Received 4 May 2023, accepted 25 May 2023, date of publication 29 May 2023, date of current version 6 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3280985

RESEARCH ARTICLE

An Empirical Study on Multimodal Activity Clustering of Android Applications

SUNGMIN CHOI[1,2], HYEON-TAE SEO[3], AND YO-SUB HAN[1]
[1]Department of Computer Science, Yonsei University, Seoul 03722, Republic of Korea
[2]Software Center, CTO, LG Electronics, Seoul 050061, Republic of Korea
[3]KT Research and Development Center, Seoul 06763, Republic of Korea

Corresponding author: Yo-Sub Han (emmous@yonsei.ac.kr)

This research was supported by the IITP grant (No. 2021-0-00354) and the AI Graduate School Program (No. 2020-0-01361) funded by the Korea government (MSIT).

ABSTRACT Recently, researchers have started looking at Android activities to improve user interface (UI) design. Since similar activities in Android have similar functional behaviors, activity clustering is a fundamental step toward efficient Android app development. Well-grouped activities are useful not only for UI design, but also for app design, development, and testing. However, there are no studies on activity clustering yet, and no activity dataset with labels and categories. The purpose of this study is to use the Rico dataset to know (i) whether the Rico dataset can be used for activity clustering, (ii) how useful activity attributes expressed in XML are for activity clustering, and (iii) how useful fusion with activity image and attributes is for activity clustering. We generate various activity latent vectors using a CNN autoencoder for the Rico dataset. Then, we produce a sequence-to-sequence latent vector from the semantic properties of the Rico dataset. Finally, by fusing the two models, we propose an activity clustering approach using multimodal learning. Since there are no labels in the dataset, we make 2000 labeled data for evaluation. The experimental results show that the activity clustering works well by fusing the semantic activity latent vector and the seq2seq latent vector. Especially, activity attributes such as component and position information are effective for activity clustering and help to boost the performance better than real activity images or Rico. Research findings on clustering and newly created labeled data can be a starting point for various studies on Android activity.

INDEX TERMS Activity clustering, autoencoder, CNN, deep learning, sequence-to-sequence, Rico.

I. INTRODUCTION

Over the past decades, the smartphone market has grown tremendously. Notably, Android has grew to the largest market share with many applications developed on the mobile operating system. Then, several research areas have arisen as a result of innovation in the market. For instance, there are several research findings on graphical user interface (GUI) searches with respect to specific forms [1], [2], [3], fast GUI skeleton implementation [4], automatic analysis for apps [5], [6], bug and crash detection [7], [8], and automated testing [9], [10].

It is useful to know the function of each app screen. In the case of Android, an app screen is called an activity.[1] An activity is the main Android component that interacts with the user. Thus, one can analyze app activities better by examining user interface (UI) components. More importantly, similar activities often exhibit similar behaviors because similar activities are made up of similar UI components, which give rise to similar behaviors. This implies that if one finds a bug or malware during an automated test of a particular activity, then it is likely that the same bug or malware exists in other apps with similar activities [11], [12]. For instance, if a bug occurs in the *EditText* component, the activity using the *EditText* component in other apps can also fall victim to the bug.

The associate editor coordinating the review of this manuscript and approving it for publication was Alberto Cano.

[1]https://developer.android.com/guide/components/activities/intro-activities

Identifying similar activities makes debugging or designing specific activities easier. It is therefore beneficial to study the problem of grouping similar activities of Android apps.

The problem of identifying similar activities can be classified as a *clustering* problem. Clustering is the task of grouping a set of data, such that data in the same group share similar characteristics, in contrast to the data in other groups. Clustering is a class of unsupervised learning method that has been extensively applied and studied in computer vision [13], [14] as well as audiovisual [15] and multimodal [16], [17], [18] deep learning. Until now, clustering has been applied to well-known image and text datasets such as MNIST, STL-10, and CIFAR-10. However, almost no progress has been made on activity datasets.

There are a few recent studies for activity classification instead of clustering. Rosenfeld et al. [19] proposed activity classification using the KStar algorithm [20]. They predefined seven activity types and used 15 features to classify the activities. Amalfitano et al. [21] automatically detected specific GUIs labeled as Gate GUIs, such as the Login Gate GUI or the Network Settings Gate GUI. They used textual information of XML as feature vectors and classified gate GUIs using a Naïve Bayesian (NB) classifier.

Meanwhile, researchers realized that data-driven models are useful for creating adaptive UIs and predicting performance [22], [23]. Deka et al. [1] presented the Rico dataset consisting of mobile app designs for making better apps. They trained an autoencoder [24] for UI layout similarity, and executed a query-by-example search over UIs using the nearest neighbor. Liu et al. [2] expanded the Rico dataset, and generated semantic annotations for mobile app UIs by detecting UI components and training a convolutional neural network (CNN) [25] to distinguish between icon classes. They also trained an autoencoder for semantic screenshot images and executed a query-by-example search over UIs using a ball tree. The Rico dataset is the de facto standard for UI design and layout generation [26], [27], [28], [29] in Android apps. We notice that if the Rico data is clustered properly, then we can use this clustered data for malware detection [30], [31], [32], UI generation [4], [26], and automated testing [21], [33], [34], [35]. However, there are no prior studies on activity clustering for the Rico dataset. This leads us to study an effective clustering method for the Rico dataset, and we study the following research questions:

- **RQ1:** *Is it feasible to use the Rico dataset for activity clustering as image data?*
- **RQ2:** *How useful is the semantic annotation of the Rico dataset for activity clustering as text data?*
- **RQ3:** *How useful is the hybrid approach that uses both images and text of the Rico dataset for activity clustering?*

Based on the Rico dataset, we build an unsupervised activity clustering model. This is because often there are no labeled data available for app-related clustering. For fair evaluation, we manually make two types of labeled test datasets from the unlabeled Rico dataset. To the best of our knowledge, this is the first attempt to use multimodal learning and a sequence-to-sequence (seq2seq) autoencoder based on the attention mechanism for Android activity clustering. We also publish two new datasets [36]. Note that activity clustering is not simply the end goal but the entry point for new, innovative Android research from a software engineering perspective [37].

The remainder of this paper is organized as follows. Sections II and III provide the necessary backgrounds and related work, respectively. Section IV presents the experimental design, and Section V describes training, test, and validation datasets from the Rico dataset. Section VI explains our experiments and results. Then, Section VII discuss the threats to the validity of our study. Finally, in Section VIII, we conclude the paper with possible future studies based on our findings.

II. BACKGROUND

We construct an unsupervised activity clustering model which generates various latent vectors using a CNN autoencoder and seq2seq autoencoder. This section provides the necessary background and concepts related to activity clustering model.

A. ANDROID ACTIVITY

An Android activity is a single screen with a UI in an app. An activity interacts with users and has a lot of information including UI components, UI hierarchy and interactions with other activities. Android provides a variety of UI components and UI controls. These UI components can be used in analyzing activities because activities are typically made up of UI components, and UI information can be extracted from activity in XML using UIAutomator[2] or AndroidViewClient.[3] Android activity can be expressed in two formats: activity images and activity properties including UI information.

B. MULTIMODAL CLUSTERING

Clustering is a representative type of unsupervised learning. Even in the absence of a clear criterion for classifying objects, similar objects can be grouped together by considering the attributes or features of the given data. Multimodal clustering synchronously performs several clusterings with multimodal vectors in different shared spaces such as audiovisual clustering, image clustering, and video-caption clustering [15], [16], [17]. Multimodal fusion integrates information from multiple modalities, with the goal of predicting an outcome measure such as a class (through classification) or a continuous value (through regression) [16], [18]. Recently, there has been active research on fusion methods and clustering algorithms for multimodal clustering [38]. Multimodal clustering algorithms aim to find a way to integrate clustering results for each modality. To achieve this, multimodal clustering algorithms

[2]https://developer.android.com/training/testing/ui-automator
[3]https://github.com/dtmilano/AndroidViewClient

typically combine features extracted from each modality and cluster the entire data based on them.

C. AUTOENCODER

An autoencoder [24] is a type of neural network in which the output layer has the same dimensionality as the input layer. An autoencoder converts the input into a reduced representation, which is stored in the middle layer, and trains the representation to reconstruct the input. Here, the encoding part of the autoencoder seeks to encode important hidden features present in the input data, and the features of the input can be extracted during the process of reducing reconstruction errors.

1) Seq2seq AUTOENCODER

The seq2seq model is an effective model for generating sequential outputs from sequential inputs based on the encoder–decoder architecture, and it has gained popularity in neural machine translation [39], [40]. Both the encoder and decoder use recurrent neural networks (RNNs) such as long short-term memory networks (LSTM) [41] and gated recurrent units (GRU) [42], to handle sequential inputs of variable length. The encoder of a seq2seq model can transform the inputs of variable length to a fixed-size context vector by encoding the sequence information.

2) CNN AUTOENCODER

CNNs are effective in extracting features from images by maintaining the spatial information of the images [25]. A CNN contains a convolution layer that reflects the activation function after applying a filter to the input data, and a pooling layer to reduce the size of the output. This allows the image to be extracted as a feature with a reduced dimensionality. A convolutional autoencoder is an autoencoder that employs a CNN for the unsupervised learning of images [43]. Like most autoencoders, the convolutional autoencoder is composed of an encoder and decoder. The encoder has a convolution layer and a pooling layer to train the features of an image. The decoder contains a deconvolutional layer [44] to increase the size of the feature map in order to restore the image.

III. RELATED WORK

A. RICO DATASET

Deka et al. [1] constructed the Rico dataset, which is the largest repository of mobile app designs. The dataset consists of 10,811 user interaction traces and 72,219 UIs from 9,772 Android apps across 27 Google Play categories. For each app, Rico offers its own collection of UIs and individual user interaction traces. Each UI is annotated with a low-dimensional vector representation that encodes a layout based on the distribution of text and images. Liu et al. [2] generated semantic annotations for 72k unique UIs in the Rico dataset. They used code-based heuristics and structure-based patterns to identify different types of UI components and text button concepts. From the resulting data, they derived 25 types of UI components and 197 text button concepts. To classify icons, they trained a vision-based, deep-learning adapted CNN architecture. This trained architecture identified 99 icon classes.

Lee et al. [26] built a tool, GUIComp, to assist GUI prototyping based on the Rico dataset. The tool helps to resolve issues that can happen for users with little design experience by recommending example designs. Pandian et al. [28] used the Rico dataset to generate the UI layout hierarchy from the UI screen. They utilized Deep Neural Networks (DNNs) to identify UI elements, locations, and dimensions using the Rico dataset, and generated UI layout trees.

