location information. Compared with command and widget classification, location phrase generation is a more challenging task because it needs to compose an appropriate free-form phrase for many similar GUI contexts and widgets. We show two failure examples in Figure 3. In Figure 3(a), user selects text in address bar, but the model predicts the location as "toolbar". In Figure 3(b), the model predicts "at right popup" while the human label is "in window". The wrong predictions could be caused by the fact that both address bar and toolbar appear at the top of application window, and both window and popup are container widgets and share similar visual features.

Our model can accurately predict command and widget classes. Most confusions come from the commands and the widgets with similar visual features, such as scroll down/up, dropdown versus button, window versus page. Location phrase generation is a more challenging task, but the generation results by our model are acceptable.

TABLE IV
IMPACT OF SINGLE-, TWO- AND THREE-INPUT IMAGE SEQUENCES

	Command				Widget				Location			
Input	**Prec**	**Recall**	**F1**	**Accu**	**Prec**	**Recall**	**F1**	**Accu**	**BLEU**	**ROUGH**	**METEOR**	**CIDEr**
CropCR	0.76	0.72	0.73	0.74	0.71	0.66	0.68	0.70	0.63	0.63	0.47	0.88
Origin	0.65	0.62	0.63	0.67	0.68	0.65	0.66	0.66	0.63	0.63	0.47	0.89
SimMap	0.70	0.67	0.68	0.73	0.80	0.76	0.77	0.77	0.62	0.62	0.46	0.87
CropCR + Origin	0.75	0.73	0.74	0.76	0.73	0.74	0.73	0.75	0.66	0.66	0.48	1.40
CropCR + SimMap	0.77	0.73	0.74	0.78	0.81	0.78	0.79	0.79	0.63	0.63	0.47	0.86
Origin + SimMap	0.77	0.73	0.73	0.77	0.83	0.78	0.80	0.80	0.63	0.63	0.46	0.99
CropCR + Origin + SimMap	0.82	0.83	0.82	0.84	0.87	0.84	0.85	0.87	0.77	0.76	0.51	2.85

B. Impact of joint learning (RQ2)

1) Motivation: Our model takes three input image sequences: original frames, cropped change regions, and similarity maps, which are designed to play complementary roles to represent user action context and features. In this RQ, we investigate the impact and synergy of the three input image sequences by ablation study.

2) Method: We develop six variants of input: three single-input variants (cropped change regions (CropCR), originial frames (Origin) or similarity maps (SimMap)), three two-inputs variants (Crop-CR+Origin, CropCR+SimMP, or Origin+SimMap). These variants use the same model configuration as the full model but ablate the corresponding sub-model(s) for the absent input sequences. The command classifier, widget classifier and location generator remain unchanged. We use the same data for training model variants as for the full model (Origin+CropCR +SimMap). All model variants are trained for 10 epochs until convergence. We perform 5-fold cross validation and report the average performance metrics.

3) Result: Table IV reports the performance of the six input variants and the full model. With single-input sequence, the model achieves only about 0.7 F1-score and accuracy for predicting commands and widgets. The cropped change region is good at predicting commands, with 74% accuracy and 73% F1-score. Similarity map has better performance than original frame, especially for predicting widgets. The grayscale similarity map contains much less details than the original RGB frame, but this excludes much noise irrelevant to the action which is beneficial. In addition, the similarity map can display accurate contour of a change region instead of a rectangle bonding box, which is important for recognizing widgets. For example, "Button" usually has more smooth edge than "Text", and "Checkbox" is a small rectangle in general. In contrast, original frame contains too many details, which often blur the most important visual features related to user actions, especially for those commands or widgets involving small change regions, for example, type a short word.

Adding one more input sequence can improve the overall model performance, especially adding cropped change regions or similarity maps to original frames. For example, original-frame-only-input has poor performance for recognizing small widgets such as "Button", "Checkbox" and "Icon", but adding similarity map can significantly improve the performance (+0.14 in F1-score and +0.14 in accuracy), because similarity map indicates which region on the original frame the model should pay more attention to. Furthermore, two-inputs model variants achieve very close performance and the gap between the best and the worst model variants narrow when using two input sequences. With the use of three input sequences (i.e., full model), we observe further 0.06-0.08 increase in F1-score and accuracy for predicting commands and widgets.

Each input image sequence has its contributions for recognizing user actions. Combining all three input streams and formulating them as joint learning can improve the overall model performance.

