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

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

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 hyper-parameter 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 $\{f_0, f_1, .., f_{N-1}, f_N\}$, 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