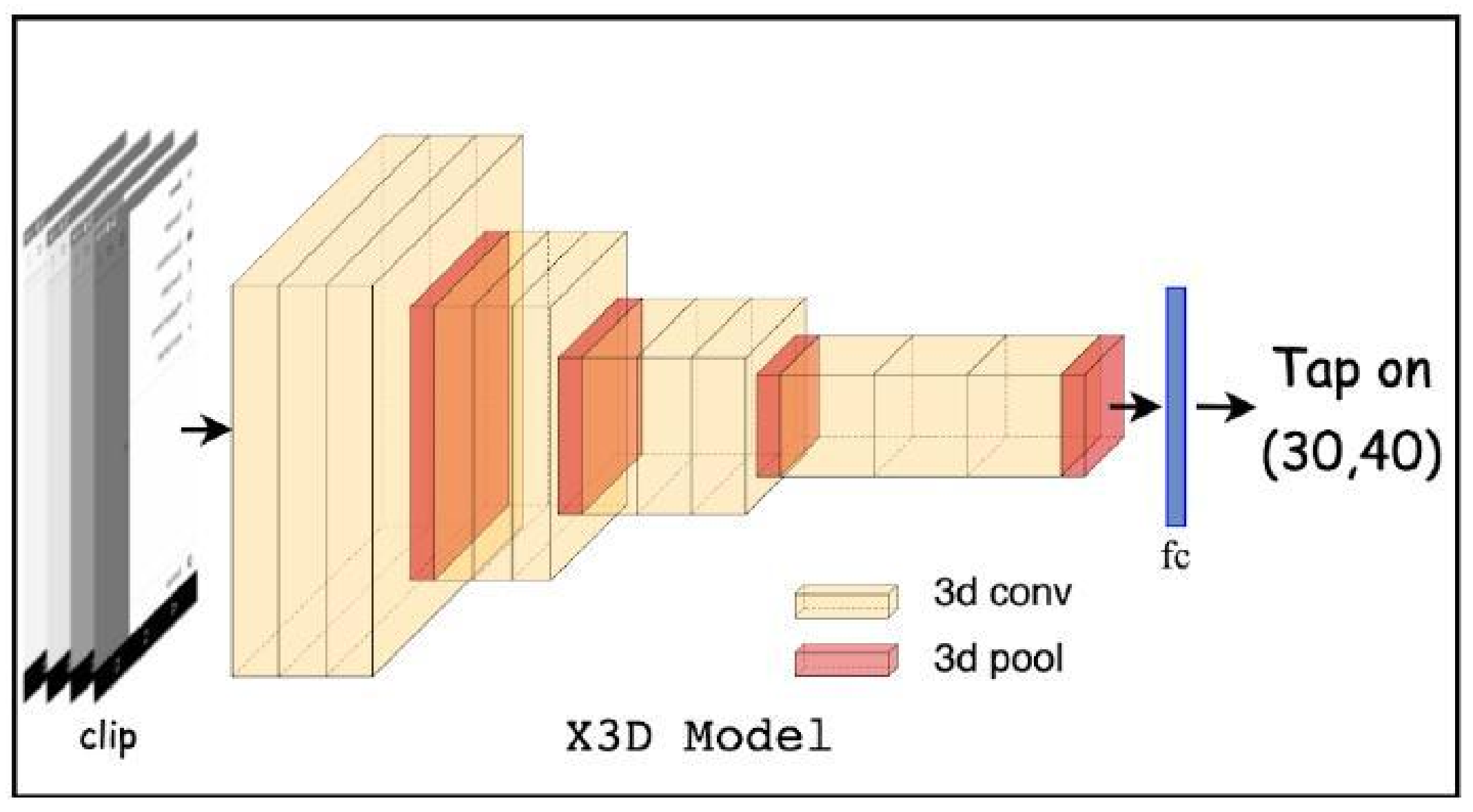