C. Impact of multi-task learning (RQ3)

1) Motivation: Our model generates a structured natural language description in the form of [command] [widget] [location]. Considering the potential latent associations between commands, widgets and locations, we train the model in a multi-task learning setting. In this RQ, we want to compare our method with independent learning and traditional video captioning.

2) Method: For independent learning, we optimize the command classifier, the widget classifier and the location phrase generator separately, rather than optimizing the combined loss of the three outputs. For traditional video captioning [45], we train an LSTM decoder to generate an unstructured sentence (e.g., "click button in popup") for an action screencast, rather than separate command, widget and location. In both variant settings, the encoder remains the same as the original model. The variant models are trained in the same way as the original model. We perform 5-fold cross validation and report the average performance metrics.

3) Result: Table V presents the performance of the two variant settings and the original model. Independent learning results in marginal decrease in F1-score and accuracy for predicting commands and 0.04 decrease in F1-score and accuracy for predicting widgets, compared with multi-task learning. For location phrase generation, independent learning results in relatively larger performance degradation (-0.09 BLEU, -0.06 ROUGE, -0.02 METEOR and -1.39 CIDEr), compared with multi-task learning. This result agrees with the nature of multi-task learning. It is an inductive transfer mechanism whose principle goal is to improve generalization performance while keeping the model prediction performance [46]. To gain

TABLE V
PERFORMANCE OF MULTI-TASK LEARNING VERSUS INDEPENDENT LEARNING OR TRADITIONAL VIDEO CAPTIONING (UNSENTGEN)

	Command				Widget				Location			
Input	**Prec**	**Recall**	**F1**	**Accu**	**Prec**	**Recall**	**F1**	**Accu**	**BLEU**	**ROUGH**	**METEOR**	**CIDEr**
Independent	0.83	0.80	0.81	0.83	0.85	0.79	0.81	0.83	0.66	0.66	0.49	1.46
UnSentGen	0.32	0.30	0.30	0.50	0.19	0.17	0.17	0.37	0.59	0.59	0.45	0.80
Our model	0.82	0.83	0.82	0.84	0.87	0.84	0.85	0.87	0.77	0.76	0.51	2.85

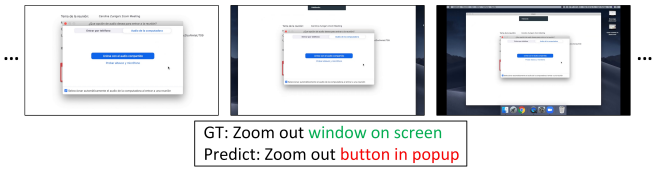

Fig. 4. Failure example of traditional video captioning

the global optimization of all subtasks, tradeoffs would be made on the performance of some subtasks. In our work, multi-task learning keeps the performance of command and widget classification, while the more challenging location phrase generation task benefits from the command and widget classification in the multi-task learning setting.

For traditional video captioning (UnSentGen row), we observe many strange outputs such as "zoom out button in popup" in Figure 4, which is an impossible combination of command and widget. Such outputs result in very low F1-score and accuracy for predicting commands and widgets. This is because command and widget classification often require different spatio-visual features to make accurate prediction. In our structured prediction, two separate MLPs are used for command and widget classification respectively, which can extract command- or widget-specific features. In contrast, the unstructured sentence generation by traditional video captioning relies on one common feature vector for generating all three types of action information, which is challenging to satisfy all at the same time. Furthermore, the unstructured sentence generation has to generate a longer sentence from a larger vocabulary (combining command labels, widget labels and location words) than the three separate predictions. This inevitably increases the task difficulty.

Multi-task learning slightly improves the performance of command and widget classification, but it benefits the more challenging location phrase generation task. Traditional video captioning cannot reliably generate an unstructured sentence which contains all necessary command, widget and location information.

V. PILOT STUDY: BUG REPRODUCTION

Having evaluated our model's performance for structured user action recognition from screencasts, we would like to demonstrate a practical application on bug reproduction that our model enables.

