# DescribeCtx: Context-Aware Description Synthesis for Sensitive Behaviors in Mobile Apps

Shao Yang[1] Yuehan Wang[2] Yuan Yao[3] Haoyu Wang[4] Yanfang (Fanny) Ye[5] Xusheng Xiao[1]

[1]Case Western Reserve University, [2]University of Illinois at Urbana-Champaign

[3]State Key Laboratory for Novel Software Technology, Nanjing University

[4]Beijing University of Posts and Telecommunications, [5]University of Notre Dame

[1]{sxy599,xusheng.xiao}@case.edu, [2]yuehanw2@illinois.edu, [3]y.yao@nju.edu.cn, [4]haoyuwang@bupt.edu.cn, [5]yye7@nd.edu

## ABSTRACT

While mobile applications (i.e., apps) are becoming capable of handling various needs from users, their increasing access to sensitive data raises privacy concerns. To inform such sensitive behaviors to users, existing techniques propose to automatically identify explanatory sentences from app descriptions; however, many sensitive behaviors are not explained in the corresponding app descriptions. There also exist general techniques that translate code to sentences. However, these techniques lack the vocabulary to explain the uses of sensitive data and fail to consider the context (i.e., the app functionalities) of the sensitive behaviors. To address these limitations, we propose DescribeCtx, a context-aware description synthesis approach that trains a neural machine translation model using a large set of popular apps, and generates app-specific descriptions for sensitive behaviors. Specifically, DescribeCtx encodes three heterogeneous sources as input, i.e., vocabularies provided by *privacy policies*, behavior summary provided by the *call graphs* in code, and contextual information provided by *GUI texts*. Our evaluations on 1,262 Android apps show that, compared with existing baselines, DescribeCtx produces more accurate descriptions (24.96 in *BLEU*) and achieves higher user ratings with respect to the reference sentences manually identified in the app descriptions.

## CCS CONCEPTS

• **Theory of computation** → **Program analysis**; • **Security and privacy** → **Software security engineering**.

## KEYWORDS

mobile apps; description synthesis; static analysis; deep learning

**ACM Reference Format:**

Shao Yang, Yuehan Wang, Yuan Yao, Haoyu Wang, Yanfang (Fanny) Ye, Xusheng Xiao. 2022. DescribeCtx: Context-Aware Description Synthesis for Sensitive Behaviors in Mobile Apps. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE'22), May 22–27, 2022, Pittsburgh, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/-3510003.3510058

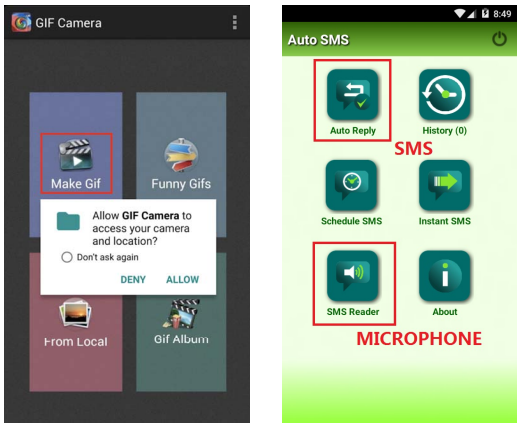


*ICSE '22, May 22–27, 2022, Pittsburgh, PA, USA*



ACM ISBN 978-1-4503-9221-1/22/05…$15.00

https://doi.org/10.1145/3510003.3510058




(a) GIF camera (b) Auto SMS

**Figure 1: Example apps' GUIs: (a) permission request; (b) GUI context for SMS and microphone permissions.**

## 1 INTRODUCTION

Mobile applications (i.e., apps) are becoming capable of handling nearly all kinds of needs from users [63]. However, certain behaviors of apps are less desirable or even harmful [28, 35, 44, 60, 86], such as disclosing users' location in the background [22, 44, 61, 86]. Mainstream smartphone platforms (i.e., Android and iOS) require users to grant permissions for permission requests at run time [36, 55] (Figure 1(a)). However, as shown in research studies [23, 45, 74, 75], most apps provide only information about *what* permissions they request, rather than *how and why* they will use the permission-protected data, causing users to make uninformed decisions on their privacy.

Recognizing these limitations, existing research efforts [56, 59] have been made to provide descriptions by identifying sentences in app descriptions. App descriptions describe apps' functionalities and features, and may include sentences that provide justifications for certain permissions. However, as most app descriptions or privacy policies do not explain *every* sensitive behavior of the app, these methods can only provide descriptions for a small subset of sensitive behaviors. Our empirical study on 1,292 popular apps from Google Play [27] shows that only 37.37% permission uses are explained in app descriptions. For example, "Wish" is an uprising shopping app with more than 500M downloads. It requests multiple dangerous permissions (such as LOCATION, STORAGE, CONTACTS, and

CAMERA) without providing any information regarding how these dangerous permissions will be used in the app description.

Alternatively, there exist techniques [33, 40, 46, 62, 82] that generate comments for code, or translate permission-protected API calls based on a pre-defined description model. Unfortunately, these techniques mainly generate functionality descriptions by utilizing the words in comments and identifiers, and *lack the vocabulary* to explain the uses of sensitive data. More importantly, these techniques fail to consider the *context of sensitive behaviors*, i.e., what functionality justify the apps' sensitive behaviors (e.g., car navigation justifying the use of gps locations), making the generated sentences less accurate and informative.

In this paper, we propose **a context-aware description synthesis approach, DescribeCtx, to generate app-specific descriptions for sensitive behaviors**. Specifically, inspired by the recent advances of neural machine translation [12, 16, 50, 66], DescribeCtx trains a sequence-to-sequence (seq2seq) model by using the apps' code and contextual features extracted from a large set of popular apps. *The synthesized descriptions can be used to directly improve the descriptions shown in permission requests (e.g., the prompts for requesting permissions), or assist developers in writing customized messages for explaining the permission requests, helping promote better security practices in mobile apps.*

Due to the unique characteristics of mobile apps, there are three major challenges to train DescribeCtx: ① identifying the representative features for the sensitive behaviors in the code and the apps' contextual information, ② building the vocabulary for describing the sensitive behaviors, and ③ extracting the reference sentences to supervise the model learning. To address these challenges, DescribeCtx is built upon the following key insights:

- *Representation of sensitive behaviors*: sensitive behaviors are triggered by permission-protected API calls. Call graphs that lead to permission-protected API calls, referred to as *permission-protected-API call graph* (PACG), have been shown to be effective in representing the contexts of sensitive behaviors for detecting undesired behaviors [20, 78, 82, 83]. Thus, the PACG of a sensitive behavior provides a representative summary of the sensitive behavior's context in the code.
- *Representation of contextual information*: an app's GUI includes the contextual information that describes the app's functionality and justifies the uses of certain permissions. As shown in Figure 1(b), the GUI provides the contextual information of the apps' functionality for sending SMSs, and we can infer that the intention of the "Auto Reply" button is to reply SMSs. Thus, the text in the GUI where a sensitive behavior is triggered provides the contextual information for the sensitive behavior.
- *Vocabularies for sensitive behaviors*: as shown in existing studies [14, 59, 69], privacy policies contain information such as what kind of data will be collected and what permissions are required in this app. Thus, privacy policies can provide the vocabulary for the apps' sensitive behaviors.
- *Reference sentences for sensitive behaviors*: in order to assess the quality of the synthesized descriptions during both training and testing stages, we need high-quality reference sentences that describe sensitive behaviors in apps. As indicated by recent studies [56, 59], app descriptions contain sentences that explain permission uses. Since these descriptions are provided by the app developers who own the apps' code and are mainly used to promote the apps, they are more concise and easy for the users to understand. Thus, the sentences that describe sensitive data uses in app descriptions can be used as the reference sentences for sensitive behaviors.

Based on these key insights, DescribeCtx represents the features of apps' PACGs, GUI texts, and privacy policies as the sources and the reference sentences in app descriptions as the targets, and learns the conversion from the sources to the target. Furthermore, apps may lack permission descriptions for sensitive behaviors in their privacy policies. As shown in existing work [13, 23, 68] and our empirical results in Section 5.2, most apps use permissions for similar purposes, and thus DescribeCtx borrows the relevant permission descriptions from the training apps for these apps.

We evaluate DescribeCtx using popular Android apps downloaded from Google Play.[1] Within our affordable effort, we examine 8,814 apps in total, and then curate an evaluation dataset consisting of 1,292 popular Android apps among them (20.86 *GB*, 16.93 *MB* on average). All of these apps contain permission descriptions in privacy policies and reference sentences of sensitive behaviors in app descriptions, so that DescribeCtx can use the permission descriptions to train the seq2seq model and use the reference sentences to evaluate DescribeCtx. We measure the performance using both standard metrics in machine translation (*BLEU* [57] and *ROUGE-L* [42]) and user studies.

The results show that DescribeCtx achieves *BLEU* score 24.47 when apps' own privacy policies are missing and *BLEU* score 25.75 when they are available, which is at least 15.31% better than the state-of-the-art description synthesis approaches: Code2Vec [4, 5] and DescribeMe [82]. Furthermore, the user study results show that on average, DescribeCtx achieves higher ratings for syntactical correctness and semantic closeness than the baselines, with relative improvements over 5.13% and 9.91%, respectively. These results indicate that as most permissions are used for similar purposes, even if apps do not provide descriptions in privacy policies, **the guidance of behavior representation, GUI contextual text, and the relevant permission descriptions of other apps' privacy policies can be leveraged to synthesize accurate descriptions**.