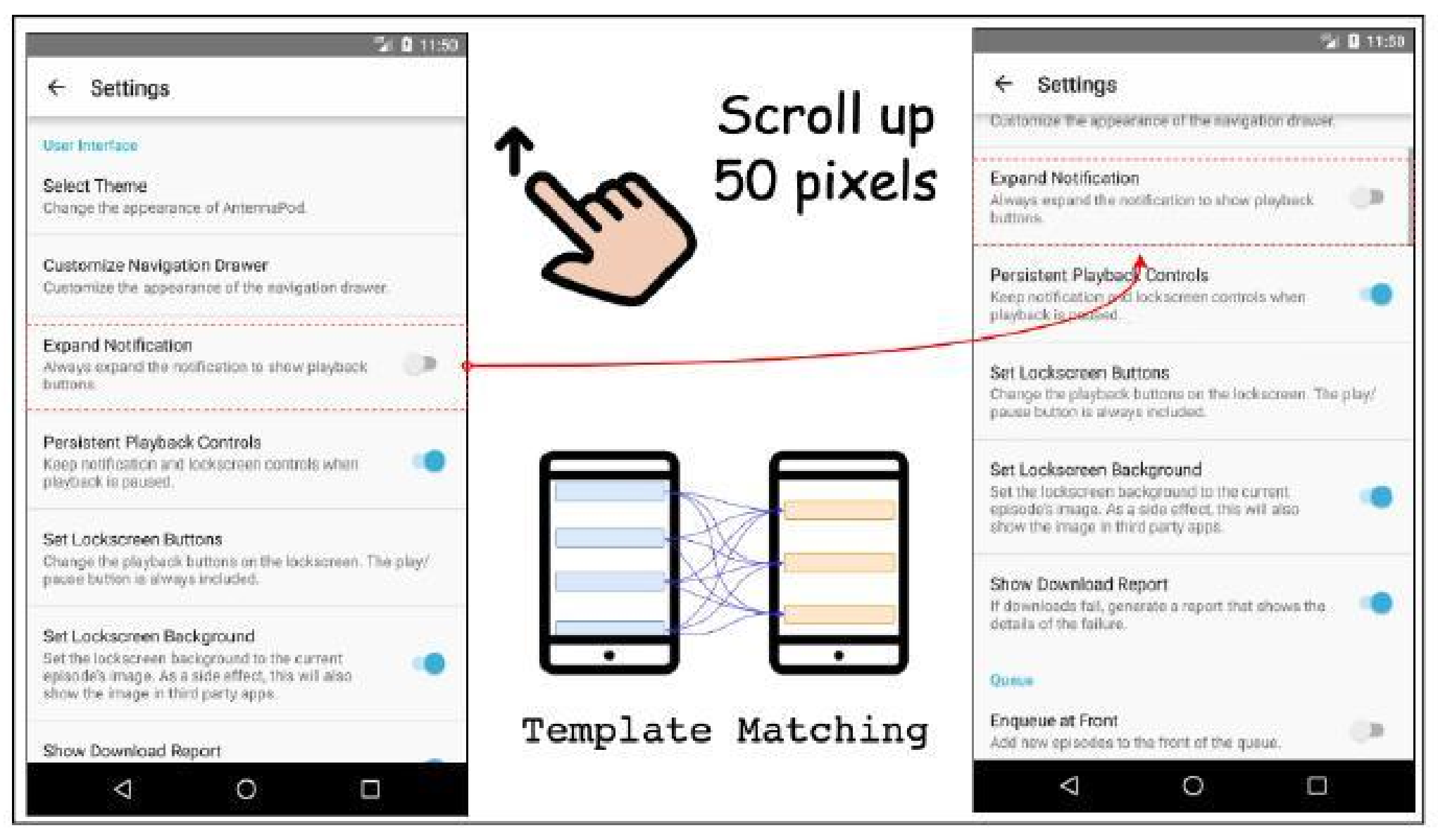