A. Motivation

Bug reproduction is an essential software engineering task [5], [6], [7], [8]. There are two common ways to report a bug: text description and screencast demonstration. The study [5] shows that many users are not professional enough to write a high quality bug report with clear and complete steps to reproduce. This not only increases the burden of software maintainers to understand and analyze the bugs, but also creates a barrier to automated bug reproduction techniques based on the steps-to-produce descriptions in bug reports [6], [7]. In contrast, recording screencasts lowers the bar for ordinary users to report bugs and it can cover every single detail of a bug (when and how it occurs). GitHub has announced that video upload is available on its platform [47], meaning that video data is becoming more and more important in software community. However, the image nature of screencasts makes it hard to integrate video content with existing text-based tools. In this work, we conduct a pilot study of this potential usage. In particular, we investigate if the generated human-understandable action scripts from bug screencasts can help people reproduce the bugs.

B. A Screencast-to-ActionScript Tool

We develop a screencast-to-actionscript (S2AS) tool for our pilot study. As shown in Figure 5, S2AS integrates our SeeAction model with a heuristic-based video segmentation method and a UI widget detection method, thus constituting a full reverse-engineering process that converts an input action screencast into a script of structured user actions, such as those shown in Figure 1.

Based on our video labeling experience and the observation of our labeled action fragments, we design a simple image similarity based method for video segmentation, driven by the fact that when a user action occurs on the UI, there will be more or less pixel changes. As illustrated in Figure 5, for an action screencast in our dataset, user actions result in low screen similarities during the action time periods, while the periods without user actions have high screen similarities and remain stable over time. Based on this observation, S2AS segments the screencast into a series of action fragments by identifying the time points when the screen similarities start and finish changing.

For widget detection, we use an existing technique UIED [28] which can detect the bounding box of a widget and recognize its class. For detected text widgets, UIED uses Google OCR [48] to convert text widget images to texts. S2AS applies UIED to an action fragment in the reverse order (from

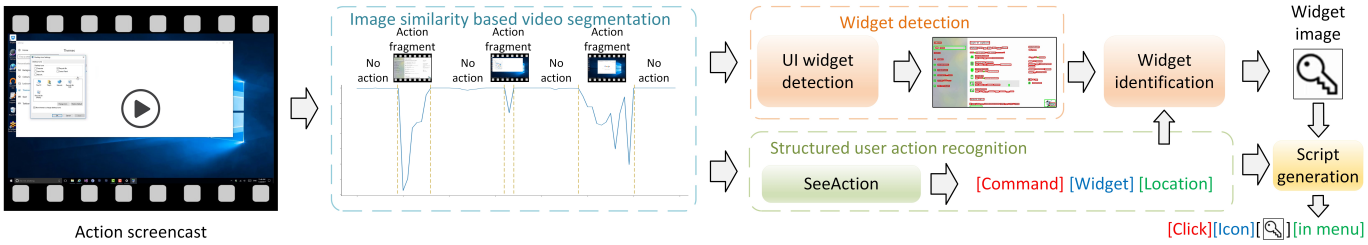

Fig. 5. Our Screencast-to-ActionScript Tool

the last frame to the first). UIED detects all widgets on a UI image. If UIED detects a widget on a frame whose bounding box overlaps with the change region identify by SeeAction for that frame and that has the same class as the widget class predicted by SeeAction, S2AS identifies this widget as the target widget of the user action. If the target widget is a non-text widget, S2AS crops the widget image by its bounding box. For text target widget, it takes the OCRed text as the widget information. Finally, it outputs a complete structured user action based on the structured user action predicted by SeeAction and the identified widget information, such as [click] [Icon] [ ] [in menu]. S2AS applies SeeAction and UIED to all action fragments and generates a script of user actions for the input screencast.

C. Pilot Study

1) Bug Reproduction Dataset: We randomly sample 100 bug reports with bug screencasts on Firefox Bugzilla [44]. These bugs were reported during January 2022 - January 2024 and have no overlap with our labeled dataset. They are related to a wide range of Firefox functionalities (e.g., website browsing, web page inspection, menu bar customization). By inspecting the text description of the collected bug reports, we summarize them to five quality grades, which include High Quality (HQ), Low Quality - Ambiguous Step (LQ-AS), Low Quality - Vocabulary Mismatch (LQ-VM), Low Quality - Missing Step (LQ-MS), and No Steps to Reproduce (N-S2R). High quality means the bug reports have detailed, clear and complete steps to reproduce the bugs. N-S2R means the complete absence of S2R description in bug report. The three low quality grades are defined according to the S2R quality issues revealed in [5]. Ambiguous step means a step matches more than one GUI component or event. Vocabulary mismatch means a step does not match any application interaction. Missing step means a required step is not described in the bug report. A bug report may suffer from several low quality issues. We annotate the most significant issue that prevents the bug reproduction as a bug's quality grade.

