

# Data-Driven Accessibility Repair Revisited: On the Effectiveness of Generating Labels for Icons in Android Apps

Forough Mehralian  
School of Information and Computer  
Sciences  
University of California, Irvine, USA  
fmehrali@uci.edu

Navid Salehnamadi  
School of Information and Computer  
Sciences  
University of California, Irvine, USA  
nsalehna@uci.edu

Sam Malek  
School of Information and Computer  
Sciences  
University of California, Irvine, USA  
malek@uci.edu

## ABSTRACT

Mobile apps are playing an increasingly important role in our daily lives, including the lives of approximately 304 million users worldwide that are either completely blind or suffer from some form of visual impairment. These users rely on screen readers to interact with apps. Screen readers, however, cannot describe the image icons that appear on the screen, unless those icons are accompanied with developer-provided textual labels. A prior study of over 5,000 Android apps found that in around 50% of the apps, less than 10% of the icons are labeled. To address this problem, a recent award-winning approach, called *LabelDroid*, employed deep-learning techniques to train a model on a dataset of existing icons with labels to automatically generate labels for visually similar, unlabeled icons. In this work, we empirically study the nature of icon labels in terms of distribution and their dependency on different sources of information. We then assess the effectiveness of *LabelDroid* in predicting labels for unlabeled icons. We find that icon images are insufficient in representing icon labels, while other sources of information from the icon usage context can enrich images in determining proper tokens for labels. We propose the first context-aware label generation approach, called *COALA*, that incorporates several sources of information from the icon in generating accurate labels. Our experiments show that although *COALA* significantly outperforms *LabelDroid* in both user study and automatic evaluation, further research is needed. We suggest that future studies should be more cautious when basing their approach on automatically extracted labeled data.

## CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in accessibility**; • **Software and its engineering** → **Software usability**.

## KEYWORDS

Accessibility, Deep Learning, Android, Alternative Text, Screen Reader

## ACM Reference Format:

Forough Mehralian, Navid Salehnamadi, and Sam Malek. 2021. Data-Driven Accessibility Repair Revisited: On the Effectiveness of Generating Labels for Icons in Android Apps. In *Proceedings of the 29th ACM Joint European*

*Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21)*, August 23–28, 2021, Athens, Greece. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3468264.3468604>

## 1 INTRODUCTION

There is an increased onus on app developers to make their products accessible for users with a wide range of disabilities, including the approximately 304 million users worldwide that are blind or visually impaired [21]. Blind users rely on screen readers to interact with apps. Built-in screen readers in mobile devices facilitate interactions with mobile apps by reading the screens out loud for the users. Similar to Alt-Text for web images [3], the embedded textual labels for GUI images are essential for enabling usage of these apps by the blind. These labels are even more critical for *functional* icons [9], images that developers utilize to convey the availability of an action, not to convey information. Without a proper description of the functionality initiated by icons, screen-reader users are unable to interact with an app. App accessibility is thus directly affected by the lack of informative icon labels.

Recently, several projects have studied the extent of accessibility issues in mobile apps [18, 22, 47], among others demonstrating widespread violation of label-based accessibility guidelines in Android apps. Missing labels, duplicate labels, and non-informative labels are different types of label-based accessibility concerns, among which missing labels is the most prominent one. Missing label occurs when an icon is not accompanied with a textual label (a.k.a., *content description* in Android) describing its functionality. Ross et al. [47] report that in 50% of their assessed apps, less than 10% of icons were labeled.

The increasing awareness of this accessibility issue has instigated some recent efforts towards alleviating it [22, 26, 59]. Notably, Chen et al. [22] developed a promising method, called *LabelDroid*, which employs deep-learning techniques to train a model on a dataset of existing icons with labels to automatically generate labels for visually similar, unlabeled icons.<sup>1</sup>

For data-driven approaches, such as *LabelDroid*, data exploration is a prerequisite. Perfect model architectures may deliver misleading or unexpected results if the dataset is not carefully examined. Although prior works have empirically studied the severity of the missing labels [18, 47], none have studied (1) the characteristics of natural language labels for the labeled icons extracted from thousands of automatically explored apps in terms of their categories, uniqueness, distribution, and dependency to other icon properties, which can be relevant to the problem of automatic label generation, (2) effectiveness of existing repair approaches in predicting different categories of labels, and (3) impact of incorporating different sources of information in generating icon labels.

<sup>1</sup>*LabelDroid* received the ACM SIGSOFT Distinguished Paper Award at ICSE 2020 [11].



This work is licensed under a Creative Commons Attribution International 4.0 License.  
*ESEC/FSE '21, August 23–28, 2021, Athens, Greece*  
© 2021 Copyright held by the owner/author(s).  
ACM ISBN 978-1-4503-8562-6/21/08.  
<https://doi.org/10.1145/3468264.3468604>

To fill this gap, we conducted a large empirical study on icons extracted from 9,658 android apps to understand the characteristics of icons and labels in Android apps. We then assessed the effectiveness of learning and generating natural language labels using this dataset.

Our empirical study reveals that the dataset of automatically extracted icon labels is highly imbalanced, resulting in a severe data-driven bias in the LabelDroid model. It is striking that the introduced bias in LabelDroid is toward predicting *predefined labels* that are shipped with icons in the widgets and templates of Android’s Standard Development Kit. Since in practice it is extremely unlikely for these icons to be unlabeled, generating labels for them is pointless. We found that excluding these predefined labels drops the LabelDroid’s accuracy by 34%.

Besides, our empirical study shows the necessity but insufficiency of images in representing icon labels. We found that incorporating different information sources for icons can enrich their representation by providing their usage context, substantially improving the identification of correct tokens in labels.

These findings subsequently informed the development of COALA—a *deep learning (DL)* approach to generate context-aware labels for icons in Android apps. COALA automatically extracts high-quality labels for icons from the raw dataset of app screens and layouts, from which it learns how to incorporate different sources of information and translate the image to a textual label. It then utilizes the learned model to generate informative labels for unlabeled icons. Our experiments show that COALA outperforms LabelDroid by 24% in generating labels for unlabeled icons that exactly match the ground truth.

This paper makes the following contributions:

- An empirical study of the nature of labels and how different sources of information contribute to predicting a correct label;
- An analysis of data imbalance, and how it invalidates the results reported in the evaluation of LabelDroid [22];
- COALA, the first context-aware label generation approach to generate textual labels for Android icons and its implementation which is publicly available [6];
- Experimental results corroborating the superiority of COALA in comparison to LabelDroid in generating high-quality labels for icons.

The remainder of this paper is organized as follows. Section 2 provides the background of this study using an illustrative example. Section 3 explains the empirical study of natural language labels and other sources of information of icons. Then, Section 4 describes COALA in detail, which will be evaluated in Section 5 along with the existing deep learning model, LabelDroid. Section 6 explains the threats to the validity of the research. The paper concludes with a discussion of the related research and avenues of future work.

## 2 BACKGROUND

An Android app’s user interface (UI) is implemented in terms of one or more *activity* components, where each activity represents a screen. Figure 1 shows an activity for a messenger app along with snippets of its XML *layout* and source code. A *layout* file specifies the placement and design of UI elements in an activity. UI elements such as `ImageView` are objects in the XML tree structure of layout.