In summary, the paper makes the following major contributions:

- A novel approach, DescribeCtx, that synthesizes natural language descriptions for sensitive behaviors in apps.
- A novel set of techniques that extract features from GUIs, privacy policies, and PACGs of apps and a novel model that combines these features to synthesize descriptions for sensitive behaviors.
- An evaluation on 1,292 apps that demonstrates the effectiveness of DescribeCtx and the improvement over the state-of-the-art.
- A prototype implementation of DescribeCtx and the results that are both publicly available [2].

[1] We focus on Android apps due to its market dominance [63] and open source ecosystem, but the general idea is applicable to other smartphone platforms such as iOS.

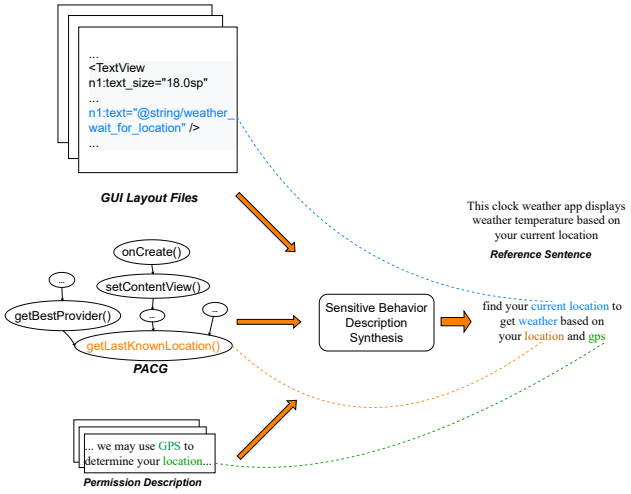



**Figure 2: Motivating example of DescribeCtx.**

## 2 BACKGROUND

### 2.1 App Descriptions and Privacy Policies

An app description is a short summary that describes the app's functionality, justifying the sensitive data it will need. For example, the "Facebook" app declares that its functionality include "share photos, videos, and your favorite memories, find local social events, and make plans to meet up with friends", and "backup photos by saving them in albums". This implies that users have to provide the app with the corresponding permissions to access the camera, the location data, and the storage. A privacy policy is legally required by Google Play. In most cases, a privacy policy should specify the data being collected and the user rights. Additionally, if the app uses any permission in the following dangerous permission groups, i.e. `CALENDAR`, `CONTACTS`, `LOCATION`, `CAMERA`, `SMS`, `STORAGE`, `PHONE`, `MICROPHONE`, the developers have to disclose the use of the permission because these permission groups collect personal sensitive data from the app users.

### 2.2 Attentive Seq2seq Model

From a probabilistic perspective, translation from a given source sentence $x = (x_1, ...x_n)$ to a target sentence $y = (y_1, ...y_n)$, is equivalent to finding the maximum conditional probability, i.e. $argmax_y p(y|x)$. In neural machine translation, a parameterized model, such as seq2seq [64], is trained to maximize the conditional probability of sentence pairs. A typical seq2seq model has two recurrent neural networks (RNNs) [25]. One RNN (encoder) is used to transform source natural language sequences into vector representations. The other RNN (decoder) is used to transform the vector representations to the target natural language sequences. As the length of input sequence grows, Cho et al. [15] show that the performance of traditional seq2seq architecture deteriorates drastically. In order to address this issue, Bahdanau et al. [12] extend the typical seq2seq model with attention mechanism, where the attentive seq2seq adaptively learns the weights of each input word at the decoding stage.

## 3 MOTIVATION EXAMPLE

Figure 2 shows how DescribeCtx extracts features and synthesizes the description for the sensitive behavior triggered by the permission-protected API `getLastKnownLocation`, which requires the permission `ACCESS_COARSE_LOCATION`. This example is simplified from the app "mobi.infolife.ezweather.widget.liveweather_forcast". DescribeCtx extracts three types of features from the app. The first feature is the text in the GUIs, referred to as *GUI contextual text*, which provides rich information to inform users about the apps' purposes and expected functionalities [7, 34, 51, 76]. The second feature is the text in the PACG, which is a call graph that leads to the permission-protected API `getLastKnownLocation`, and contains not only direct method calls, but also edges representing Inter-Component Communications (ICC) [41, 53, 54, 85], lifecycle method calls [9, 70], and multithreading method calls [81]. The third feature is the permission description of `LOCATION` in its privacy policy (i.e., "...we may use GPS to determine your location...").

These three types of features form three heterogeneous input sequences, and are fed into DescribeCtx's synthesis model, which is trained using popular apps in Google Play. As shown in Figure 2, the synthesized description is quite similar to the reference sentence, achieving a *BLEU* score of 46.06. The colored lines show how the three types of features contribute to the words in the synthesized description. If DescribeCtx discards GUI contextual text from the model and retrains the model, the synthesized description (i.e., "the app can use your location and gps") will lose some important app-specific information (i.e., weather). In fact, we have conducted an empirical study on 1,262 apps to measure the *word uniqueness* of the GUI contextual text, and the results show that GUI contextual text can provide 10.46% unique words for the synthesized descriptions. Furthermore, if DescribeCtx further discards both GUI contextual text and PACG from the model, the synthesized description becomes very short and general (i.e., "access location and gps"). This demonstrates the important roles played by privacy polices, GUI contextual texts, and PACGs in DescribeCtx.

## 4 APPROACH

### 4.1 Overview

Figure 3 shows the overview of DescribeCtx. DescribeCtx consists of two phases: training phase and prediction phase. The training phase has two major steps: (1) feature extraction and (2) sensitive behavior-description learning. In the feature extraction step, DescribeCtx accepts the APK files and privacy policies of training apps as input, and extracts three text sequences (i.e., PACGs, GUI contextual texts, and privacy policies) from each app. In the sensitive behavior-description learning step, DescribeCtx trains a description synthesis model based on the three inputs and the reference sentences in app descriptions. In the prediction phase, given an app with sensitive behaviors, DescribeCtx extracts the PACGs and the GUI contextual texts from the app's APK file. Then, the permission description selection module checks whether the app's privacy policy provides descriptions for the permissions used by the sensitive behaviors. If the privacy policy contains related permission descriptions, it directly extracts such descriptions; otherwise, it uses the most relevant permission descriptions from the

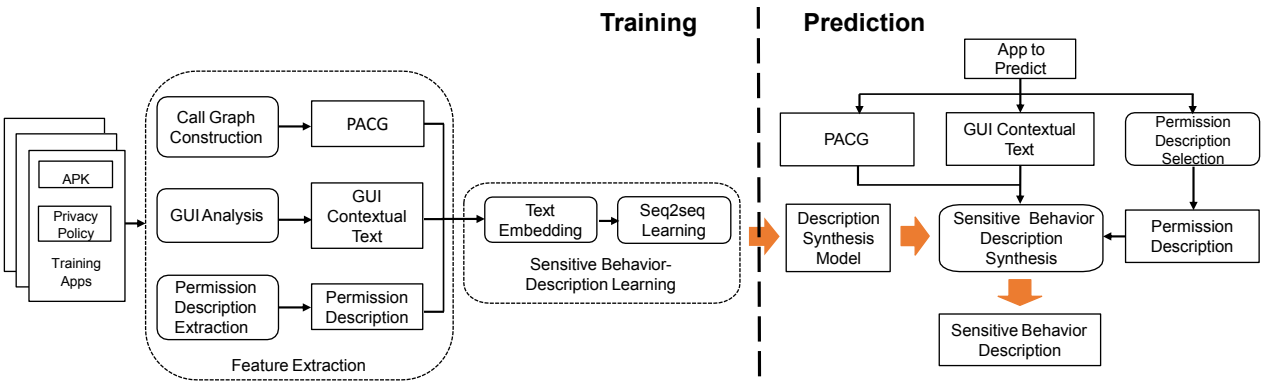



**Figure 3: Overview of DescribeCtx.**

privacy policies of the training apps. Using the PACG, the GUI contextual text, and the permission description as input, DescribeCtx synthesizes a description for each sensitive behavior of the app.

## 4.2 Feature Extraction

DescribeCtx applies static analysis and textual analysis techniques to extract three types of features: PACG, GUI contextual text, and privacy policy.

*4.2.1 Permission-Protected-API Call Graph (PACG).* In Android apps, building a call graph for a sensitive behavior needs to consider the unique characteristics of Android runtime. Android apps are component-based and event-driven applications, where apps' code is developed as components (e.g., activities for defining the GUIs and services for performing background jobs) and executed when user events (e.g., clicking buttons) or system events (e.g., receiving a phone call) occur [24, 29, 78]. In particular, the communications between components (i.e., Inter-Component Communications (ICC) [41, 53, 54, 85]), lifecycle methods triggered when apps' execution states change (e.g., open or close) [9, 70], and multithreading methods used to run worker threads cause apps' executions to split into different components or threads [81], forming implicit calling relationships among the methods in the apps' code.