Two authors annotate the bug report qualities independently, and their initial annotations have 0.81 inter-rater agreement by Cohen's kappa [49]. For disagreements, they discuss to decide the final grade. As shown in Table VI, only 23 out of 100 bug reports have high-quality S2R descriptions, and 31 bug reports do not provide S2R description at all (the bug report may simply leave a statement "see the attached video"). For the rest 46 low-quality bug reports, 27, 10 and 9 of them have significant issues in missing steps, vocabulary mismatch and ambiguous steps, respectively. Our results are similar to the observation of S2R quality issues in [5].

TABLE VI
BUG REPORT QUALITY GRADES AND REPRODUCTION RESULTS

Q-Grade	HQ	LQ-AS	LQ-VM	LQ-MS	N-S2R	ALL
#S2R-QG	23	9	10	27	31	100
ReproS2R	23	3	4	8	0	38
ReproGAS	15	5	6	16	19	61

Q-Grade: quality grade, #S2R-QG: the number of bug reports that have different quality grades of Steps to Reproduce, ReproS2R: the number of bug reports that are correctly reproduced by the Steps to Reproduce descriptions, ReproGAS: the number of bug reports that are correctly reproduced by Generated Action Scripts.

2) Study Procedure: We recruit six graduate students from our school to participate in our study. Through a pre-study survey, all participants use Firefox regularly and are familiar with the main functionalities involved in the bugs. The participants do not know any background knowledge of this work before finishing the experiment. We process the bug screencasts in bug reports by our S2AC tool, and generate an action script for each of them. We collect the S2R descriptions (if any) in the bug reports. The six participants are split to two groups. Three of them (ReproS2R) are given the original S2R descriptions in the bug reports for bug reproduction, while the other three (ReproGAS) use the generated action scripts by our tool. All participants screen-record their bug reproduction process. Following the practice to evaluate the success of record-and-replay tools [1], [2], [3], [4], we manually check the Externally Visible State (EVS) of the bug reproduction, and regard the reproduction as a success if and only if the EVS by reproduction is the same as the EVS of the reported bug.

3) Results and Analysis: Table VI reports the union of the bug reports that were successfully reproduced by the three participants using the original S2R descriptions (ReproS2R) or the generated action scripts (ReproGAS). Obviously, unless the ReproS2R participants watch the bug screencasts, it is impossible to reproduce the 31 bugs without the S2R descriptions (N-S2R). Among the 69 bug reports with the S2R descriptions, only 38 of them are reproduced successfully by using the original S2R descriptions, and the majority (23) are high quality bug reports. The participants have much

lower success rate (only about 30%) for reproducing bugs based on low quality S2R descriptions. For example, given an ambiguous step "Search mozilla" (LQ-AS), one can type "mozilla" in either search bar or address bar of the browser, or may search some other keywords such as "what is mozilla". Another example is vocabulary mismatch (LQ-VM) "Disable OMTC". None of the three ReproS2R participants know what OMTC (off main thread compositing) is before the experiment and are stuck in this step. Reproducing the bug reports that miss important steps (LQ-MS) is also a big challenge. For example, the S2R of the bug report may mention only "set default search engine as Google". The participants have to figure how to reach the particular setting page and change the setting.

Using the generated action scripts, the number of successfully reproduced bugs increases to 61. Actually, the overall success rate and the success rates for different quality grades are all about 60%-65%, because using the generated action scripts to reproduce the bugs has nothing to do with the presence and quality of the original S2R descriptions. 19 of 31 bugs without the S2R descriptions and about 60% of bugs with low-quality S2R descriptions are successfully reproduced based on the generated action scripts. Compared with the 30% reproduction success rate based on the original low-quality S2R descriptions, this is a promising result as a tool like S2AC could be leveraged to help "lazy" bug reporters write good-quality S2R descriptions. Of course, compared with the high-quality S2R descriptions written by human (100% reproduction success), the generated action scripts still have room for improvement.

For the whole screencast-to-actionscript process, we find that video segmentation deserves more research as the inaccurate segmentation will inevitably affect the subsequent action recognition. For example, the user scrolls up and down the web page back and forth quickly, leading to less accurate segmentation between the consecutive similar user actions. Some user actions may also be over-segmented, leading to many too fine-grained actions. For example, a "click button" action are segmented into "hover button" and "click button", and selecting a long text are segmented to multiple "select text". Although much of such less ideal segmentation can still be understood and applied by human participants, some may cause the human confusion and the reproduction failures. For example, a "click checkbox" action is segmented into two (or more) "click checkbox" actions, and the second (and following) "click checkbox" may lead to unexpected results. Such less ideal segmentation would also negatively affect the automatic replay tools [27], [4], [14], [35].