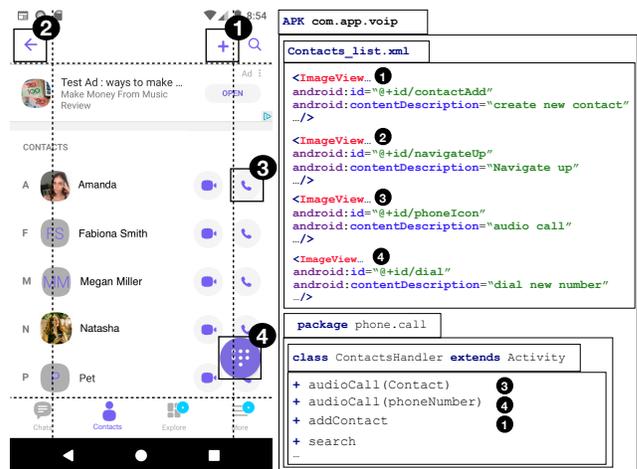


Figure 1: Icons and their content descriptions in a messenger app

Visually impaired users rely on screen readers, like Google’s *TalkBack* service, to interact with apps. Screen readers describe images for blind users by announcing the developer-provided textual label in `android:contentDescription` field of UI elements such as `ImageView` and `ImageButton`. Figure 1 illustrates four icons in a messenger app, along with their content descriptions in the layout.

Note that the icons comprising Android Standard Development Kit’s UI widgets and templates, such as *action bars*, come with *predefined* labels. The textual description of icon number 2 in Figure 1, “Navigate up”, is an example of such predefined labels.

Prior studies [47] show that missing label (i.e., content description) is a prevalent accessibility issue in Android apps, rendering screen readers inoperable. The plus icon in Figure 2 suffers from this issue. *TalkBack* screen reader describes this button as “unlabeled button”, seriously hindering a blind user’s ability to use the app. To tackle this issue, Chen et al. proposed LabelDroid [22], which predicts natural language labels for icons given their images using deep-learning techniques. In their work, the authors extracted thousands of labeled icons from 7,594 Android apps. They then trained an image captioning deep-learning model to transform icon images to natural language labels. However, images cannot fully represent icon labels. For example, while the plus icon in Figure 1 and Figure 2 are visually similar, a proper label for the former, e.g., “create new contact”, is different from the label for the latter, i.e., “add a playlist”. The distinction between two labels comes from the usage context of icons. The former is used in the contacts page of a messaging app, while the latter is used in a music player app. Our objective is to understand the nature of natural language labels for icons, study the effectiveness of the prior work, and propose an automated label generation model to alleviate the shortcomings of the prior work and motivate the need for further studies.

## 3 DATA EXPLORATION

To develop a better understanding of icon labels and whether different sources of information have predictive impact on them, we conducted an empirical study to answer the following research questions:

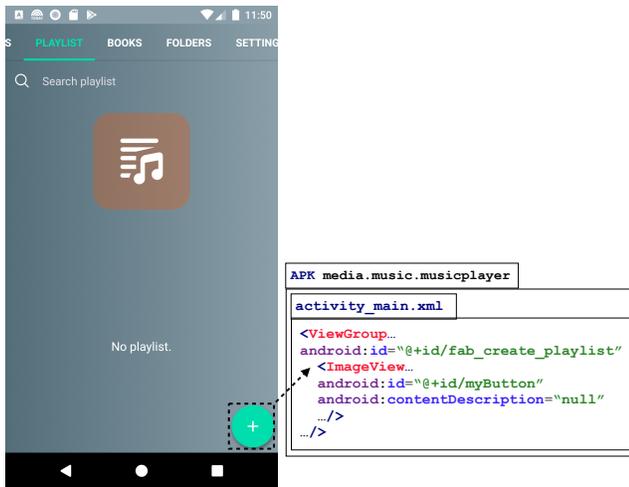


Figure 2: Plus icon in a music player app

- RQ1. What are the characteristics of labels regarding their uniqueness and distribution?
- RQ2. How similar are the labels of icons with similar images?
- RQ3. To what extent different sources of information from icon context can reveal the label?

### 3.1 Experimental Setup

We conduct our study on a set of icons extracted from 15,087 Android apps included in the LabelDroid dataset [22]. This dataset was collected through dynamic GUI exploration of apps. We extracted visible, clickable Android `ImageView` and `ImageButton` icons from XML layouts and screenshots to form our primary dataset for this study. Each icon in our dataset corresponds to a triple of  $\langle \text{image}, \text{label}, \text{usage\_context} \rangle$ . For image, we crop the screenshot based on the coordinates of an icon’s boundary box as specified in the bounds property of the icon considering the orientation of the device. For label, we use the value of `contentDescription` property associated with the icon. For contextual information, we extract several parameters in three levels, i.e., app-level (app category), activity-level (activity name, screen title), icon-level (Android id, screen region, ancestor id, siblings id/text).

The majority of contextual parameters directly map to a property in the XML layout, e.g., id and text. To find the parent and siblings of an icon, we refer to the hierarchical structure of XML layouts. Icons that share a parent node in the XML tree are considered to be siblings. For the screen title, we refer to the text property of the top, leftmost `TextView` element in the layout. For the screen region, we refer to the bounds property of an icon to determine in which of the 9 screen regions, as specified with dashed lines in Figure 1, it belongs. Activity names and package names are available in GUI exploration artifacts. We then use package names to extract app categories from Google Play using BeautifulSoup [46] crawler.

To improve data integrity, we performed text normalization steps on textual information of icons. Labels and textual parameters extracted using the above-mentioned techniques are not restricted to follow a standard or commonly accepted structure. Developers may use *camelCase* [5] or *snake\_case* [13] conventions, or even other types of characters as delimiters. We transformed the textual information to lower case, replaced all the special characters with

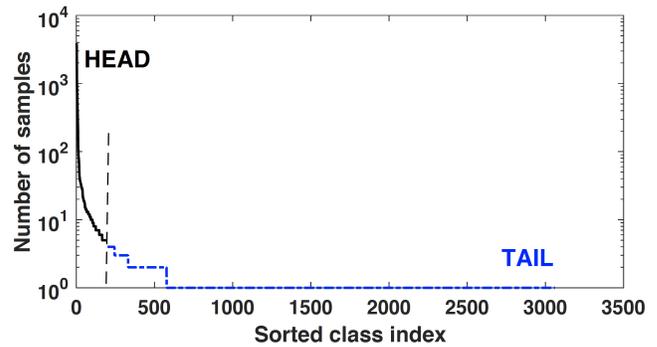


Figure 3: Imbalanced distribution of labels for icons. To the left are the few dominant classes, and to the right is the long tail. The cutoff separates the labels with more than 5 samples.

a space and applied spell correction and lemmatization to their tokens. We also filtered *meaningless labels* as introduced in the prior work [22].

In our dataset, icons with the same label in the same app would be counted once. Thus, our dataset consists of 21,864 icons extracted from 17,839 different screens of 9,658 apps.

### 3.2 RQ1. Characteristics of Labels

For this research question, we first study the distribution of icon labels. In our dataset, we found 3,061 different labels with high-class imbalance. By considering each of these 3,061 labels as a class for icons, we observed a *long-tailed* dataset as shown in Figure 3, in which 51.57% of the data comes only from 3 most frequent classes, while 93.5% of the classes have less than 5 samples in the whole dataset and 2,484 out of 3,061 occurred only once. The average number of samples per class is 7.14 and the median is 1. The gap between mean and median also indicates a left-skewed data distribution.

By further exploration of dominant classes of labels, we found that some Android icons come with a predefined label. For example, if a developer uses an up button in the action bar, similar to icon 2 in Figure 1, it comes with “Navigate up” label. We manually extracted predefined labels for all icons in Android Studio, the most widely used IDE for Android development. This analysis produced the following labels: {“navigate up”, “more options”, “next month”, “previous month”, “open navigation drawer”, “close navigation drawer”, “search”, “clear query”, “interstitial close button”}. If we exclude these labels, the average number of samples per class drops by 63.79% and changes to 2.58.