To build a PACG for a permission-protected API call in an app, DescribeCtx first constructs an Inter-Component Call Graph (ICCG) for the app, and then identifies the PACG based on the ICCG for the sensitive behavior. Based on the identified PACG, DescribeCtx extracts the texts of the nodes in the PACG as the representation for the sensitive behavior. We next formally define an ICCG and a PACG, and describe how to build them in detail.

Definition 1. *An Inter-Component Call Graph (ICCG) is a directed call graph of an app that contains both the edges that represent calling statements and the edges that represent implicit calling relationships, including multi-threading methods, Android lifecycle methods, event-driven methods, and ICC methods.*

To build an ICCG, DescribeCtx first leverages existing static call graph techniques [9, 70] to build a call graph based on calling statements and lifecycle methods. It then expands the static call graph with the edges representing more implicit calling relationships. In particular, our analysis includes three types of implicit calling relationships that are most commonly used in Android apps, including multi-threading, event-driven method, and inter-component communication (ICC). Representative callers and callees include `setOnClickListener` and `onClick`, `Thread.start` and `Thread.run`, `AsyncTask.execute` and `doInBackground`, `sendMessage` and `handleMessage`, etc. Additionally, DescribeCtx integrates ICC methods using an existing tool IC3 [53]. Based on ICCGs, we next formally define PACGs:

Definition 2. *A Permission-Protected-API Call Graph (PACG) is a subgraph of an ICCG, which is formed by the union of the paths from the dummy entry node of the ICCG to the node that represents a given sensitive API call.*

To identify a PACG for a sensitive API call in an ICCG, DescribeCtx performs Breadth-First-Search (BFS) from this API call to find all the paths leading to the dummy entry node of the ICCG, and then uses all the nodes and edges found in these paths to construct the PACG. If no paths are found, the PACG contains only the sensitive API call. In our evaluation dataset (Section 5), each ICCG contains 34,897 nodes and 83,949 edges on average, and each PACG contains only 66 nodes and 71 edges on average. Note that one app can have multiple PACGs because it may use multiple sensitive API calls when requiring one certain permission. For example, an app that requires `CAMERA` permission can invoke `takePicture` and `MediaRecorder.start` to take pictures and record videos.

**Representation for Sensitive Behaviors.** DescribeCtx generates the representation of a sensitive behavior by extracting the text from the nodes of its PACG. Note that a method call contains the text that represents the class name, the return type, the method name, and the parameters. As method names often present useful and relatively unique information while class names, return types, and parameters may be shared by many methods, DescribeCtx extracts words from only the method names. As Android apps mainly follow the camel case convention in defining method names, DescribeCtx splits each method name into a set of words based on the convention. For example, `getLastKnownLocation` is split into *get*, *last*, *known*, and *location*.

One particular challenge for extracting texts from a PACG is code obfuscation, which is widely used in Android apps. The obfuscated words (e.g., *zzab*) will make the extracted texts contain lots of noises. To address this challenge, DescribeCtx removes the obfuscated methods by leveraging two English dictionaries: NLTK [47] and WordNet [21]. Specifically, DescribeCtx looks

up each extracted word in both dictionaries to filter out the non-English words. DescribeCtx adopts two dictionaries to expand the scope of the acceptable words. For example, "app" is not listed as an English word in NLTK's word list, but it is listed in WordNet. In addition, DescribeCtx directly removes those method names that use a single character such as *a* or *b*.

*4.2.2 GUI Contextual Text.* To extract GUI contextual texts from an app, we first apply Apktool [65] to decompile an app and extract its XML layout files. To extract GUI information that is relevant to a sensitive behavior, DescribeCtx performs a static analysis to associate layout files to sensitive behaviors. First, based on the PACG of a sensitive behavior, DescribeCtx identifies the activities that invoke the sensitive API calls. In Android apps, layout files are mostly loaded in activities using methods `setContentView` or `onCreateView` with their unique layout ids. Our static analysis identifies these methods and associates the layout ids to the activities and fragments, and further associates the activities to the corresponding layout files using the layout-id mapping.

The GUI contextual text mainly consists of two types: (1) string values of any GUI widget with `text` attribute from the GUI's XML layout files, and (2) names of images and icons used in the app. Both types of text can be found in the resource folder */res/* of a decompiled APK file. For the textual GUI widgets, such as `TextView` and `Button`, we extract their `text` attributes. For the image-based GUI widgets, such as `ImageView` and `ImageButton`, we extract their `id` and `src` attributes, where the `id` attributes link the GUI widgets in the layout files to the variables in the source code, and the `src` attributes provide the names of the resource files (i.e., images and icons). We mainly use the text from the `text` and `src` attributes. If they are not available, we use the `id` attributes of the GUI widgets.

**Sensitive Behaviors without GUI Contextual Texts**. Note that for certain sensitive behaviors that have no relevant GUI contextual text, we apply a text-similarity based approach to borrow GUI contextual text from another app. Specifically, DescribeCtx computes the text similarity based on TF-IDF [11] between the extracted text of the sensitive behavior's call graph and the GUI contextual text of other apps that use the same permission. Then, DescribeCtx chooses the GUI contextual text with the highest similarity score for the sensitive behavior.

*4.2.3 Privacy Policy.* In addition to permission descriptions, privacy policies also include the information unrelated to the sensitive behaviors of the app, such as legal information and user agreements. To effectively extract permission descriptions from apps' privacy policies, DescribeCtx incorporates BERT [17], which has achieved dominating performance in natural language processing (NLP) tasks such as question answering (QA) and text classification. Specifically, DescribeCtx issues permission-related questions (e.g., "why does it use `SMS` permission?") to the QA module of BERT [73], and the QA module returns an answer (i.e., an sentence) from the privacy policy with the highest confidence. As apps may use the same permission to perform different sensitive behaviors and the permission descriptions often include multiple sentences, DescribeCtx applies this extraction process for $k$ times: each time DescribeCtx gets a sentence, DescribeCtx removes the sentence from the privacy policy and issues the same question to the QA module. To ensure the precision of the extracted permission descriptions, DescribeCtx only accepts the answers whose confidence are greater than a threshold $s$. Based on our empirical evaluations (Section 5), we set $k$ to 3 and $s$ to 0.2. Finally, to take into consideration the contexts for the sentences, DescribeCtx also extracts the previous and the next sentences of each returned sentence.

## 4.3 Sensitive Behavior-Description Learning

In this step, DescribeCtx learns a neural machine translation model using the extracted features from the training apps and the reference sentences identified from the corresponding app descriptions.

*4.3.1 Model Overview.* The overview of our behavior-description learning model is shown in Figure 4. The model adapts a seq2seq architecture but accepts three heterogeneous input sequences (i.e., word sequences from PACGs, GUI contextual texts, and privacy policies). Each input sequence is transformed by a text embedding model and passed to a bidirectional LSTM (Long-Short Term Memory) layer. Next, before combining three representations into a single vector, we apply the co-attention mechanism [48, 66] to combine the representations of the GUI contextual text and PACG. The main reason for using co-attention is that a PACG usually contains many irrelevant words for a specific sensitive behavior, and the GUI contextual text helps identify the words that are most relevant to the sensitive behavior. Then, the concatenated vector representation is used as input of the decoder. When decoding, DescribeCtx applies another attention layer, which computes the relative weights of the three input sequences as well as the relative weights of each word in terms of generating each target word to form the description.

*4.3.2 Modeling Input Sequences.* We adopt similar modeling for all the three input sequences. In the following, we take the GUI contextual text as an example. We denote the embeddings of the words in the GUI contextual text as $[x_1, x_2, \cdots, x_N]$, where $x_i \in \mathbb{R}^l$ is the embedding of the $i$-th word, $l$ is the embedding dimension, and $N$ is the maximum length of the input sequence. To obtain $x_i$, we adopt Glove [58], which is a pre-trained and widely-used word embedding model in machine translation, as our text embedding model. Then, to capture the sequential order of words, we use the widely-used bidirectional LSTM [25, 32]. Each LSTM neuron takes the word embedding $x_i$ and the output of the previous neuron $h_{i-1}$ as input, and outputs $h_i$ for the current word,

$$h_i = Bi\text{-}LSTM(x_i, h_{i-1}) \tag{1}$$

where $h_i \in \mathbb{R}^d$ of size $d$ is the hidden state of the current LSTM neuron, and it also stands for the updated feature vector for the $i$-th word. The output of the bidirectional LSTM contains the feature vectors for each input word, i.e., $f_{GUI} = [h_1, h_2, \cdots, h_N]$ where $f_{GUI} \in \mathbb{R}^{d \times N}$. Note that the embeddings of all the words including those in PACGs, GUI contextual texts, and permission descriptions, are maintained in the same lookup table.

