IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 17, 2020, accepted December 19, 2020, date of publication December 24, 2020,
date of current version January 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3047019

FCDP: Fidelity Calculation for Description-to-Permissions in Android Apps

ZHIQIANG WU[1], (Graduate Student Member, IEEE), XIN CHEN[1], AND SCOTT UK-JIN LEE[2], (Member, IEEE)

[1]Department of Computer Science and Engineering, Hanyang University, Ansan 15588, Republic of Korea
[2]Department of Computer Science and Engineering, Major in Bio Artificial Intelligence, Hanyang University, Ansan 15588, Republic of Korea

Corresponding author: Scott Uk-Jin Lee (scottlee@hanyang.ac.kr)

This work was supported by the Hanyang University through the Research Fund under Grant HY-2020-1700.

ABSTRACT Mobile app descriptions have been widely used in app markets to deliver various types of information to end-users. Even though this information may implicitly expose the dangerous permissions that allow access to sensitive data, most users cannot correctly identify and interpret the corresponding textual sentences owing to insufficient knowledge regarding Android permissions and the semantics of app descriptions. It is therefore important to assist users in understanding whether an app description accurately reflects whether the app may request dangerous permissions. To this end, we propose an approach named Fidelity Calculation for Description-to-Permissions (FCDP). It is aimed at assisting app-market auditors in assessing whether an app description indicates all information related to dangerous permissions using a quantified fidelity for providing a high-quality description to mobile users. Furthermore, we experimentally investigate the effect of different factors on FCDP, and we demonstrate that FCDP outperforms the state-of-the-art method by over 3.65% in predicting description-to-permissions. By using 64,265 Android descriptions crawled from Google Play, our in-depth analysis further indicates that most app descriptions do not entirely disclose the semantics of dangerous permissions for mobile users in the wild. It is therefore quite urgent to assist app-market auditors in regulating description writing in this regard.

INDEX TERMS Android, app descriptions, fidelity calculation, natural language processing, permissions.

I. INTRODUCTION

Android has become the most popular mobile operating system, with a 70.68% share of the global market in April 2020 [1]. Android users may download apps from Google Play, an official app market for the Android platform, with over 2.55 million mobile applications available as of the first quarter of 2020 [2]. Currently, attackers are posing severe security threats by developing several malicious apps for stealing private user information. To protect sensitive user information, Android provides a mechanism that enables end-users to grant or deny permission when a request is received. Before installing an app, users usually check its description to determine whether the functionalities of app meet their expectations. Hence, these descriptions significantly affect the decision to download and install an emerging app. Accordingly, developers should write an attractive description of the major functionalities of an app to attract more users [3]. App descriptions not only introduce these functionalities but also provide information regarding possible access to sensitive information and requests for associated permissions. Generally, the dangerous permissions are used to support app functionalities and are also implied in app descriptions, namely, the fidelity of description-to-permissions [4]. For instance, Fig. 1 shows a part of the description of Alipay,[1] the most popular finance app in China. The red-coloured words "Send/Receive money from your peers" implicitly indicate that the app may request `READ_CONTACT` permission to add friends in Alipay. Similarly, to order food from local restaurants, the app may request `LOCATION` permissions to search for nearby restaurants. Finally, users should grant `CAMERA` permission to scan the QR code for payment, as shown in Line 7 of the description. If a description has a lower fidelity regarding its declared permissions, then the app may either request more unnecessary permissions or deliberately hide some functionalities, possibly posing security threats.

When a user searches for an app in Google Play, query keywords are generally matched with app descriptions and

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

[1]https://play.google.com/store/apps/details?id=com.eg.android.AlipayGphone

YOU ARE ONLY A 'TOUCH' AWAY-
1. Send/Receive money from your peers
2. Card free payment at millions of merchants
3. Top up your mobile phone and pay your utility bills
4. Order food from local restaurants or book a taxi
5. Manage your money with wealth management products
6. Free off-site cross-bank transfer / credit card repayment and loans
7. Scan & Pay – Scan & use the QR code to pay at your local stores
……

FIGURE 1. Snippet of app description from *Alipay*.

names [5]. Only 21% of users are capable of identifying the implicitly used permissions from app descriptions before installation [6]. Other users consider that app description only introduces the functionalities and promotion of app without any security-related information, they thus prefer to download the first app displayed in the search results without reading app descriptions [7]. To protect user's privacy, dangerous permissions are no longer automatically granted during installation from Android 6.0. The users may grant or deny dangerous permissions with a pop-up dialogue at runtime based on their demand. As making decision on granting dangerous permissions is a non-trivial work for most users, the users tend to rush through permission requests by granting all requested permissions in order to enable all functionalities [8]. In order to assist end-users in understanding whether the dangerous permissions are imperative for their demands based on app descriptions, it is necessary to develop guidelines for writing appropriate app descriptions. Google Play has issued a simple policy for writing app description, which includes disclosing the usage of dangerous permissions [9]. According to it, developers should explicitly disclose whether an app requires accessing, collecting, using or sharing Personal and Sensitive Information (PSI), such as contacts, location, and SMS. However, Google Play does not provide any mechanism for identifying usages of PSI for app description in the vetting process. Hence, developers tend to only focus on presenting the major functionalities of their apps to attract more users. Consequently, app descriptions may not disclose all information regarding dangerous permissions.

To address these issues, we propose **F**idelity **C**alculation for **D**escription-to-**P**ermissions (FCDP) to predict the permissions that may be requested by an app based on textual descriptions, and to quantify the fidelity of these descriptions-to-permissions. To precisely capture the semantics from app descriptions, we employ the attention mechanism of Bidirectional Long Short-Term Memory (Bi-LSTM) to extract meaningful words for predicting 16 permissions that allow access to sensitive user information. To satisfy the aforementioned policy, our work also presents an intuitive fidelity for description-to-permissions, which aims to assist Google Play auditors in determining whether app descriptions truthfully disclose the usages of dangerous permissions. It enables auditors to provide appropriate feedback to app developers for modification based on fidelity until all usages of PSI are adequately presented in app descriptions. As a result, understandable descriptions that requires a little or no background on Android security can be provided to end-users.

With further analysis, we have investigated the impacts of the different factors on predicting permissions from app descriptions including models, corpora and the dimensionality of word embedding. We also have applied the proposed method to assess the fidelity of description-to-permissions in the wild.

The main contributions of this study are the following:

- We propose FCDP, which is an effective approach for predicting potentially requested permissions from app descriptions and calculating a fidelity metric that quantitatively measures the correlation among the predicted permissions and requested permissions in the source code.
- Using FCDP, we conducted an empirical study to demystify the effect of different deep learning factors on predicting permissions from app descriptions.
- We performed a large-scale measurement by applying FCDP to 64,265 apps to measure the fidelity of app descriptions in the wild, and then discussed the distribution of benign and malicious Android apps.

The remainder of this article is organized as follows. Section II presents the necessary background material on Android permissions and deep learning. Section III reviews related studies and highlight their strengths and weaknesses. The proposed approach is introduced in Section IV. The experimental study and corresponding evaluation are presented in Section V. Section VI discusses the threats to the validity of our study. Finally, Section VII concludes this article.

II. BACKGROUND

To better understand our methodology, we first introduce the permission mechanism in Android operating system, then illustrate the necessity of controlling dangerous permissions. Subsequently, the mathematical derivations of Long Short-Term Memory (LSTM) and attention mechanism (the deep-learning model that we employ) are explained in detail.

A. ANDROID PERMISSION MECHANISM

To protect private user information, the Android platform provides a permission mechanism to control access to such data. Android apps must request permissions to access specific user/device data and system features, such as the Internet, SMS, and camera [10]. For instance, a communication app such as WeChat[2] must have `CAMERA` permission for making video calls and taking pictures. According to the

[2]https://play.google.com/store/apps/details?id=com.tencent.mm

privacy level of the data to be handled, permissions are classified as follows:

- Normal permissions cover areas where apps should access data or resources outside their application sandbox, but where there is minimal risk to user privacy or the operation of other apps, and users cannot revoke these permissions.
- Signature permissions are granted by the system at installation time for apps that have the same certificate signs.
- Dangerous permissions allow an app to access sensitive user information that can be easily exposed by developers.

AndroidManifest.xml file is a configuration file that describes essential app information, including components, permissions, intent filters, and Android build tools. The permissions requested to run the app should be declared in manifest file with tag name `uses-permissions` [11], [12]. The corresponding functionalities that require permission to access private data can only be invoked after user authorization.