This observation alerts us to the erroneous conclusions we may draw from the evaluation of a label prediction approach. To facilitate the explanation of the issue, imagine 90% of the data has label X. In that case, a model that just predicts X is already 90% accurate and its effectiveness may not be interpreted realistically. This issue would be exacerbated if our goal was to label unlabeled icons, but the model is only good at predicting dominant labels that happen to be the aforementioned predefined labels.

In addition to dominant classes, it is also important to pay attention to low-frequency labels. The long tail of label distribution demonstrates the labels for which the dataset may not be representative enough. When we exclude predefined labels, 31.47% of

the remaining labels have only one sample in the dataset. That means, a proper label for them cannot be simply retrieved from the previously seen data, challenging a deep learning model for prediction of labels.

**Observation 1:** The data is highly skewed towards a limited set of labels, threatening the validity of the evaluation of a label prediction model. Furthermore, low-frequency and unique labels in the long tail challenge the construction of an effective learning-based prediction model.

When we studied the distribution of tokens in the whole dataset, we observed the same long-tailed distribution. That is, the tokens of predefined labels are the dominant classes. We further studied the tokens for unique labels to determine to what extent the tokens of unique labels can be derived from previously seen tokens. We found that for 53.99% of unique labels, *all* of the tokens were observed in the previously seen tokens, and for 86.5% of the unique labels, at least one non-trivial token was previously seen. By trivial tokens, we mean stop words such as “for”, “the”, “to”, etc. Thus, a token-based label prediction model may hold promise in correctly generating low-frequency and unique labels.

**Observation 2:** Substantial portion of tokens comprising the unique labels can be found in the existing vocabulary of tokens, suggesting a token-based prediction model may be effective in correctly generating low-frequency and unique labels.

### 3.3 RQ2. Labels of Visually Similar Icons

In order to study labels of icons with similar images, we trained an image classifier and annotated the icons with their image class. Liu et al. [38] identified 99 common image classes shared across apps through manual open coding of Android icons. They then trained a Convolutional Neural Network (CNN) to classify icon images. Their model is 94% accurate on average in predicting class of icon images. Rico dataset [25] provides the output of this image classifier for icons existing in their dataset of Android screens. We augmented our dataset with the class of icons to study the diversity of labels of visually similar icons.

On average, each class contains 225.4 icons with 20.03 different labels. For example, among 178 icons in class “add” (the plus icon), 61 different labels exist. In terms of tokens, the average number of unique tokens for labels in each class of icons is 28.6 with the median of 16. For instance, there are 61 unique tokens comprising the labels of icons in class “add”.

**Observation 3:** While image similarity restricts the set of probable labels and tokens for icons, there is still substantial level of diversity among labels and tokens for visually similar icons.

### 3.4 RQ3. Labels and Icon Information

We study the relationship between tokens of icon labels and other contextual information from the icon. We extract the contextual information in three levels: (1) *App level* that contains the categories

**Table 1: Correlation,  $\rho$ , and Mutual Information,  $MI$ , between icon information,  $C$ , and tokens in labels,  $T$ .**

Information Sources		$MI(T, C)$	$\rho(T, C)$
App level	Category	0.0908	0.1469
Activity level	Screen title	0.3474	0.0953
	Activity name	0.3924	0.1808
Icon level	Android id	0.7337	0.4221
	Screen region	0.2840	0.3187
	Siblings id	0.5673	0.3266
	Siblings text	0.4519	0.2814
	Parent id	0.5699	0.426

of apps, (2) *Activity level* including screen title and activity name, and (3) *Icon level* that contains the identifier name of the icon itself, its parents, and its siblings along with the region that the icon is located and the texts of its siblings.

To measure the dependency between two random variables, we calculate their *Mutual Information (MI)* [41]. It quantifies the amount of information obtained about one random variable through observing another random variable. In the context of our problem, MI determines which parameter,  $C$ , has the highest likelihood of predicting correct tokens in the label,  $T$ .  $MI(C, T)$  is defined as  $H(T) - H(T|C)$ , where  $H$  is the *entropy*, representing the uncertainty in a random variable. Although MI tells us how important the parameters are in predicting tokens, it does not tell us whether the contextual information is a predictor of presence or absence of tokens in icon labels. For that, we calculate the Pearson correlation coefficient,  $\rho(T, C)$ , to see how changes in the icon information result in *predictable* changes in the tokens.

Table 1 summarizes the result of this experiment. As shown in Table 1, we observe different degrees of correlation between the various sources of information and the tokens. The identifier name of the icon and its parent have the highest correlation with tokens in the labels. While app category does not appear to associate with tokens. Apart from app category, activity-level parameters have least correlations.

**Observation 4:** Different information sources exhibit different degrees of effectiveness in empowering a probabilistic model in predicting tokens for icon labels.

### 3.5 Summary

Our findings in data exploration motivated us to conduct further studies on the models and develop COALA, the first context-aware label generation approach.

First, the empirical study showed that the dataset of icons and labels in Android is highly imbalanced towards predefined labels. Thus, we will study the impact of learning on such data on the model fairness.

Second, the empirical study showed that while a significant subset of labels in the dataset are unique, substantial portion of tokens comprising these unique labels can be found in the existing vocabulary of tokens. To overcome the challenge posed by unique and low-frequency labels for a DL approach, we devise a learning model to generate labels in terms of their constituent tokens.

Third, the empirical study showed that while there exists a wide variety of labels for visually similar icons, there are other sources

of information available for improving the representation power of a probabilistic model. Our approach, in turn, leverages additional sources of information of an icon in addition to its graphical representation to determine the probable tokens in its label. This is akin to the intuition that sighted users can easily distinguish the functionality of similar icons by virtue of their knowledge of each icon’s usage context.

In the following section, we describe the architecture of COALA. We then study further research questions related to the fairness and effectiveness of DL models in generating textual labels for icons in Section 5.

## 4 COALA

In the following section, we introduce COALA—a *deep learning (DL)* approach to generate **context-aware labels** for icons in Android apps. Figure 4 provides an overview of COALA, which consists of two main modules: *Data Pre-Processing* and *DL Architecture*.

Data Pre-Processing module in COALA is responsible for extracting a dataset of labeled icons along with their information in their usage context. After finding icon specifications from thousands of XML layouts, it filters improper and duplicate icons similar to the prior work [22] and creates a dataset of icons for the DL module.

The DL Architecture is responsible for encoding the icons to be later decoded to textual labels. The encoding step has two phases: *Image Encoder* and *Context Encoder*, each of which is tailored to compute the embedding of a specific type of data. These representations are then fused in a *Fully Connected Layer* to prepare a vector, from which *Label Decoder* generates the corresponding textual label.

In the remainder of this section, we describe the details of each module in our DL Architecture as illustrated in Figure 4.

### 4.1 Image Encoder Module

Embedding visual data into low-dimensional space has been extensively studied using the *Convolutional Neural Networks (CNN)* [35]. This type of network receives input images as a matrix of their pixel values and extracts a feature vector of the input image through various convolutional and pooling layers.

COALA encodes images in the same fashion. However, instead of training a CNN model from scratch, it utilizes the *transfer learning* technique. Transfer learning allows us to leverage pre-trained CNN models such as ResNet [31], without the need for dealing with technical challenges of training a model from scratch on a big

enough set of training data for sufficient amount of time. That means, we leverage the knowledge of a pre-trained ResNet model and re-purpose it for icon image classification, a.k.a., fine-tuning.

