IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 18, 2019, accepted April 15, 2019, date of publication April 22, 2019, date of current version May 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2912210

AC-Net: Assessing the Consistency of Description and Permission in Android Apps

YINGLAN FENG[1,2], LIANG CHEN[1,2], (Member, IEEE), ANGYU ZHENG[1,2], CUIYUN GAO[3], (Student Member, IEEE), AND ZIBIN ZHENG[1,2], (Senior Member, IEEE)

[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[2]National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou 510006, China
[3]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077

Corresponding author: Liang Chen (chenliang6@mail.sysu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61702568 and Grant U1711267, in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355, and in part by the Fundamental Research Funds for the Central Universities under Grant 17lgpy117.

ABSTRACT With Android applications (apps) becoming increasingly popular, there exist huge risks lurking in the app marketplaces as most malicious software attempt to collect users' private information without their awareness. Although these apps request users' authorization for permissions, the users can still face privacy leakage issues due to their limited knowledge in distinguishing permissions. Thus, accurate and automatic permission checking is necessary and important for users' privacy protection. According to previous studies, analyzing app descriptions is a helpful way to examine whether some permissions are required for apps. Different from those studies, we consider app permissions from a more fine-grained perspective and aim at predicting the multiple correspondent permissions to one sentence of app description. In this paper, we propose an end-to-end framework for assessing the consistency between descriptions and permissions, named Assessing Consistency based on neural Network (AC-Net). For evaluation, a new dataset involving the description-to-permission correspondences of 1415 popular Android apps was built. The experiments demonstrate that AC-Net significantly outperforms the state-of-the-art method by over 24.5% in accurately predicting permissions from descriptions.

INDEX TERMS Android security, app descriptions, app permissions, consistency assessment, text classification, deep learning.

I. INTRODUCTION

As the mobile internet is booming, mobile apps are becoming an integral part of peoples' daily life, providing functions including file sharing, message sending, and recreation. Users download apps from app markets, such as Apple's App Store and Google Play Store, and install them with permissions granted on their mobile phones. Some of these granted permissions (*e.g.*, contact list and photo gallery access) are related to users' privacy. They allow the apps to obtain sensitive information, leading to potential information leakage and privacy breach [1]. For example, in March 2018, the Facebook-Cambridge Analytica data scandal was reported involving the collection of 87 million Facebook users' personally identifiable information through apps [2], [3], which triggered heated public discussion and made people pay much more attention to the protection of private information.

To tackle these problems, the previous Android versions adopt the install-time permission mechanism, *i.e.*, show a list of claimed permissions to users and request for their authorization before installation. However, most of the users only focus on the functionality itself, but few patiently check and make out the declared permissions [4]. Recently, the official Android system proposes a new runtime permission mechanism that users are asked to grant the dangerous permissions at runtime instead of being notified of the permissions during installation [5], [6]. Even so, users may lack the knowledge to decide which permissions are necessary for an app. For example, facing a map app, some users would grant its malicious READ_CONTACT permission request, leading to possible privacy leakage for them.

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

To detect malicious app and assist users' decision-making process, some previous work resorts to exploring the relationships between source code and privacy-related resources such as requested permissions [7], [8], privacy policies [9]–[12], and app description topics [13]. In this paper, we aim at disclosing apps' privacy to users and focus on exploiting app descriptions to check permissions' necessity, commonly termed as *description-to-permission fidelity* [14]. In addition to helping users understand whether an app is over-privileged, such fidelity checking can also assist developers in writing high-quality descriptions [15], and enhance the trustworthiness of the app stores.

Accurately predicting permission usage from app descriptions is challenging. Although unsupervised semantics-based methods are labor-saving, they generally require sufficient semantic information (*e.g.*, API documents [16] and descriptions [14]), and are prone to generate false correlations, *i.e.*, encountering some patterns with high relevance score but actually meaningless. For instance, if many anti-virus apps declare the permission GET_TASK, the noun ''antivirus'' may frequently appear with this permission together, thereby forming a relational pattern which may not reasonable [14]. Supervised methods tend to achieve better performance by learning knowledge from labeled data, but they usually need huge manual labor in building training dataset (*e.g.*, description-privacy pairs). To alleviate the labor cost in supervised methods, we focus on accurately measuring description-to-permission fidelity by using a relatively small training dataset.

In our work, we leverage the natural language processing (NLP) technique and propose an end-to-end framework for **A**ssessing **C**onsistency based on neural **Net**work, named **AC-Net**. AC-Net takes app descriptions as input and outputs the consistency between app descriptions and permissions. We propose a novel learning model called TextGRU to learn semantic representations of app descriptions better. Instead of directly adopting deep learning models, we combine multilevel features during implementation. Through learning, the model predicts the permissions corresponding to a description sentence.