B. ACTIVITY CLASSIFICATION

Rosenfeld et al. [19] classified activities using machine learning. They identified seven activity types from 100 Android apps—Splash, Advertisement, Login, Portal, Mail, Browser, and To Do List—and constructed 15 feature vectors from clickable, horizontal, vertical, and text field elements. Their approach used pre-defined categories to label activities rather than grouping similar items. They checked the accuracy of activity-type prediction using several different classification models such as decision trees, k-nearest neighbors, logistic regression, random forests, multi-layer perceptron, and KStar. Their study showed that KStar yields the best result.

Amalfitano et al. [21] utilized a machine learning approach to determine the class to which a given GUI belongs. They defined a special class of GUI called the Gate GUI. Examples of Gate GUI include the Login GUI and the Network Settings GUI. Feature vectors are extracted from the XML GUI representation, provided by the UIAutomator. They hypothesized that the descriptions of GUIs of the same Gate GUI class are likely to share common textual information. Then they chose the most frequent terms among the GUIs belonging to the same Gate GUI class. They built a labeled dataset consisting of GUI descriptions belonging to 5000 real Android apps. Finally, they classified gate GUIs using a Naïve Bayesian classifier.

Ardito et al. [35] developed a testing framework based on app and activity classifiers. In their framework, the classifiers played a role in determining the type of application under test and the purpose of each screen. For the latter, they defined eight classes from 70 apps—Advertisement, Login, Portal, List, To Do, Browser, Map, and Messages—and obtained 100 labeled activities.

The previous studies [19], [21], [35] classified activity via supervised learning instead of clustering via unsupervised learning. They focused on specific goals such as app testing, and used datasets smaller than the Rico dataset.

IV. EXPERIMENTAL DESIGN

A. RESEARCH QUESTIONS

Our study aims to address the following research questions:

RQ1: Is it feasible to use the Rico dataset for activity clustering as image data? There are few studies on activity

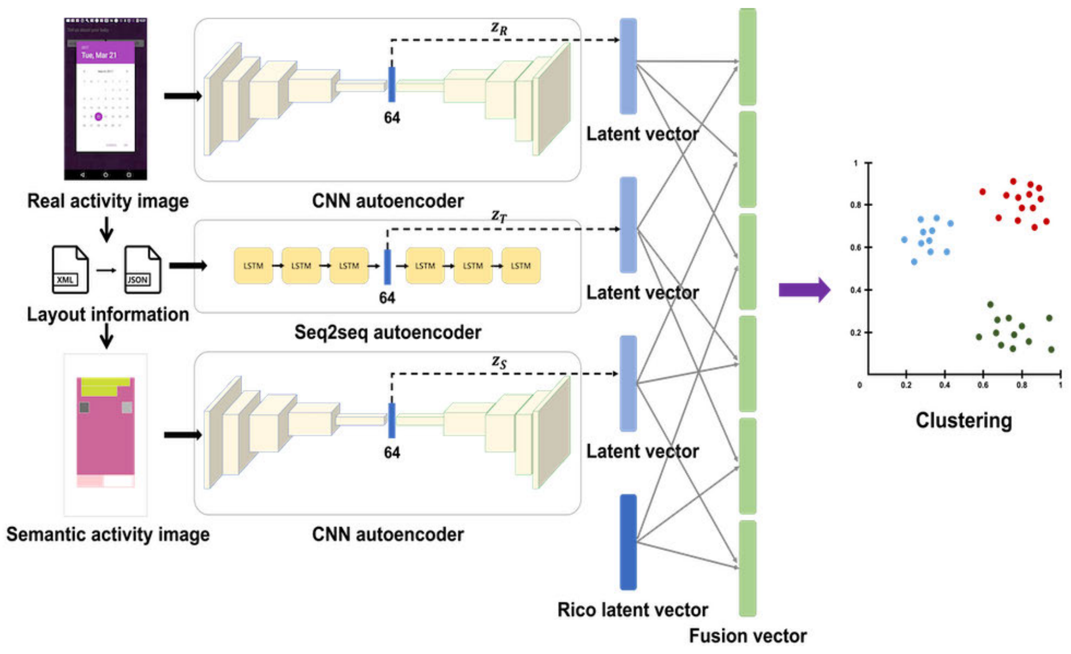


FIGURE 1. Illustration of the proposed clustering model. In the first stage, the proposed model trains the features of the layout information using the attention-based seq2seq autoencoder and the features of the activity image using the CNN autoencoder. In the second stage, we perform clustering by fusing the latent vectors of each autoencoder and the Rico latent vector, which is obtained from the Rico dataset itself.

clustering, so we apply image clustering techniques on activities. Since an activity can be represented as an image, we want to see how much performance is achieved by applying image clustering on the images. Then, by comparing the clustering results of well-known datasets and the Rico dataset, we check whether activity clustering using the Rico dataset is feasible.

RQ2: How useful is the semantic annotation of the Rico dataset for activity clustering as text data? This research question is based on the fact that an activity contains UI components. The UI properties of Android activities are expressed using XML in the layout format. We use the XML information of semantic annotations [2] for clustering. Unlike RQ1, which focuses on using images, RQ2 probes the effectiveness of using text in clustering.

RQ3: How useful is the hybrid approach that uses both images and text of the Rico dataset for activity clustering? The purpose of this research question is to find out the effectiveness of fusing the results of RQ1 and RQ2. This means that activities are represented by images and text, and RQ3 uses both to check the clustering result. With this research question, we want to explore how activity attributes such as UI information expressed in XML affect clustering.

B. CLUSTERING APPROACH

This section explains an overview of our model consisting of two stages as shown in Algorithm 1 and Figure 1. Figure 1 shows the proposed clustering model consisting of two stages. In the first stage, the proposed model is training the features of the layout information using an attention-based seq2seq autoencoder. At the same time, the model is also training the features of the activity image using a CNN autoencoder. The outputs of the encoder are latent vectors that capture the most important features of the layout information and the activity images, respectively. In the second stage, the proposed model performs clustering by fusing the latent vectors of each autoencoder and the Rico latent vector. By fusing the latent vectors of the layout information, activity images, and Rico, the proposed model can be clustered based on their features.

Algorithm 1 Activity Clustering Process

```
Input: Real activity images: $R$, Semantic activity images: $S$,
    UI component information: $T$, The number of clusters:
    $K$; Maximum iterations: $MaxIter$
Output: Clustering results
 1: Extract training and test data from Rico dataset
 2: $encodedText \leftarrow encode(T)$
 3: $iter \leftarrow 0$
 4: while $iter \neq MaxIter$ do
 5:     $z_R, z_S \leftarrow$ CNN autoencoder for $R$ and $S$
 6:     $z_T \leftarrow$ seq2seq autoencoder for $encodedText$
 7:     $iter \leftarrow iter + 1$
 8: end while
 9: Select $z_R$, $z_S$, and $z_T$ with the lowest loss
10: Fusion with $z_R$, $z_S$, and $z_T$
11: K-Means and GMM with $K$
```

The Rico dataset provides unlabeled activity data and contains semantic annotations consisting of images and JavaScript object notation (JSON) files [1], [2]. The Rico

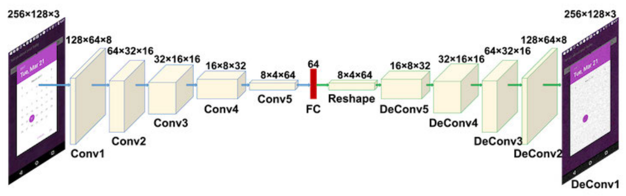


FIGURE 2. Illustration of a CNN autoencoder. The model is designed to extract features of reduced dimension by reconstructing real and semantic activity images.

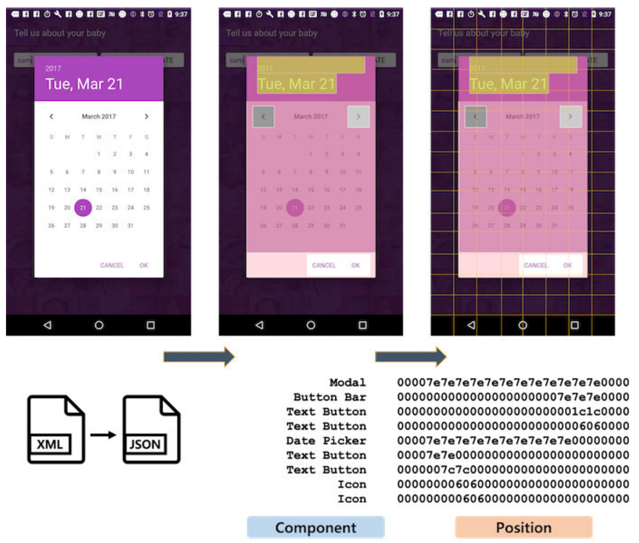
dataset also provides 64-dimensional pre-trained vectors obtained via a deep autoencoder for each activity. We denote the pre-trained vectors as Rico.

Algorithm 1 describes the process of the proposed clustering model. The inputs to the model include Android activity images, layout information, the number of clusters, and the maximum number of iterations. After processing the proposed clustering model, the clustering results can be obtained. A CNN autoencoder is used to extract features for real activity images and semantic activity images, provided by the Rico dataset (line 5). A seq2seq autoencoder is also used to extract the latent vector from the JSON files (line 6), which includes layout information with components and the corresponding position information (line 2). Once the maximum number of iterations is completed, we select the real activity latent vector, semantic activity latent vector, and seq2seq latent vector with the lowest loss (line 9). Then, each latent vector is fused (line 10), and the resulting fused vector is used for clustering (line 11).

In general, the CNN autoencoder shows a higher performance in image training compared to seq2seq, but the training speed is very slow. On the other hand, the seq2seq autoencoder can be trained faster than the CNN autoencoder. However, if the sequences are of variable length, the distances in vector space increase. This implies that if the number of components in similar activities is different, then similar activities would be classified as different—a potential flaw using the seq2seq autoenconder in the clustering of the activities. Thus, the latent vectors, which are output from each autoencoder, are fused together to overcome the shortcomings of the CNN autoenconder and the seq2seq autoencoder. Then, these fused latent vectors are used as input data for clustering.

C. IMAGE CNN AUTOENCODER

A CNN autoencoder shown in Figure 2 is used to extract features for real activity images and semantic activity images provided by the Rico dataset. Using the image as input, we reconstituted the original image via deconvolution, having reduced the image to a fixed sized, lower dimension via convolution and pooling. Therefore, the input and output are the same dimension. For better training, we follow the process of Ciregan et al. [45]; we resize input images to 256 × 128 to reduce the memory consumption and increase the batch size in training.



```
      Modal    00007e7e7e7e7e7e7e7e7e7e7e7e0000
 Button Bar    00000000000000000000007e7e7e0000
Text Button    0000000000000000000000001c1c0000
Text Button    00000000000000000000000060600000
Date Picker    00007e7e7e7e7e7e7e7e7e7e00000000
Text Button    00007e7e000000000000000000000000
Text Button    0000007c7c0000000000000000000000
       Icon    00000006060000000000000000000000
       Icon    00000000606000000000000000000000
```

FIGURE 3. Illustration of the partitioning of the activity component. To express position information, the total boundary of the activity is divided into 128 cells. If a component is included in the corresponding cell, it is represented in hexadecimal form.