To that end, we prepared a dataset of icon images with their annotated classes from Rico dataset [25]. Using a tool in prior work [38], Rico augmented more than 66,000 screenshots with semantic annotations which consist of icon classes. We extracted annotated icons in Rico dataset and manually checked the consistency of images with their annotated class. We also downsampled dominant classes of icon images to have a balanced dataset. We trained and fine-tuned a pre-trained ResNet18 classifier [31] using this data and used it in COALA for encoding icon images.

### 4.2 Context Encoder Module

The input of Context Encoder is the sources of information from usage context of the icon as shown in Table 1. The purpose of this module is to embed these parameters into a feature vector. In choosing a proper model for Context Encoder, we should consider three main characteristics of parameters. First, these parameters have two different types: categorical and textual. Second, textual parameters have a variable length. Third, often only a subset of parameters are available for a given icon.

To support both categorical and textual parameters, Context Encoder utilizes two different input embedding components, namely: one hot encoder [28] and a word embedding model, specifically GloVe [45].

One hot encoder maps the categorical parameters, i.e., *category* and *screen region*, to a binary vector. *Category* can take either one of the 53 categories that exist in Google Play or none if it was removed from Google Play by the time we checked. *Screen region* is one of the 9 zones, as shown in Figure 1, that an icon can belong to. Thus, the encoded vector should be at least 63 bits in length. However, it is zero-padded to have the same length as the vector representation of other parameters.

For textual parameters, similar to [40], Context Encoder first summarizes all the parameters into one sentence by joining the textual phrases with a dot, “.”, as a delimiter. For example, for the plus icon in Figure 2, joining the page title, i.e., “playlist”, and its cleaned android id, i.e., “create playlist”, results in the summary of “playlist.create playlist.” for textual parameters of the icon (we filtered “fab” token since it specifies the icon type, i.e., floating action button, and is not informative). Then, a pre-trained GloVe model is responsible for mapping each token in the summarized sentence to its vector representation. GloVe is an unsupervised pre-trained model that has proven its ability in capturing syntax and semantic regularities using vector arithmetic [45]. This is mainly because (1) it is not needed for the model to learn the exact vocabulary of these tokens, and (2) using a semantic preserving word embedding enables the model to better generalize to unseen tokens whose synonyms exist in the dataset.

Given the vector representation of icon information, Context Encoder utilizes a *Recurrent Neural Network (RNN)* [32]. RNNs are known for their chain-like structure, which makes them capable of learning from variable-length sequences of data. Specifically, we use a type of RNNs, *Long Short-Term Memory (LSTM)* networks, shown to be superior to the standard RNNs by avoiding the long-term dependency problem caused by Vanishing Gradients [20].

LSTMs consist of a chain of repeating modules, a.k.a., *LSTM cells*. The vector representation of icon information passes through the

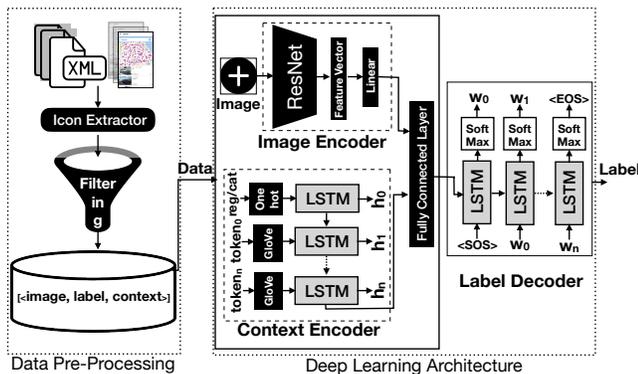


Figure 4: Overview of COALA framework

LSTM cells in which four neural network layers interact in a special way. Several adjustable weights control the information each cell remembers, forgets, or passes to the next cell (a.k.a., hidden state). During training, the model tunes these internal weights towards decreasing the overall training loss.

The output of the last LSTM cell is fused in a fully connected layer with the image embedding to provide the input for the Label Decoder module.

### 4.3 Label Decoder Module

Given the image and other information of an icon, the Label Decoder is responsible for generating natural language labels. Icon labels are variable-length sequences of tokens. Thus, a proper model should be able to generate accurate tokens of labels sequentially. To that end, COALA employs an LSTM network with an internal loop that lets it iteratively generate icon labels token by token.

At each time step, the Label Decoder chooses the most probable token from a vocabulary of tokens. COALA builds this vocabulary based on frequent tokens in the labels of the training set. This vocabulary also includes sos (Start of Sequence), eos (End of Sequence), and UNK (Unknown). Then, each label will be represented by a sequence of ids of its comprising tokens surrounded by sos and eos tokens. UNK stands for tokens of labels that are not in the vocabulary.

Label Decoder initializes the hidden state of the first LSTM cell with the encoder output. Similar to the LSTM network of Context Encoder module, four internal neural network layers with adjustable weights regulate the information flow through the network. Label Decoder also sends an sos token to the first LSTM cell to signal the start of the label generation process. Next the LSTM cells get the previously generated token as input and calculate the score of choosing each token in the vocabulary. A Softmax [14] layer gets the LSTM cell output to transform the scores to a probability function. The most probable token is the final output at each step. Label Decoder terminates this procedure when eos token is the output of current time step.

To train DL model, Label Decoder calculates cross-entropy loss function to measure the extent to which the predicted probability diverges from the actual token. COALA trains the whole DL architecture end-to-end. Thereby, this loss function back propagates through the whole network, i.e., encoder and decoder, to adjust the internal weights.

During training, the aforementioned process of passing the last generated token to the next LSTM cell can be followed. However, this process results in model instability and slow convergence [27]. To alleviate these issues, COALA uses *Teacher Forcing* strategy. Teacher forcing is a training-time procedure in which the model receives the ground truth output  $y_t$  as input at time step  $t + 1$  [56]. This means Label Decoder passes the  $t_{th}$  token of the target label as the input to the  $t + 1_{th}$  LSTM cell during training. However, in the testing phase, we pass the previously generated tokens to the next LSTM cell and we only use the ground truth labels to calculate the performance of the model.

## 5 DL MODEL ASSESSMENT

To study the impact of imbalanced training data on the state-of-the-art model, LabelDroid, and also the effectiveness of our context-aware approach, we study the following research questions:

**RQ4.** How effective is LabelDroid in practice?

**Table 2: Details of COALA dataset**

	App	Activity	Icon
Train	7,728	14,230	17,462
Test	965	1,821	2,274
Validation	965	1,788	2,128
Total	9,658	17,839	21,864

**RQ5.** To what extent is the DL architecture of LabelDroid capable of coping with imbalanced data?

**RQ6.** How effective is the context-aware model of COALA in label prediction? To what extent COALA outperforms the context-agnostic model of LabelDroid?

**RQ7.** To what extent the labels provide an informative explanation of the icon functionality?

**RQ8.** How long does it take for COALA to train and predict labels?

## 5.1 Experimental Setup

**5.1.1 Datasets.** We split the dataset of icons introduced in Section 3.1 into three separate sets with respect to apps. In this way, train, validation, and test set are respectively 80%, 10%, and 10% of apps selected randomly. Table 2 shows the details of our dataset. Note that we run our experiments 5 times using different random partitioning of the data to minimize evaluation bias. This means in a different random partitioning, the number of icons and screens may be slightly different in each partition since we split based on apps.