Furthermore, we pay more attention to the degree of consistency between an app's description and declared permissions instead of providing a simple ''yes'' or ''no'' answer as in existing studies. Such an output style has multi-fold advantages: 1) For app evaluation systems of app markets, fidelity measurement based on the degree of consistency is more tolerant for some app descriptions lack of details; 2) For users, they can personally choose the apps with acceptable fidelity besides the fewer apps with 100% fidelity; and 3) During mobile app development, developers can update app descriptions sentence-by-sentence according to the degree of consistency.

The main contributions of this paper are summarized below:

- We leverage techniques in the field of NLP and propose a deep learning based framework named AC-Net. It is the first attempt to apply deep learning techniques in assessing the description-to-permission consistency of mobile apps.
- We manually labeled permissions of nearly 25,000 description sentences for 1,415 popular Android apps for evaluation of our model.[1] Comprehensive experiments show that AC-Net achieves significantly better performance than the state-of-the-art methods, with over 24.5% improvement in accurate consistency assessment.
- Our labeled dataset is released online[2] for making our experiments reproducible and promoting future research. To the best of our knowledge, the scale of our released dataset, both the number of app's description sentences and the number of permissions, is the largest in the field of fidelity measurement between descriptions and permissions.

The rest of this paper is organized as follows. Section II introduces the background knowledge and motivating examples. Next, we illustrate the detailed design of our AC-Net framework in Section III. Section IV presents the design of the empirical study followed by the experimental results shown in Section V. Section VI discusses the threat to our study, and then Section VII reports on related work. Finally, we conclude the paper in Section VIII.

II. BACKGROUND AND MOTIVATION

To assist readers in better understanding the motivation of our task and the involved methodology, we first introduce permission-based privacy protection mechanisms, app permissions, and descriptions, then illustrate the importance of well assessing the consistency of the two aspects, and finally explain some NLP knowledge related to our proposed methodology.

A. ANDROID ARTIFACTS

1) ANDROID SECURITY MECHANISMS

The core design of the Android security architecture is that by default no app has the privilege to perform any operation that adversely affects other apps or users [1]. This includes reading or writing the user's private data (such as contacts or SMS), reading or writing files from other apps, accessing network, keeping the device awake, etc. Since each Android app runs in a process sandbox, the app must explicitly share resources and data, declaring which permissions are required to obtain additional functionality that is not provided by the base sandbox. Apps statically declare the permissions they need to get access to the resources, and the Android system prompts the user for authorization.

The existing permission-based access control for Android security is vulnerable to attack, such as privacy leakage (when users grant permissions to malware and unknowingly allow

[1] This dataset is created for ensuring the correctness of the correspondence between descriptions and permissions, which is discussed in Section VI. In practice, the scale of the dataset can be easily extended for more app descriptions and permissions.

[2] https://github.com/LinkyVon/AC-Net

access to personal information) and even might be bugged or be controlled (some malware can monitor the phone call and SMS without user's consent or send various commands to remotely control the device) [1].

2) APP PERMISSIONS

Apps request permissions either during their installation time (for Android 5.1.1 and below) or runtime (for Android 6.0 and higher).

Install-time Permission Requests: Android introduces a permission-based security model to limit access to system resources and private data. All the needed permissions must be declared in the AndroidManifest.xml file and granted during install time. If the user denies these permissions request, the system will cancel the installation of the app. On the contrary, the app will have access to all those permissions. However, there are quite a few developers overly specify their permission requirements [17], and many users often fail to make a better judgment due to their less expertise, which increases the attack surface and causes additional security risks from apps breach [18].

Runtime Permission Requests: From Android 6.0 (API level 23, runtime requests version) onward, system permissions are divided into several levels of protection. The most important levels are Normal and Dangerous.

The system will automatically grant access for ***Normal Permissions*** which have little risk to user's privacy. These permissions cannot be revoked as well. For instance, users are not notified about the authorization of the permission KILL_BACKGROUND_PROCESSES as it belongs to ***Normal Permissions***.

For ***Dangerous Permissions***, users can grant them during app runtime instead of in the period of app installation, which seems to be good news for users. However, Wijesekera *et al.* [19] found that at least 80% of the participants would tend to block at least one permission request and most of them expressed a desire to prevent over a third of requests following their expectations. Without necessary permissions granted, the user experience will be influenced by the pop-up permission dialogues and limited functionalities.

Unlike normal and dangerous levels, ***Signature Permissions*** SYSTEM_ALERT_WINDOW and WRITE_SETTINGS are particularly sensitive. If an app requires one of these permissions, it must send an intent with action to request the user's approval.