In Android 5.1 and lower, users should grant all requested permissions at installation time. However, users usually do not understand the permission mechanism and are unlikely to deny any unwanted dangerous permission. If a user does not want to grant certain permission for an app, the user has to cancel the installation process. From Android 6.0 (API level 23), to control accessing PSI, users may approve dangerous permissions for an app at runtime [13], [14]. Hence, the user may grant/deny dangerous permissions even after installation of an app, and features associated with denied permissions are disabled. Normal and signature permissions, such as `INTERNET` and `ACCESS_NETWORK_STATE`, are automatically granted at installation time.

B. DEEP LEARNING

1) LONG SHORT-TERM MEMORY (LSTM)

Recurrent Neural Network (RNN) is an artificial neural network, in which cell states may express context through input sequences. RNN is widely used with temporal inputs, as in text generation [15], speech recognition [16], and text classification [17], [18]. However, RNN is known to have vanishing and exploding gradients in the long-term memory of temporal information. To address these issues, two variants of RNNs have been proposed: LSTM [19], [20] and Gated Recurrent Unit (GRU) [21]. The former has considerably better performance on datasets with long sequences than the latter. Long-term dependencies are resolved using the gates in the LSTM cells to control either the transition or forgetting of data. As shown in Fig. 2, an LSTM cell consists of a forget gate f_t, an input gate, i_t and an output gate o_t at the t-th time stamp.

The forget gate f_t determines which piece of information to drop from the previous cell state C_{t-1} using the following formula:

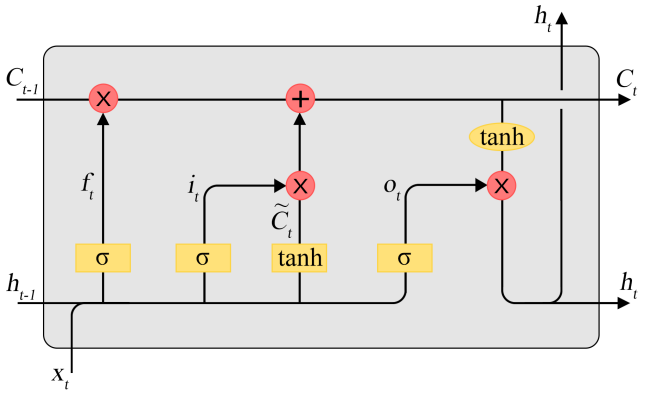


FIGURE 2. Internal structure of an LSTM cell at timestamp t.

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \tag{1}$$

where σ is a sigmoid function, x_t denotes the input at timestamp t, and W_f and b_f denote the weight matrix and bias for the forget gate, respectively. The output of the forget gate is a numerical vector between 0 and 1 for each element in the cell state C_{t-1}. If the output is close to 1, the corresponding element should be retained entirely. In contrast, 0 indicates that the element should be dropped.

The next step is to determine what new information should be stored in the cell state. Initially, the input gate i_t determines which values will be updated. Subsequently, a tanh layer generates a vector of new candidate values $\widetilde{C}_t$ that can be added to the state:

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right) \tag{2}$$

$$\widetilde{C}_t = \tanh\left(W_C\left[h_{t-1}, x_t\right] + b_C\right) \tag{3}$$

The cell state C_t at timestamp t consists of the retained state from the previous cell state C_{t-1} by the forget gate f_t and the new candidate state $\widetilde{C}_t$ from the input gate i_t:

$$C_t = f_t \odot C_{t-1} + i_t * \widetilde{C}_t \tag{4}$$

Finally, the output gate o_t controls the part of the input x_t that should be output based on the last hidden state h_{t-1} and cell state C_t. The hidden state h_t at timestamp t is calculated as:

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{5}$$

$$h_t = o_t \odot \tanh\left(C_t\right) \tag{6}$$

2) ATTENTION MECHANISM

The attention mechanism proposed by Bahdanau *et al.* [22] is important in various applications involving temporal inputs, such as image recognition [23] and natural language processing (NLP) [24], [25]. In text classification, not all words have the same semantic contribution to a sentence. Hence, the attention mechanism is employed to extract semantically important words and aggregate their representations to form a sentence vector [26].

The attention mechanism can automatically learn a weight α_{ij} to capture the correlation between the encoder hidden state h_i and the decoder hidden state s_i. These attention weights are used to construct a context vector c_i as input to the decoder. This vector c_i at time j of a decoder is computed as a weighted sum of all hidden states, as shown in Eq. 7. The weight α_{ij} of each hidden state h_j is calculated in Eq. 8 and 9, where, $v_a \in \mathbb{R}_n$, $W_a \in \mathbb{R}_{n\times n}$ and $U_a \in \mathbb{R}_{n\times 2n}$ are weight matrices in the attention mechanism.

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j \tag{7}$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T} \exp(e_{ik})} \tag{8}$$

$$e_{ij} = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right) \tag{9}$$

III. RELATED WORK

In this study, we investigate the relationship between requested permissions in the source code and descriptions of apps on Google Play. Several studies have been concerned with this problem. Herein, we present related work and indicate the drawbacks of existing methods. Moreover, we compare these methods with the proposed approach.

A. PERMISSIONS-BASED ANDROID APP ANALYSIS

Android employs a unique permission mechanism to protect sensitive resources. The declared permissions do not always require the use of a specific set of APIs. Moreover, permissions are not always described well in the documentation, causing developers to either lack relevant knowledge or misuse APIs during development [27], [28]. Karim *et al.* proposed ApMiner to recommend permissions for an app based on given sets of Android APIs [29]. ApMiner relies on association rule discovery to identify co-occurrence patterns of Android APIs and permissions from markets, and recommends a set of suitable permissions for developers according to the used APIs in source code. Although such technique can help developers to declare proper permissions, it could not provide clear reasons why the permissions are declared in the source code. Holavanalli *et al.* [30] proposes a method to analyze the flow of permissions among sources and sinks [31], named Flow Permissions. It allows users to vet and grant explicit/implicit information flows in apps. Unfortunately, the end-users have to identify malicious flows themselves without any detailed instruction, which can be quite difficult. Sarma *et al.* also employed machine learning to detect rare appearing permissions in its category, to inform users whether the risks of installing an app is commensurate with the expected benefit [32]. However, this approach does not provide a clear guideline on handling rarely appearing permissions for users. Rashidi *et al.* proposed DroidNet to run apps under probation mode without granting any permission upfront [33]. It provides recommendations of peer expert users on granting permissions, which may prevent granting some unnecessary permissions in the app for inexperienced users. However, this approach heavily depends on responses from expert users and makes it impossible for inexperienced users to understand the decisions without a proper textual explanation.

In all these related studies, the source code of apps was analyzed to determine whether the developers employed proper permissions according to the used APIs. The end-users still cannot understand the purposes of permissions/APIs without a clear explanation while they are required to grant permission. In contrast to these studies, we bridge the semantic gap between the description and the requested permissions in Android apps to assist Google Play auditors and users in understanding the fidelity of description-to-permissions with an indicator and regulating description writing.

B. CONSISTENCY OF DESCRIPTION TO PERMISSIONS

To understand whether an app description truthfully declares the usage of dangerous permissions for its functionalities and meets the expectations of the end-users, Pandita *et al.* [34] proposed WHYPER to bridge the semantic gap between user expectations and app descriptions. They transformed the sentences in the app descriptions into first-order-logic (FOL) representations to analyze Android API documents, thus facilitating the risk assessment of mobile applications. However, WHYPER is not able to cover all textual patterns associated with permissions from API documents [4]. Moreover, it can detect only a small portion of dangerous permissions (i.e., `READ_CALENDAR`, `READ_CONTACTS`, `RECORD_AUDIO`). In 2014, CHABADA was proposed by Gorla *et al.* to identify outlier apps based on the theme of their descriptions [35]. It employed Latent Dirichlet Allocation (LDA) to identify the main topics for each description. Then, through unsupervised One-Class Support Vector Machine (SVM) anomaly detection, CHABADA identifies outliers with abnormal API usages in the same clusters. This approach cannot handle polysemous words and simple app descriptions that do not have any semantics for clustering topics. AutoCog [4] was proposed by Qu *et al.* to assess the consistency among app descriptions and their source code. It extracts noun phrases from app descriptions using Explicit Semantic Analysis (ESA) based on the frequency and computes the semantic relatedness of texts with permissions. AutoCog covers 11 permissions and outperforms WHYPER. It also provides notice to developers whether textual sentences accurately reflect permissions by matching pairs of noun phrases. However, AutoCog generates the large number of false positives under the following situations: 1) the app description is excessively simple; 2) the app description does not contain any common patterns of sensitive permissions; 3) the app requests permissions without using corresponding APIs [37]. ACODE [5] employed a keyword-based technique to infer whether an app description clearly indicates the usages of permissions to access sensitive information. Polysemy profoundly affects semantic reasoning. For instance, the word "contact" can be a verb, as in the sentence "contact with us by email," whereas

TABLE 1. Comparison table of related works.