Moreover, as we aim to study the effects of imbalanced data on LabelDroid, inspired by prior work [23], we created balanced datasets by downsampling dominant classes of data. A parametric Sigmoid function on the inverse label frequency manages the balanceness of the dataset. Sigmoid parameter adjusts the number of frequent labels included in the sampled data by controlling the steepness of its curve. We experimented with 5 different parameters of Sigmoid, resulting in balanced datasets of size {2,557, 3,931, 7,599, 11,601, 15,495}. The smallest dataset is completely balanced with one instance for each label.

**5.1.2 DL Implementation and Configurations.** Our DL model is implemented in PyTorch [44], a popular open-source Machine Learning library for Python. We utilized Adam optimizer [34] to update the internal weights iteratively based on the cross-entropy loss function. To prevent the predefined labels from overwhelming the network during training and producing a biased model, we adapted weighted cross-entropy [37] to enforce the model learn from the labels in minority. Each DL model has several configurable parameters, a.k.a., hyperparameters, that can impact the performance of the model. To tune these hyperparameters of the model, we also performed a guided grid search strategy to choose the values that had the best performance on the validation data. The details of our configurations are available on COALA’s website [6].

**5.1.3 Evaluation Metrics.** We evaluate the effectiveness of DL models using 4 evaluation metrics that are commonly used for image captioning problems, namely: BLEU [42], METEOR [19], ROUGH [36], CIDEr [52]. We also report *exact match*, i.e., the percentage of data for which the generated label is an exact match of the ground truth. That means, it only awards the model if *all* the tokens of the ground truth appear in the generated label with the same order. BLEU score, however, focuses on n-gram overlaps to measure the quality of the

**Table 3: LabelDroid’s effectiveness in generating predefined/non-predefined labels in their test set.**

	Exact_match	BLEU-1	BLEU-2	BLEU-3	METEOR	ROUGE_L	CIDEr
Predefined	0.90	0.90	0.90	0.82	0.69	0.91	4.54
Non-predefined	0.17	0.24	0.18	0.18	0.12	0.27	0.89
All	0.51	0.55	0.53	0.41	0.32	0.55	2.58

generated label. It calculates precision for n-grams, denoted by BLEU-1, BLEU-2, and BLEU-3, for n in {1, 2, 3}.

BLEU score has some drawbacks, for example in not considering sentence structure or word meanings, which has led to the advent of other evaluation metrics. METEOR (Metric for Evaluation of Translation with Explicit ORdering) is based on the harmonic mean of unigram precision and recall [19]. ROUGH is a set of recall-oriented measures, from which we use ROUGH\_L that is based on the Longest Common Subsequence in the ground truth and the generated label [36]. CIDEr (Consensus-based Image Description Evaluation) leverages term frequency-inverse document frequency (tf-idf) to measure the similarity of the ground truth and the generated label [52].

The implementation of these metrics is available in a Python library, called NLGEval library [51], which we have used to evaluate COALA.

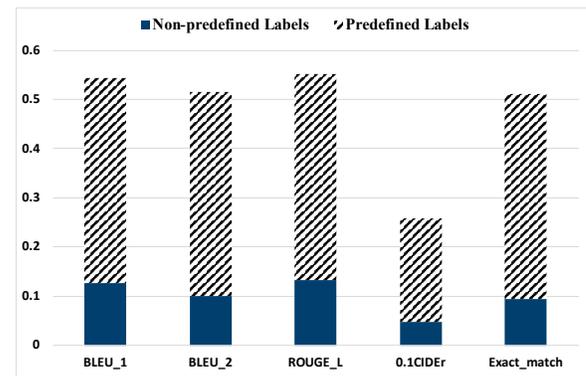
## 5.2 RQ4. LabelDroid’s Effectiveness

Our empirical study (Section 3) revealed a severely imbalanced label distribution for icons. We conducted an experiment to study whether overlooking this data-driven bias leads to misinterpretation of prior work’s effectiveness. For that purpose, we trained LabelDroid [22] on their dataset using their default configurations and evaluated the model on generating (1) *predefined* labels, which as introduced in Section 3.2 correspond to the default catalog of icons in widgets that come with Android Studio, and (2) *non-predefined* labels, i.e., all the icons in the test set except the ones with predefined labels. Among 1,876 icons in their test set, 866 of them have predefined labels and the remaining 1,010 icons have non-predefined labels. Note that for this research question, we use the original dataset of LabelDroid to only study the impact of data balance-ness and keep their approach as close as possible to their original version. For next questions, we use the dataset introduced in Section 5.1.1, which is the extended version of LabelDroid’s dataset since the dataset they used in their work lacks additional sources of information from icons.

Table 3 summarizes the results of this experiment. As shown in Table 3, the effectiveness of LabelDroid, in all metrics, is significantly higher in generating predefined labels than the non-predefined labels. The unfortunate outcome is that this variation impacts the overall result: resulting in an incorrect interpretation of the model’s effectiveness in predicting proper labels for unlabeled icons.

For a better illustration of the impact of imbalanced data on the overall effectiveness of LabelDroid, consider the stacked bar chart of Figure 5. Here, each bar indicates the ratio of correctly predicted labels according to a different metric. Within each bar, the solid blue fill indicates the ratio of correctly predicted non-predefined labels, while the dashed fill indicates the ratio of correctly predicted predefined labels. Figure 5 clearly shows the overall evaluation is highly impacted by the model’s effectiveness on predefined labels.

This observation indicates that the imbalanced data produced a biased model for predicting predefined labels. But the issue is substantially more severe than it may appear at first blush, since there is no point in predicting icons with predefined labels. After all, these are the labels of icons that are shipped with the popular Android Studio. In practice, it is a rare occurrence for Android Studio icons to appear in apps with no labels. This would only occur if the developer intentionally removes the label automatically associated with the icon by the IDE. The ultimate goal of label prediction is to generate labels for unlabeled icons, which often have non-predefined labels.

**Figure 5: Biased effectiveness of LabelDroid towards predefined labels**

## 5.3 RQ5. Impact of Balanced Data on LabelDroid’s Effectiveness

To further evaluate LabelDroid, we also studied its performance if it were to be trained on balanced data in predicting non-predefined labels. To that end, we performed balance sampling with regard to distinct labels of the training data to downsample dominant labels. Then, we trained a new model on the newly sampled training set and evaluated the model on the test set.

Figure 6 depicts the results of training LabelDroid on balanced data using CIDEr metric. The orange dashed line corresponds to the effectiveness of the model trained on balanced data. We should note that in balanced sampling, as well as changing the data distribution, we are reducing the amount of training data, which affects the model’s ability to learn. To monitor this variable, we also performed random downsampling on the training data to make the dataset have the same number of training data as in our balanced datasets. In figure 6, the blue solid line depicts the performance of the model trained on randomly sampled data. Figure 6 shows that having balanced training data does not improve LabelDroid’s effectiveness. The general trend in the blue solid line shows that reducing the training data slightly degrades the model’s effectiveness. However, increasing the data balance-ness drastically drops

**Table 4: Comparison of COALA and LabelDroid effectiveness in generating non-predefined labels**

model	Exact_match	BLEU-1	BLEU-2	BLEU-3	METEOR	ROUGH_L	CIDEr
COALA	0.38	0.4	0.2	0.15	0.21	0.489	1.34
LabelDroid	0.14	0.21	0.125	0.09	0.12	0.24	0.7

the model’s effectiveness. The same behavior was observed in the model’s effectiveness in generating non-predefined labels and when we used other evaluation metrics. However, to comply with the page limits, those results are available on COALA’s website [6].

This concludes that learning from balanced data did not provide any remarkable improvement in the effectiveness of LabelDroid in generating non-predefined labels.