*4.3.3 Feature Combination.* To eliminate the irrelevant information in a PACG, we adopt the parallel co-attention mechanism to simultaneously update the PACG feature and the GUI contextual text feature with the guidance of each other. In the following, we show how to update the PACG features based on the GUI contextual text features, and the opposite direction can be analogously

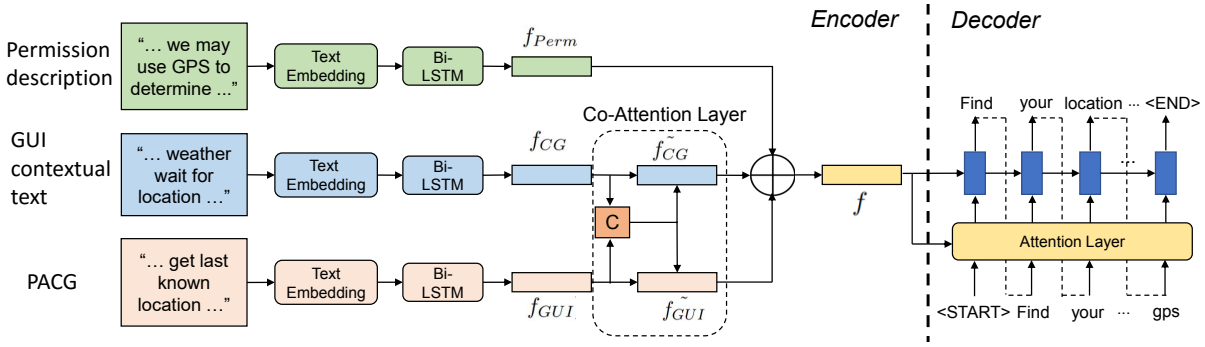



**Figure 4: Overview of the behavior-description learning model.**

obtained. Here, the attention improves the PACG feature vector by highlighting the words that are relevant to GUI contextual text. Specifically, we use $f_{GUI} \in \mathbb{R}^{d \times N}$ and $f_{CG} \in \mathbb{R}^{d \times M}$ to denote the feature vectors of the input PACG and GUI contextual text, where $N$ and $M$ are the maximum lengths of the input GUI contextual text and PACG, respectively. We then compute the correlation matrix $C \in \mathbb{R}^{N \times M}$ as follows,

$$C = tanh(f_{GUI}^T W_c f_{CG}) \tag{2}$$

where $W_c \in \mathbb{R}^{d \times d}$ is the parameter to learn. Note that there are $N$ and $M$ feature vectors for the GUI contextual text and the PACG, and this $C$ matrix contains the similarities/correlations between these $N \times M$ pairs of feature vectors.

Based on the above correlation matrix, we can correlate $f_{GUI}$ and $f_{CG}$ by transferring the features for each other. In particular, we update $f_{CG}$ with the guidance of $f_{GUI}$,

$$\begin{aligned} H_{CG} &= tanh(W_{CG} f_{CG} + (W_{GUI} f_{GUI}) C) \\ a_{CG} &= softmax(W_h H_{CG}) \\ \tilde{f_{CG}} &= \sum_{i=1}^{M} (a_{CG}^{(i)} f_{CG}^{(i)}) \end{aligned} \tag{3}$$

where $\tilde{f_{CG}} \in \mathbb{R}^d$ indicates the final attentive feature vector for the PACG, and $W_{GUI}, W_{CG} \in \mathbb{R}^{r \times d}, W_h \in \mathbb{R}^{1 \times r}$ are learnable parameters. $a_{CG}$ stands for the attention/importance for each word to the final PACG feature vector. Similarly, we can obtain the feature vector $\tilde{f_{GUI}}$ from $f_{GUI}$. In practice, we update $f_{GUI}$ and $f_{CG}$ in parallel. For the permission description from the privacy policy, since it usually contains human-written, high-quality sentences, we directly use the output of the last LSTM neuron as its feature vector. We denote it by $f_{Perm}$, and obtain the final output vector $f = [f_{Perm}; \tilde{f_{CG}}; \tilde{f_{GUI}}]$ of the encoder.

*4.3.4 Attentive Decoder.* When generating the target sequence, the relative importance/weights of the three input sequences are not necessarily equal. For example, the privacy policy is more important when generating words related to the description of permissions, and the GUI context and call graph explain more on the specific behaviors. Therefore, DescribeCtx applies an attentive decoder to learn the relative importance of three input sequences when generating each word of the target sequence. In particular, we also use LSTM to generate the behavior description. Suppose the hidden state of the LSTM neuron is $s_t \in \mathbb{R}^d$ at timestep $t$ (i.e., when generating the $t$-th word). The attention distribution of the three input sequences is then calculated as follows,

$$\begin{aligned} e_i^t &= v^T tanh(W_f f_i + W_s s_t) \\ a_i^t &= softmax(e_i^t) \end{aligned} \tag{4}$$

where $f_1 = f_{Perm}, f_2 = \tilde{f_{CG}}, f_3 = \tilde{f_{GUI}}$, and $W_f, W_s \in \mathbb{R}^{r \times d}, v \in \mathbb{R}^r$ are parameters to learn. The normalized $a_i^t$ determines how much attention should be given to the $i$-th input sequence when generating the $t$-th word in the description. Then, we can produce the overall representation $c_t = \sum_{i=1}^{3} a_i^t f_i$ of the three input sequences when generating the $t$-th word.

## 4.4 Sensitive Behavior Description Synthesis

Based on the trained model, given an app that contains sensitive behaviors, DescribeCtx applies the feature extraction techniques to extract the features from the app, and feeds the features into the model to synthesize descriptions for each sensitive behavior. Given the fact that the privacy policy of an app does not necessarily contain all related permission descriptions, *the permission description selection module in DescribeCtx is used to choose relevant permission descriptions from the training apps* when the related permission descriptions are missing. It computes the TF-IDF similarity between the GUI contextual text and the permission descriptions of the training apps, and chooses the top-$k$ similar sentences as the permission description.

# 5 EVALUATION

We evaluate the effectiveness of DescribeCtx on real world apps. Specifically, we aim to answer the following research questions:

- RQ1: How effective is DescribeCtx in generating descriptions for sensitive behaviors, with and without app-specific privacy policies?
- RQ2: How does DescribeCtx compare with the existing work?
- RQ3: How do different techniques in extracting features affect the effectiveness of DescribeCtx?
- RQ4: How effective is DescribeCtx in generating descriptions for sensitive behaviors, from users' perspectives?

**Table 1: Effectiveness results of DescribeCtx and existing work. DescribeCtx performs significantly better than the existing work. DescribeCtx$_p$ can further improve DescribeCtx when the app-specific privacy policies are available.**

| Approach | Metric | CALENDAR | CAMERA | CONTACT | LOCATION | MICROPHONE | SMS | STORAGE | Average |
|---|---|---|---|---|---|---|---|---|---|
| DescribeCtx | BLEU | 24.27 ± 1.90 | 24.91 ± 2.77 | 24.10 ± 2.56 | 26.79 ± 3.45 | 24.03 ± 3.35 | 24.55 ± 2.92 | 24.01 ± 3.33 | 24.47 ± 3.02 |
| | ROUGE-L | 22.76 ± 1.37 | 23.16 ± 1.90 | 25.78 ± 1.48 | 23.64 ± 1.80 | 19.53 ± 1.70 | 26.45 ± 1.60 | 22.84 ± 2.35 | 22.88 ± 2.00 |
| DescribeCtx$_p$ | BLEU | 26.94 ± 3.48 | 26.12 ± 2.74 | 24.71 ± 3.50 | 27.85 ± 3.44 | 25.60 ± 2.90 | 24.55 ± 2.92 | 24.29 ± 3.20 | 25.75 ± 3.10 |
| | ROUGE-L | 23.44 ± 1.47 | 23.39 ± 1.91 | 26.14 ± 1.65 | 23.82 ± 2.00 | 21.88 ± 2.01 | 26.45 ± 1.60 | 23.10 ± 2.44 | 23.47 ± 2.20 |
| Code2Vec [5] | BLEU | 9.19 ± 1.50 | 8.12 ± 0.56 | 2.27 ± 0.21 | 10.87 ± 1.39 | 11.21 ± 1.06 | 3.71 ± 1.03 | 8.44 ± 1.00 | 8.49 ± 0.97 |
| | ROUGE-L | 13.35 ± 1.44 | 6.28 ± 0.38 | 11.93 ± 0.72 | 16.00 ± 1.13 | 9.38 ± 0.51 | 16.20 ± 3.93 | 17.38 ± 1.41 | 13.01 ± 0.98 |
| Code2Vec$^+_{pp}$ [5] | BLEU | 15.22 ± 1.85 | 13.42 ± 1.55 | 17.89 ± 2.15 | 15.54 ± 1.68 | 15.19 ± 1.92 | 12.11 ± 1.40 | 12.74 ± 1.72 | 14.36 ± 1.76 |
| | ROUGE-L | 18.96 ± 1.28 | 15.64 ± 1.65 | 24.76 ± 1.72 | 19.15 ± 1.18 | 16.80 ± 1.34 | 21.88 ± 1.83 | 17.73 ± 1.76 | 18.19 ± 1.59 |
| DescribeMe [82] | BLEU | - | - | 20.90 ± 1.40 | - | 14.16 ± 1.24 | 4.88 ± 0.30 | - | - |
| | ROUGE-L | - | - | 9.31 ± 0.25 | - | 17.31 ± 0.90 | 8.17 ± 0.62 | - | - |
| DescribeCtx$^-_{ctx}$ | BLEU | 17.78 ± 2.47 | 16.45 ± 2.01 | 23.22 ± 2.63 | 19.30 ± 2.55 | 18.48 ± 2.48 | 16.63 ± 2.22 | 17.74 ± 3.20 | 18.38 ± 2.62 |
| | ROUGE-L | 20.06 ± 1.52 | 13.73 ± 1.14 | 23.21 ± 1.86 | 19.47 ± 1.39 | 14.38 ± 1.41 | 24.66 ± 2.58 | 18.56 ± 2.40 | 17.74 ± 1.79 |
| DescribeCtx$^-_{GUI}$ | BLEU | 17.78 ± 2.47 | 16.91 ± 2.09 | 23.31 ± 2.65 | 21.17 ± 2.94 | 19.03 ± 2.51 | 16.63 ± 2.22 | 18.61 ± 3.20 | 18.79 ± 2.48 |
| | ROUGE-L | 20.06 ± 1.52 | 14.52 ± 1.20 | 23.77 ± 1.90 | 19.88 ± 1.45 | 17.91 ± 1.90 | 24.66 ± 2.58 | 18.70 ± 2.35 | 18.76 ± 2.17 |
| DescribeCtx$^-_{pp}$ | BLEU | 6.94 ± 0.44 | 4.13 ± 0.23 | 11.64 ± 2.18 | 5.67 ± 0.49 | 4.89 ± 0.40 | 1.62 ± 0.04 | 2.60 ± 0.28 | 4.83 ± 0.05 |
| | ROUGE-L | 9.27 ± 0.90 | 7.04 ± 0.67 | 19.88 ± 1.45 | 9.64 ± 1.07 | 8.87 ± 0.88 | 4.79 ± 0.46 | 7.94 ± 0.61 | 9.27 ± 0.10 |
| DescribeCtx$_{cp}$ | BLEU | 22.46 ± 2.28 | 23.37 ± 2.90 | 24.10 ± 2.56 | 25.88 ± 3.28 | 22.90 ± 2.87 | 24.55 ± 2.92 | 22.75 ± 3.25 | 23.70 ± 3.00 |
| | ROUGE-L | 22.37 ± 1.45 | 21.57 ± 1.89 | 25.78 ± 1.48 | 23.50 ± 1.80 | 19.04 ± 1.65 | 26.45 ± 1.60 | 21.99 ± 2.49 | 22.19 ± 2.08 |