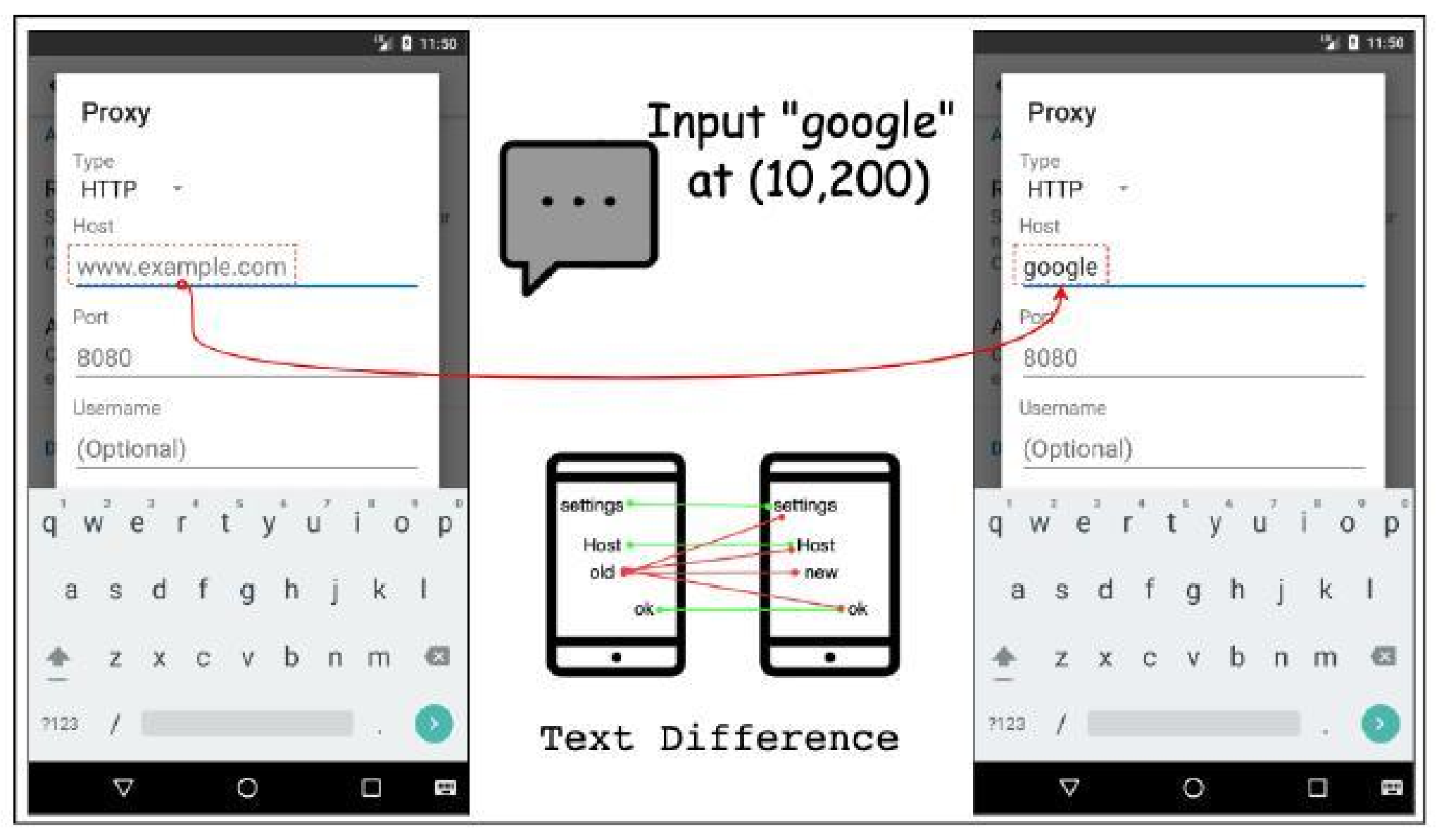