3) APP DESCRIPTIONS

App descriptions are a crucial channel for developers to communicate apps' information to users. Besides including app functionality, app descriptions usually implicitly deliver the permission usage information. An example of app description for the *CloudMagic Email* (*com.cloudmagic.mail*), a communication app, is shown in Fig. 1. The feature list elaborates the functionalities provided by the app and can indicate the app's permission usage. For example, the description sentence "*Download attachments in background*" illustrates that the app provides file downloading service, which means that the app needs WRITE_EXTERNAL_STORAGE permission.

Works with Gmail, Exchange, Yahoo, Outlook, iCloud, Google Apps, Office 365 and any IMAP account!

Accolades:
★ "Winner of the Internet's highest honor" - The Webby Awards People's Voice.
★ "The best mobile email app" - The Wall Street Journal.

Feature List:
- Connected with your favorite tools. Save emails to Wunderlist, Todoist, Evernote, OneNote, Trello, Zendesk, Salesforce.com, Asana, Instapaper, OmniFocus and many more apps.
- Attach files from Dropbox, iCloud Drive and other file storage services.
- Download attachments in background.

...

FIGURE 1. Snippet of the app description for *CloudMagic Email*.

S1: Works with Gmail, Exchange, Yahoo, Outlook, iCloud, Google Apps, Office 365 and any IMAP account.
S2: Accolades: "Winner of the Internet's highest honor" - The Webby Awards People's Voice.
S3: "The best mobile email app" - The Wall Street Journal.
S4: Feature List: Connected with your favorite tools.
S5: Save emails to Wunderlist, Todoist, Evernote, OneNote, Trello, Zendesk, Salesforce.com, Asana, Instapaper, OmniFocus and many more apps.
S6: Attach files from Dropbox, iCloud Drive and other file storage services.
S7: Download attachments in background.

FIGURE 2. Description sentences corresponding to the description snippet in Fig. 1. The red fonts indicate permission-related words, and the first two characters of each sentence denote the sentence identifier.

B. MOTIVATING EXAMPLE

Assessing the consistency between app description and permissions bridge the semantic gap between app-stated functions and the actual app permission usage. In our work, AC-Net quantifies the correspondence between descriptions and permissions, thereby can alert potentially malicious or redundant permission declaration in advance.

To show how AC-Net works, we use the app *CloudMagic Email*[3] as an example, where we successfully detect suspicious permission usage based on app descriptions. *CloudMagic Email*'s declared permissions include:

p_1: GET_ACCOUNTS;
p_2: WRITE_EXTERNAL_STORAGE;
p_3: READ_CALENDAR;
p_4: READ_CONTACTS;
p_5: ACCESS_FINE_LOCATION.

To illustrate that AC-Net can address the problem of false and misleading advertising likewise, we take the permission p_6: RECORD_AUDIO, which is not declared by this app but within the measuring range of AC-Net, as an example.

We split description into seven sentences, as depicted in Fig. 2, and predict the permission usage of each sentence. The predicted results can be shown as a sentence-permission matrix, *i.e.*, Table 1. Each row in the matrix represents the probability distributions of the permission usage expressed by the corresponding sentence, with the last row summarizing the permission distribution of all the sentences.

[3]This is a free communication app. The version we crawled was released on December 21, 2015 and had an average rating of 4.47 out of 89281 downloads. This app renamed as 'Newton Mail' is still available in Google Play now.

TABLE 1. Prediction results of permission usage based on the description sentences in Fig. 2. Each row represents the probability distributions of the studied permissions, with the last row summarizing the probabilities of permission usage expressed by the whole description. The bold values indicate the maximum ones by column, and the column highlighted in grey means that the corresponding permission is not declared by this app but within the detecting range of the system.

Sentence	p_1	p_2	p_3	p_4	p_5	p_6
s_1	**0.96**	0.12	0.03	0.00	0.00	0.00
s_2	0.02	0.00	0.00	0.00	0.00	0.00
s_3	0.11	0.00	0.00	**0.13**	0.00	0.00
s_4	0.64	0.02	0.00	0.10	0.00	0.00
s_5	0.78	0.89	**0.58**	0.12	**0.01**	0.00
s_6	0.77	0.34	0.01	0.00	0.00	0.00
s_7	0.00	**0.95**	0.00	0.00	0.00	0.00
TOTAL	**0.96**	**0.95**	**0.58**	**0.13**	**0.01**	**0.00**

According to the matrix, It can be found that the description strongly suggests the use of permissions p_1 and p_2, but weakly relates to the permissions p_4 (READ_CONTACTS) and p_5 (ACCESS_FINE_LOCATION). These weak relationships are attributed to the unspecified location and contact access in the description. In addition, since the permission p_6 (RECORD_AUDIO) has no relation with the description at all, AC-Net helps to clear off the suspicion that the app overstates its functionalities. For end users, the assessment result can help them make decisions following their own privacy preferences. For the app developers, they can improve the description or modify the requested permissions accordingly.