## 5.1 Evaluation Subjects and Setup

We collected real-world apps of different categories from Google Play [27] as our evaluation subjects, which are then grouped based on their permission groups [18, 26]. Note that all permissions are protecting user understandable sensitive data [7, 56, 59, 71, 72]. Thus, we focus on 12 dangerous permissions in 7 groups, which are the representative permissions that protect user understandable sensitive data. Since we need to use the permission descriptions in privacy policies for training and the explanations of permission uses in app descriptions for assessing the quality of the synthesized descriptions, we choose apps that use at least one of the dangerous permissions and satisfy the following two requirements:

- *Permission Descriptions in Privacy Policy*: for each app, if it provides a link for a privacy policy, we apply relevant keyword search to check whether the privacy policy contains sentences to explain the permission.
- *Permission Explanation in App Description*: for each chosen permission, we apply relevant keyword search to check whether their app descriptions contain sentences that may explain the uses of the permission. We then manually inspect these sentences to determine whether they can be used as reference sentences.

In total, we examine 8,814 apps, and use 1,262 qualified apps (20.86 *GB*, 16.93 *MB* on average). The detailed number of apps in each permission group can be found in Table 2. Note that we exclude the `PHONE` permission group because its permissions (i.e., `READ_PHONE_STATE`, `CALL_PHONE`, `READ\WRITE_CALL_LOG`) are barely explained in apps' descriptions (only 3 apps). We compile a list of permission-protected APIs (from API level 21 to level 29) from the Android Developer website. We also manually inspect the permission-API mappings provided by PScout [10] and add more APIs if the APIs access sensitive data and are not deprecated after API level 20. In total, we obtain 171 unique permission-protected APIs. For each app, DescribeCtx is applied to build an ICCG and extract PACGs from the ICCG for each permission-protected API call. On average, DescribeCtx extracts 4.6 PACGs from each app.

**Training and Prediction**. For each permission-protected API in an app, we extract its PACGs, GUI contextual text, and permission description as an input triple, and use the reference sentence as the output. In total, we obtain 1,556 data samples from the 1,262 apps. To evaluate the effectiveness of our model, we perform ten-fold cross validation. For the parameters, we set the word embedding size $l = 100$ and the LSTM state size $d = 256$. The maximum lengths of the text sequences extracted from GUI contextual text, PACGs, and privacy policy are all set to 50. Based on our empirical results (Section 5.4), DescribeCtx uses top-3 relevant permission descriptions from the training apps when an app's own privacy policy does not provide related permission descriptions. The model is trained using stochastic gradient descent with the Adam optimizer.

**Evaluation Metrics**. We measure the effective of DescribeCtx using two widely-used metrics for machine translation: *BLEU* [57] and *ROUGE* [42]. *BLEU-n* (*n* from 1 to 4) score $\in [0, 100]$ represents the percentage of the *n*-grams from the synthesized description that also co-occur in the reference sentence, where score 100 means a perfect match. The *BLEU* metric used in our evaluations is the average of *BLEU-n* (*n* from 1 to 4). For *ROUGE*, we use *ROUGE-L*, which takes the longest common subsequences in reference sentences and synthesized descriptions into account. Similar to *BLEU*, *ROUGE-L* score/percentage is also in the range of $[0, 100]$ with 100 indicating a perfect match.

**User Study**. As *BLEU* and *ROUGE-L* do not fully consider semantic meanings and grammatical structures, we further conduct two user studies to evaluate whether the synthesized descriptions are similar to the reference sentences. The first study recruits students with Android usage experiences in their daily lives and Android development experiences, and the second study leverages Amazon Mechanical Turk (MTurk) [6] to recruit users with more diverse backgrounds. MTurk is a crowdsourcing marketplace which enables researchers to easily harness the collective insights from the survey participants with different backgrounds and experiences. Having two different groups with participants from various backgrounds is more representative and bias-resistant. For both user studies, we compare the synthesized descriptions generated by DescribeCtx with the baselines DescribeCtx$^-_{ctx}$ and Code2Vec$^+_{pp}$.

**Table 2: Permission groups and reference sentences.**

| Permission Group | # Apps | # Refs | Coverage |
|---|---|---|---|
| STORAGE | 1,126 | 462 | 41.11% |
| LOCATION | 586 | 336 | 57.34% |
| MICROPHONE | 558 | 217 | 39.25% |
| CAMERA | 877 | 328 | 36.94% |
| CONTACT | 356 | 138 | 38.76% |
| SMS | 107 | 24 | 22.43% |
| CALENDAR | 202 | 51 | 25.74% |
| Average | – | – | 37.37% |

DescribeCtx$^{-}_{ctx}$ uses the features of permission and privacy policies (so that it can build the vocabularies for sensitive behaviors), but does not use PACGs and GUI contextual text to represent sensitive behaviors. Code2Vec$^{+}_{pp}$ uses code2vec [4, 5] to model code and generate sentence, and extends it by incorporating the privacy policy as input. We also provide an introduction to help the participants better understand the purpose of the survey.

For the first user study, we recruited 8 graduate students and 4 undergraduate students. The participants are provided with a reference sentence and three synthesized descriptions, and asked to provide ratings on whether each synthesized description is similar to the reference sentence from two aspects: syntactical correctness and semantic closeness. For each aspect, the students rate each synthesized description using a 5-point scale: Very Bad (1), Bad (2), So-so (3), Good (4), and Very Good (5). To reduce the bias of the user study, we randomly shuffle the three sentences produced by the three approaches. In total, each student is given a survey of 778 quadruples of one reference sentence and three synthesized sentences, and each synthesized sentence is rated by 6 students.

For the second user study, we resort to MTurk for an open survey. We divide 1,556 data samples into 50 splits (each containing 31 or 32 samples), and send out 300 splits (i.e., 6 for each split). Each sample consists of a reference sentence as the question and three synthesized sentences from the three approaches as the answers. The order of the three answers is randomly shuffled. Rating the synthesized sentences can be time-consuming, and MTurk participants are perhaps more impatient. Thus, instead of rating the synthesized sentences, for each question, MTurk participants are asked to pick one or more answers that look like the reference sentence in terms of semantic closeness. To ensure the quality of the collected answers, we constrain that each sample is displayed for at least 10 seconds before it can be moved to the next sample. For each data split, we also intentionally plug in an additional data sample with one obvious answer and two irrelevant answers. If the user chose the irrelevant answers, we consider all his/her answers invalid and remove them all. There are 275 unique users participating in our study, and 174 out of them are considered valid.

The example survey, the consent form, the detailed statistics of the recruited students, and more details of the user studies are available at our project website [2].

## 5.2 RQ1: Overall Effectiveness

Table 1 shows the effectiveness results of our approach. We first compare DescribeCtx, which uses relevant permission descriptions from the privacy policies of training apps, with DescribeCtx$_p$, which uses the permission descriptions from the apps' own privacy policies. On average, DescribeCtx achieves 24.47 *BLEU* score with a standard deviation of 3.02, and achieves 22.88 *ROUGE-L* score with a standard deviation of 2.00. DescribeCtx$_p$ achieves slightly higher *BLEU* (25.75 with a standard deviation of 3.10) and *ROUGE-L* scores (23.47 with a standard deviation of 2.20). These results indicate that **most popular apps use permissions for similar purposes, and thus with the guidance of behavior representation and GUI contextual text, the relevant permission descriptions of other apps' privacy policies can be leveraged to synthesize accurate descriptions**.