Items	Ours	WHYPER [34]	CHABADA [35]	AutoCog [4]	ACODE [5]	AC-Net [36]
# Permissions	16	3	-	11	11	16
Method	BiLSTM + Attention	FOL	SVM	Textual Patterns	Keyword-based	GRU
Fidelity Quantification	Yes	No	No	No	No	No

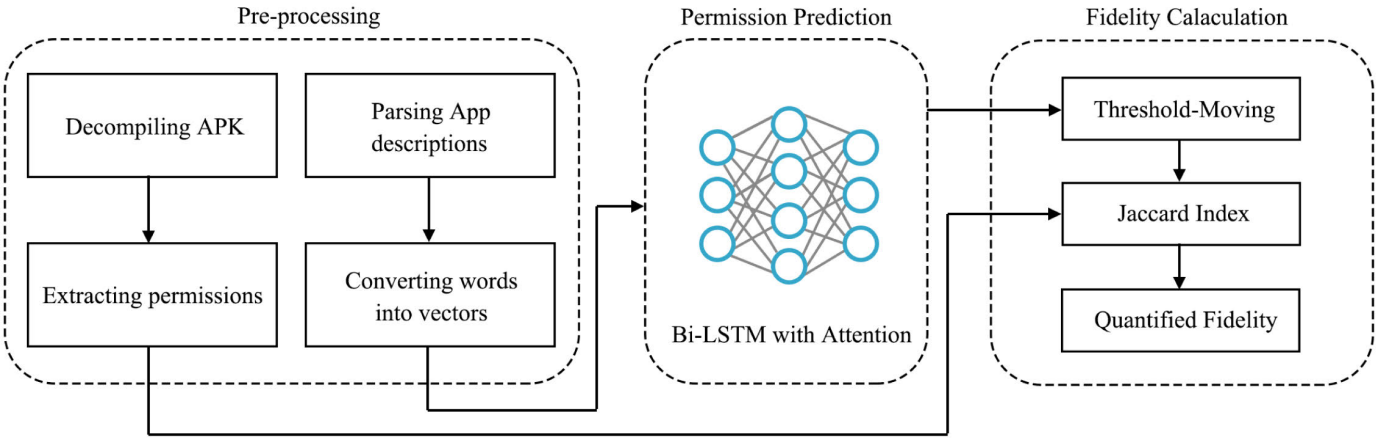


FIGURE 3. Overall architecture of FCDP.

it is a noun in the sentence "add friends from your contacts," indicating the usage of the permission `READ_CONTACTS`. Feng *et al.* proposed AC-Net [36], which uses the GRU model to predict permissions from app descriptions. It covers 16 dangerous permissions, which is currently the maximum number of labeled permissions. However, only app descriptions that declare at least one permission are considered, and therefore, the actual situation in the wild is not captured.

Table 1 compares the characteristics of approaches between our study and other related works. These related works only investigated whether the target permissions are disclosed in the app descriptions. They did not quantify the fidelity of description-to-permissions, which cause that the stakeholders are difficult to utilize the results of those works for further decision. Among the aforementioned methods, AC-Net [36] is closest to our present study. However, unlike AC-Net, our proposed method captures the semantics of app descriptions using the attention mechanism in deep learning. In addition, our study is the first attempt to quantify the fidelity of description-to-permissions so that users/auditors can be assisted in vetting an app through an intuitive quantitative measure.

IV. PROPOSED APPROACH

In this section, we present the proposed approach in detail. Fig. 3 shows the architectural overview of FCDP, which consists of three major parts: pre-processing of app and descriptions, construction of the deep learning model for permission classification, and calculation of the fidelity measure for the correlation between predicted results and requested permissions in the source code.

A. PRE-PROCESSING

To calculate the fidelity of description-to-permissions, the requested permissions should be extracted from the AndroidManifest.xml file, and textual descriptions should be converted into numerical vectors. Thus, pre-processing is divided into two parts.

1) PERMISSION RETRIEVAL

Android Package Kit (APK) files are decompiled using apktool[3] to obtain the AndroidManifest.xml file, in which the requested permissions are declared. In this study, we predict 11 permission groups (involving 16 sensitive permissions as in [36]) from app descriptions, as shown in Table 2. The end-users concern these 16 permissions which allow the app to access sensitive data and operations [38]. We generate a vector P_{truth} to record the ground truth of the permissions for each app. If the candidate permission occurs in the XML file with the `<uses-permission>` tag, the value of the corresponding permission should be set to 1. Otherwise, the value is set to 0, implying that the app did not declare the particular permission in AndroidManifest.xml file.

2) TEXTUAL PRE-PROCESSING

To feed the descriptions into the deep learning model, the app descriptions should be split into sentences, and

[3]https://ibotpeaches.github.io/Apktool/

TABLE 2. Labeled permissions in AC-Net [36].

Permission Groups	Permissions
Storage	WRITE_EXTERNAL_STORAGE
Contacts	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
Location	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
Camera	CAMERA
Mircophone	RECORD_AUDIO
SMS	READ_SMS
	SEND_SMS
Call Log	READ_CALL_LOGS
Phone	CALL_PHONE
Calendar	READ_CALENDAR
Settings	WRITE_SETTING
Tasks	GET_TASKS
	KILL_BACKGROUND_PROCESS

then each sentence should be represented by numerical vectors. The textual pre-processing consists of the following sub-tasks:

a: SENTENCE SEGMENTATION

Given an app description may consist of sequential human-readable sentences, URLs, email addresses of developers, and emoji. However, URLs, email addresses, and emoji have no semantic value for predicting permissions. Thus, we employ regular expressions to remove irrelevant text from the raw description. To accurately predict the permissions from the descriptions. The tokenization from NLTK[4] is applied to split the descriptions into sentences based on sentence boundaries, such as list bullets and punctuation marks (i.e., '*', '?', '!', '⋆'). After sentence segmentation, all punctuation marks are also removed by regular expressions, as they have no semantics in the subsequent analysis. Finally, a given description can be split into a set of plain-text sentences $D_i = \{S_0, S_1, \ldots S_{n-1}\}$, where i denotes the i-th description in the dataset, and n is the number of sentences in the i-th description.

b: REMOVAL OF STOP WORDS AND STEMMING

In English, some words are extremely common and appear frequently but have little value in matching a sentence to candidates. These words are called stop words [39] and may be entirely excluded from the vocabulary in the descriptions. Stop words usually are pronouns, be verbs, or prepositions (e.g., "I", "am", and "it"). These words have no semantic value for NLP, furthermore, they increase the dimensions of features. We thus remove 179 common stop words in each sentence using the NLTK library.

Moreover, vocabularies with similar semantics would appear with different variants in English, such as "require," "requires," and "required." The deep learning model determines each variant to have individual semantics, leading to excessive consumption of computational resources owing to the increase in the feature dimensions for word embedding. To address this issue, we first convert the letters of all sentences into lowercase. Subsequently, we apply the Snowball stemmer[5] for word stemming. Removing stop words and stemming can potentially improve performance, as there remain fewer and only meaningful tokens. Finally, each sentence S_n in D_i only retains the words that may represent its semantics with a variable length.

c: WORD VECTOR GENERATION

To convert textual sentences into numerical vectors that can be recognized by computers, we use Word2Vec [40] to prepare a word embedding matrix pre-trained on 86,320 descriptions. Word2Vec takes a large corpus of text as input and produces a vector space. Each unique word in the corpus is assigned a corresponding vector in the space. To avoid rare words and neologisms, we consider a word valid for training if it appears at least 10 times in the entire dataset [40]. Word2Vec generates a set of numerical vectors with a dimensionality of 100 for each valid word, otherwise, a zero vector is assigned for invalid words that are excluded.