Moreover, one description sentence may imply more than one permission request in some cases. For example, S_5 strongly relates to p_1, p_2, and p_3. Although S_5 is most involved with p_2 (with probability at 0.89), its moderate relation to p_3 determines that the whole description probably requests p_3. According to the one-to-one correspondence in previous studies [16], the entire description would be unrelated to p_3. Thus, predicting the multiple correspondent permissions of one description sentence is helpful for accurate description-permission fidelity measurement.

C. NLP PRELIMINARIES

1) WORD EMBEDDING

Word embeddings are a set of language modeling techniques that represent words into a continuous vector space with much lower dimension. Word2vec is a typical word embedding proposed by Mikolov *et al.* [20] and has been widely adopted in the field of NLP [21]. The vector representations obtained by Word2Vec are learned by two-layer neural networks to reconstruct linguistic contexts of words. The distance (*e.g.*, cosine distance) between two word vectors in the low-dimension space indicates the semantic similarity of these words. GloVe [22] utilizes global word-word co-occurrence counts to learn the word vector representations. Mikolov *et al.* [23] further proposed crawl to improve the quality of word vectors using a combination of known tricks, such as position-dependent weighting and phrase representations.

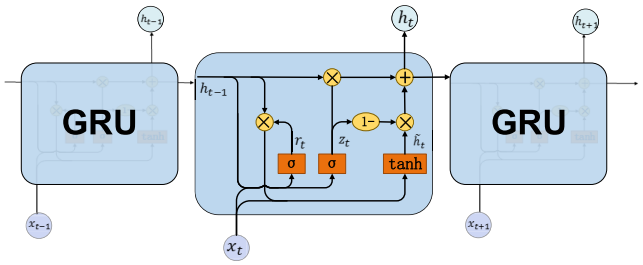

FIGURE 3. Illustration of a simple RNN model with gated recurrent units (GRUs).

2) RECURRENT NEURAL NETWORK (RNN)

Recurrent neural networks (RNN) [24], where the internal states can better express the contextual information by processing sequences of inputs, is commonly used in the NLP field [25]–[27]. As shown in Fig. 3, overall, it consists of a hidden state h, receiving the current input x_t and activated by the state of the previous time stamp. At each time stamp, the recurrent hidden state h_t of RNN is updated by

$$h_t = \phi(h_{t-1}, x_t)), \tag{1}$$

where ϕ is a non-linear activation function, typically set as a complex recurrent unit, *e.g.*, Long short-term memory (LSTM) [24] and Gated recurrent unit (GRU) [26]. LSTM and GRU are two popular extensions of RNN, and we choose GRU in this paper due to its simpler structure and good performance in many NLP tasks [28], [29].

GRU is one recurrent unit on sequence modeling using a gating mechanism to learn long-term dependencies [26]. As shown in Fig. 3, for the t-th time stamp, the two gates (*i.e.*, the reset gate r_t and the update gate z_t) in GRU are computed by

$$r_t = \sigma(\mathbf{W}_r \cdot [h_{t-1}, x_t]), \tag{2}$$
$$z_t = \sigma(\mathbf{W}_z \cdot [h_{t-1}, x_t]), \tag{3}$$

where σ is the logistic sigmoid function, and $\mathbf{W}_r$ and $\mathbf{W}_z$ are weight matrices. The reset gate determines whether the candidate hidden state $\tilde{h}_t$ combines the new input x_t with the previous state h_{t-1}:

$$\tilde{h}_t = \tanh(\mathbf{W} \cdot [r_t h_{t-1}, x_t]). \tag{4}$$

Finally, the update gate controls how much the hidden state h_t updates based on candidate state $\tilde{h}_t$ and previous state h_{t-1}. The activation of the t-th state h_t is computed by

$$h_t = z_t h_{t-1} + (1 - z_t)\tilde{h}_t. \tag{5}$$

III. DESIGN OF AC-NET

In this section, we present AC-Net in details. Figure 4 gives an overview of our framework. There are two main phases of the AC-Net: model training of the deep neural network for classification and consistency assessment. In the training phase, we take in a set of app's description along with their each permission labels (whether it is indicated or not) as input. The descriptions are then subject to preprocessor and