In general, DescribeCtx performs well in all permission groups (*BLEU* > 24), even in the CALENDAR and SMS groups where we have a limited number of training samples (only 24 and 51 as shown in Table 2). The possible reason is that the permissions in these groups are used for similar purposes (e.g., "send SMS" or "send text message" for SEND_SMS, and "add event" for WRITE_CALENDAR). The lowest *ROUGE-L* score is from the MICROPHONE group. One possible reason is that those apps use microphones for quite diverse purposes including detecting surrounding noise, recording speech, having audio or video chat, or even playing games.

**Case Study**. We provide some examples in Table 3. The first two examples are from the CAMERA group. We can observe although these two apps use cameras for different purposes (take photos vs. scan QR code), DescribeCtx is able to synthesize non-trivial and high-quality descriptions for both of them. This is due to the fact that we take app-specific contextual information as input. For example, DescribeCtx extracts "photo" and "gallery" from the GUI of the first app and extracts "Scan the QR code" from the GUI of the second app. Additionally, the second app's privacy policy does not provide any permission description for this behavior, and DescribeCtx addresses this problem by extracting the permission descriptions from the most similar apps in our training set (e.g., "Our app uses camera permission for scanning purpose") and connecting them with the GUI contextual text. Two more examples from the STORAGE and LOCATION groups are shown in the table, where unique words are extracted from apps' GUIs to synthesize the descriptions. The last example is from the app "com.lily.times.tweenshusky.all" which uses the microphone to record user's speech and repeats it using the voice of husky dog. DescribeCtx precisely identifies the sensitive behavior of the app (i.e., using the microphone to record your voice). However, app-specific words such as "husky" and "funny voice" from the app's GUI contextual text are ignored given the fact that they are very rare in the training data. In summary, these examples show that **apps' GUI contextual texts and relevant permission descriptions in privacy policies can provide app-specific information to improve the synthesized descriptions**.

## 5.3 RQ2: Comparison with Existing Work

Next, we compare DescribeCtx with the following three lines of the state-of-the-art approaches. (1) Approach that identifies sentences in app descriptions to explain sensitive behaviors [56,

**Table 3: Synthesized examples from different permission groups. The bold words represent the unique words that are extracted from apps' GUIs and used by DescribeCtx to synthesize the descriptions.**

| App | Permission Group | Reference Sentence | DescribeCtx | BLEU |
|---|---|---|---|---|
| app.tocus.photoframe.missyouphotoframe | CAMERA | Take your photos from your gallery or from the camera | Take photos from your gallery or take from the camera | 62.31 |
| com.citylife.cachier | CAMERA | Scan the QR code of the cards and quickly receive the information | You can **scan the qr code** with phone camera | 35.04 |
| nl.fotoview.shatter | STORAGE | Save and share your image in JPG or PNG format | You can **save image** to your phone and **share** with social media on social | 31.17 |
| com.cedarfair.canadaswonderland | LOCATION | The app uses your location to keep you up to date with offers, events and activities relevant to your location | This app will read your location information you can **search** location and **nearby service** | 25.35 |
| com.lily.times.tweenshusky.all | MICROPHONE | Talk to husky dog and he will repeat everything you say with a funny voice | The app uses microphone to record your voice | 12.73 |

59]. (2) Two closest related approaches Code2Vec [4, 5] and DescribeMe [82]. Code2Vec leverages only the source code of the app, constructs abstract syntax tree (AST) of each method, and uses attentive seq2seq model to generate summary for the source code. DescribeMe leverages both GUI contexts and call graphs of the app, and uses pre-defined templates to generate sensitive permission descriptions. (3) Traditional comment generation approach that does not consider app contexts (i.e., GUI contextual text and call graph) or app-specific vocabulary (i.e., privacy policy), or uses code context (i.e., call graphs) solely [33, 40, 46, 62]. Specifically, these methods can be seen as variants of DescribeCtx: DescribeCtx$^{-}_{ctx}$ that uses only privacy policies of the apps as input, DescribeCtx$^{-}_{GUI}$ that uses both privacy policies and PACGs as inputs, and DescribeCtx$^{-}_{pp}$ that uses app contexts as inputs.

**Comparison with Sentence Identification**. In this comparison, we measure the number of dangerous permissions requested by the apps (8,814 apps) and the number of reference sentences provided to explain them, as shown in Table 2. As we can see, each app requests 3.3 dangerous permissions on average, and the reference sentences cover only 1.2 permissions, which results in the average coverage being only 37.37%. This shows that app descriptions do not always have the descriptions for all the dangerous permissions requested by the apps. In contrast, DescribeCtx can synthesize descriptions for all permissions.

**Comparison with Code2Vec and DescribeMe**. The comparison results of these two approaches are shown in Table 1. We can see that DescribeCtx outperforms Code2Vec for all permission groups in terms of *BLEU* and *ROUGE-L* scores (at least 43% better). As Code2Vec lacks the vocabulary from privacy policies, we also feed the privacy policies into Code2Vec's learning model, resulting in a variant of Code2Vec: the Code2Vec$^{+}_{pp}$ approach. Code2Vec$^{+}_{pp}$ is much better than Code2Vec, achieving a *BLEU* score of 14.36 and a *ROUGE-L* score of 18.19. Nonetheless, DescribeCtx still outperforms Code2Vec$^{+}_{pp}$ by 41.3% for the *BLEU* score and 20.5% for the *ROUGE-L* score. These results indicate that GUI contextual texts and PACGs of the apps contribute more on synthesizing accurate sensitive behavior descriptions than using the source code representation alone with Code2Vec. For DescribeMe, we compare only three permission groups (i.e., SMS, STORAGE, and CONTACT) since we can only find these three templates from their paper[2]. We can see that DescribeCtx significantly outperforms DescribeMe on both *BLEU* and *ROUGE-L* scores in these three permission groups, especially the SMS group (*BLEU* scores: 24.55 versus 4.88).

**Comparison with Traditional Comment Generation**. As shown in Table 1, DescribeCtx outperforms the three competitors for all permission groups. DescribeCtx$^{-}_{ctx}$ only achieves a *BLEU* score of 18.38 and a *ROUGE-L* score of 17.74 on average. This is significantly lower than DescribeCtx (dropping relatively by 24.88% and 22.47%, respectively). DescribeCtx$^{-}_{GUI}$ achieves a *BLEU* score of 18.79 and a *ROUGE-L* score of 18.76 on average. These scores are slightly higher than those of DescribeCtx$^{-}_{ctx}$, indicating the effectiveness of using PACGs instead of the permission names. Still, DescribeCtx achieves much higher *BLEU* and *ROUGE-L* scores in 6 out of 7 permission groups (i.e., CALENDAR, CAMERA, LOCATION, MICROPHONE, SMS, and STORAGE), indicating that GUI contextual text plays an important role for DescribeCtx to generate app-specific descriptions in these six permission groups. For DescribeCtx$^{-}_{pp}$, it performs relatively poor, indicating the importance of incorporating the vocabulary of privacy policies.

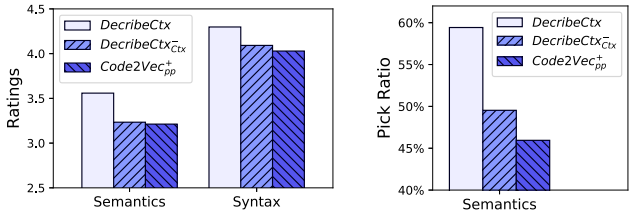
Overall, these results demonstrate that **DescribeCtx outperforms the state-of-the-art techniques in terms of *BLEU* and *ROUGE-L* scores**.

## 5.4 RQ3: Impact of Feature Extractions

We next evaluate the impacts of different design choices in the feature extraction.

**Impact of PACGs**. We first design DescribeCtx$_{cp}$ that uses a call path instead of a PACG as its features for behavior representation. As shown in Table 1, DescribeCtx outperforms DescribeCtx$_{cp}$ by 3.15% and 3.02% on average in terms of *BLEU* and *ROUGE-L* scores. For the permission groups SMS and CONTACTS, the scores of both methods are almost identical, as the GUI contextual text in these two permission groups already contains sufficient information for explaining sensitive behaviors (e.g., "send sms" or "open contact"). Overall, this experiment indicates that compared to call

[2]We have contacted the authors but they were not able to provide the complete set of the templates used in their study.




(a) User study A (b) User study B

**Figure 5: User ratings for the synthesized descriptions of different approaches. The proposed DescribeCtx is better than the two competitors in both user studies.**

paths, our PACGs contain more useful information that can be used to synthesize sensitive behavior descriptions.

**Impact of Permission Descriptions Extraction**. Existing work provides various techniques that can be used to extract permission descriptions from apps' privacy policies [14, 39, 56, 73]. We compare DescribeCtx's permission description extraction technique with a keyword-based search approach and two text classification techniques using BERT [73] and CNN [39]. We manually label the sentences that describe permission uses in the privacy policies as the ground truths, and compute the precision, recall, and $F_1$ values of each technique. The results show that DescribeCtx achieves the best $F_1$ score (0.86) among all the compared approaches, while the keyword-based search technique achieves the worst $F_1$ score (0.65). These results clearly demonstrate the superiority of DescribeCtx's permission description extraction technique over the existing ones.