B. PERMISSION PREDICTION

Herein, we present the proposed learning model for predicting permissions. We employ Bi-LSTM with attention mechanism Fig. 4 shows an overview of the model design. The input of the model consists of a sequence of word indices for the sentence S in the description, which is combined with the matrix of word vectors in the embedding layer to be fed into deep learning model for text classification. Finally, the model outputs a vector of probabilities as a prediction for each sentence, indicating whether the sentence introduces any detectable permissions.

1) EMBEDDING LAYER

Textual sentences cannot be directly fed into a neural network. Hence, we generate two dictionaries to represent textual sentences by numerical indices. One dictionary maps each word from the pre-trained Word2Vec model to its word vector, represented as W_{word}. The other represents words with integer indices for all vocabularies in Word2Vec as W_{index}. The embedding layer uses these two dictionaries to generate a word embedding of each sentence into a set of numerical vectors. However, a sentence length varies in natural language. To retain the semantics of sentences as much as possible, we set the maximum sentence length among all descriptions to fix the input length. Zero padding is applied to sentences with length less than the maximum length. Thus, each sentence is represented as a tensor $S_i = [w_0, w_1, w_2, \ldots w_{L-1}]$ where w_i is an $L \times D$ word vector. The output of the embedding layer is a set of numerical vectors in the form of a tensor $T^{L \times D \times N}$, where L denotes the maximum length of the sentences in the dataset, D is the dimensionality of the word

[4]https://www.nltk.org/

[5]https://snowballstem.org/

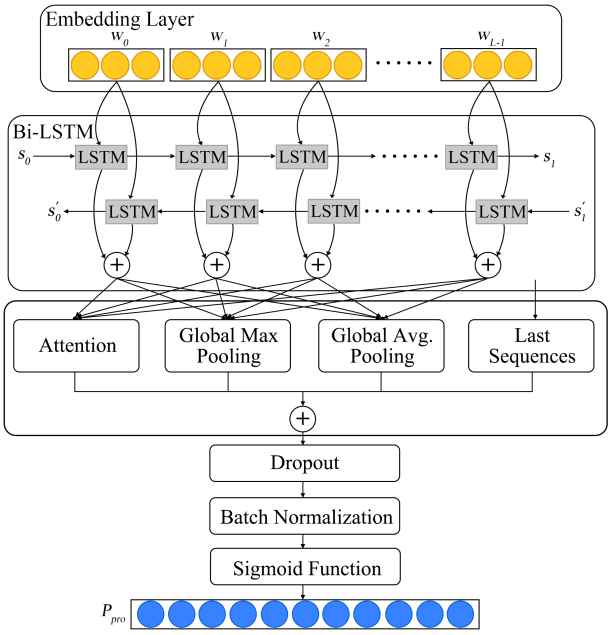


FIGURE 4. Model design of Bi-LSTM with attention mechanism.

embedding (which is 100 in our model), and N is the number of sentences in the input.

2) Bi-LSTM LAYER

In the app descriptions, the semantics depend not only on the preceding words but also on subsequent words at time t. Hence, we employ Bi-LSTM to capture context features. The one LSTM layer is a forward pass and handles the input from time 0 to t. The other one is a backward pass and captures features from previous words, it has the opposite direction of that of the previous layer. The hidden state h_t of a Bi-LSTM cell at time t consists of the hidden state of the forward pass $\overrightarrow{h_t}$ and that of the backward pass $\overleftarrow{h_t}$.

In this layer, we obtain the hidden states of the entire sequence for each time t. To accelerate the computation and prevent overfitting, the hidden states are split into two parts: the hidden states h_w of the entire sequence for each time, and the last sequence h_l at time t.

3) ATTENTION LAYER

To reduce the dimensionality of features and avoid overfitting, the hidden states h_w of the entire sequence are passed to the attention, global max pooling, and global average pooling layers. The attention mechanism is used to accurately capture text semantics. Moreover, the output of the attention layer is concatenated with that of the other two pooling layers and the last sequence h_l to reduce dimensionality. Thus, the concatenated output h can be expressed as follows:

$$h = h_{att} \oplus h_{max} \oplus h_{avg} \oplus h_l \tag{10}$$

4) OUTPUT LAYER

To prevent overfitting, we adopt a dropout rate of 0.4, and batch normalization [41] to randomly drop 40% of the units from the previous layer. Subsequently, the changed hidden states are passed to the fully connected layer to determine the usage of permissions. As a sentence may declare no or multiple permissions, each class can be considered as a binary classification. Thus, the sigmoid function regulates the output into the interval [0, 1]. Consequently, the probability of permission P_{pro} for the sentence S_i can be expressed as:

$$P_{pro} = \sigma(h_n) \tag{11}$$

where h_n is the normalized data from the batch normalization layer.

5) COMPILING MODEL

After model construction, we compile the model with the following settings: The word vector W_{word} is mapped to an embedding matrix W_{emb} that contains only the word vectors by the sequence of W_{index} as pre-trained word embeddings. Description-to-permissions is a multi-label classification, indicating that each permission is a binary classification [42]. Hence, we utilize binary cross-entropy as loss function to estimate the difference between the ground truth and the predicted values of each permission:

$$L(y, \widetilde{y}) = \sum_{i}^{N} \left[y_i \log(\widetilde{y_i}) + (1 - y_i) \log(1 - \widetilde{y_i}) \right] \tag{12}$$

where y is the ground truth of each sentence, $\widetilde{y}$ denotes the predicted result from the deep learning model, and N is the number of training samples.

C. FIDELITY CALCULATION

1) PERMISSION DETERMINATION

Given an unknown application, its description should be processed and predicted by Sections IV-A and IV-B. The results of the given description are presented as an $n \times 11$ tensor. To compare the predicted results with the requested permission P_{truth} at the app level, the sentence results are integrated into a vector P_{pred}. Moreover, the predicted labels cannot be determined by the default threshold (0.5) in an imbalanced dataset. We used threshold moving to determine the permission declared in the app description if p_i satisfies Eq. 13, where $p_i \in P_{pred}$, and Num_i^- and Num_i^+ denote the number of negative and positive samples for the i-th permission in the dataset, respectively. Finally, the predicted permissions are expressed as a vector P_{label}.

$$\frac{p_i}{1 - p_i} \times \frac{Num_i^-}{Num_i^+} > 1 \tag{13}$$

2) FIDELITY CALCULATION

We determine the number of intersected permissions in the predicted labels P_{label} and the ground truth P_{truth}

as a consistency degree. Hence, we apply the Jaccard Index [43], a statistical method to calculate a fidelity measure for the correlation between P_{label} and P_{truth} as follows:

$$J\left(P_{label}, P_{truth}\right) = \frac{|P_{label} \cap P_{truth}|}{|P_{label} \cup P_{truth}|} \tag{14}$$

The value of $J\left(P_{label}, P_{truth}\right)$ denotes the proportion of consistent permissions in the prediction and ground truth, and may reflect the fidelity of description-to-permissions at the app level.

V. EMPIRICAL STUDY

To evaluate the proposed approach, we set up experiments and evaluation criteria for highly imbalanced datasets to address the following four research questions.

- **RQ1**: Does FCDP outperform other deep learning models, such as AC-Net and textCNN, for permission prediction from app descriptions?
- **RQ2**: What is the effect of using different corpora on permission prediction?
- **RQ3**: Do the dimensions of word embedding significantly affect prediction performance?
- **RQ4**: What is the fidelity of descriptions in the wild? What is the difference in the fidelity of description-to-permissions between benign and malicious apps?

To answer these questions effectively, we performed both in-the-lab and in-the-wild experiments. The former is aimed at evaluating the performance of the proposed approach and identifying the factors that may affect permission prediction from app descriptions using an existing dataset. The latter is used to investigate the distribution of description fidelity in the wild.

To compare the influence of multiple factors, we leverage t-test [44] that is most commonly applied to determine if the means of two sets of data are significantly different, to identify whether the performances corresponding to two experimental samples are significantly different (regarding RQ 1–3). Thus, we propose the following hypotheses for the t-test:

- **Null hypothesis** H_0: The detection performance for permission P_i on condition A is similar to that on condition B.
- **Alternative hypothesis** H_1: The detection performance for permission P_i on condition A is significantly better than that on condition B.

Here, P_i is detectable permission in our approach, whereas conditions A and B denote two different configurations of the training models. The significance level α was set to 0.05. We reject the null hypothesis H_0 if the *p-value* calculated by the t-test is less than the significant level α ($p\text{-}value < \alpha$), and consider conditions A and B have significant difference. Otherwise, we accept the null hypothesis.