D. ACTIVITY COMPONENT PARTITIONING

The JSON files that include layout information are reused for activity component partitioning. Android activities have XML layout information, which is converted to JSON to be annotated semantically in the Rico dataset. As a result, the Rico JSON files contain activity information, such as class, ancestors, resource-id, text, bounds, and components. Each activity is expressed using components and bounds, which are organized in a hierarchical structure. We used 25 UI components defined by Liu et al. [2] to express activity: Advertisement, Background Image, Bottom Navigation, Button Bar, Card, Checkbox, Date Picker, Drawer, Icon, Image, Input, List Item, Map View, Modal, Multi-Tab, Number Stepper, On/Off Switch, Pager Indicator, Radio Button, Slider, Text, Text Button, Toolbar, Video, and Web View. Components are represented in a hierarchical order, as depicted in Figure 3, to generate input sequence data for an activity.

As each activity component varies according to the bounds given to each component, the partition embedding which encodes the position information of each boundary in the sequence can be utilized for component partitioning. To express the position of the component on the screen, the total boundary of the activity was divided into 128 cells, as shown in Figure 3. If the number of cells is less than 128, the size of one cell will be large enough to contain many components, such that the cell cannot properly represent the relationship between the position and the components. In addition, if the number of cells is greater than 128, the size of one cell will be too small, such that the one cell does not contain a component and the length of the vector increases unnecessarily.

When a component is included in its corresponding cell, the cell displays 1, otherwise the cell displays 0. However,

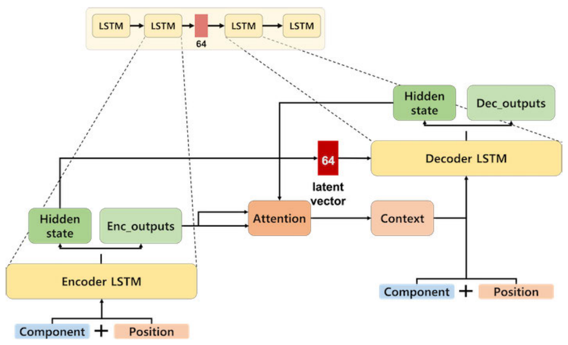


FIGURE 4. Seq2seq autoencoder based on attention mechanism. The component and position information are simultaneously reconstructed. The attention mechanism of the proposed seq2seq autoencoder only utilizes the hidden states of the top RNN layers in both the encoder and decoder.

when the subsequent length of the position (128 digits) is too long, the autoencoder performance becomes worse because the input is too long. Therefore, the length of the position input is reduced to 32 digits by converting from binary to hexadecimal.

Suppose that $X = (x_1, x_2, \ldots, x_m)$ is a component input sequence of length m. The sequence is first embedded into $e_X = (e_{x_1}, e_{x_2}, \ldots, e_{x_m})$. The position of the input symbols $P = (p_1, p_2, \ldots, p_m)$ is also embedded into $e_P = (e_{p_1}, e_{p_2}, \ldots, e_{p_m})$. The final representation of the input sequence is generated by combining the two representations as follows:

$$e_X \mathbin{+\!\!+} e_P = (e_{x_1} \mathbin{+\!\!+} e_{p_1}, e_{x_2} \mathbin{+\!\!+} e_{p_2}, \ldots, e_{x_m} \mathbin{+\!\!+} e_{p_m}), \quad (1)$$

where $[\mathbin{+\!\!+}]$ denotes the concatenation operator. For instance, as shown in Figure 3, the boundary of the *Modal*, which is the component in the top layer of the activity, encompasses most of the partitions. x_1 is *Modal* and p_1 is *00007e7e7e7e7e7e7e7e7e7e7e0000* in Figure 3. Therefore, $e_{x_1} \mathbin{+\!\!+} e_{p_1}$ contains the information of *Modal* and *00007e7e7e7e7e7e7e7e7e7e7e0000*.

E. Seq2seq AUTOENCODER BASED ON ATTENTION MECHANISM

The input data for the seq2seq autoencoder is the sequence data. To extract features from activities using an autoencoder, an autoencoder that takes an input sequence and produces the reconstructed input sequence (the purpose of the seq2seq autoencoder) needs to be constructed.

Given an input sequence, the encoder generates a fixed-size latent vector that represents the sequence. The decoder of the seq2seq autoencoder utilizes another LSTM to reconstruct the input sequence from the encoder output. As a result, the inputs and outputs have the same dimension. However, as the length of the input sequence increases, it becomes very difficult for the decoder to generate an accurate input sequence using a fixed-size latent representation vector. To deal with this problem, when generating sequences in the decoder, an attention mechanism is applied to train the weights that determine which components are important in the input sequence [46], [47]. In particular, it only uses the hidden states of the top RNN layers in both the encoder and decoder [46]. After computing the hidden state of the decoder at each timestep, the attention matrix is computed. Therefore, when the sequence is generated through the attention mechanism, the information of the input sequence can be continuously used, thereby improving the output string generation accuracy for a long input sequence.

As shown in Figure 4, the model is trained to reconstruct component and position information simultaneously by using the seq2seq autoencoder based on the attention mechanism. By using the attention mechanism to generate input sequences with both component and position information, the model is able to train the weights that determine the importance of each component in the input sequence. This leads to improved output sequences generated by the decoder.

F. CLUSTERING

Various input modalities are combined for clustering of Android activities. Examples include latent vectors for Rico and seq2seq, Rico and real activity, Rico and semantic activity, seq2seq and real activity, seq2seq and semantic activity, and real and semantic activity as shown in Figure 1.

1) WEIGHTED FUSION FUNCTION

In order to fuse various latent vectors, we use the sum and concatenation functions widely utilized in multimodal clustering [16]. And two latent vectors are fused by applying weights to each latent vector. The weights are w and $(1 - w)$, with $0 < w < 1$, to a precision of one decimal place.

Suppose $e_I = (e_{I_1}, e_{I_2}, \ldots, e_{I_m})$ is a latent vector, and $e_S = (e_{S_1}, e_{S_2}, \ldots, e_{S_m})$ is another latent vector with the same dimension as e_I:

- The weighted sum function computes the elementwise sum of the feature maps as follows:

$$\begin{aligned} sum(e_I, e_S) = (&w \cdot e_{I_1} + (1 - w) \cdot e_{S_1}, \\ &w \cdot e_{I_2} + (1 - w) \cdot e_{S_2}, \\ &\ldots, w \cdot e_{I_m} + (1 - w) \cdot e_{S_m}). \end{aligned} \quad (2)$$

- The weighted concatenation function constructs the output by concatenating the input feature maps as follows:

$$\begin{aligned} cat(e_I, e_S) = (&w \cdot e_{I_1} \mathbin{+\!\!+} (1 - w) \cdot e_{S_1}, \\ &w \cdot e_{I_2} \mathbin{+\!\!+} (1 - w) \cdot e_{S_2}, \\ &\ldots, w \cdot e_{I_m} \mathbin{+\!\!+} (1 - w) \cdot e_{S_m}), \end{aligned} \quad (3)$$

where $[\mathbin{+\!\!+}]$ denotes the concatenation operator.

2) CLUSTERING ALGORITHM

The experiments are conducted with two classic clustering approaches to determine which approach is the more effective clustering algorithm for clustering activities. K-Means clustering is a distance-based algorithm that identifies K centroids, before allocating each data point to the nearest cluster, while minimizing the sum of distances between the points and the centroid [48]. Gaussian mixture models (GMMs) assume that there are a fixed number of Gaussian distributions, and that each of these distributions represents a cluster [49]. Hence, a GMM tends to group the data points belonging to a single distribution. This approach differs from K-Means clustering since the GMMs explain the variance and return the probability of a data point to fall into each of the clusters.

G. METRICS

We evaluate the performance of our proposed model using three metrics: purity, normalized mutual information (NMI), and adjusted rand index (ARI), which are widely used as clustering evaluation metrics [16], [48].

Purity is the rate of the number of objects that were correctly classified in the range [0, 1]. To compute purity, each cluster is paired up with the class that overlaps the most. NMI is a clustering validation metric which estimates the quality of clustering. NMI measures the mutual information between the correct and predicted labels and is normalized to scale the results between 0 (no mutual information) and 1 (perfect correlation). The rand index (RI) calculates the degree of similarity between two clusters. This measure counts the number of pairs assigned to the same or different clusters for the predicted and actual clusters, respectively. ARI negatively adjusts the result when two objects with the same label are placed in different clusters. ARI has a value close to 0 for random labeling, regardless of the number of clusters and samples. On the other hands, ARI is guaranteed to have a value of exactly 1 when the clusters are equal.

V. RICO DATASET

The Rico dataset is the largest public repository of mobile app designs to date. It contains 72k UI screenshots with annotations about the UI elements in both textual and visual form. Training and evaluation datasets were generated based on the Rico dataset, which provides unlabeled activity data and contains semantic annotations consisting of images and JSON files.

A. TRAINING DATASET

In the 66,261-item dataset, a total of 65,538 data points were used, excluding data without component labels in the JSON file. In the Rico dataset, the layout information in the Android XML format was converted to JSON by adding semantic annotations and including components and bounds. In the case of images, we use the real activity images and semantic activity images created through the converted JSON. For text, if components consist of one or more words, we generate an input sequence for the activity by combining the words of the component. For instance, the *List Item* component is converted to *ListItem*. Position information are generated including the boundary of a component paired with each component of the generated input sequence. The vocabulary size of the input sequence is 25 (which is the same as the number of UI components defined by Liu et al. [2]) and the vocabulary size of the position information is 3,311. The 2000 data elements created for the test and validation datasets were removed and the rest of the data was used as training data. In total, there were 63,538 elements in the training dataset.

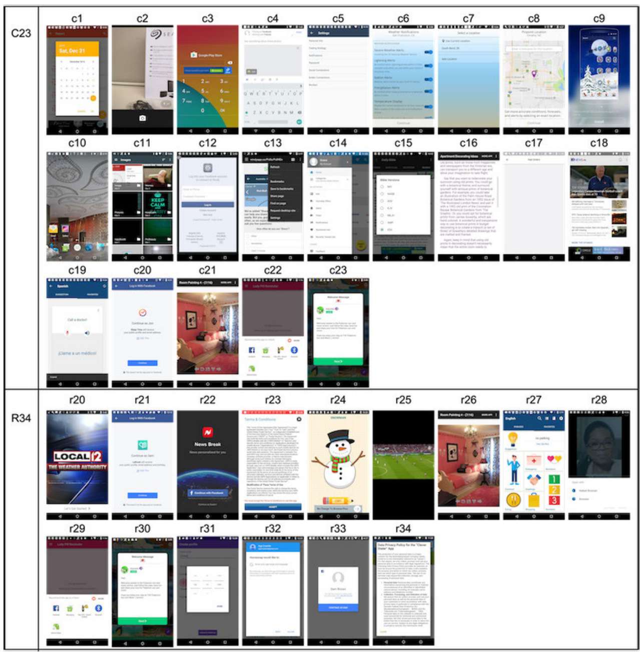


FIGURE 5. Test dataset, showing the categories and sample images. c1–c19 are the same as r1–r19.