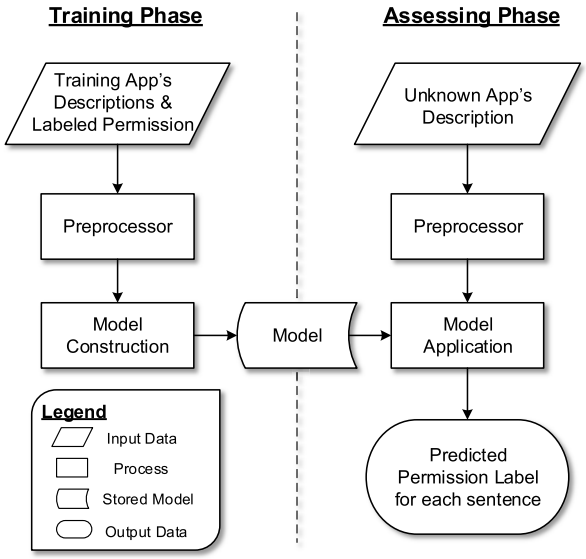

FIGURE 4. Framework overview of AC-Net.

model construction. We pre-process the descriptions at first in the preprocessor to format the sentences. The preprocessed sentences can be fed into the proposed neural network, named TextGRU, to train the classifier in the model construction step. In the assessing phase, an unknown app's description is adopted as input and again pre-processed as in the training phase. The model application process finally applies the learned model to classify every sentence and output the binary probability distributions for each declared permission. The output indicates the correlation between the app description and permissions of this unknown app. Detailed explanations are presented in the following.

A. PREPROCESSING

The preprocessor accepts the natural-language description of both unknown app and collected apps. Then, it converts the sentences into sequences of word indices consistently for further model training or assessment. In particular, the preprocessor performs the following tasks:

1) SENTENCES SPLIT

Accurate detection of sentence boundaries is critical for splitting app description into sentences. In simplistic English, characters such as '.', '!', '?' mark the end of a sentence, and others like '⋆', '•' or '*' may list bullet points are also considered as sentence separators. In particular, we use regular expressions to filter decimals, ellipsis, shorthand notations, URLs, and E-mail addresses at first, since these characters have other legal usages except for sentences separation. After sentences splitting, we'll get a set of sentences $\mathbf{S} = \{s_0, s_1, ..., s_n\}$, including N items.

2) STOP WORDS REMOVAL AND STEMMING

Some high-frequency words which appear to be of little value in analyzing permission usage are excluded from the vocabulary entirely. These words are called stop words. We remove a group of stop words, which contains 127 generic stop words in NLTK [30] and Top 50 words in Domain-specific stop words list created by Watanabe *et al.* [31]. After that, we apply to the stemming, *i.e.*, convert English words into its purest form, to improve the performance of text classification tasks. Here we use Porter Stemmer tools in NLTK to employ stemming.

3) WORD VECTORS INITIALIZATION

We initialize word vectors with our pre-trained word embeddings trained on 69941 Android app descriptions collected from Google Play. The vectors have a dimensionality of 100 and were produced by Word2Vec model which is a shallow, two-layer neural network. Compared to the somewhat popular embeddings such as GloVe [22] (400 thousand word vectors trained on Wikipedia) and crawl [23] (2 million word vectors trained on Common Crawl), ours fully retains domain-specific characteristics of statements. We compute an index mapping words to known embeddings as a dictionary. Then create another word index mapping words to integers for all input text. We truncate the sequences to a maximum length of L words. So far, we format our sentences to a tensor $\mathbf{T}^{N\times L}$ that can be fed into a neural network. The individual items in it are integers of word index.

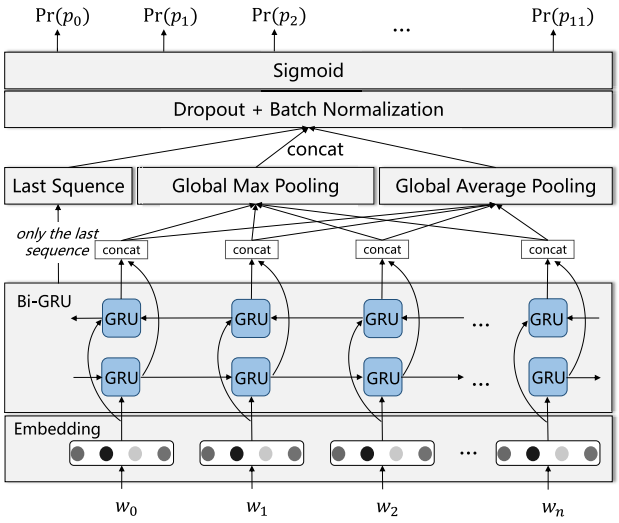

FIGURE 5. Model architecture of TextGRU.

B. MODEL CONSTRUCTION

We propose a deep neural network model for text classification called TextGRU. The model, shown in Figure 5, is a variant of the RNN architecture with the GRU. The input of the network is one of the sentences s_i, which is a sequence of word indices $[w_0, w_1, ..., w_n]$. The output of the network contains the binary probability distributions for each permission. We use $Pr(p_k)$ to denote the probability whether the sentence suggests permission p_k. We divide the model into four layers and give details for each layer in implementation.