A. EXPERIMENTAL SETUP

1) DATASET

We collected 105,090 Android app descriptions from Google Play in November 2019. The dataset is released in Github.[6] The approach in previous studies was to collect app descriptions that introduced at least one permission. However, not all descriptions introduce permissions, and therefore the fidelity of app descriptions cannot be accurately assessed by using the aforementioned approach. To address this issue, we retained all descriptions as our dataset, which has a tremendous difference with others. These descriptions cover all app categories (a total of 37 categories) in Google Play, where the category "Parenting" with 1073 apps, represents the minimum size among all categories in the dataset.

In this study, we concentrate on analyzing descriptions in English. Therefore, we applied Compact Language Detector 3 (cld3) library[7] to extract English descriptions. This library was developed using a neural network where there can be false positives, particularly regarding short descriptions or neologisms. Non-English descriptions are filtered out in the following two circumstances:

1) If the majority of sentences in the description are detected as non-English, then the entire description should be removed.
2) If the description contains multiple languages, indicating that the description may contain multiple versions of languages, then only non-English sentences are removed.

After pre-processing, a total of 86,320 English descriptions satisfy our requirements as a corpus for training word vectors. The corpus contains 1,372,032 sentences and 41,797 unique words. It will be fed into deep learning model for word embedding. As manually labeling description-to-permissions is labor-intensive, we selected the labeled dataset of AC-Net provided by Feng *et al.* [36] to train our model. It contains 1,415 unique apps with 24,726 sentences. This dataset labels a total of 16 sensitive permissions in 11 groups, as shown in Table 2. If these permissions are misused, private user data are at high risk of leakage. For convenience of training, the permissions were divided into 11 groups according to usage. We predicted the permissions from the app description based on the 11 permissions groups.

2) EVALUATION SETUP

To equally evaluate the proposed model with others, we ran all compared models in the same environment with the same pre-trained word embedding, training and testing dataset. A random seed is assigned to split training and testing dataset, which ensures that the models can obtain the same data for each epoch. Moreover, we also set the unified hyper-parameter that is uncovered in the prior research for each model such as batch size and learning rate. We then conducted experiments using 10-fold cross-validation to train

[6]https://github.com/hhhwwwuuu/FCDP
[7]https://github.com/google/cld3

TABLE 3. The performance of different models.

Permissions	ROC-AUC						PR-AUC					
	FCDP	AC-Net	bi-LSTM	textCNN	RCNN	HAN	FCDP	AC-Net	bi-LSTM	textCNN	RCNN	HAN
STORAGE	**0.9452**	0.9388	0.9063	0.8464	0.8886	0.9054	0.5868	**0.5893**	0.5096	0.4677	0.4307	0.4750
CONTACTS	**0.9547**	0.9296	0.9111	0.8848	0.9081	0.9067	**0.6129**	0.5790	0.4898	0.5081	0.4565	0.4971
LOCATION	0.9782	**0.9792**	0.9610	0.9368	0.9480	0.9551	0.8010	**0.8416**	0.7403	0.7254	0.7019	0.7209
CAMERA	**0.9822**	0.9720	0.9439	0.9395	0.9410	0.9478	**0.7055**	0.5977	0.5678	0.6171	0.5246	0.5493
MICROPHONE	**0.9869**	0.9725	0.9217	0.9153	0.9234	0.9267	**0.5476**	0.4479	0.3734	0.3745	0.4037	0.3593
SMS	**0.9979**	0.9971	0.9898	0.9883	0.9893	0.9833	**0.8047**	0.7747	0.7757	0.7038	0.6915	0.7064
CALL LOG	**0.9903**	0.9895	0.9707	0.9524	0.9078	0.9793	0.3089	0.2164	0.2513	0.0847	0.2553	**0.3373**
PHONE	**0.9975**	0.9954	0.9961	0.9864	0.9821	0.9945	**0.6688**	0.6430	0.5179	0.5045	0.5320	0.5274
CALENDAR	**0.9993**	0.9992	0.9988	0.9991	0.9987	0.9992	0.7363	0.7344	0.6407	**0.7477**	0.6189	0.6900
SETTINGS	**0.9528**	0.9422	0.9010	0.7642	0.8805	0.8635	**0.3898**	0.3874	0.3087	0.1763	0.2900	0.3012
TASKS	**0.8624**	0.8494	0.7866	0.6491	0.7970	0.8204	**0.4285**	0.3784	0.3004	0.1760	0.2509	0.2437
Avg.	**0.9679**	0.9604	0.9352	0.8966	0.9240	0.9347	**0.5992**	0.5627	0.4978	0.4623	0.4687	0.4916

and evaluate our classifier. We randomly shuffled the dataset and selected 20% of sentences from the dataset as the testing set for evaluation. The rest of the dataset was equally divided into 10 sets for the training model. In each fold of the cross-validation, nine sets were sequentially used for training, and the remaining set was used for validation. The experiments were conducted on DGX-Station under Ubuntu 18.04. We used four NVIDIA Tesla V100 for training, and one GPU for testing.

B. EVALUATION METRICS

The dataset provided by AC-Net is highly imbalanced for each permission and includes lots of negative samples that do not contain any semantic of detected permissions. For instance, CALENDAR permission is introduced in 287 sentences (1.16%). Since accuracy cannot reflect the prediction performance for imbalanced dataset [45], we selected two other metrics: Area Under Curve of Receiver Operating Characteristic curve (ROC-AUC) [46], and Area Under Curve of Precision–Recall Curve (PR-AUC) [47].

The ROC curve is a chart that visualizes the trade-off between the true positive rate (TPR) and false-positive rate (FPR) on the different threshold [46]. ROC-AUC is the area under the ROC curve. Higher TPR with lower FPR indicates better performance on each threshold, and a ROC curve close to the top-left side indicates that the current model has the best performance. That is, the ROC-AUC can be used to evaluate classification performance and compare this performance with that of other classifiers. Eq. 15 can be used to calculate the ROC-AUC, where P and N are the numbers of positive and negative samples, respectively. The ω_1 function returns 1 if the positive probability is lower than the negative probability from the classifier. Otherwise, it returns 0. The ω_2 function returns 1 if the positive probability equals the negative probability.

$$ROC = 1 - \frac{1}{P \times N} \sum_{x^+ \in D^+} \sum_{x^- \in D^-} \left(\omega_1 - \frac{1}{2} \omega_2 \right) \quad (15)$$

Similarly, the precision-recall curve [47] is used to measure the trade-off between precision and recall on a different threshold. The PR-AUC is also used to evaluate the performance of classifiers. It can be calculated as follows:

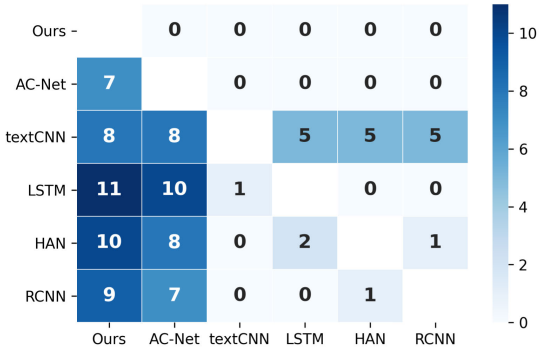
$$\text{PR-AUC} = \sum_n (R_n - R_{n-1}) P_n \quad (16)$$

where R_n and P_n denote recall and precision on the n-th threshold, respectively.

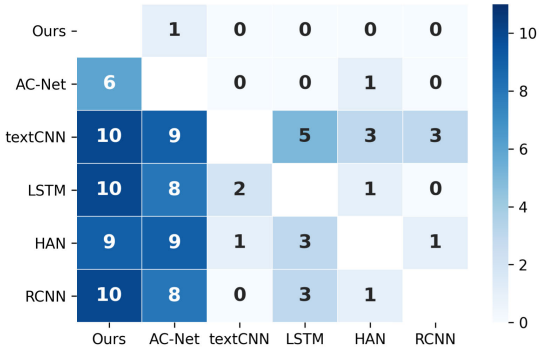
C. RQ1: THE PERFORMANCE OF OUR APPROACH

To evaluate the effectiveness of FCDP in predicting permissions from app descriptions, we compared it with the following deep learning models: AC-Net [36], Bi-LSTM, Convolutional Neural Networks for text classification (textCNN) [48], Hierarchical Attention Networks (HAN) [24] and Recurrent Convolutional Neural Networks (RCNN) [49]. To perform a valid comparison, we set identical hyper-parameters for each model, including corpus for training word vectors, the dimensionality of word embeddings, and different random seeds for shuffling and splitting datasets.