(a) TAP

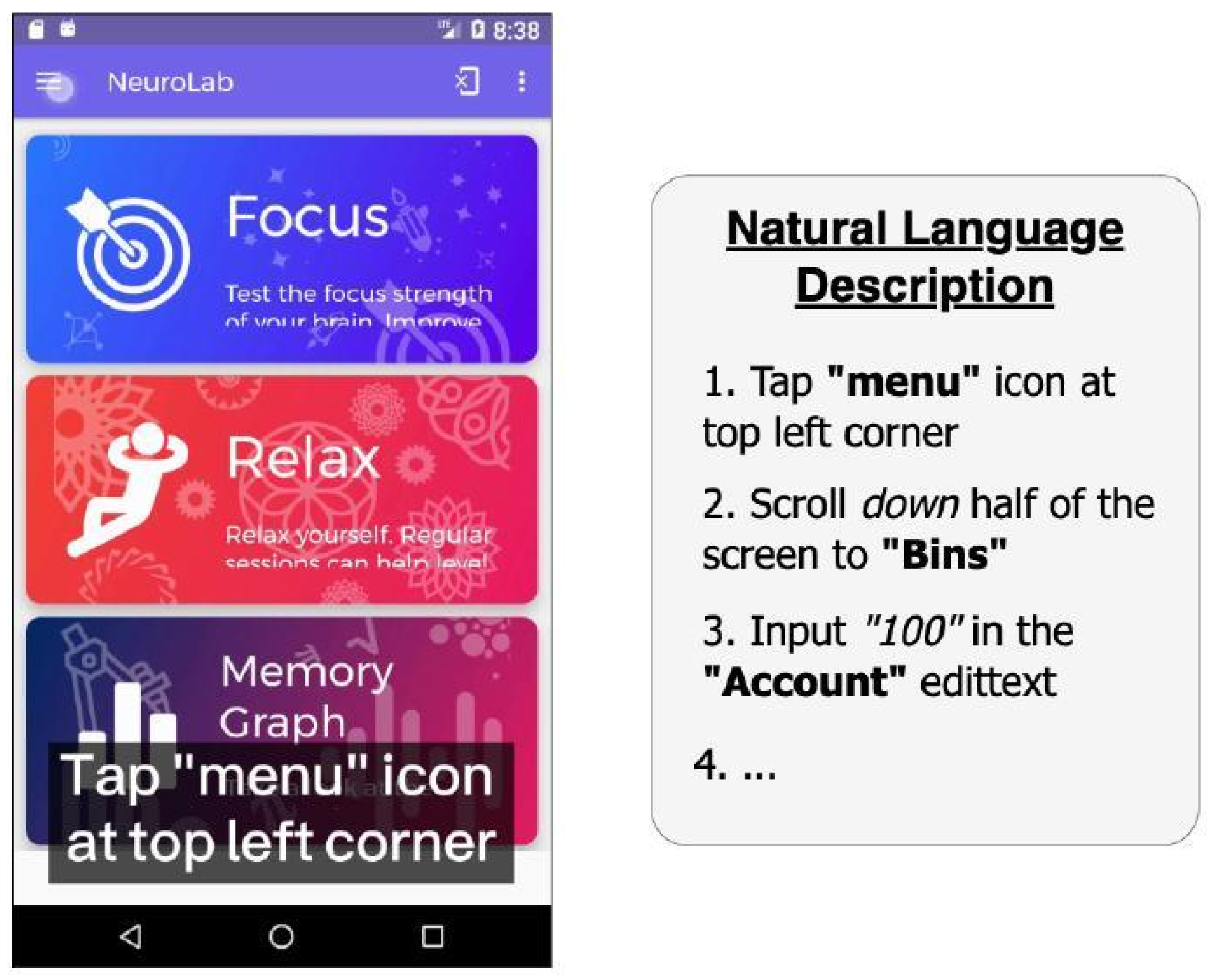


(b) SCROLL



(c) INPUT

Fig. 5: Approaches of Action Attribute Inference.



Natural Language Description

1. Tap **"menu"** icon at top left corner
2. Scroll *down* half of the screen to **"Bins"**
3. Input *"100"* in the **"Account"** edittext
4. ...

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*,