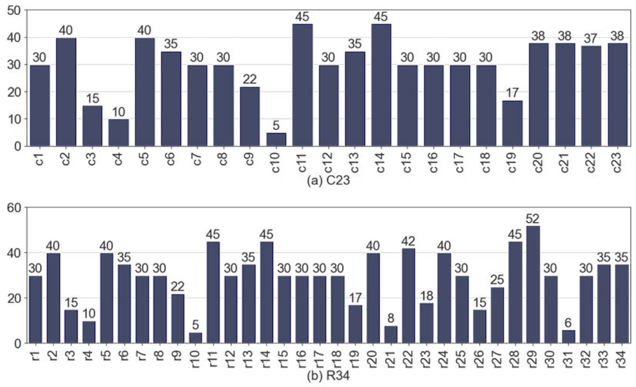


FIGURE 6. The distribution of the number of test images over the C23 and R34.

B. TEST AND VALIDATION DATASET

Two types of labeled datasets were constructed from the Rico dataset and manually categorized for evaluation. Because an activity contains too much information for untrained humans to accurately distinguish an activity, two experts examined: a senior engineer with more than 13 years of Android development experience and a junior engineer with 5 years of Android development experience, both of whom developed Android products in the company. We have classified 2000 images from 10,000 randomly-selected images, based on common patterns, structures, and behaviors [2], [19].

TABLE 1. Each category description for C23 and R34. C and R stand for category and revision, respectively. The category order has no special meaning.

Label	Description
c1	c1 displays a type of calendar, including the date picker component.
c2	c2 shows a type of camera. This type is similar to the image type (c21), but this type includes a camera icon.
c3	c3 denotes a type of dialer with 12 grids containing numbers.
c4	c4 is an input method editor (IME) type with an input function.
c5	c5 presents a list type containing a list item component.
c6	c6 is a list type with an on and off switch that does not contain a list item component. The Rico dataset contains specific screens such as c6 and c19, thus, these screens have been categorized separately.
c7	c7 does not include a list item component. In addition, c7 contains the location icon on the first line.
c8	c8 displays a type of pinpoint location on a map.
c9	c9 shows a type of theme containing a page indicator component.
c10	c10 is a homescreen with app icons.
c11	c11 is an image list type with grid.
c12	c12 presents a type of input form with an input component. This type has an editable text field and must receive input to function. This type includes the login activity.
c13	c13 is similar to c5 in that it has a list of item components. The difference is that c13 is the menu list type, and c5 is displayed on the full screen.
c14	c14 appears to be similar to that of c13; the main difference between c13 and c14 is the location of the menu list. c13 has a menu list in the upper right, and c14 has a navigation menu in the upper left of the screen.
c15	c15 is the selection dialog type. This type looks like a popup, except that we have to choose it.
c16	Depending on the components included in the activity, we divided the text-related categories into three categories: c16, c17, and c18. c16 is a type in which the screen consists of text only.
c17	c17 is a type consisting of screens with little or no text.
c18	c18 is a type consisting of a screen with text and images.
c19	c19 contains a specific screen that is difficult to classify as c18 and c21.
c20	c20 is a combination of text, images, and buttons. This category type is novel, in that the click of a button will launch a new intent. c20 is divided into r20, r21, r22, and r23.
c21	c21 provides a type that contains almost any image. c21 is divided into r24, r25, r26, and r27.
c22	c22 displays a type of "open and shared with". c22 is divided into r28 and r29.
c23	c23 represents any type of popup. c23 is divided into r30, r31, r32, r33, and r34.
r20	r20 is an advertisement type with a button to move to another activity at the bottom of the screen.
r21	r21 is a type of continued name. This type contains login activity without an input form.
r22	r22 is a combination of text, images, and buttons, excluding r20 and r21.
r23	r23 is a combination of text and buttons, except for r21. r23 looks like c16 or c18, but unlike c16 and c18, r23 has a button to launch a new intent.
r24	r24 is a typical image view type.
r25	r25 is a type of film.
r26	r26 is a snapshot.
r27	r27 is a specific type of the Rico that is difficult to include in r24.
r28	r28 denotes the "open with".
r29	r29 shows the "share with".
r30	r30 is a typical popup type.
r31	r31 is a popup with a date picker and number of stepper components.
r32	r32 has special purposes (automatically login), such as allow, access, and deny.
r33	r33 is the popup version of r21.
r34	r34 is a popup type with size of almost full screen.

TABLE 2. Number of datasets by category out of 2000 labeled data. C23 is included in R34.

Category	# of validation dataset	# of test dataset
C23	700	700
R34	1000	1000

Figure 5 shows the categories and the sample images. Table 1 shows the description and characteristics of each category classified by the criteria. The criteria are as follows. i) Activities were classified according to the characteristics of the UI components occupying the screen (c1–c12, c19, c21, c22), and ii) classified according to the ratio of the UI components to the screen (c13–c18). iii) We classified activities by inferring the user's purpose or intention for the screen through the UI components of the screen (c20, c23). The test and validation datasets were categorized into 23 categories. Then, another five researchers with at least 2 years of Android development experience and a computer science background reviewed the results, some categories were further broken down by location and size of UI components; for example, c20 is divided into r20–r23; c21 is divided into r24–r27; c22 is divided into r28, r29; and c23 is divided into r30–r34. Finally, the test and validation datasets were categorized as C23 and R34, where C and R stand for category and revision, and 23 and 34 denote the number of category, respectively.

As shown in Table 2, C23 consists of 700 validation and test datasets each, and R34 consists of 1000 validation and test datasets each. Figure 6(a) and Figure 6(b) show the distribution of the number of images for C23 and R34 test datasets. Each category consists of 5 to 52 images. Both C23 and R34 test datasets have an average of about 30 images in each category. c11 and c14 in C23, and r29 in R34 have the greatest number of images and c10 (r10) has the least, i.e. 5 per category.

VI. EXPERIMENT RESULTS

The machine we used to train and test the model is a workstation with Intel Xeon W-2145 CPU, 32GB RAM, and an NVIDIA TITAN Xp 12GB GPU. The operating system of the machine was Ubuntu 18.04. The model was implemented with PyTorch [50] using the Adam [51] optimizer, with a learning rate of 10^{-3} without decay. The software used by the model is listed in [36].

a: Seq2seq AUTOENCODER

The size of the hidden unit was set to 64 by considering the 64-dimensional pre-trained vectors of the semantically annotated images provided by Rico. The number of layers of the encoder and decoder was set to 1. The unidirectional LSTM is used, where the size of the component input sequence embedding is 14. On the other hand, the position information has a fixed length and a larger vocabulary size, so the position embedding size was set to 50. The model was trained over 60 epochs with a batch size of 32, with each epoch lasting approximately 1 min. The loss converged at 50 epochs.

b: CNN AUTOENCODER

The encoder consists of five convolutional layers with dimensions of 8×3, 16×3, 16×3, 32×3, and 64×3 (channel × kernel size) and one size of padding. A max pooling layer of size and stride 2 is applied after every convolution layer. Finally, to express the data as a latent vector of 64 dimensions, a fully connected (FC) layer is added and encoded in 64 dimensions. The decoder consists of the same layers as the encoder, but in reverse order, with deconvolution layers converting the latent vector back to the original input data. The model was trained over 100 epochs with a batch size of 32.

We first trained the seq2seq model and the convolution model. The latent vector provided by the Rico dataset and the latent vectors of each trained model are fused by the sum and concatenation methods, as described in Section IV-F. Additionally, a weight between 0.1 and 0.9 to one decimal place (inclusive) is given to fuse the two latent vectors. We have repeated the whole experiment 30 times and computed the average score.

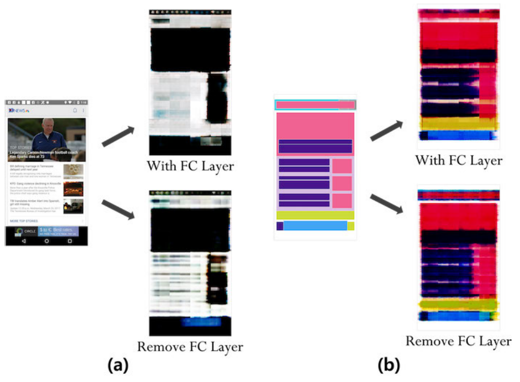


FIGURE 7. Performance of the CNN autoencoder for different types of images: (a) a real activity image and (b) a semantic activity image, used to evaluate the performance of unsupervised feature learning. The CNN autoencoder model without an FC layer shows similar performance when processing a real or semantic activity image. However, when applying an FC layer, the real activity image displays the worst result.

A. RQ1: IS IT FEASIBLE TO USE THE RICO DATASET FOR ACTIVITY CLUSTERING AS IMAGE DATA?

To answer RQ1, we conduct experiments for image clustering. We compare the clustering results of the well-known datasets (MNIST,[4] STL-10,[5] and CIFAR-10[6]) and the Rico dataset. This allows us to indirectly check if activity clustering is possible using the Rico dataset. The summary of the well-known datasets is in Table 4.

Table 3 shows K-Means and GMM of purity, NMI, and ARI for each dataset. A CNN autoencoder as mentioned in Section IV-C is used to extract features from datasets. The MNIST dataset shows the best results. The Rico dataset outperforms STL-10 and CIFAR-10. Since STL-10 and CIFAR-10 are already used as representative datasets for clustering, it is reasonable to use Rico dataset as a dataset for clustering as well.

The real activity image shows worse results than the semantic activity image in Table 3. Figure 7 shows the performance of the proposed CNN autoencoder for each type of image. In both the real and semantic activity images, better learning results are achieved without the FC layer in the model, which is added to flatten the output of the last convolutional layer to 64 dimensions. By the way, applying the FC layer to the model yields slightly different results. In particular, the real activity images show worse results than the semantic activity images. These results show that it is difficult to train the neural network using real activity images. Therefore, it is very hard to train a very low-size feature map for clustering using only activity images.