Table 3 shows the results of the evaluation under different models. Overall, the proposed approach outperforms AC-Net on all permissions groups in terms of ROC-AUC except for `LOCATION`. PR-AUC of the proposed model also surpassed that of AC-Net on 7 out of 11 permission groups. However, the ROC-AUC and PR-AUC of the proposed model are quite close to those of the others, with a negligible difference. In such cases (i.e., with a 0.001 difference), we cannot meaningfully compare the prediction performance for a permission, particularly for the permissions `SMS`, `PHONE`, and `CALENDAR`. To verify that the performance of the proposed model is better than other models, we trained each model 10 times to obtain their ROC-AUC and PR-AUC as samples of t-test. Then, t-test is performed to determine whether the performances have the significant difference in each permission group of every model pair. 11 t-tests were performed for each model pair, and a total of 15 model pairs for each evaluation metrics (i.e. ROC-AUC and PR-AUC) among 6 deep learning models have been tested. Hence, we performed 330 t-tests for RQ 1. The results integrated into a heatmap are shown in Fig. 5, where the ROC-AUC and



(a) ROC-AUC



(b) PR-AUC

FIGURE 5. Significant difference among six deep learning models.

PR-AUC results of the models are seen. The value of each cell in Fig. 5 denotes the number of significant difference in the performances of 11 detected permission groups among two models. For instance, the cell with value 6 indicates that 6 out of 11 permission groups of our model significantly outperform AC-Net based on PR-AUC, as shown in Fig. 5 (b). The models exhibited similar detecting performance for the other permissions. Thus, the proposed model is significantly better than AC-Net in predicting permissions from descriptions in terms of both ROC-AUC and PR-AUC. Furthermore, it outperforms the other text-classification models.

Answer to RQ 1: The proposed model can effectively predict permissions from app descriptions. It slightly outperforms the state-of-the-art model, AC-Net (by over 3.65% in terms of PR-AUC) and significantly surpasses the other simple models.

D. RQ2: COMPARISON THE PERFORMANCE BETWEEN DIFFERENT CORPORA

To evaluate the effect of different corpora on FCDP, we used four corpora to generate the word vectors: the domain corpus in Section IV-A, two common corpora in NLP, namely, Wikipedia [50] and Twitter [51], and random initialization. The method of random initialization updates the word vectors during training without feeding any external corpus.

As shown in Table 4, the prediction performance for 8 out of 11 permissions was better by using the domain corpus than by using the other corpora in terms of ROC-AUC and PR-AUC. Since imbalanced dataset affects the FPR, the difference in ROC-AUC among corpora is small and cannot reflect the effect on positive samples. Moreover, the results cannot indicate which corpus has a significant effect on predicting permissions with fine differences. For instance, the Wikipedia corpus results in a ROC-AUC of 0.9980 for the `SMS` permission, which represents a difference of 0.0001 compared with the domain corpus. Thus, we again perform *t-tests* to determine whether there is a significant difference in terms of PR-AUC among the corpora, as shown in Fig. 6. The performance achieved by using a corpus is significantly better than that achieved by random initialization for generating word vectors.

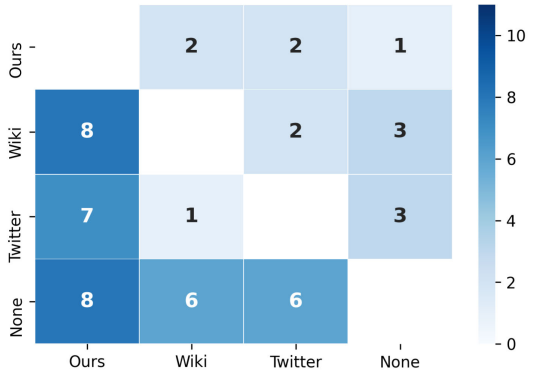


FIGURE 6. Significant difference among domain corpus, Wikipedia, Twitter and random initialization.

Answer to RQ 2: Compared with random initialization, using a corpus for word embedding is statistically significant for description analysis. In particular, the domain corpus is superior to a common external corpus in NLP because the domain corpus contains more semantics for predicting permissions.

E. RQ3: THE IMPACT OF DIMENSIONALITIES OF WORD EMBEDDINGS

Dimensionality affects the quality of word embedding [52]. We further characterize the effect of the dimensionality of word embeddings on the prediction performance for permissions from app descriptions. Table 5 shows the results of the evaluation under different dimensionality configurations. We selected 50, 100, 200, and 300 dimensions (where 100 dimensions were employed in our approach) in the evaluation. As before, we applied the t-test to determine whether there is a significant difference among different configurations, as shown in Fig. 7. The performance achieved by dimensionality of 100 is significantly better than that by other configurations.

TABLE 4. Effect of different corpora.

Permissions	ROC-AUC				PR-AUC			
	Ours	Wikipedia	Twitter	None	Ours	Wikipedia	Twitter	None
Storage	**0.9452**	0.9331	0.9430	0.8799	**0.5868**	0.5534	0.5750	0.4450
Contacts	**0.9547**	0.9169	0.9174	0.9156	**0.6129**	0.5540	0.5555	0.4587
Location	0.9782	0.9816	**0.9819**	0.9155	0.8010	**0.8206**	0.8094	0.6725
Camera	**0.9822**	0.9570	0.9651	0.9600	**0.7055**	0.5582	0.5537	0.5883
Mircophone	**0.9869**	0.9832	0.9656	0.9194	0.5476	0.4095	0.4537	**0.5788**
SMS	0.9979	**0.9980**	0.9978	0.9902	0.8047	**0.8393**	0.8180	0.7109
Call Log	**0.9903**	0.9887	0.9901	0.9505	**0.3089**	0.2245	0.2206	0.2323
Phone	**0.9975**	0.9967	0.9936	0.9955	**0.6688**	0.6309	0.6275	0.6371
Calendar	**0.9993**	0.9985	0.9986	0.9983	**0.7363**	0.4564	0.5384	0.7341
Settings	**0.9528**	0.9356	0.9300	0.8645	**0.3898**	0.3642	0.3459	0.2669
Tasks	0.8624	**0.9093**	0.8945	0.7785	**0.4285**	0.3426	0.3465	0.2842
Average	**0.9679**	0.9635	0.9616	0.9244	**0.5992**	0.5231	0.5313	0.5108

TABLE 5. Effect of different dimensionalities of word embedding.

Permissions	ROC-AUC				PR-AUC			
	50d	100d(Ours)	200d	300d	50d	100d(Ours)	200d	300d
Storage	0.9384	**0.9452**	0.9348	0.9317	0.5688	**0.5868**	0.5661	0.5546
Contacts	0.9394	**0.9547**	0.9496	0.9487	0.6015	**0.6129**	0.5854	0.5843
Location	0.9780	**0.9782**	0.9673	0.9634	0.7756	**0.8010**	0.7700	0.7560
Camera	0.9742	**0.9822**	0.9766	0.9749	0.6528	**0.7055**	0.6896	0.6812
Mircophone	0.9681	**0.9869**	0.9789	0.9789	0.5454	**0.5476**	0.5507	0.5534
SMS	0.9975	**0.9979**	0.9977	0.9974	0.7749	**0.8047**	0.7924	0.7737
Call Log	**0.9932**	0.9903	0.9907	0.9910	0.2301	0.3089	0.2969	**0.3422**
Phone	0.9974	0.9975	**0.9976**	**0.9976**	0.6275	0.6688	0.7014	**0.7133**
Calendar	0.9986	0.9993	0.9991	**0.9994**	0.5362	0.7363	0.6715	**0.7669**
Settings	0.9378	**0.9528**	0.9184	0.9105	0.3683	0.3898	**0.3907**	0.3784
Tasks	**0.8655**	0.8624	0.8494	0.8402	**0.4287**	0.4285	0.3672	0.3700
Average	0.9626	**0.9679**	0.9600	0.9576	0.5554	**0.5992**	0.5802	0.5885

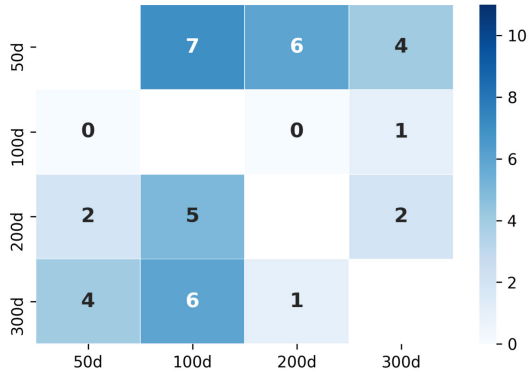


FIGURE 7. Significant difference in using different dimensionalities for word embedding.