1) EMBEDDING LAYER

In this layer, we leverage our word index and the dictionary mentioned above to compute embedding weights. The Embedding Layer maps the integer inputs to the initial embedding weights which can be fine-tuned during training to improve the performance. At this point, the output of this layer is a tensor $\mathbf{T}^{N\times L\times D}$, where D denotes the dimensionality of weights and is the same as the dimension of the pre-trained word embeddings.

2) BI-GRU LAYER

In this layer, we utilize Bidirectional GRU (Bi-GRU) to capture the feature of future as well as past context. Bi-GRU extends the unidirectional GRU networks by introducing the layer in which the recurrent process is in the opposite direction. We denote the current hidden state outputs of the forward pass and backward pass as $\overrightarrow{h}_t$ and $\overleftarrow{h}_t$ separately. Then the output is concatenated by

$$h_t = \overrightarrow{h}_t \oplus \overleftarrow{h}_t. \tag{6}$$

3) CONCAT LAYER

In Concat Layer, we process the output of Bi-GRU Layer in three different ways and merge them along the concatenation axis. As illustrated in Figure 5, the Last Sequence part only accepts the GRU hidden state output at the last time step. At the same time, the Global Max Pooling part and the Global Average Pooling part can both access the hidden state output for each input time step, and then make global max pooling operation and global average pooling operation for temporal data respectively. After that, we merge them into a concatenated vector $\mathbf{h}_c$:

$$\mathbf{h}_c = \mathbf{h}_L \oplus \mathbf{h}_{GMP} \oplus \mathbf{h}_{GAP}. \tag{7}$$

4) OUTPUT LAYER

After applying dropout at the rate of 0.3 and normalizing the activations of the previous layer at each batch for preventing overfitting, we use the sigmoid function as the activation in a fully connected layer to limit each item of final output $P_r(p)$ into [0, 1], since $P_r(p_k)$ is considered as the probability that the input sentence indicates the permission p_k.

$$P_r(p_k) = \sigma(\mathbf{W}|_{rate}\hat{\mathbf{h}_c}), \tag{8}$$

where $\mathbf{W}|_{rate}$ randomly set input units to 0 by the probability of *rate* and $\hat{\mathbf{h}_c}$ is normalized data.

5) MODEL TRAINING

We consider the decision of each permission as a binary classification instead of macro multi-class classification. To learn the parameters of the network, we should calculate the binary cross-entropy [32] of predictions and true labels for each class and minimize the sum. The definition of loss function as follows:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{C}[y_j^i \ln \hat{y}_j^i + (1-y_j^i)\ln(1-\hat{y}_j^i)], \tag{9}$$

where $\hat{y}_j^i$ is prediction probability that the i-th sentence indicates the j-th permission and y_j^i is the corresponding ground-truth label. N and C denote the number of training samples and class number severally.

C. MODEL APPLICATION: CONSISTENCY ASSESSMENT

The assessment process is based on the trained model. When a new app comes, after preprocessing its app description and performing text classification on each sentence, the model outputs the binary probability distributions of each declared permission for each sentence. In addition to the sentence level, we consider the maximum values of each permission as consistency degree for the entire app. Rather than a simple "yes" or "no" answer, we pay more attention to the degree of consistency between app description and declared permissions, whether the sentence level or the app level.

IV. EMPIRICAL STUDY DESIGN

In this section, we illustrate the overall experimental design before presenting the detailed results. Specifically, we first show the research questions we aim to answer, then the experimental datasets, and finally the evaluation metrics we used to assess the results.

RQ1: Can AC-Net accurately predict app permissions based on descriptions? We compare AC-Net with baseline methods to answer this question.

RQ2: Can AC-Net outperform commonly-used classification models such as SVM?

RQ3: What is the impact of word embeddings on the model performance? Here, we compare the word embeddings pre-trained on our corpus, and the existing popular embeddings.

RQ4: What is the performance of AC-Net when evaluated using WHYPER's dataset [16]? We also include one available dataset which only considers three permissions, besides validating on the dataset built by us and involving more app permissions.

A. DATASET

We collected 69,713 Android apps from Google Play in December 2015, which is a snapshot of popular apps declaring at least one permission. Following previous research [14], [16], [31], we selected partial app permissions for study. Table 2 illustrates the 16 permissions involved in our work, which can be classified into the official three protection levels declared by Google: dangerous permissions, normal permissions, and signature permissions [33].