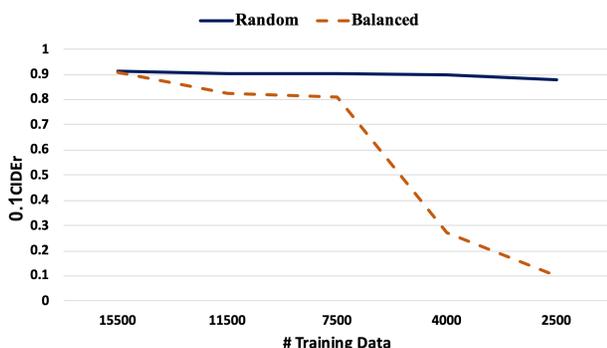
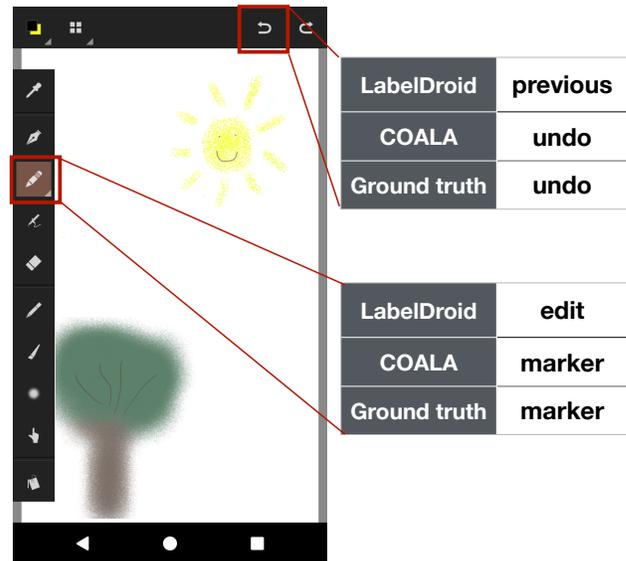
#### 5.4 RQ6. Effectiveness of COALA

The ultimate goal of COALA is to generate labels for unlabeled icons. Thus, our main objective is to evaluate COALA’s effectiveness in generating non-predefined labels and compare it with the prior work [22].

Table 4 summarizes the effectiveness of COALA and LabelDroid in generating non-predefined labels and shows the superiority of COALA over LabelDroid in all metrics. Figure 7 illustrates two icons in a painting app in our test set for which COALA was able to generate correct labels, but LabelDroid failed.

In this app, there are different icons for different painting tools such as a marker in a vertical, left toolbar, as well as other essential icons in the horizontal, top toolbar. It is clear that without considering the context of these icons, generating the correct label may be impossible. For example, the Marker icon in the vertical toolbar has been widely used to denote the “Edit” icon. However, by considering the adjacent icons, COALA was able to detect the icon as a painting tool. For the Undo icon, in addition to commonly co-occurred icons, existing hints in their textual information, such as Android *id*, enabled the model to generate accurate labels. More examples of LabelDroid’s failures, for which COALA was able to generate correct labels are available on our website [6].

We also examined common failure scenarios for COALA. As shown in Figure 8, COALA has generated incorrect labels, particularly for icons 1 and 2. Due to the black-box nature of DL models, pinpointing the exact reason for these failures is not possible. However, examining the icons gives us an overview of probable culprits. In the first snapshot, the contextual information for minus icon is not

**Figure 6: Overall evaluation of LabelDroid model trained on re-sampled data****Figure 7: Examples of inability of the context-agnostic model, LabelDroid, in generating correct labels.**

informative. Thereby, COALA was unable to recognize its functionality. Furthermore, the existence of the token “wheel” in the identifier name of its sibling may have confused the model to generate a wrong label. There are also some failures in generating labels for infrequent icons, e.g., icon 2. Not having sufficient training data to help the model learn the icon is a probable cause of this failure.

Besides these model failures, there are certain cases that are not mistakes, yet penalize the effectiveness of our model in the manner evaluated here. Case in point, consider icon 3 in Figure 8, where COALA has generated more valid tokens, possibly conveying more useful information to a blind user than the ground truth. Additionally, the generated labels may be semantically valid alternatives, as in icons 4 and 5 in Figure 8. There are also some cases, such as icon 6, in which the ground truth is invalid, not the generated label. These examples indicate that in practice COALA is more effective than the NLP metrics suggest, since they are simply comparing the generated labels against the ground truth. These metrics do not account for situations in which COALA either generates a semantically equivalent label as the ground truth label, or the ground truth itself is wrong. This observation motivated us to conduct a user study to better evaluate COALA.

#### 5.5 RQ7. Informative Explanation for Users

In addition to the automated evaluation of models using metrics described in Section 5.1.3, we conducted a user study to understand the quality of automatically generated labels by LabelDroid and COALA. Since manual investigation of all icons in our test set was not practical, we randomly selected a sample from the test data

	1	2	3
			
LabelDroid	Remove	<UNK>	Refresh
COALA	Go	<UNK> menu	Refresh result
Ground truth	Decrease Value	Filter call	Refresh
	4	5	6
			
LabelDroid	Open	Navigate up	Pause
COALA	Open menu	Hide sidebar	Pause
Ground truth	Open navigation drawer	Collapse	Congratulations

Figure 8: Examples of failures in label generation using COALA and LabelDroid

based on the image classes of icons. This ensures having a representative and balanced dataset of icons with different images (e.g., plus, backward-arrow, etc.). We used our icon image classifier introduced in Section 4.1 to get image classes and randomly selected up to 10 icons from each class, resulting in 198 icons from 61 distinct image classes. Next, we highlighted the icon under investigation on the screenshot of the app as shown in Figure 9. For each icon, we show four types of labels (1) *ground truth*, which is the content description extracted from the XML layout, (2) the generated label by LabelDroid, (3) the generated label by COALA, and (4) a *random* label, different from the ground truth, selected from all labels in the dataset. We then used Google form to display the icons on the screenshots, as well as four different labels to the users and get their responses. We shared the survey on social media and asked volunteers to rate the quality of the labels from 1 to 5 after reading an instruction (the instruction is available at [6]). To avoid bias, we collected and aggregated responses of at least three different users for each icon, resulting in 730 answers in total.

Prior to analysis, we filtered out the unreliable data as follows:

- **Incomprehensible icons.** The highlighted area on a screenshot may not specify a proper icon due to capturing the screenshot of an app at a transition point during app exploration. Alternatively, the user may not be able to determine the functionality of a designated icon from the screenshot. We remove such images from our analysis by asking users if the icon on the screenshot is valid and understandable.
- **Inconsistent scores.** We expect users to assign the same score to the same labels of an icon; otherwise, there is inconsistency in scores. For example, the ground truth and COALA’s candidates for an icon can both be “collapse”, and users should assign the same score to both of them. We remove all scores of users who have inconsistent scores since such users are not reliable.
- **Insufficient rating.** We have a threshold of three responses for each icon. Thus, if applying the prior filtering steps drops the number of responses for an icon below three, we remove the icon entirely.

Our filtering criteria removed 216 responses, resulting in 514 responses for 156 icons. We then aggregated the users’ responses

Table 5: Statistical analysis of scores. Given the significance level of 0.05, the scores of COALA’s labels are significantly better than the scores of LabelDroid’s labels.  $\bar{\mu}_{Diff}$  is the average of difference between score lists.