Answer to RQ3: The dimensionality of word embeddings slightly affects the prediction performance for permissions from descriptions. A smaller or larger dimensionality may result in losing semantic features in the generation of the word vectors. Considering the trade-off between training overhead and performance, 100 dimensions for word embedding are suitable for the domain corpus.

F. RQ4: THE DISTRIBUTION OF FIDELITY IN THE WILD

To investigate the actual number of app descriptions that accurately reflect the requested permission in the wild, we have collected numerous English descriptions and their APK files from Google Play. We also have collected a set of Android malware released between January 2017 and September 2020 from AndroZoo [53], in order to understand the different distribution of fidelity on benign and malicious apps. AndroZoo is a weekly updated Android repository with over 10 million apps from various sources. All apps in AndroZoo are scanned by VirusTotal[8] that contains more than 50 anti-virus scanners to detect Android malware.

Benign Apps: We randomly collected 65,811 app descriptions in English from Google Play, and then used VirusTotal to scan the collected apps. We only selected the apps that were not flagged by any scanner. It is important to avoid the false positives that are generated by anti-virus scanners. As a result, 63,063 apps formed a benign set.

Malware Apps: Since Google Play also detects and removes malicious apps from their market, we only collected 28 malware out of 65,811 apps during random collection in Google Play. We then extracted more malicious apps from AndroZoo which still exists in Google Play. However, app descriptions can be modified and updated by developers as new functionalities are added with upgrades. To ensure that the selected malicious apps from AndroZoo can correctly match the corresponding version of descriptions in Google Play, we compared SHA256 of APK files to check

[8]https://www.virustotal.com/gui/home

whether two apps are identical. If the hash values of two apps matched, the app and its description were included in the malicious set. Otherwise, we tested the latest app from Google Play with VirusTotal for malware detection. The app is considered malware if at least 14 scanners flagged it as malware [54], [55]. After scanning, there are 1,202 apps in the malicious set.

In addition, we considered all valid English descriptions, even descriptions not introducing any permission, to reflect the actual fidelity of description-to-permissions in the wild. To the best of our knowledge, we are the first to consider all descriptions without the limitation of permission semantics. Finally, we retained 64,265 apps with their descriptions as our wild dataset.

Once the wild dataset was prepared, we predicted the permissions from the descriptions and calculated the corresponding fidelity to reflect the degree of matching description to permissions. Fig. 8 shows the distribution of fidelity for benign and malicious apps. It demonstrates that 4,613 benign and 348 malicious apps are completely disclosed all requested permissions in their descriptions. Whereas, the medians of fidelity are 0.636 and 0.818 for benign and malicious sets, respectively. The distribution of fidelity on benign apps indicates that average 63.6% usages of dangerous permissions are explicitly/implicitly presented in each description. On the contrary, the descriptions of malicious apps on the market show the higher fidelity on matching descriptions to permissions in Google Play. According to the investigation, two possible reasons may cause this bias. Firstly, as Google Play store removes all detected malware, we only have a small portion of malware with their descriptions in Google Play. Secondly, these malware which still are available in Google Play are elaborately pretended as benign apps to bypass the various detection techniques. Their descriptions intentionally disclose fictitious descriptions of functionalities, and the implied permissions in descriptions are used for other malicious purposes in implementation. However, the app description is to introduce its major functionalities and privacy usages for users before installation. It is necessary to regulate app description writing and present the authentic and understandable usages of PSI in descriptions.

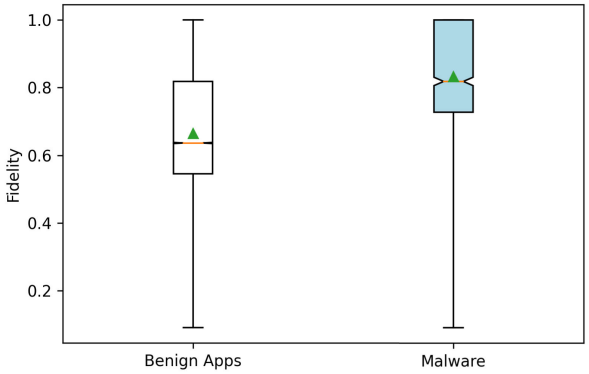


FIGURE 8. Distribution of Fidelity on Benign and Malicious Apps.

Answer to RQ 4: In our wild dataset, only 4,961 out of 64,265 app descriptions (7.72%) completely matched to their requested permissions in source code, where their fidelity is 1. The fidelity of benign and malicious apps is 0.636 and 0.818, respectively. The usages of dangerous permission cannot be accurately reflected in descriptions. Developers declare some unnecessary permissions in the manifest file without API invocations to decrease fidelity. As a result, app descriptions cannot be a feature to detect Android malware since descriptions is unable to precisely match to the implementation in source code. However, Google Play auditors can employ the fidelity of description-to-permissions to regulate description writings as an indicator in order to provide understandable descriptions with the usages of dangerous permissions for users.

VI. THREATS TO VALIDITY

We discuss the main issues that may threaten the validity of this study.

A. INTERNAL VALIDITY

The main threat to internal validity could be an imprecise classification in predicting permissions from app descriptions. The performance of prediction depends on the developers who wrote the app descriptions. According to the analysis of the descriptions collected in this study, some of these are excessively concise and do not introduce any permission. For instance, *Tbliya*[9] is an online shopping app. Its description only contains one sentence that introduces the app for online shopping, which cannot infer any dangerous permission. However, this app actually requests `Location` and `Storage` permissions when manual examining. To investigate the relationship between introducing permissions and the number of sentences, we randomly selected 50 apps with less than 10 sentences. If the number of sentences is less than 5, the app description may only contain useless illustrations that cannot provide any semantic information for predicting permissions. In the dataset, 32,712 descriptions (21.72%) of all collected descriptions consist of less than five sentences, representing an outlier in the prediction process owing to the lack of permission semantics.

Developers usually introduce the major functionalities of an app in the description to attract end-users, whereas secondary functionalities tend to remain unmentioned even if they involve sensitive permissions [5]. The lack of secondary functionalities in the descriptions also affects the prediction performance for permissions. Moreover, this situation may also cause the major functionalities to occupy a large portion of the dataset, which may therefore become more imbalanced. To overcome the issues discussed above, we discuss several viable strategies for regulating description writing as possible future research. 1) Google Play needs to assess

[9]https://play.google.com/store/apps/details?id=tbliya.com

whether all usages of dangerous permissions are explicitly declared in descriptions by using FCDP or other similar methods, and give feedback to developers for modifying the descriptions. 2) According to the results of analyzing API calls, the purposes of permission usages will be converted into natural language based on code context. 3) Developing a description writer to automatically generate a description that properly discloses the purposes of requesting dangerous permissions without human intervention.

B. EXTERNAL VALIDITY

Our collected benign descriptions in this study represent a small portion of the apps in Google Play. To overcome the potential bias from the dataset, we randomly collected the samples in each category from Google Play without considering the popularity and whether the descriptions contain any permission to ensure that resulting dataset may represent the real fidelity of description-to-permissions. Moreover, the quality of labeled dataset affects the prediction performance of deep learning models owing to the distribution of each labeled permission and human inspection. Unfortunately, we could not expand the dataset with the same annotation guidelines.

VII. CONCLUSION

In this article, we proposed an approach called FCDP that measures the fidelity of description-to-permissions on Google Play to bridge the semantic gap between app descriptions and requested permissions. As a result, it was demonstrated that FCDP outperforms the state-of-the-art method AC-Net by over 3.65% in precisely predicting permissions on PR-AUC.