For widget detection, we encounter some difficulty of adjusting two important parameters of UIED, $mingrade$ and $minelearea$. Too small value results in many noisy bonding boxes, and too large value misses many small widgets. After the experiments, we set $mingrade = 40$ and $minelearea = 3$ which achieves the best result on our dataset. But UIED still detects some noisy widgets or misses some widgets, which result in the wrong or incomplete action scripts affecting the bug reproduction. For example, an icon is misclassified as text, resulting in the mismatch of widget class, and consequently a missing step in the generated action script.

Our pilot study, albeit by no means conclusive, demonstrates the promise of non-intrusive screencast-to-actionscript reverse-engineering tool for bug reproduction. It also reveals the challenges in building the tool pipeline and the improvement for video segmentation and widget identification which we leave as our future work.

VI. Related Work

HCI actions can be recorded using instrumentation-based or computer vision based methods. Instrumentation-based methods are all more or less intrusive, relying on source code or customized OS [3], [19], or OS or GUI framework accessibility support [50], [17], [18], or runtime customization [1], [2]. Computer-vision based methods offer an alternative non-intrusive way, and receive growing attention in recent years. For examples, ActionNet [26] extracts primitive mouse and keyboard commands between two consecutive frames in an action screencast. V2S [27] recognizes tap command in mobile apps usage videos based on tap indicators. Other works investigate the extraction of GUI widgets (e.g., Waken [34], Prefab [51]) or application contents (e.g., source code [52], [53]). Our work is the first non-intrusive method to extract command, widget and location as a whole structured action, and our approach does not limit action fragment lengths, nor does it assume special visual indicators.

Recognizing HCI actions can help the analysis of developer behavior [17], [52], [54]. It is also the foundation for UI automation tasks, for example, record-and-replay testing [1], [2], [3], [55] and robotic process automation [9]. A wide range of UI automation tools are available, such as Selenium [10], Appium [11], Robotium [56], Monkeyrunner [57], UIAutomator [12]. All these tools rely on OS or GUI framework accessibility support. In contrast, visual script based UI automation such as Sikuli [35], AirTest [14], UiPath [13] and UI.Vision [58] use computer vision techniques to match widgets to be operated. Although they are platform independent, widget matching is sensitive to changes of screen size and resolution and widget appearance [4], [1]. All these UI automation techniques focus on action replay. In contrast, our work focuses on non-intrusive action recognition from screencasts.

Our approach can be regarded as a vision-to-text task. In computer vision community, image captioning [45], [59], [60] and video captioning [61], [62], [63], [64] are two typical vision-to-text tasks. Image captioning techniques have been adopted for generating GUI code from GUI design images [65], [66] or generating widget labels to improve accessibility [67], [68], [69], [70], [71]. However, these works generate unstructured text for describing the image or video. Our experiment in RQ3 show this cannot satisfy the need for complex user action recognition. Video segmentation [25], [24], video retrieval [72], [73] and object detection [74], [75]

are related techniques, which can be integrated with our approach to build the whole screencast-to-actionscript pipeline. Object detection models have been used for random GUI testing [76], and GUI widget specific detection method has also been proposed [28]. Other remotely related work includes predicting screens with code [77], GUI visual quality [78], game visual quality [79] and actionable GUI areas [80]. These works make prediction on static GUI images, while our goal is action recognition from screencasts.

VII. Threats to Validity

One threat to internal validity is the manual labeling bias of the screencast dataset. In order to minimize the errors, two authors label the whole dataset independently and finally reach agreement by discussion. We find that the disagreements mainly come from judging location of user action. For example, two authors give label of "in search bar" and "in textbox" to a video fragment, but they refer to the same location in Firefox, which has low impact to user action representation. Another internal threat is from the participants' bias of pilot study. None of the participant know the knowledge of our work before completing the experiment.