H0 (Null Hypothesis)	$\bar{\mu}_{Diff}$	p-value
Ground truth - COALA = 0	0.76	5.14e-6
Ground truth - LabelDroid = 0	1.08	8.68e-10
COALA - LabelDroid = 0	0.33	1.7e-2
Ground truth - Random = 0	2.78	2.94e-25
COALA - Random = 0	2.02	7.49e-20
LabelDroid - Random = 0	1.67	6.43e-16

by calculating the average of all the scores for each icon. Therefore, for each type of labels in {ground truth, COALA, LabelDroid, and Random} there is a list of 156 scores corresponding to each icon, which we call *score list*. The average of all score lists of ground truth, COALA, LabelDroid, and Random are 3.91, 3.15, 2.83, and 1.13, respectively.

To determine if the differences observed between the means of score lists are statistically significant, we performed hypothesis testing. Since the scores failed the Shapiro-Wilk normality test [48], we performed non-parametric testing using Wilcoxon signed-rank test [55] with significance level of 0.05. Table 5 shows the result of this analysis. As seen on the upper half of the table, the quality of ground truth labels is better than the quality of labels generated by COALA and LabelDroid, since the mean of ground truth’s score list is significantly better than the mean of COALA’s and LabelDroid’s score list (p-value equal to 5.14e-6 and 8.68e-10 respectively). Similarly, it shows that the quality of labels generated by COALA is significantly higher than the quality of labels generated by LabelDroid (p-value=1.7e-2). Moreover, the lower half of the table shows that labels of ground truth, COALA, and LabelDroid have higher quality than random labels.

An outcome of this analysis is that although COALA significantly improves the state-of-the-art technique in generating natural language labels for icons, it is still not as good as the labels provided by actual developers. This observation suggests that there is still room for improvement and further research in this area.

## 5.6 RQ8. Performance

To answer this research question, we evaluated the time required to train a new model and used the resulting model to generate a label for an icon. We ran the experiments on a Ubuntu computing cluster using an NVIDIA GP102 GPU and 128G memory. It took 241 minutes on average for COALA to train a new model on our dataset. This time includes evaluating the model at each time step on the validation set for model selection purposes. However, it took only 17 milliseconds on average for the trained model to generate the label of an icon given its specification. This indicates that COALA is efficient for use in a variety of settings, including automated repair of inaccessible apps, and inclusion in screen readers to dynamically resolve unlabeled icons.

## 6 THREATS TO VALIDITY

**Sampling bias:** The selection of Android apps in this study may introduce bias. We mitigated this threat by exploring another dataset, RICO [25]. We obtained similar results as that reported here. The results of our study on RICO dataset are available online [6]. Moreover, both LabelDroid and RICO datasets consist of more than 20

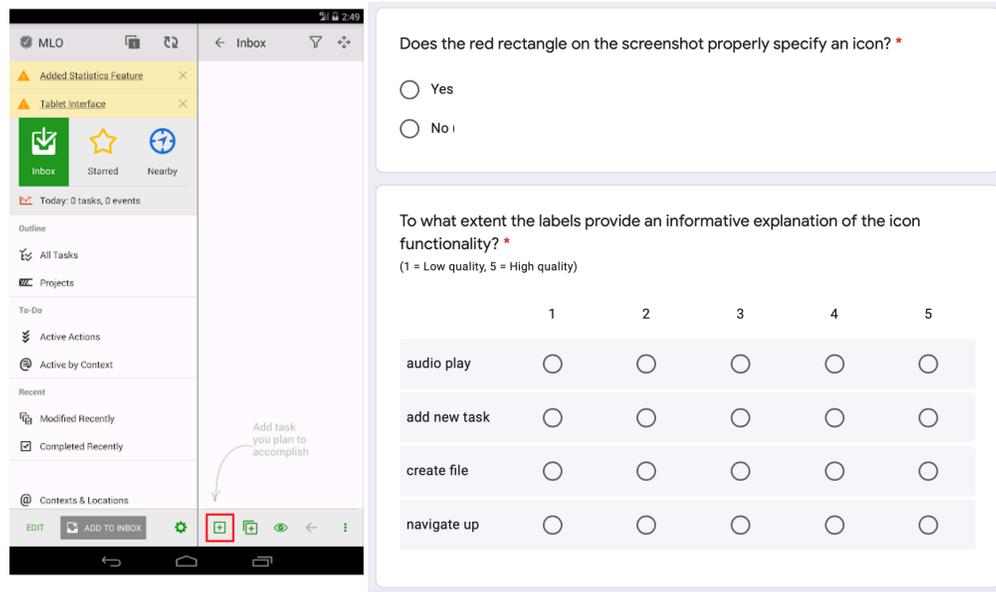


Figure 9: A sample question from the user study.

thousands apps selected from various categories of Google Play store.

**Learner bias:** For the empirical study on DL models, we use two architectures, COALA and LabelDroid. One possible threat to the validity of our results is the choice of the neural-network modules and hyper parameters of our models. For COALA, our focus was studying the impact of incorporating different sources of information using a well-known architecture. Also, for training LabelDroid, we used their original implementation and hyper-parameters.

**Evaluation bias:** We evaluated LabelDroid and COALA under the same evaluation metrics used in LabelDroid’s publication and consistent with natural language processing literature. We report the effectiveness of models on a test set, left out from the whole dataset of labeled icons. However, the generalizability of models on unlabeled icons needs to be studied further. This signifies the need for creating high quality benchmarks of icons in Android apps in future. We further reduce the evaluation bias by running our experiments 5 times and averaging the results.

## 7 RELATED WORK

Accessibility issues have been extensively studied for websites [29, 33] and more recently for mobile apps either in specific categories such as e-government [50], smart cities [39], and health [54] or in general [18, 24, 43, 47, 49, 53, 57]. The increased awareness of the prominence of accessibility issues has motivated the development of several accessibility guidelines, and accessibility assessment and repair tools.

**Accessibility guidelines:** The World Wide Web Consortium (W3C) [17] is the main organization in determining protocols and standards for websites whose primary initiative is to develop accessibility standards. They have provided detailed tutorials for constructing inclusive web pages [16]. For mobile apps, Google and Apple, the primary organizations facilitating the app marketplace, have published accessibility guidelines for Android and iOS developers [4, 7]. Despite the existence of these guidelines, according

to [18], developers are still not aware of the issues or find it costly to address them.

**Android accessibility evaluation and repair tools:** Accessibility evaluation tools leverage static analysis and/or dynamic analysis techniques to report various accessibility issues. Lint [10] is a static tool that checks project files and warns the developers about missing labels. Espresso [8], Robolectric [12] and Accessibility Scanner [1] are based on Accessibility Testing Framework of Android [2], with the capability to dynamically scan the app for accessibility issues [15]. PUMA [30], MATE [26], and IBM Mobile Accessibility Checker (MAC) [57] are other dynamic testing frameworks that check accessibility issues at runtime.

Despite several tools for accessibility assessment, only a few repair tools are available to fix accessibility issues for blind users. To enable runtime accessibility repair and enhancement for Android, Zhang et al. [58] proposed interaction proxies to layer on top of the original implementation of app. In their subsequent work [59], they utilize this platform for social annotation of GUI elements for missing labels. Different from their work, COALA is capable of automatically generating labels for icons by learning from the previously labeled ones.

Furthermore, Liu et al. [38] utilize a deep learning classifier to semantically annotate icon images based on around 100 categories they defined for the icons. Although their initiative was not fixing accessibility issues, screen readers can take advantage of their proposed textual annotations for unlabeled icons. Unlike their work, COALA generates textual labels not only from icon images but also from their usage context. In this work, we leverage from their annotated icons to fine-tune an image classifier which is capable of embedding icon images.

The most relevant work to our study is LabelDroid [22], which is a context-agnostic model for generating labels for icons. As discussed heavily throughout the paper, LabelDroid’s insufficient representation of Android icons only by their images, as well as its biased model negatively affect its effectiveness.