**Impact of Top-$k$ Relevant Permission Descriptions**. In this experiment, we compare the results of DescribeCtx using different $k$ values, ranging from 1 to 6. The results show that, DescribeCtx achieves the best results when $k$ is 2 or 3. To mitigate the overfitting issue, $k = 3$ is used in our evaluations. The *BLEU* scores achieved by DescribeCtx drop sharply when $k \geq 4$. The probable reason is that when $k$ becomes larger, more noises are introduced and it becomes more difficult to synthesize accurate descriptions.

## 5.5 RQ4: User Study

Figure 5 shows the results of our two user studies, where we compare DescribeCtx with DescribeCtx$^{-}_{ctx}$ and Code2Vec$^{+}_{pp}$. Figure 5(a) shows the average ratings from the recruited students of different approaches. We can see that DescribeCtx achieves higher ratings for both syntactical correctness and semantic closeness than the two competitors. For example, compared with DescribeCtx$^{-}_{ctx}$, the relative improvements of DescribeCtx are 5.13% (4.30 versus 4.09) and 9.91% (3.55 versus 3.23), respectively. Figure 5(b) shows the results from MTurk workers. Here, we collect the pick ratios of different approaches, i.e., the percentage of sentences selected as the most semantically similar one to the reference sentences. The results show that users are more inclined to choose the synthesized sentences from DescribeCtx. Compared with DescribeCtx$^{-}_{ctx}$, DescribeCtx improves it by 19.97% (57.06% versus 47.56%).

We also observe some inconsistencies between user ratings and *BLEU*/*ROUGE* scores. We show two examples in Table 4. The first example uses the CALENDAR permission, and its *BLEU* score is 21.56, which is much lower than the average. However, the user ratings indicate that this is an accurate description as its semantic rating is much higher than the average (i.e., 4.33 versus 3.55). The reason is that *BLEU*/*ROUGE-L* scores measure the word-level similarity between the synthesized description and the reference sentence, whereas user ratings could better cover the semantics. For the second example, although the *BLEU* score of the synthesized description is much higher than the average, the semantic rating is only 2.83 as the keyword "usb" is missing. These examples illustrate the necessity of conducting the user study.

Overall, these two user studies show that **the synthesized descriptions of DescribeCtx are better than the existing approaches from users' perspectives**.

# 6 THREAT TO VALIDITY

The main internal threat comes from mistakenly labelling the reference sentences for the sensitive behaviors. To mitigate the threat, each sentence is verified by at least two authors, and we further check the GUIs and the code when we are not sure whether a sentence is providing justification to a permission. Another inherent limitation is that DescribeCtx does not consider dynamic-loading GUI texts in the feature extraction phase. This would cause DescribeCtx miss some GUI contextual texts of the sensitive behaviors. We mitigate this issue by using similar GUI contextual texts from other training samples.

The main external threat comes from the representativesness of our evaluation dataset. We have used 12 dangerous permissions in 7 groups that protect user understandable sensitive data, and chose 1,262 apps that contain both reference sentences and privacy policies as we need to use the apps' reference sentences and privacy policies to assess the effectiveness of DescribeCtx. To mitigate the threat for generalizing sentences for the apps that do not contain privacy policies describing their requested permissions, DescribeCtx leverages the training apps' privacy policies to complement the missing privacy policies for the apps. We also exclude the PHONE permission group due to lack of training data. These threats can be further mitigated by contacting the developers of the apps to provide the accurate descriptions.

# 7 DISCUSSION

**Application of DescribeCtx**. The synthesized descriptions of DescribeCtx can help users understand the functionalities of apps when the original app descriptions are incomplete or absent. Also, the synthesized descriptions can be used to check whether an app provides only low-qualify general descriptions for permission uses by comparing the synthesized descriptions with the provided descriptions [45], and serve as the basis for developers to provide customized permission descriptions.

**Sensitive Behaviors**. DescribeCtx focuses on the sensitive behaviors resulted by invoking sensitive APIs. However, apps may use sensitive data in Android's *intent* and *uri* messages, or through the inputs from GUIs [7, 34, 51]. Such issues can be mitigated by integrating existing static/dynamic analysis techniques to expand the scope of the sensitive behaviors.

**Code Obfuscation and Third-party Libraries**. In our evaluation subjects, we observe that at least 78.53% of the apps (991 out of 1262) apply obfuscation techniques on their code. As demonstrated

Table 4: Example synthesized descriptions with contradictory *BLEU* scores and user ratings.

| BLEU | Rating (Semantics/Syntax) | Reference Sentence | DescribeCtx |
|---|---|---|---|
| 21.56 | 4.33/4.17 | You can view all schedules in calendar, or create a schedule | You can read calendar and add event to calendar |
| 32.97 | 2.83/4.00 | Download recordings to a usb storage | Download files to your phone storage |

in Section 5.2, while DescribeCtx discards obfuscated names in call graphs, DescribeCtx is still able to effectively synthesize descriptions with the help of the contextual information provided by GUI text. Also, Section 5.4 shows that DescribeCtx$_{cp}$ that uses call paths instead of call graphs are more sensitive to obfuscations due to the limited nodes in the call paths. Currently, DescribeCtx does not consider third-party libraries as apps generally do not explain the uses of libraries to users. These issues can be addressed by requesting developers of third-party libraries to provide their own descriptions on the permission uses.

**Malicious Apps**. DescribeCtx is designed to synthesize descriptions for sensitive behaviors, but not to detect malicious behaviors that deliberately evade detection [19, 67, 70, 86]. For malicious apps that aim to manipulate permission-protected data, DescribeCtx can still synthesize descriptions for them by training a model on malicious apps with reference sentences (e.g., malware family description) that explain the malicious behaviors. For malicious behaviors that do not concern about the sensitive data protected by permissions, such as root phones and C&C (Command and Control), we can adopt the advanced detection techniques [49, 76, 78, 79], which is out of the scope of DescribeCtx.

**Reference Sentences**. We consider the app descriptions provided by the app developers as the most reliable and easy-to-access source for reference sentences. To mitigate the risk that developers may lie about the permission uses, we choose only high-quality popular apps and manually inspect the chosen apps' functionality to verify that the provided app descriptions can be used. Crowd-sourcing approaches may also be adopted to collect reference sentences especially for the apps that lack such sentences in the app descriptions. However, obtaining an accurate reference sentence requires deploying apps and inspecting code, which is a non-trivial effort. Nonetheless, we conduct such an evaluation by first randomly selecting 125 apps that are not included in our evaluation dataset. Then, two authors spent over 70 hours to verify the apps and write the behavior descriptions, by deploying them in Android 8.0 virtual machine and inspecting the source code via JADX [1]. In total, we obtain 64 sensitive behaviors from these apps by excluding those requiring login, leading to failures, or triggering sensitive behavior in the background. Among them, 53 (83%) author-written descriptions are semantically consistent with the synthesized sentences by DescribeCtx. This result shows that DescribeCtx synthesizes sentences that can well describe the sensitive behaviors themselves.

## 8 RELATED WORK

**Extracting Descriptions for Sensitive Behavior**. There exists a line of work [56, 59] that adapts NLP techniques and machine learning techniques to extract sentences in app descriptions for explaining the permission uses. However, these sentences usually cannot cover all the permissions requested by the apps. Unlike these approaches, DescribeCtx incorporates app descriptions and privacy policies into the neural machine translation model to synthesize descriptions for sensitive behaviors. Recent work [82] generates descriptions using behavior description model for the call graphs of sensitive APIs. Unlike DescribeCtx, this approach uses only words from the code, failing to consider app-specific functionality described in privacy policies and GUIs. In addition, some work [8, 14] identifies sentences that describe what types of information have been used in the apps, while DescribeCtx identifies the permission descriptions for the sensitive behaviors.

**Code Comment Generation**. Earlier code comment generation work adopts template-based or heuristic-based techniques to generate comments for source code [31, 62]. However, these proposals have been shown to be less effective and flexible compared to data-driven techniques. Later, deep learning techniques such as convolutional neural networks and recurrent neural networks have been adopted to translate a small piece of the source code into comments [3, 33, 37, 40, 46]. However, these proposals do not consider the context of sensitive behaviors and are not as effective as DescribeCtx, as shown in Section 5.3 and Section 5.4.

**Seq2seq and Attention Mechanism**. Seq2seq learning and the attention mechanism [12, 64, 66] have become the de facto standard in many applications including neural machine translation, text summarization, and question answering. As a promising deep learning model, seq2seq model has also been used in addressing plenty of other software engineering tasks, such as commit message generation [38, 77], comment generation [33, 40, 46], API migration [30, 52], and program synthesis [43, 80]. In this work, we also adopt the attentive seq2seq model for synthesizing natural language descriptions. Different from existing work, we encode three-way information in our seq2seq model, and pay special attention to learning the context of sensitive behaviors via applying co-attention [48, 84] on the call graph and GUI words.

## 9 CONCLUSION

We have presented a context-aware neural machine translation approach, DescribeCtx, that trains a seq2seq model using a large number of apps, and generates descriptions for sensitive behaviors in the apps based on app-specific contextual information. DescribeCtx provides static analysis techniques and text analysis techniques to extract features from apps' call graphs, GUI layout files, and privacy policies, forming three heterogeneous input sequences for training the model. Our evaluations on 1,292 real apps show that DescribeCtx achieves high *BLEU*/*ROUGE-L* scores with respect to the reference sentences in the app descriptions. Users also provide relatively high ratings for DescribeCtx in terms of syntactic and semantic aspects of the synthesized descriptions.