Threats to external validity include the selection of desktop applications and AI models, and widget detection accuracy. We use 5 popular desktop applications in our work and further evaluation is required on other types of applications such as video game and mobile applications. In terms of AI models, we applied 3D CNNs and LSTM in this work to support video action recognition. However, it is easy to replace those modules with alternative AI models such as Transformer[81] and BERT[82] Furthermore, the pilot study involves multiple steps of processing and the error accumulation of each step (especially widget detection by UIED) makes it not perfect enough to support all practical applications. This work focuses on generating structured natural language description from screencasts and we will try to optimize the pipeline by improving the performance of each step.

VIII. Conclusion and Future work

This work proposes a novel computer vision based model to reverse-engineer structured user actions from only the visual information recorded in screencasts. Our model extracts the spatio-temporal features jointly from the three complementary data streams, original frames, change regions and similarity maps, and predicts the command, widget and location in a multi-task setting. We evaluate our model on a large dataset of 7260 video-action pairs from five widely-used applications. Our results confirms the high accuracy of our model (even in the challenging across application training-testing setting) and the effectiveness of our joint-learning and multi-task learning design. We develop a prototype screencast-to-actionscript tool based on our model, and demonstrate its effectiveness in generating high-quality, human-understandable action scripts for bug reproduction through a pilot study on 100 Firefox bugs.

SeeAction's functionality extends beyond bug report reproduction. Its human-readable and structured HCI action descriptions can support a variety of downstream software engineering tasks, including UI automation and testing, broadening its applicability and potential impact. In the future, we will investigate more robust video segmentation and GUI widget detection methods to improve the screencast-to-actionscript pipeline for non-intrusive UI automation, and further explore its potential applications in various software engineering tasks.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 86–96.

[6] Y. Zhao, T. Yu, T. Su, Y. Liu, W. Zheng, J. Zhang, and W. G. Halfond, "Recdroid: automatically reproducing android application crashes from bug reports," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 128–139.

[7] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically discovering, reporting and reproducing android application crashes," in *2016 IEEE international conference on software testing, verification and validation (icst)*. IEEE, 2016, pp. 33–44.

[8] 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.

[9] 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.

[10] https://www.selenium.dev, Selenium, 2021.

[11] http://appium.io, Appium, 2021.

[12] https://developer.android.com/training/testing/ui-automator, UIAutomator, 2021.

[13] https://www.uipath.com, UiPath, 2021.

[14] https://airtest.netease.com/home, Netease, 2021.

[15] 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.

[16] 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.

[17] L. Bao, D. Ye, Z. Xing, X. Xia, and X. Wang, "Activityspace: a remembrance framework to support interapplication information needs," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 864–869.

[18] L. Bao, Z. Xing, X. Xia, and D. Lo, "Vt-revolution: Interactive programming video tutorial authoring and watching system," *IEEE Transactions on Software Engineering*, vol. 45, no. 8, pp. 823–838, 2018.

[19] 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.

[20] https://developer.android.com/studio/test/espresso-test-recorder, Google, 2021.

[21] https://www.wikihow.com, wikiHow, 2021.

[22] 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.

[23] http://ase-conferences.org/ase/past/ase2020/conf.researchr.org/details/ase-2020/ase-2020-a-mobile-2020--workshop-/7/Mobile-App-Testing-and-Analysis-Getting-There-in-Industrial-Cases.html, A-mobile, 2020.

[24] H. Xu, A. Das, and K. Saenko, "R-c3d: Region convolutional 3d network for temporal activity detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5783–5792.

[25] Y. Zhao, Y. Xiong, L. Wang, Z. Wu, X. Tang, and D. Lin, "Temporal action detection with structured segment networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2914–2923.

[26] D. Zhao, Z. Xing, C. Chen, X. Xia, and G. Li, "Actionnet: Vision-based workflow action recognition from programming screencasts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 350–361.

[27] 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.

[28] 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.

[29] https://chatgpt.com/, chatgpt, 2024.

[30] Y. Zhao, I. Misra, P. Krähenbühl, and R. Girdhar, "Learning video representations from large language models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 6586–6597.

[31] J. Bi, N. M. Nguyen, A. Vosoughi, and C. Xu, "Misar: A multimodal instructional system with augmented reality," *arXiv preprint arXiv:2310.11699*, 2023.

[32] Z. Wang and A. C. Bovik, "Modern image quality assessment," *Synthesis Lectures on Image, Video, and Multimedia Processing*, vol. 2, no. 1, pp. 1–156, 2006.

[33] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[34] N. Banovic, T. Grossman, J. Matejka, and G. Fitzmaurice, "Waken: reverse engineering usage information and interface structure from software videos," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, 2012, pp. 83–92.