Summary for RQ1. By comparing the clustering results of the well-known datasets and the Rico dataset, it was checked that activity clustering is possible using the Rico dataset. It makes sense to use Rico, which shows better results than

[4]http://yann.lecun.com/exdb/mnist/
[5]https://cs.stanford.edu/ acoates/stl10/
[6]https://www.cs.toronto.edu/ kriz/cifar.html

TABLE 3. Clustering results (%). Rico is a latent vector provided by the Rico dataset itself, generated from a dense layer. Real activity and semantic activity images are provided by the Rico dataset, and each latent vector is generated by a CNN autoencoder. C23 and R34 are the labeled datasets for the tests described in Section V-B.

	Purity		NMI		ARI	
Data	K-Means	GMM	K-Means	GMM	K-Means	GMM
MNIST	50.7	**61.3**	53.4	**74.0**	42.7	**62.3**
STL-10	22.1	22.4	14.3	17.2	9.1	11.1
CIFAR-10	20.5	20.0	13.2	13.2	7.9	7.8
Rico-C23	34.3	34.6	53.5	54.3	21.6	25.9
Real activity-C23	25.1	23.0	46.3	45.9	16.8	16.7
Semantic activity-C23	40.0	38.7	59.9	58.7	32.9	31.2
Rico-R34	39.3	43.4	58.7	56.0	27.8	27.2
Real activity-R34	30.9	30.3	52.5	52.1	21.3	21.0
Semantic activity-R34	**43.9**	42.8	**63.6**	62.8	**34.4**	33.5

TABLE 4. Dataset used in the experiments.

Dataset	# Examples	# Classes
MNIST	70,000	10
STL-10	13,000	10
CIFAR-10	60,000	10
RICO	66,261	23 & 34

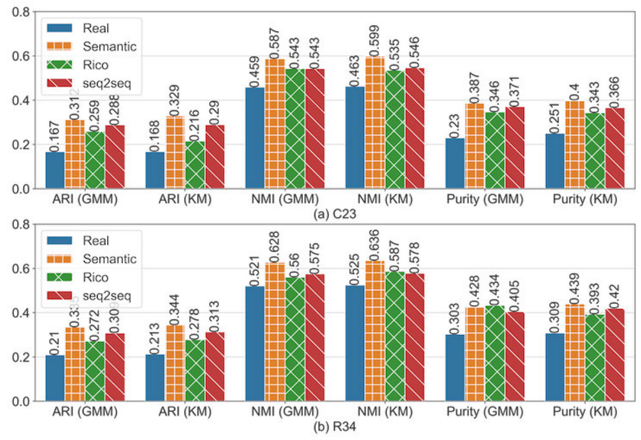


FIGURE 8. Clustering results based on four different types: real activity image, semantic activity image, Rico, and seq2seq. These clustering results are recorded for different clustering methods, for the different dataset categories (C23 and R34), and according to different evaluation metrics. From best to worst performance, the images can be ranked as follows: semantic activity image, seq2seq, Rico, and real activity image.

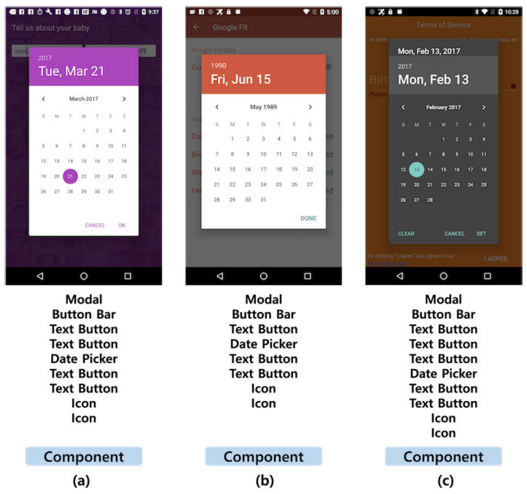


FIGURE 9. Example of a limitation of the seq2seq autoencoder. It looks like (a), (b), and (c) are all contained in the c1 cluster. However, because the components that make up each activity are different, the length of the entire encoded sequence is different for (a)–(c). As a result of these varying sequence lengths, the seq2seq autoencoder may incorrectly classify the activities into different clusters.

STL-10 and CIFAR-10, which are already used as a representative dataset for clustering [13], [52], [53]. As a result, it was found that activity clustering using the Rico dataset is feasible.

B. RQ2: HOW USEFUL IS THE SEMANTIC ANNOTATION OF THE RICO DATASET FOR ACTIVITY CLUSTERING AS TEXT DATA?

We leverage XML information in semantic annotations [2] to determine how activity attributes affect clustering. As mentioned in Section IV-D and IV-E, the XML information is transformed into component and position information, which is then used as an input to the seq2seq autoencoder.

As shown in Figure 8(a) and Figure 8(b), the clustering performance using the seq2seq latent vector is second only to the semantic activity latent vector. The clustering performance of the seq2seq latent vector was expected to be equal to or better than the Rico latent vector because inputs for seq2seq contain both the components and their spatial information. In general, seq2seq performed better than Rico. For instance, the ARI score of seq2seq using K-Means clustering for C23 is 7.4% points higher than Rico, and 3.7% points higher for GMM clustering on R34. The data in the Rico do not include the actual component; the image is simply divided into text and nontext. Therefore, it is possible for activities that are not actually similar to be classified into the same class. Because seq2seq autoencoder considers component and hierarchical structure order as well as the positional information of a component, it clusters similar activities more accurately.

However, in specific cases (*c1 issue*), Rico displayed better results than seq2seq. This is because the distance tends to

TABLE 5. Top five ARI, NMI, and Purity scores for the C23 and R34. * is the result of single modals. The Fusion column is expressed in the form "fusion_method (clustering_algorithm weight)". $\mathcal{G}$ and $\mathcal{K}$ stand for GMM and K-Means. R, S, re, and se represent Rico, seq2seq, real activity and semantic activity, respectively. $\mathbin{+\!\!\!+}$ and $+$ mean concatenation and sum functions.

	C23					
	Fusion	ARI	Fusion	NMI	Fusion	Purity
1	R $\mathbin{+\!\!\!+}$ S ($\mathcal{G}$ 0.7)	0.357	S + se ($\mathcal{G}$ 0.9)	0.604	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.7)	0.433
2	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.6)	0.346	S + se ($\mathcal{K}$ 0.7)	0.603	R + S ($\mathcal{K}$ 0.7)	0.431
3	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.7)	0.345	S + se ($\mathcal{K}$ 0.9)	0.603	re + se ($\mathcal{G}$ 0.1)	0.429
4	R + S ($\mathcal{G}$ 0.7)	0.344	S $\mathbin{+\!\!\!+}$ se ($\mathcal{G}$ 0.9)	0.602	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.6)	0.424
5	R + S ($\mathcal{K}$ 0.7)	0.342	R + se ($\mathcal{G}$ 0.8)	0.602	S $\mathbin{+\!\!\!+}$ se ($\mathcal{G}$ 0.8)	0.420
*	R ($\mathcal{G}$)	0.259	R ($\mathcal{G}$)	0.543	R ($\mathcal{G}$)	0.346
*	S ($\mathcal{K}$)	0.290	S ($\mathcal{K}$)	0.546	S ($\mathcal{G}$)	0.371
*	re ($\mathcal{K}$)	0.168	re ($\mathcal{K}$)	0.463	re ($\mathcal{K}$)	0.251
*	se ($\mathcal{K}$)	0.329	se ($\mathcal{K}$)	0.599	se ($\mathcal{K}$)	0.400

	R34					
	Fusion	ARI	Fusion	NMI	Fusion	Purity
1	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.7)	0.396	S + se ($\mathcal{G}$ 0.9)	0.642	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.7)	0.513
2	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.6)	0.380	S $\mathbin{+\!\!\!+}$ se ($\mathcal{G}$ 0.9)	0.642	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.8)	0.501
3	R + S ($\mathcal{K}$ 0.7)	0.376	re $\mathbin{+\!\!\!+}$ se ($\mathcal{K}$ 0.1)	0.642	R + S ($\mathcal{K}$ 0.7)	0.498
4	R $\mathbin{+\!\!\!+}$ S ($\mathcal{G}$ 0.7)	0.374	R $\mathbin{+\!\!\!+}$ S ($\mathcal{K}$ 0.7)	0.642	R + S ($\mathcal{G}$ 0.7)	0.494
5	R + S ($\mathcal{G}$ 0.7)	0.373	S $\mathbin{+\!\!\!+}$ se ($\mathcal{K}$ 0.9)	0.641	R $\mathbin{+\!\!\!+}$ S ($\mathcal{G}$ 0.7)	0.492
*	R ($\mathcal{K}$)	0.278	R ($\mathcal{K}$)	0.587	R ($\mathcal{G}$)	0.434
*	S ($\mathcal{K}$)	0.313	S ($\mathcal{K}$)	0.578	S ($\mathcal{K}$)	0.420
*	re ($\mathcal{K}$)	0.213	re ($\mathcal{K}$)	0.525	re ($\mathcal{K}$)	0.309
*	se ($\mathcal{K}$)	0.344	se ($\mathcal{K}$)	0.636	se ($\mathcal{K}$)	0.439

increase in the vector space if the sequences are of different lengths. For example, as shown in Figure 9(a), most activity components in the c1 category consist of: *Modal, ButtonBar, TextButton, TextButton, DatePicker, TextButton, TextButton, Icon*, and *Icon*. In the case of some activities, one *TextButton* is missing, as shown in Figure 9(b), or one *TextButton* has been added, as shown in Figure 9(c). Because the seq2seq autoencoder encodes the entire input sequence, even if two input components are similar but the lengths of the two sequences are different, they may not produce similar feature vectors. To demonstrate this limitation, Figure 9 shows three similar activities that are not grouped into the same class.

Summary for RQ2. Activity attributes such as UI components and positional information improve the performance of activity clustering. In particular, using textual data shows better performance than real activity images or Rico. However, due to the limitations of the seq2seq autoencoder, a *c1 issue* may occur.

C. RQ3: HOW USEFUL IS THE HYBRID APPROACH THAT USES BOTH IMAGES AND TEXT OF THE RICO DATASET FOR ACTIVITY CLUSTERING?

Table 5 and Figure 10 show the clustering results of the fused vectors. Table 5 displays the top five ARI, NMI, and Purity scores of multimodals and single modals for the C23 and R34. Figure 10 demonstrates all clustering results including weighted fusion functions and clustering methods for C23 and R34. In both Table 5 and Figure 10, multimodal clustering generally performs better than single modal clustering.

From Table 5, the best performance is fusion with the seq2seq latent vector or the semantic activity latent vector using both C23 and R34. The fusion model outperforms the single models because the fusion model complements the weaknesses of the single models by fusing the Rico, seq2seq, and semantic activity latent vectors.

However, not all multimodal clustering methods perform better than single-modal clustering methods. In some cases, a model that fuses a real activity latent vector to another latent vector performs worse than single-modal models as shown in Figure 10. This discrepancy occurs because the neural network struggles to learn to cluster activities using real activity images, as shown in Figure 7. Using the real activity latent vector in the model degrades the performance of the model when fused with vectors having different properties, such as Rico and seq2seq latent vectors. As a result, multimodal clustering is beneficial for activity clustering based on the semantic activity latent vector or seq2seq latent vector.