We applied FCDP to 64,265 Android apps to assess their fidelity of description-to-permissions in the wild. Our results demonstrated that this fidelity could not be used as a metric for detecting Android malware. As approximately 21.72% of the descriptions in the dataset did not contain any semantic about permissions, and secondary functionalities involved sensitive permissions were not always mentioned in the descriptions. Our findings indicated that more effort should be invested to assist Google Play auditors in regulating app description writing and permission requirements recommendations by using the fidelity of description-to-permissions. Moreover, the end-users also can utilize the fidelity to understand whether the app truthfully introduces their usage of sensitive permissions without requiring the background of Android security.

REFERENCES

[1] Statcounter. *Mobile Operating System Market Share Worldwide*. Accessed: May 28, 2020. [Online] Available:https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] Statista. *Distribution of Free and Paid Android Apps in the Google Play Store*. Accessed: May 29, 2020. [Online]. Available: https://www.statista.com/statistics/266211/distribution-of-free-and-paid-and roid-apps
[3] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. N. Leung, "Enhancing the description-to-behavior fidelity in Android apps with privacy policy," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 834–854, Sep. 2018.
[4] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "AutoCog: Measuring the Description-to-permission fidelity in Android applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2014, pp. 1354–1365.
[5] T. Watanabe, M. Akiyama, T. Sakai, and T. Mori, "Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps," in *Proc. 11th Symp. Usable Privacy Secur. (SOUPS)*, 2015, pp. 241–255.
[6] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. 8th Symp. Usable Privacy Secur. (SOUPS)*, 2012, pp. 1–14.
[7] H. Wang, J. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (UbiComp)*, 2015, pp. 1107–1118.
[8] B. Rashidi, C. Fung, and E. Bertino, "Android malicious application detection using support vector machine and active learning," in *Proc. 13th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2017, pp. 1–9.
[9] Google. *Privacy, Security, and Deception*. Accessed: May 29, 2020. [Online] Available:https://play.google.com/about/privacy-security-deception
[10] X. Liu, Y. Leng, W. Yang, C. Zhai, and T. Xie, "Mining Android app descriptions for permission requirements recommendation," in *Proc. IEEE 26th Int. Requirements Eng. Conf. (RE)*, Aug. 2018, pp. 147–158.
[11] I. A. Dogru and M. Önder, "AppPerm analyzer: Malware detection system based on Android permissions and permission groups," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, no. 3, pp. 427–450, Mar. 2020.
[12] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, "On the (un) reliability of privacy policies in Android apps," 2020, *arXiv:2004.08559*. [Online]. Available: http://arxiv.org/abs/2004.08559
[13] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.
[14] D. C. Nguyen, E. Derr, M. Backes, and S. Bugiel, "Short text, large effect: Measuring the impact of user reviews on Android app security & privacy," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 555–569.
[15] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *Proc. 32rd AAAI Conf. Artif. Intell.*, 2018, pp. 5141–5148.
[16] Z. Tang, Y. Shi, D. Wang, Y. Feng, and S. Zhang, "Memory visualization for gated recurrent neural networks in speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 2736–2740.
[17] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-CNN," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, 2018, pp. 1063–1072.
[18] M. Al-Smadi, O. Qawasmeh, M. Al-Ayyoub, Y. Jararweh, and B. Gupta, "Deep recurrent neural network vs. Support vector machine for aspect-based sentiment analysis of Arabic hotels' reviews," *J. Comput. Sci.*, vol. 27, pp. 386–393, Jul. 2018.
[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[20] P. Liu, X. Qiu, and X. Huang, "Adversarial multi-task learning for text classification," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2017, pp. 1–10.
[21] E. Pan, X. Mei, Q. Wang, Y. Ma, and J. Ma, "Spectral-spatial classification for hyperspectral image based on a single GRU," *Neurocomputing*, vol. 387, pp. 150–160, Apr. 2020.
[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.
[23] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4438–4446.
[24] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2016, pp. 1480–1489.
[25] Y. Wang, M. Huang, X. Zhu, and L. Zhao, "Attention-based LSTM for aspect-level sentiment classification," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2016, pp. 606–615.

[26] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, "An attentive survey of attention models," 2019, *arXiv:1904.02874*. [Online]. Available: http://arxiv.org/abs/1904.02874

[27] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 217–228.

[28] P. Gadient, M. Ghafari, P. Frischknecht, and O. Nierstrasz, "Security code smells in Android ICC," *Empirical Softw. Eng.*, vol. 24, no. 5, pp. 3046–3076, Oct. 2019.

[29] M. Y. Karim, H. Kagdi, and M. Di Penta, "Mining Android apps to recommend permissions," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, vol. 1, Mar. 2016, pp. 427–437.

[30] S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F. Shen, S. Y. Ko, and L. Ziarek, "Flow permissions for Android," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 652–657.

[31] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing Android sources and sinks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 14, 2014, p. 1125.

[32] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *Proc. 17th ACM Symp. Access Control Models Technol. (SACMAT)*, 2012, pp. 13–22.

[33] B. Rashidi, C. Fung, A. Nguyen, T. Vu, and E. Bertino, "Android user privacy preserving through crowdsourcing," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 773–787, Mar. 2018.

[34] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 527–542.

[35] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 1025–1035.

[36] Y. Feng, L. Chen, A. Zheng, C. Gao, and Z. Zheng, "AC-Net: Assessing the consistency of description and permission in Android apps," *IEEE Access*, vol. 7, pp. 57829–57842, 2019.

[37] L. Yu, X. Luo, C. Qian, and S. Wang, "Revisiting the description-to-behavior fidelity in Android applications," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, vol. 1, Mar. 2016, pp. 415–426.

[38] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, 2011, pp. 627–638.

[39] M. Gerlach, H. Shi, and L. A. N. Amaral, "A universal information theoretic approach to the identification of stopwords," *Nature Mach. Intell.*, vol. 1, no. 12, pp. 606–612, Dec. 2019.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2013, pp. 1–12.

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[42] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014.

[43] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou, "Following Devil's footprints: Cross-platform analysis of potentially harmful libraries on Android and iOS," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 357–376.

[44] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the Mann–Whitney u test," *Behav. Ecol.*, vol. 17, no. 4, pp. 688–690, Jul. 2006.

[45] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2018.

[46] H. Narasimhan and S. Agarwal, "Support vector algorithms for optimizing the partial area under the ROC curve," *Neural Comput.*, vol. 29, no. 7, pp. 1919–1963, Jul. 2017.

[47] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 233–240.

[48] Y. Kim, "Convolutional neural networks for sentence classification," 2014, *arXiv:1408.5882*. [Online]. Available: http://arxiv.org/abs/1408.5882

[49] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2267–2273.

[50] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[51] H. Saif, M. Fernandez, Y. He, and H. Alani, "Evaluation datasets for Twitter sentiment analysis," in *Proc. Int. Workshop Emotion Sentiment Social Expressive Media*, 2013, pp. 9–21.

[52] Z. Yin and Y. Shen, "On the dimensionality of word embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 887–898.

[53] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 468–471.

[54] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Jun. 2019, pp. 47–62.

[55] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 14, 2014, pp. 23–26.

ZHIQIANG WU (Graduate Student Member, IEEE) received the B.S. degree in computer science from Shanghai Polytechnic University, China, in 2015, and the M.S. degree from Hanyang University, Ansan, South Korea, in 2017, where he is currently pursuing the Ph.D. degree in computer science with the Department of Computer Science and Engineering. His research interests include Android apps analysis, code smells on security, and software refactoring on mobile apps.

XIN CHEN was born in Liaoning, China, in 1995. She received the B.S. degree in software engineering from the Harbin University of Science and Technology, China, in 2017. She is currently pursuing the M.S. degree leading to the Ph.D. Program with the Department of Computer Science and Engineering, Hanyang University, Ansan, South Korea. Her research interests include software smell detection, code refactoring, and code plagiarism.

SCOTT UK-JIN LEE (Member, IEEE) received the B.S. degree in software engineering and the Ph.D. degree in computer science from The University of Auckland, New Zealand. After the Ph.D. degree, he was a Postdoctoral Research Fellow with the Commissariat à l'Énergie Atomique et aux Énergies Alternatives, France. He has been with the Department of Computer Science and Engineering, Hanyang University, ERICA Campus, South Korea, since 2011. He has been serving as an Associate Professor of bio artificial intelligence with the Department of Computer Science and Engineering. His research interests include software engineering, formal methods, and quality assurance. He is also a member of the Korean Institute of Information Scientists and Engineers and the Korean Society of Computer and Information. He has served as an editor, the technical chair, and a committee member for several journals and conferences.