## ACKNOWLEDGMENTS

Xusheng Xiao's work is partially supported by the National Science Foundation under the grants CCF-2046953 and CNS-2028748. Yuan Yao is partially supported by the Collaborative Innovation Center of Novel Software Technology and Industrialization. The work was done when the second author was at Nanjing University. Xusheng Xiao and Yuan Yao are the corresponding authors.

## REFERENCES

[1] Jadx: Dex to java decompiler, 2020. https://github.com/skylot/jadx/releases.
[2] Describectx project website, 2021. https://github.com/DescribeCTX/DescribeCTX.
[3] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A convolutional attention network for extreme summarization of source code. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2091–2100, 2016.
[4] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. Code2seq: Generating sequences from structured representations of code. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. Code2Vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages (POPL)*, 3:40:1–40:29, 2019.
[6] Amazon. Amazon mechanic turk, 2021. https://www.mturk.com/.
[7] Benjamin Andow, Akhil Acharya, Dengfeng Li, William Enck, Kapil Singh, and Tao Xie. UiRef: Analysis of sensitive user inputs in android applications. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 23–34, 2017.
[8] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck. In *Proceedings of the USENIX Security Symposium*, 2020.
[9] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, page 259–269, 2014.
[10] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. PScout: Analyzing the android permission specification. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 217–228, 2012.
[11] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., USA, 1999.
[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
[13] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 73–84, 2010.
[14] Andow Benjamin, Mahmud Samin, Yaseer, Wang Wenyu, Whitaker Justin, Enck William, Reaves Bradley, Singh Kapil, and Xie Tao. PolicyLint: Investigating internal privacy policy contradictions on google play. In *Proceedings of the USENIX Security Symposium*, pages 585–602, 2019.
[15] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of the Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST)*, pages 103–111, 2014.
[16] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.
[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, 2019.
[18] Android Official Documentation. App permissions: Permissions overview.
[19] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the USENIX Security Symposium*, pages 21–21, 2011.
[20] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. Graph embedding based familial analysis of android malware using unsupervised learning. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 771–782, 2019.
[21] Christiane Fellbaum, editor. *WordNet An Electronic Lexical Database* . The MIT Press, 1998.
[22] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, pages 3–14, 2011.
[23] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 1–14, 2012.
[24] Dominik Franke, Corinna Elsemann, Stefan Kowalewski, and Carsten Weise. Reverse engineering of mobile application lifecycles. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 283–292, 2011.
[25] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.
[26] Google. Android permission groups, 2020. https://developer.android.com/reference/android/Manifest.permission_group.
[27] Google. Google Play Store, 2020. https://play.google.com/store?hl=en.
[28] Google. Potentially harmful applications (phas), 2020. https://developers.google.com/android/play-protect/potentially-harmful-applications.
[29] Google. Application Fundamentals, 2021. https://developer.android.com/guide/components/fundamentals.
[30] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. DeepAM: migrate apis with multi-modal sequence to sequence learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3675–3681, 2017.
[31] Sonia Haiduc, Jairo Aponte, and Andrian Marcus. Supporting program comprehension with source code summarization. In *Proceedings of the international conference on software engineering (ICSE)*, pages 223–226, 2010.
[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[33] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 200–210, 2018.
[34] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. SUPOR: Precise and scalable sensitive user input detection for android apps. In *Proceedings of the USENIX Security Symposium*, pages 977–992, 2015.
[35] AV-TEST Institute. Malware statistics, 2020. https://www.av-test.org/en/statistics/malware/.
[36] iOS Official Documentation. Human interface guideilnes: Requesting permission.
[37] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2073–2083, 2016.
[38] Siyuan Jiang, Ameer Armaly, and Collin McMillan. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 135–146, 2017.
[39] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
[40] Alexander LeClair, Siyuan Jiang, and Collin McMillan. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 795–806, 2019.
[41] Li Li, Alexandre Bartel, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. IccTA: Detecting inter-component privacy leaks in android apps. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 280–291, 2015.
[42] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 74–81, 2004.
[43] Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, Luke Zettlemoyer, and Michael D. Ernst. Program synthesis from natural language using recurrent neural networks. Technical Report UW-CSE-17-03-01, University of Washington Department of Computer Science and Engineering, 2017.
[44] Bin Liu, Hongxia Jin, and Ramesh Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 337–350, 2015.
[45] Xueqing Liu, Yue Leng, Wei Yang, Wenyu Wang, Chengxiang Zhai, and Tao Xie. A large-scale empirical study on android runtime-permission rationale messages. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 137–146, 2018.
[46] Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. Neural-machine-translation-based commit message generation: how far are we? In *Proceedings of the ACM/IEEE International Conference on Automated*

*Software Engineering (ASE)*, pages 373–384, 2018.

[47] Edward Loper and Steven Bird. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, 2004.

[48] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, pages 289–297, 2016.

[49] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. CHEX: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 229–240, 2012.

[50] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, 2015.

[51] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and Xiaofeng Wang. UIPicker: User-input privacy identification in mobile applications. In *Proceedings of the USENIX Security Symposium*, pages 993–1008, 2015.

[52] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N Nguyen. Exploring API embedding for API usages and applications. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 438–449, 2017.

[53] Damien Octeau, Daniel Luchaup, Matthew Dering, Somesh Jha, and Patrick McDaniel. Composite constant propagation: Application to android inter-component communication analysis. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 77–88, 2015.

[54] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. Effective inter-component communication mapping in android with Epicc: An essential step towards holistic security analysis. In *Proceedings of the USENIX Conference on Security*, page 543–558, 2013.

[55] Jason D. O'GRADY. New privacy enhancements coming to ios 8 in the fall., 2014. http://www.zdnet.com/new-privacy-enhancements-coming-to-ios-8-in-the-fall-7000030903/.

[56] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: towards automating risk assessment of mobile applications. In *Proceedings of the USENIX Security Symposium*, page 527–542, 2013.

[57] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[58] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[59] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. AutoCog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1354–1365, 2014.

[60] Vaibhav Rastogi, Yan Chen, and William Enck. AppsPlayground: Automatic security analysis of smartphone applications. In *Proceedings of ACM Conference on Data and Application Security and Privacy (CODASPY)*, page 209–220, 2013.

[61] Sanae Rosen, Zhiyun Qian, and Z. Morely Mao. AppProfiler: A flexible method of exposing privacy-related behavior in android applications to end users. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, page 221–232, 2013.

[62] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 43–52, 2010.

[63] Statista. Global mobile os market share, 2020. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

[65] Connor Tumbleson and Ryszard Wisniewski. Apktool, 2017. https://ibotpeaches.github.io/Apktool/.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Annual Conference on Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[67] VirusShare. VirusShare, 2021. http://virusshare.com/.

[68] Haoyu Wang, Jason Hong, and Yao Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, pages 1107–1118, 2015.

[69] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D. Breaux, and Jianwei Niu. GUILeak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 37–47, 2018.

[70] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, page 1329–1341, 2014.

[71] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David A. Wagner, and Konstantin Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1077–1093, 2017.

[72] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. Contextualizing privacy decisions for better prediction (and protection). In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*, pages 1–13, 2018.

[73] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, pages 38–45, 2020.

[74] Xusheng Xiao, Nikolai Tillmann, Manuel Fahndrich, Jonathan De Halleux, and Michal Moskal. User-aware privacy control via extended static-information-flow analysis. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 80–89, 2012.

[75] Xusheng Xiao, Nikolai Tillmann, Manuel Fahndrich, Jonathan De Halleux, and Michal Moskal. User-aware privacy control via extended static-information-flow analysis. *Automated Software Engineering Journal (ASEJ)*, 22(3):333–366, 2015.

[76] Ke Xu, Yingjiu Li, Robert H. Deng, and Kai Chen. DeepRefiner: Multi-layer android malware detection system applying deep neural networks. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 473–487, 2018.

[77] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. Commit message generation for source code changes. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 3975–3981, 2019.

[78] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. AppContext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 303–313, 2015.

[79] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X. Sean Wang. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the ACM conference on Computer & communications security (CCS)*, pages 1043–1054, 2013.

[80] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 440–450, 2017.

[81] Yujie Yuan, Lihua Xu, Xusheng Xiao, Andy Podgurski, and Huibiao Zhu. RunDroid: recovering execution call graphs for android applications. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 949–953, 2017.

[82] Mu Zhang, Yue Duan, Qian Feng, and Heng Yin. Towards automatic generation of security-centric descriptions for android apps. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 518–529, 2015.

[83] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-aware android malware classification using weighted contextual API dependency graphs. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1105–1116, 2014.

[84] Suwei Zhang, Yuan Yao, Fent Xu, Hanghang Tong, Xiaohui Yan, and Jian Lu. Hashtag recommendation for photo sharing services. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 5805–5812, 2019.

[85] Jinman Zhao, Aws Albarghouthi, Vaibhav Rastogi, Somesh Jha, and Damien Octeau. Neural-augmented static analysis of android communication. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 342–353, 2018.

[86] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the IEEE Symposium on Security and Privacy (S & P)*, pages 95–109, 2012.