[35] http://sikulix.com, Sikulix, 2021.

[36] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[37] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[38] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[39] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4566–4575.

[40] K. Soomro, A. R. Zamir, and M. Shah, "A dataset of 101 human action classes from videos in the wild," *Center for Research in Computer Vision*, vol. 2, no. 11, 2012.

[41] 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.

[42] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[43] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick, "Microsoft coco captions: Data collection and evaluation server," *arXiv preprint arXiv:1504.00325*, 2015.

[44] https://bugzilla.mozilla.org/home, Bugzilla, 2021.

[45] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[46] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[47] https://github.blog/2021-05-13-video-uploads-available-github, GitHub, 2021.

[48] https://cloud.google.com/vision/docs/ocr, Google, 2021.

[49] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[50] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, "Tracking real-time user experience (true) a comprehensive instrumentation solution for complex systems," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 2008, pp. 443–452.

[51] M. Dixon and J. Fogarty, "Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1525–1534.

[52] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou, "Extracting and analyzing time-series hci data from screen-captured task videos," *Empirical Software Engineering*, vol. 22, no. 1, pp. 134–174, 2017.

[53] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too long; didn't watch! extracting relevant fragments from software development video tutorials," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 261–272.

[54] D. M. Hilbert and D. F. Redmiles, "Extracting usability information from user interface events," *ACM Computing Surveys (CSUR)*, vol. 32, no. 4, pp. 384–421, 2000.

[55] J. Chen, A. Swearngin, J. Wu, T. Barik, J. Nichols, and X. Zhang, "Extracting replayable interactions from videos of mobile app usage," *arXiv preprint arXiv:2207.04165*, 2022.

[56] https://github.com/RobotiumTech/robotium, Robotium, 2021.

[57] https://developer.android.com/studio/test/monkeyrunner, Monkeyrunner, 2021.

[58] https://ui.vision, UI.Vision, 2021.

[59] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4651–4659.

[60] J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: Fully convolutional localization networks for dense captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4565–4574.

[61] J. Xu, T. Yao, Y. Zhang, and T. Mei, "Learning multimodal attention lstm networks for video captioning," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 537–545.

[62] X. Li, Z. Zhou, L. Chen, and L. Gao, "Residual attention-based lstm for video captioning," *World Wide Web*, vol. 22, no. 2, pp. 621–636, 2019.

[63] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen, "Video captioning with attention-based lstm and semantic consistency," *IEEE Transactions on Multimedia*, vol. 19, no. 9, pp. 2045–2055, 2017.

[64] Y. Pan, T. Yao, H. Li, and T. Mei, "Video captioning with transferred semantic attributes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6504–6512.

[65] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2018, pp. 1–6.

[66] C. Chen, T. Su, G. Meng, Z. Xing, 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.

[67] 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.

[68] F. Mehralian, N. Salehnamadi, and S. Malek, "Data-driven accessibility repair revisited: on the effectiveness of generating labels for icons in android apps," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 107–118.

[69] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan, "Widget captioning: Generating natural language description for mobile user interface elements," *arXiv preprint arXiv:2010.04295*, 2020.

[70] G. Li and Y. Li, "Spotlight: Mobile ui understanding using vision-language models with a focus," *arXiv preprint arXiv:2209.14927*, 2022.

[71] S. Feng, M. Xie, Y. Xue, and C. Chen, "Read it, don't watch it: Captioning bug recordings automatically," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2349–2361.

[72] Y. Yan, N. Cooper, O. Chaparro, K. Moran, and D. Poshyvanyk, "Semantic gui scene learning and video alignment for detecting duplicate video-based bug reports," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[73] 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.

[74] 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.

[75] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[76] 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.

[77] A. Bergh, P. Harnack, A. Atchison, J. Ott, E. Eiroa-Lledo, and E. Linstead, "A curated set of labeled code tutorial images for deep learning," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1276–1280.

[78] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "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.

[79] J. Ye, K. Chen, X. Xie, L. Ma, R. Huang, Y. Chen, Y. Xue, and J. Zhao, "An empirical study of gui widget detection for industrial mobile games," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1427–1437.

[80] F. YazdaniBanafsheDaragh, *Deep-GUI: Towards Platform-Independent UI Input Generation with Deep Reinforcement Learning*. University of California, Irvine, 2020.

[81] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[82] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.