Summary for RQ3. In general, fusing the methods of RQ1 and RQ2 results in better performance compared to using RQ1 and RQ2 methods separately. In particular, we confirm that activity clustering with the fusion based on the semantic activity latent vector or seq2seq latent vector perform well. Note that the semantic activity latent vector and seq2seq latent vector play important roles in RQ3, which also had significant influence in RQ1 and RQ2, respectively.

D. DISCUSSION

1) WEIGHTED FUSIONS

Figure 10 illustrates the performance according to all evaluation metrics when latent vectors are fused. All graphs show similar trends across all metrics. The performance of

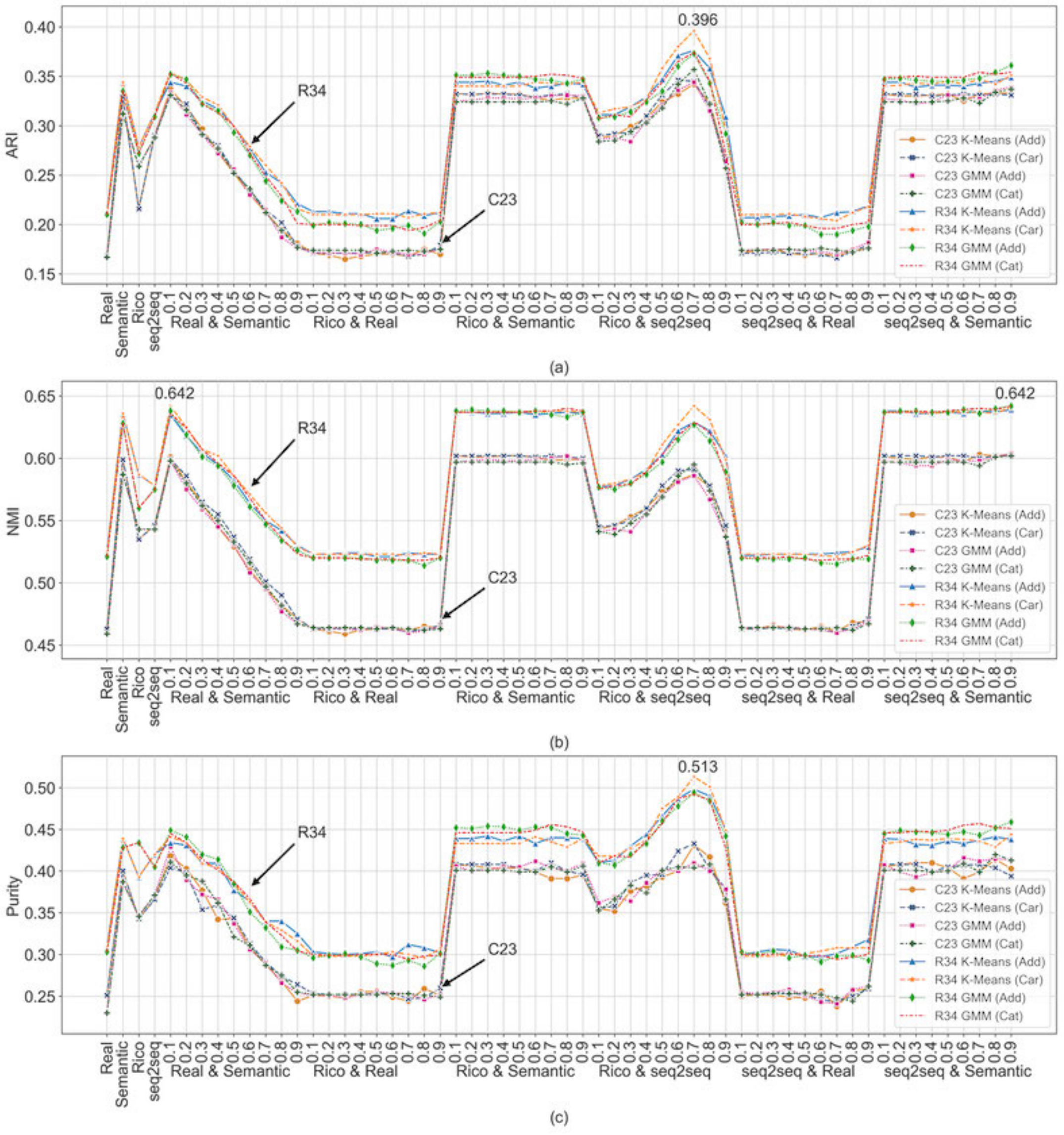


FIGURE 10. Performance of the models tested by C23 and R34 according to multiple evaluation metrics: (a) ARI, (b) NMI, and (c) Purity. Fusing semantic activity vectors with other latent vectors improves performance.

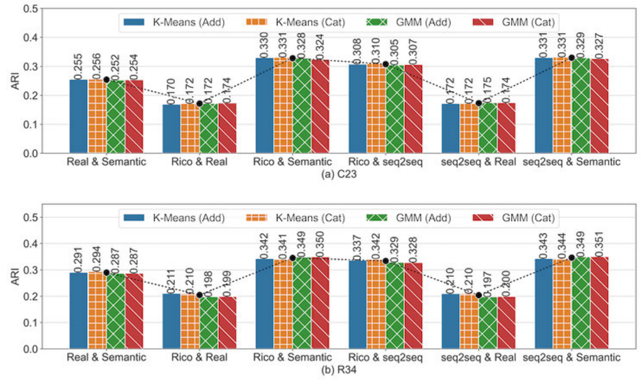


FIGURE 11. The mean values of ARI for each fusion model.

the fusion with the real activity and semantic activity latent vectors decreases as the weight of the real activity latent vector increases. This means that the weight of the semantic activity latent vector is an important factor and it is crucial not to fuse with real activity. The fusion with real activity and Rico latent vectors and the fusion with seq2seq and real activity latent vectors show relatively low performance. This is because the performance is affected by the low achievement of the real activity latent vector as shown in Figure 7. The fusion with the semantic activity latent vector or seq2seq latent vector resulted in improved performance relative to other fusion types.

Figure 11 shows the mean values of ARI for each fusion model. The mean values of NMI and purity also show similar tendencies. The fusion with Rico and semantic latent vectors and the fusion with seq2seq and semantic latent vectors show the best performance in C23. For R34, fusions with Rico and seq2seq latent vectors show the best performance along with the previous two fusions.

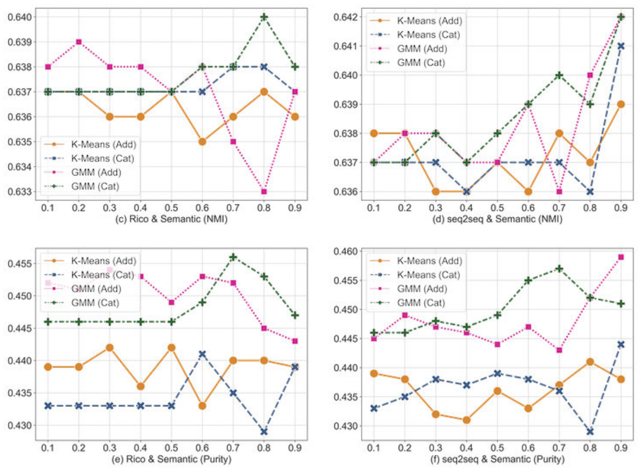


FIGURE 12. NMI and Purity of fusion with Rico and semantic activity latent vectors and fusion with seq2seq and semantic latent vectors according to weight in R34.

Figure 11 also shows that the fusion functions and clustering algorithms do not significantly affect the performance. Figure 12 illustrates the metrics of the fusion with Rico and semantic activity latent vectors and the fusion with seq2seq and semantic activity latent vectors according to weight in R34. As the weights change, there seems to be a difference in performance depending on the fusion function and the clustering method, but the difference is only about 1–2%. However, the GMM shows better performance in general.

We created two categories for the test dataset, C23 and R34. Figure 10 shows that the R34 category is better classified than the C23 category, and that using the R34 category in future studies may yield better results.

2) SEARCH RESULTS

Nearest neighbor search results on the all-element Rico dataset were compared with Rico, seq2seq, real and semantic activity latent vectors, and two multimodals (fused real and semantic activity latent vectors and fused Rico and seq2seq latent vectors) which performed activity clustering best according to the NMI and ARI evaluation metrics. Figures 13–15 represent the top three search results for Rico, seq2seq, fused Rico and seq2seq, real activity, semantic activity, and fused real and semantic activity latent vectors side-by-side over the same set of queries. Figure 13, Figure 14, and Figure 15 show the results for the cluster categories c5, c9, and c17, which show different results for the same query. As shown in Figure 13, Rico and seq2seq models gave worse search results, with the fused Rico and seq2seq latent vectors displaying better search results for c5 category. Figure 15 describes the search results for c17 category, where the real activity model appeared better than the other single modals. In this case, only one image was found, so semantic activity performed poorly. In Figure 14 representing c9 category, the real activity model missed all three images, but all others searched satisfactorily.

Summary of search results. Generally, multimodal models exhibit better performances than a single-modal. Among the single-modal, the results of the seq2seq and semantic activity models were superior to those of the Rico and real activity models.

3) LIMITATIONS

A limitation is that only 25 UI components are used to represent activities in Section IV-D, while there are many UI components provided by Android. For consistency, we used the same 25 components extracted by Liu et al. [2]. Due to differences between the actual UI component used in the activities and the component that we used, the performance of the model was lower than expected.

Difficulties may also occur in combining modalities in multimodal deep learning. In this study, we used sum and concatenation functions to fuse the two modalities. However, varying levels of noise and conflicts between modalities remain a problem in multimodal deep learning. Therefore, it is essential to explore the combination of different source modalities to improve performance [54].

The last limitation is the choice of the clustering algorithm. We experimented with only two clustering algorithms: K-Means and GMM clustering. Traditional clustering algorithms can often yield good results on general problems, but this is not guaranteed, as in the case of activity clustering. Finding a better clustering algorithm is an interesting future research proposal.

E. EXAMPLE OF USING ACTIVITY CLUSTERING

We present a concept of applying activity clustering in app testing. While the previous models set up an exploration strategy and tested the app, our approach explores the testing strategy based on activity.

Figure 16 shows the proposed example model using activity clustering. The application under test (AUT) is the input to the model, and the example model uses activity clustering to classify the activities of the AUT. UIAutomator or AndroidViewClient is used to extract activity image and activity UI information from activity, and the extracted image or UI information is used as input to activity clustering. The activity is classified through the learned activity clustering model and is assigned to one of the categories of R34. This guarantees that the model has the most suitable test method for each category. Thus, when the activity is classified into categories, the recommended testing method defined for each category is initiated. The example model repetitively classifies activities in this manner, one-by-one.