Dangerous Permissions cover the areas where apps could read or potentially write user's private data or resources. The 13 of 24 permissions which are the top risks with the highest user concerns [34] are included. We consider the WRITE_SETTINGS permission, one of the two ***Signature Permissions*** and also related to user-sensitive features. It allows an app to read or write system settings and even modify the permission status. Although ***Normal Permissions*** poses little risk to privacy information, they may

TABLE 2. App permissions included in our dataset.

Protection Levels	Permission Groups	Permission
DANGEROUS	STORAGE	WRITE_EXTERNAL_STORAGE
	CONTACTS	GET_ACCOUNTS
		READ_CONTACTS
		WRITE_CONTACTS
	LOCATION	ACCESS_FINE_LOCATION
		ACCESS_COARSE_LOCATION
	CAMERA	CAMERA
	MICROPHONE	RECORD_AUDIO
	SMS	READ_SMS
		SEND_SMS
	CALL_LOG	READ_CALL_LOGS
	PHONE	CALL_PHONE
	CALENDAR	READ_CALENDAR
SIGNATURE	SETTINGS[1]	WRITE_SETTINGS
NORMAL	TASKS[2]	GET_TASKS
		KILL_BACKGROUND_PROCESS

[1,2] Permission Groups are not required for these two levels of protections. Here we use the objects accessed by these permissions to represent the group, just for the convenience of the latter representation.

not comply with user's preferences as users cannot revoke these permissions after installation. We include two of these normal permissions that were also discussed in previous studies [14], [31]. Following these studies, we hypothesize that the access of these permissions should be well reflected in apps' descriptions.[4] Moreover, we classify the 16 permissions into 11 permission groups according to Google's official grouping strategy for dangerous permissions [33] and each permission's influence on users' phones for other permission levels, as shown in Table 2. In our work, we assess the consistency between permissions and descriptions based on the 11 permission groups.

In AC-Net, we pre-train the input word embeddings using app descriptions of all the collected apps. Overall, the descriptions contain 783,199 sentences and 39,800 unique words. We experimentally set the embedding dimension as 100. As manually labeling app permissions related to all description sentences is rather labor-intensive, we decided to choose a subset of these apps as our experimental dataset for model learning. We follow two principles below during app selection:

1) Popularity - to ensure that the app descriptions are credible. Also, the safety of these apps is crucial for users because privacy leakage in them can influence more users. Here, we removed the apps with user rating counts fewer than the average (*i.e.*, 24,715) in our collection.
2) Rating and category - to involve apps with both positive feedback and negative feedback, and mitigate the bias caused by only one category. To do so, we picked 100 top-rated apps and 100 most poorly-rated apps for each permission. The selected apps cover 40 categories, with the quantity distribution shown in Table 3.

[4] Although some apps may have sketchy or even incredible descriptions, detecting misleading descriptions is out of the scope of our work. We aim at accurately predicting app permissions based on descriptions only without combining other app repositories such as code and API documentation, which is more technically changeable.

TABLE 3. Categories of selected apps.

Category	#*Apps*	Category	#*Apps*
Action	53	Music	6
Adventure	14	Music & Audio	54
Arcade	53	News & Magazines	14
Board	2	Personalization	69
Books & Reference	35	Photography	49
Business	13	Productivity	103
Card	7	Puzzle	25
Casino	8	Racing	23
Casual	61	Role Playing	20
Comics	2	Shopping	27
Communication	111	Simulation	20
Education	16	Social	65
Educational	2	Sports	39
Entertainment	76	Strategy	20
Finance	22	Tools	197
Health & Fitness	35	Transportation	17
Libraries & Demo	3	Travel & Local	45
Lifestyle	43	Triva	1
Media & Video	46	Weather	10
Medical	3	Word	5

#*Apps*: Number of apps that belongs the category

Different from previous research [16], [31], which chooses apps on each permission independently and only considers one permission per app, our selected apps are intersected by different permissions. Thus, after filtering non-English text, it resulted in a subset containing total 1,415 unique apps with 24,726 sentences as shown in Table 5.

B. EVALUATION SETUP

In this part, we introduce the experimental dataset and the evaluation metrics.

1) EXPERIMENTAL DATASET

Although there exist public datasets containing the correspondence between description sentences and app permissions, they all involve few permissions. So we determined to manually build the experimental dataset, in which we focus on the 11 permission groups depicted in Table 2 and label the permissions reflected by each description sentence. We invited 14 participants for the labeling task. They are all computer science students who are familiar with app usage and development, including app permission invoking. Their average development experience is 1.7 years, with the maximum at about three years and the minimum at one year. We ensured that each sentence was assigned at least two students for the permission labeling, and more students were included when the agreement could not be reached. In our work, one sentence needs at most three persons to label, and the sentences labeled with permissions are called ***permission sentences***. The whole process lasted 5 to 10 days, and the average participation time of the students was 6 hours.

