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