## 8 CONCLUSION AND FUTURE WORK

Missing labels seriously hinder blind users' ability to interact with mobile apps. In this work, we studied the characteristics of icon labels and demonstrated how overlooking the imbalanced nature of labels result in a biased deep-learning model. We also presented COALA, a context-aware label generation approach for icons in Android. Our experimental results show that by incorporating additional sources of information, COALA could outperform the prior work [22] in automatically generating labels for unlabeled icons.

In future, we will explore incorporating additional sources of information from source code to enrich icon representation and improve the accuracy of our model by studying different DL models. We also aim to integrate our model with (1) IDE analysis tools, such as Lint, to not only detect missing labels, but to also recommend fixes, and (2) screen readers to facilitate blind users' interactions with apps.

Our research artifacts are available to the public [6].

## ACKNOWLEDGMENTS

This work was supported in part by award number 1823262 from the National Science Foundation and an Exploration award from the School of Information and Computer Sciences at the University of California, Irvine. We are grateful to the authors of LabelDroid [22] for graciously sharing their research artifacts with us. We would like to thank the anonymous reviewers of this paper for their detailed feedback, which helped us improve the work.

## REFERENCES

- [1] 2021. Accessibility Scanner. <https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor>
- [2] 2021. Accessibility Testing Framework for Android. <https://github.com/google/Accessibility-Test-Framework-for-Android>
- [3] 2021. Alternative Text. <https://webaim.org/techniques/alttext/>
- [4] 2021. Build more accessible apps. <https://developer.android.com/guide/topics/ui/accessibility>
- [5] 2021. Camel case. [https://en.wikipedia.org/wiki/Camel\\_case](https://en.wikipedia.org/wiki/Camel_case)
- [6] 2021. COALA. <https://github.com/fmehralian/COALA>
- [7] 2021. Content. <https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/content/>
- [8] 2021. Espresso : Android Developers. <https://developer.android.com/training/testing/espresso>
- [9] 2021. Functional Images. <https://www.w3.org/WAI/tutorials/images/functional/>
- [10] 2021. Improve your code with lint checks. <https://developer.android.com/studio/write/lint>
- [11] 2021. LabelDroid - ICSE Best Paper Award. <https://tinyurl.com/yxnydyovk>
- [12] 2021. robolectric/robolectric. <https://github.com/robolectric/robolectric>
- [13] 2021. Snake case. [https://en.wikipedia.org/wiki/Snake\\_case](https://en.wikipedia.org/wiki/Snake_case)
- [14] 2021. Softmax. <https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html>
- [15] 2021. Test your app's accessibility. <https://developer.android.com/guide/topics/ui/accessibility/testing>
- [16] 2021. Web Accessibility Tutorials: Images Concepts. <https://www.w3.org/WAI/tutorials/images/>
- [17] 2021. Web Content Accessibility Guidelines (WCAG). <https://www.w3.org/WAI/standards-guidelines/#wcag>
- [18] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE.
- [19] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*.
- [20] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* (1994).
- [21] Rupert Bourne, Jaimie Adelson, Seth Flaxman, Paul Briant, Michele Bottone, Theo Vos, Kavin Naidoo, Tasanee Braithwaite, Maria Cicinelli, Jost Jonas, Hans Limburg, Serge Resnikoff, Alex Silvester, Vinay Nangia, and Hugh Taylor. 2020. Global Prevalence of Blindness and Distance and Near Visual Impairment in 2020: progress towards the Vision 2020 targets and what the future holds. *Investigative Ophthalmology and Visual Science* (2020).
- [22] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, and Guoqiang Li. 2020. Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE.
- [23] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- [24] Rafael Jeferson Pezzuto Damaceno, Juliana Cristina Braga, and Jesús Pascual Mena-Chalco. 2018. Mobile device accessibility for the visually impaired: problems mapping and recommendations. *Universal Access in the Information Society* (2018).
- [25] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschan, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual Symposium on User Interface Software and Technology*.
- [26] Marcelo Medeiros Eler, José Miguel Rojas, Yan Ge, and Gordon Fraser. 2018. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation*. IEEE.
- [27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [28] John T Hancock and Taghi M Khoshgoftaar. 2020. Survey on categorical data for neural networks. *Journal of Big Data* (2020).
- [29] Vicki L Hanson and John T Richards. 2013. Progress on website accessibility? *ACM Transactions on the Web* (2013).
- [30] Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. 2014. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*.
- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- [32] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* (1997).
- [33] Shaun K Kane, Jessie A Shulman, Timothy J Shockley, and Richard E Ladner. 2007. A web accessibility report card for top international university web sites. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility*.
- [34] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.
- [36] Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.
- [37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*.
- [38] Thomas F Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning design semantics for mobile apps. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*.
- [39] Higinio Mora, Virgilio Gilart-Iglesias, Raquel Pérez-del Hoyo, and Maria Dolores Andújar-Montoya. 2017. A comprehensive system for monitoring urban accessibility in smart cities. *Sensors* (2017).
- [40] Son Nguyen, Hung Phan, Trinh Le, and Tien N Nguyen. 2020. Suggesting Natural Method Names to Check Name Consistencies. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE.
- [41] Liam Paninski. 2003. Estimation of entropy and mutual information. *Neural computation* (2003).
- [42] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*.
- [43] Kyudong Park, Taedong Goh, and Hyo-Jeong So. 2014. Toward accessible mobile application design: developing mobile application accessibility guidelines for people with visual impairment. *HCI Korea* (2014).
- [44] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
- [45] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing*. <http://www.aclweb.org/anthology/D14-1162>
- [46] Leonard Richardson. 2007. Beautiful soup documentation. *April* (2007).
- [47] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*.
- [48] Patrick Royston. 1992. Approximating the Shapiro-Wilk W-test for non-normality. *Statistics and computing* (1992).
- [49] Navid Salehnamadi, Abdulaziz Alshayban, Jun-Wei Lin, Iftekhar Ahmed, Stacy Branham, and Sam Malek. 2021. Latté: Use-Case and Assistive-Service Driven Automated Accessibility Testing Framework for Android. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

- [50] Leandro Coelho Serra, Lucas Pedrosa Carvalho, Lucas Pereira Ferreira, Jorge Belimar Silva Vaz, and André Pimenta Freire. 2015. Accessibility evaluation of e-government mobile applications in Brazil. *Procedia Computer Science* (2015).
- [51] Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. 2017. Relevance of Unsupervised Metrics in Task-Oriented Dialogue for Evaluating Natural Language Generation. (2017).
- [52] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- [53] Christopher Vendome, Diana Solano, Santiago Liñán, and Mario Linares-Vásquez. 2019. Can everyone use my app? An Empirical Study on Accessibility in Android Apps. In *2019 IEEE International Conference on Software Maintenance and Evolution*. IEEE.
- [54] Fahui Wang. 2012. Measurement, optimization, and impact of health care accessibility: a methodological review. *Annals of the Association of American Geographers* (2012).
- [55] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. Springer, 196–202.
- [56] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* (1989).
- [57] Shunguo Yan and PG Ramachandran. 2019. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing* (2019).
- [58] Xiaoyi Zhang, Anne Spencer Ross, Anat Caspi, James Fogarty, and Jacob O Wobbrock. 2017. Interaction proxies for runtime repair and enhancement of mobile application accessibility. In *Proceedings of the 2017 CHI conference on human factors in computing systems*.
- [59] Xiaoyi Zhang, Anne Spencer Ross, and James Fogarty. 2018. Robust Annotation of Mobile Application Interfaces in Methods for Accessibility Repair and Enhancement. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*.