To find the most suitable test method for each category, we can match the predefined classes of Rosenfeld et al. [19] and Ardito et al. [35] to R34, then apply previously selected testing methods on the classification result. For example, the r1 class uses a random strategy, the r2 class uses an enhanced random strategy, and the r5 class uses the DFS strategy. It is also possible to extend the model by applying new test methods for each activity.

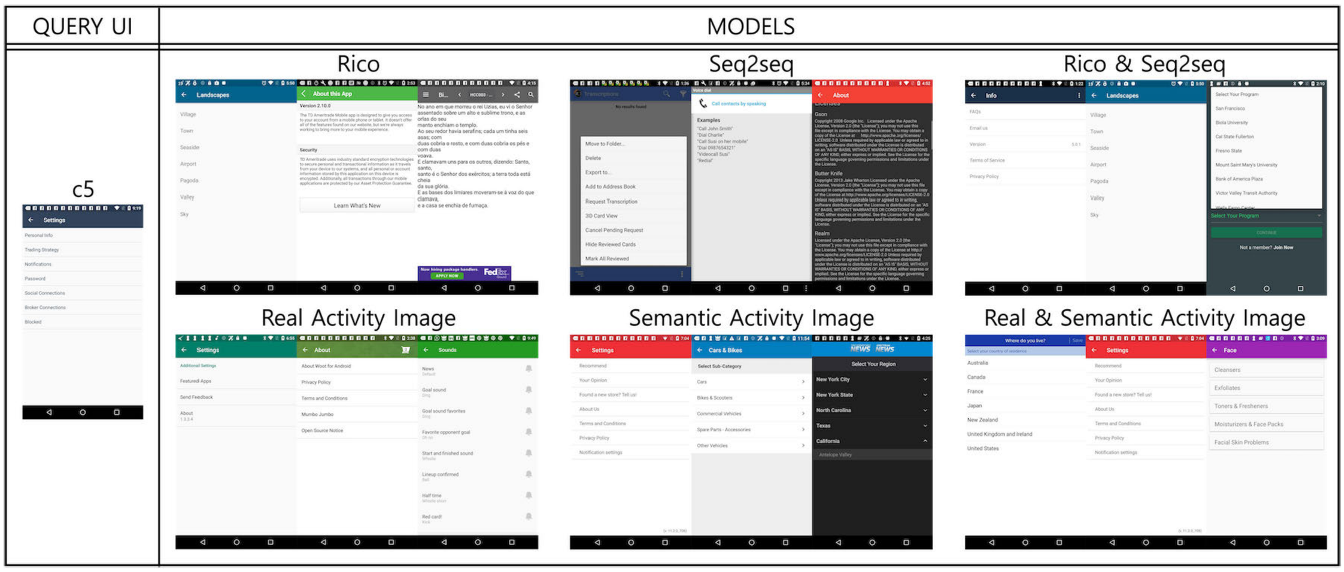


FIGURE 13. Comparison of the nearest neighbor search results on the all-element Rico dataset with Rico, seq2seq, real and semantic activity image convolution, and two multimodals which performed best according to the NMI and ARI evaluation metrics. Rico and seq2seq models gave worse search results, with the fused Rico and seq2seq latent vectors displaying better search results. Generally, the multimodal models performed better than single-modals.

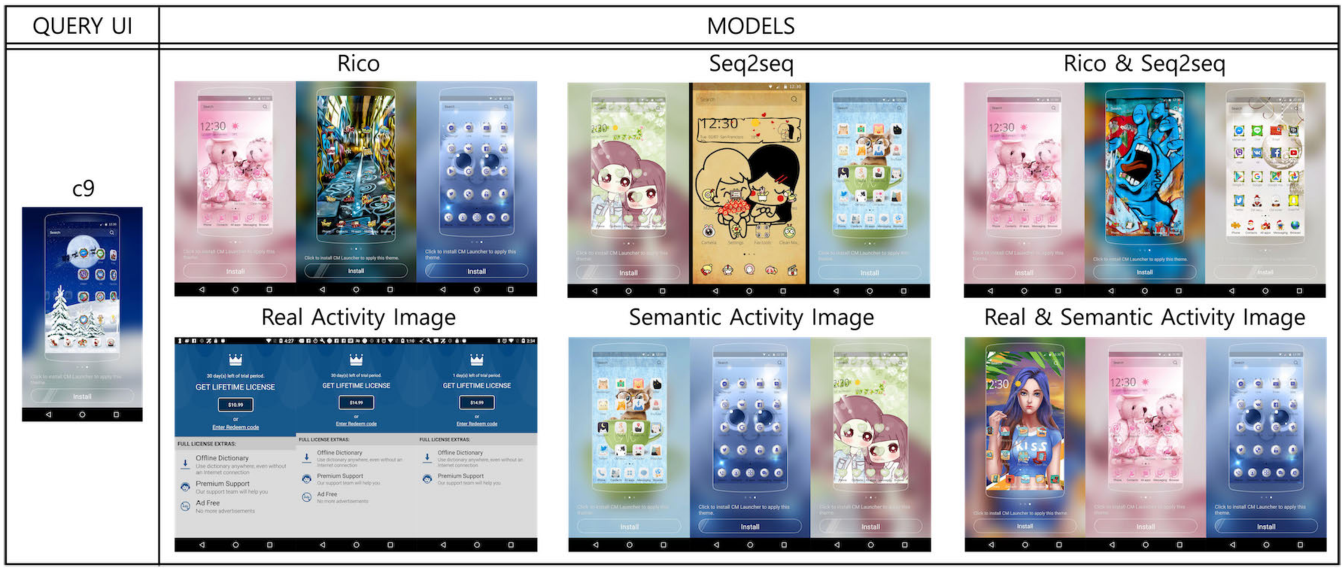


FIGURE 14. Comparison of the nearest neighbor search results on the all-element Rico dataset with Rico, seq2seq, real and semantic activity image convolution, and two multimodals which performed best according to the NMI and ARI evaluation metrics. The real activity model missed all three images, but all others searched satisfactorily.

VII. THREATS TO VALIDITY

This section distinguishes between threats to construct, external, and internal validity as follows.

A potential threat to construct validity is related to CNN and seq2seq autoencoder, which are representative models of image and text, respectively. We adopt the CNN autoencoder for activity images and the seq2seq autoencoder for activity attributes. There may be models that are more suitable for activity clustering. However, since we are the first to research activity clustering, we consider the two representative models as a starting point. Another threat to validity is based on the metrics used to evaluate clustering. Purity, NMI, and ARI have been widely used in clustering evaluation [16], [48], giving confidence that they are suitable metrics. However, it may be effective to measure clustering in another way such as Normalized Variation of Information (NVI) [55], [56]. In addition, it is interesting to consider more fine-grained metrics for each cluster.

Threats to external validity are related to generalization of our findings. Our research uses publicly available datasets

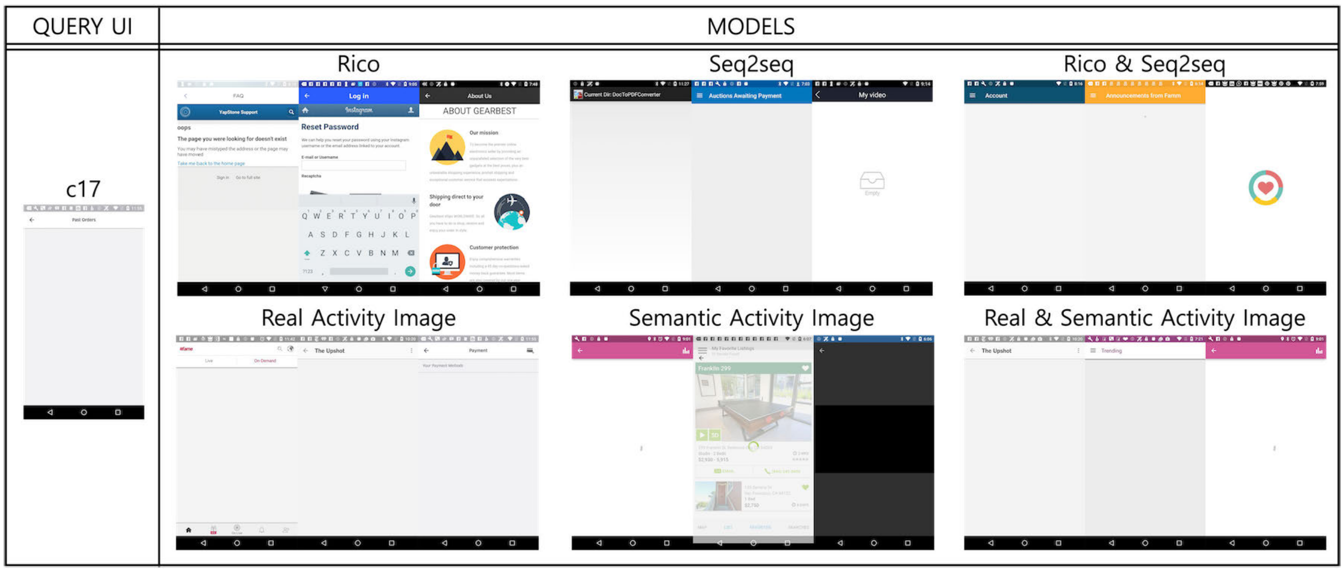


FIGURE 15. Comparison of the nearest neighbor search results on the all-element Rico dataset with Rico, seq2seq, real and semantic activity image convolution, and two multimodals which performed best according to the NMI and ARI evaluation metrics. Unusually, the real activity model showed better than other single modals. In this case, only one image was found, so semantic activity performed poorly.

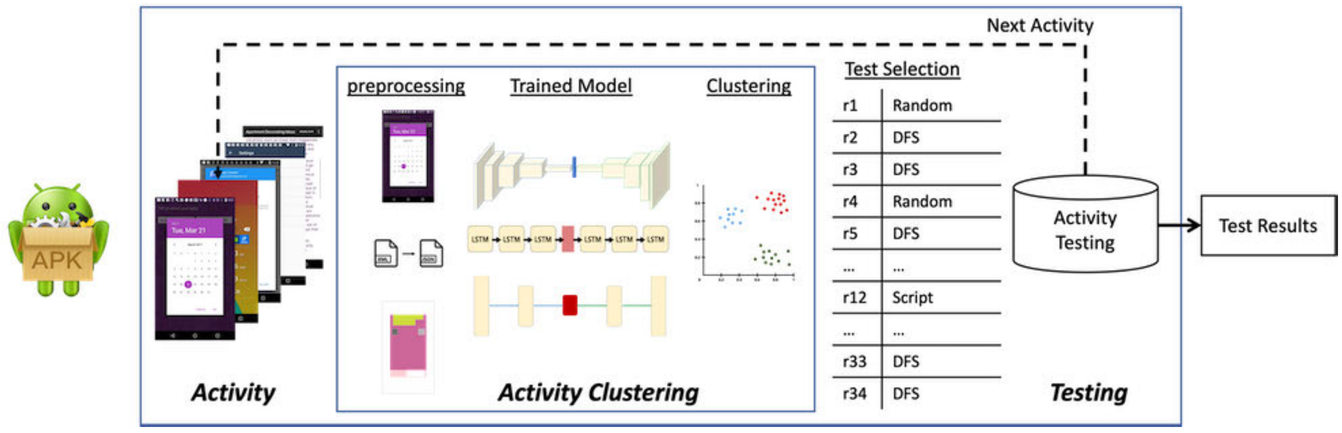


FIGURE 16. The testing process.

and clustering techniques already validated works. However, to extend the validity of our results, a wider sample of real Android apps including apps for TVs, automotives, and wearable devices should be considered as well. Thus, we cannot claim that our results generalize to other clustering algorithms and models yet.

Another possible threat to the internal validity could have been the construction of the test and validation dataset. To evaluate this threat, two experts classified 2000 images from 10,000 randomly selected images as mentioned in Section V-B. Then five other researchers checked the results. Note that we utilized common patterns, structures, and behaviors based on previous research [2], [19]. Although we have double checked our datasets classified, there could still be errors that we did not notice.

Finally, to minimize potential threats to validity, we chose open-source datasets and provided two new datasets and sources that contain our experiments [36].

VIII. CONCLUSION AND FUTURE WORK

In order to uncover the feasibility of using the Rico dataset for activity clustering and the effects of activity attributes on clustering, we proposed a new Android activity clustering model based on deep learning. A seq2seq latent vector is constructed based on properties of the activity and real and semantic activity latent vectors are obtained from a CNN autoencoder. The vectors are fused using sum and concatenation functions. Finally, unsupervised clustering is performed on the fused vectors to classify the activities. We conducted various experiments, including comparing four single modals, six fusions with various weights across two fusion methods, and two clustering approaches. Two types of labeled datasets (C23 & R34) were constructed to analyze each approach. The results show that good performance is achieved when fusing with a semantic activity latent vector or seq2seq latent vector.

We presented the limitations of our study and suggested ways to solve the limitations. For example, the accuracy of the seq2seq latent vector may suffer due to varying sequence lengths. To overcome this weakness, we constructed a CNN model for real and semantic activity images, instead of the Rico latent vector, and fused the latent vectors. Furthermore, we plan to study multimodal deep learning models that learn and cluster simultaneously [53].

Using our work as a cornerstone for activity clustering, researchers can investigate other areas based on clustering, such as GUI searching from hand-drawn sketches, GUI implementation, or bug detection in UI. In particular, in the case of automated testing, there are various areas to investigate such as applying different testing methods for each screen, generating input sequences for the Login screen, or detecting specific GUIs. We believe that our work is not limited to activity clustering and that our work can act as a foundation for further software engineering on Android research.

REFERENCES

[1] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proc. 30th Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 2017, pp. 845–854.

[2] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar, "Learning design semantics for mobile apps," in *Proc. 31st Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 2018, pp. 569–579.

[3] X. Ge, "Android GUI search using hand-drawn sketches," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion*, May 2019, pp. 141–143.

[4] 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 *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. (ICSE)*, May 2018, pp. 665–676.

[5] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy API usage patterns in Android apps: An empirical study," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, May 2014, pp. 2–11.

[6] S. T. Ahmed Rumee, D. Liu, and Y. Lei, "MirrorDroid: A framework to detect sensitive information leakage in Android by duplicate program execution," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2017, pp. 1–6.

[7] A. Zhang, Y. He, and Y. Jiang, "CrashFuzzer: Detecting input processing related crash bugs in Android applications," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–8.

[8] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically discovering, reporting and reproducing Android application crashes," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2016, pp. 33–44.

[9] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," in *Fundamental Approaches to Software Engineering*, vol. 7793, no. 19. Rome, Italy: Springer, 2013, pp. 250–265.

[10] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet? (E)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 429–440.

[11] N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting usability defects: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 848–867, Sep. 2017.

[12] H. Kim, M. Kang, S. Cho, and S. Choi, "Efficient deep learning network with multi-streams for Android malware family classification," *IEEE Access*, vol. 10, pp. 5518–5532, 2022.

[13] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5147–5156.

[14] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi, "Symmetric graph convolutional autoencoder for unsupervised graph representation learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6518–6527.

[15] D. Hu, F. Nie, and X. Li, "Deep multimodal clustering for unsupervised audiovisual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2019, pp. 9248–9257.

[16] M. Abavisani and V. M. Patel, "Deep multimodal subspace clustering networks," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 6, pp. 1601–1614, Dec. 2018.

[17] R. Zhou and Y. Shen, "End-to-end adversarial-attention network for multi-modal clustering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14607–14616.

[18] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.

[19] A. Rosenfeld, O. Kardashov, and O. Zang, "Automation of Android applications functional testing using machine learning activities classification," in *Proc. IEEE/ACM 5th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft)*, May 2018, pp. 122–132.

[20] G. John Cleary and E. Leonard Trigg, "K*: An instance-based learner using and entropic distance measure," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 108–114.

[21] D. Amalfitano, V. Riccio, N. Amatucci, V. D. Simone, and A. R. Fasolino, "Combining automated GUI exploration of Android apps with capture and replay through machine learning," *Inf. Softw. Technol.*, vol. 105, pp. 95–116, Jan. 2019.

[22] A. Sahami Shirazi, N. Henze, A. Schmidt, R. Goldberg, B. Schmidt, and H. Schmauder, "Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps," in *Proc. 5th ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, Jun. 2013, pp. 275–284.

[23] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with REMAUI (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 248–259.

[24] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2020, *arXiv:2003.05991*.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[26] C. Lee, S. Kim, D. Han, H. Yang, Y.-W. Park, B. C. Kwon, and S. Ko, "GUIComp: A GUI design assistant with real-time, multi-faceted feedback," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, Apr. 2020, pp. 1–13.

[27] K. Gupta, J. Lazarow, A. Achille, L. Davis, V. Mahadevan, and A. Shrivastava, "LayoutTransformer: Layout generation and completion with self-attention," 2020, *arXiv:2006.14615*.

[28] V. P. S. Pandian, S. Suleri, and M. Jarke, "BLU: What GUIs are made of," in *Proc. 25th Int. Conf. Intell. User Interface Companion*, Mar. 2020, pp. 81–82.

[29] D. M. Arroyo, J. Postels, and F. Tombari, "Variational transformer networks for layout generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13637–13647.

[30] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic Android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[31] H. Zhou, X. Yang, H. Pan, and W. Guo, "An Android malware detection approach based on SIMGRU," *IEEE Access*, vol. 8, pp. 148404–148410, 2020.

[32] X. Zhang, J. Wang, J. Xu, and C. Gu, "Detection of Android malware based on deep forest and feature enhancement," *IEEE Access*, vol. 11, pp. 29344–29359, 2023.

[33] J. Qin, H. Zhang, S. Wang, Z. Geng, and T. Chen, "Acteve++: An improved Android application automatic tester based on acteve," *IEEE Access*, vol. 7, pp. 31358–31363, 2019.

[34] I. Salihu, R. Ibrahim, B. S. Ahmed, K. Z. Zamli, and A. Usman, "AMOGA: A static-dynamic model generation strategy for mobile apps testing," *IEEE Access*, vol. 7, pp. 17158–17173, 2019.

[35] L. Ardito, R. Coppola, S. Leonardi, M. Morisio, and U. Buy, "Automated test selection for Android apps based on APK and activity classification," *IEEE Access*, vol. 8, pp. 187648–187670, 2020.
[36] *Test Dataset and Source*. Accessed: Feb. 23, 2023. [Online]. Available: https://github.com/hopemini/activity-clustering-multimodal-ml
[37] M. Shtern and V. Tzerpos, "Clustering methodologies for software engineering," *Adv. Softw. Eng.*, vol. 2012, pp. 1–18, May 2012.
[38] Q. Zhao, L. Zong, X. Zhang, Y. Li, and X. Tang, "A multimodal clustering framework with cross reconstruction autoencoders," *IEEE Access*, vol. 8, pp. 218433–218443, 2020.
[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014, *arXiv:1409.3215*.
[40] H. Bahuleyan, L. Mou, O. Vechtomova, and P. Poupart, "Variational attention for sequence-to-sequence models," 2017, *arXiv:1712.08207*.
[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[42] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
[43] J. Masci, U. Meier, D. C. Ciresan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2011, vol. 6791, no. 4, pp. 52–59.
[44] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2528–2535.
[45] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3642–3649.
[46] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*.
[47] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.
[48] D. C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[49] C. C. Aggarwal, *Data Mining: The Textbook*. Cham, Switzerland: Springer, 2015.
[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 32, 2019, pp. 1–15.
[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[52] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.
[53] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. Int. Conf. Mach. Learn.*, vol. 48, 2015, pp. 478–487.
[54] K. Liu, Y. Li, N. Xu, and P. Natarajan, "Learn to combine modalities in multimodal deep learning," 2018, *arXiv:1805.11730*.
[55] R. Reichart and A. Rappoport, "The NVI clustering evaluation measure," in *Proc. 13th Conf. Comput. Natural Lang. Learn.*, 2009, pp. 165–173.
[56] P. S. A. Babu, C. S. R. Annavarapu, and A. Mohapatra, "A novel method for next-generation sequence data analysis using PLSA topic modeling technique," in *Proc. 2nd Int. Conf. Adv. Comput. Commun. Paradigms (ICACCP)*, Feb. 2019, pp. 1–6.

SUNGMIN CHOI received the B.S. and M.S. degrees in computer science and engineering from the Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea, in 2002. He is currently pursuing the Ph.D. degree in computer science with Yonsei University, Seoul, Republic of Korea.

From 2002 to 2003, he was a Researcher with ITM, Seoul. Since 2004, he has been a Chief Researcher with LG Electronics, Seoul. He has been developing Linux frameworks and Android frameworks. His research interests include automated application testing, activity clustering, and Android containers.

HYEON-TAE SEO received the M.S. degree in computer science from Yonsei University, Seoul, Republic of Korea, in 2022.

Since 2022, he has been a Researcher with KT Research and Development Center, Seoul. His research interests include clustering, code repair, and code summarizations.

YO-SUB HAN received the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, in 2006. He was a Researcher with the Korea Institute of Science and Technology until 2009. In 2009, he joined Yonsei University, where he is currently a Professor with the Department of Computer Science.

His research interests include formal language theory, algorithm designs, and information retrieval.