We note that some sentences may suggest more than one permissions. For example, an app's description says "*You can either block any caller or SMS from your contacts list, call logs, and messages logs, or add unwanted number manually*". This sentence implies the usage of the

TABLE 4. Example of annotation format.

app_id	sentence_id	sentence	Permission Groups						
			STORAGE	CONTACTS	LOCATION	CAMERA	MICROPHONE	...	TASKS
1025	10	Movable manual record audio ...	0	0	0	0	1	...	0
1025	11	Support two folders to keep the ...	0	0	0	0	1	...	0
1025	12	Advance search with many option ...	0	0	0	0	0	...	0
1025	13	Automatic filter recording based ...	0	1	0	0	1	...	0

TABLE 5. Statistics of manual checking.

Permission Groups	$\#Apps$	$\#S$	$\#S_p$
STORAGE	1304	23101	1338
CONTACTS	951	17353	937
LOCATION	732	12887	724
CAMERA	406	7372	522
MICROPHONE	350	6371	319
SMS	337	6484	524
CALL_LOG	282	5457	323
PHONE	280	5445	199
CALENDAR	197	3637	289
SETTINGS	369	7016	560
TASKS	538	10203	344
TOTAL	1415	24726	4984

$\#Apps$: The number of apps that request the permission
$\#S$: The amount of sentences in the app descriptions
$\#S_p$: The number of sentences manually checked as ***permission sentence***.

permission READ_CONTACT (''*from your contacts list*''), READ_CALL_LOG (''*from call logs*''), CALL_PHONE (''*block call*'') and SMS permission group (''*block SMS*'' and ''*from messages logs*''), therefore, we annotate it as a ***permission sentence*** related to each of these permissions. We finally obtain the binary labels of the 11 permission groups for each sentence. We list some examples in Table 4 to show the annotation format. For each permission, ***permission sentences*** are labeled as ''1'', and the remaining ones are labeled as ''0''. Table 5 reports the statistics of the experimental dataset, including the number of apps, the total number of sentences, and the number of permission sentences for each permission group. As can be seen, 20.2% of descriptions sentences reflect the usage of the studied permissions.

For training our AC-Net model and evaluate its performance, we conduct comprehensive experiments by employing the repeated holdout cross-validation approach. First, we randomly select 200 app descriptions with about 3600 sentences of the pre-selected subset as the test set, and the remaining part is regarded as the train set. Then, the test set is used for the evaluation of our model as well as other comparing methods, whereas the train set is split into training and validation sets with the partition of 4:1 once again for better model training. The random sampling process is repeated 10 times. Each time, all of the comparison experiments are evaluated on the same randomly sampled test set. The final results are the average values of them.

2) EVALUATION METRICS

Determining whether a sentence of app description is a ***permission sentence***, is a classification problem. As shown in Table 5, the ***permission sentences*** represent a small part of overall sentences for every permission group. When dealing with the class imbalances, accuracy is an unsuitable metric to use. Usually, there are two candidates as metrics: The area under the receiver operating characteristic curve (ROC-AUC) and precision-recall curve (PR-AUC).

The ROC-AUC value is equivalent to the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example [35]. The higher the AUC value, the better the classification effect. The metrics ROC-AUC can be calculated as follows:

$$ROC - AUC = \frac{\sum_{i \in positive\_class} rank_i - \frac{N_p \times (N_p+1)}{2}}{N_p \times N_n}, \quad (10)$$

where N_p and N_n denote the number of positive and negative samples, and $rank_i$ is the ranking of the i-th positive sample.

The precision-recall curve shows the trade-off between precision and recall for the different threshold [36]. PR-AUC is just the area under the PR curve. It can be approximated by the integral with:

$$PR - AUC = \sum_{i=1}^{N} (R_n - R_{n-1}) P_n, \quad (11)$$

where P_n and R_n are the precision and recall at the nth threshold, and N is the number of samples.

V. EXPERIMENTAL RESULT

A. RESULTS OF RQ1: PERFORMANCE OF AC-NET

To demonstrate the efficiency of our model, AC-Net, we compare it with four relevant approaches: Keyword-based, WHYPER [16], AutoCog [14] and ACODE [31].

1) By the Keyword-based method, we consider the sentence which contains at least one keyword as the ***permission sentence***. Column ''Key-Based'' in Table 7 shows the keywords that we obtained from permission-related APIs, five words for each permission.

2) WHYPER [16] is a semantic-based approach extracting semantic graph from Android API document to correlate app description and permission semantics. They only considered three permissions. To extend the other permissions, we manually extract the semantic pattern set from Android API document like them.

3) AutoCog [14] automatically discovers a set of semantic patterns for each permission from numerous permission-related app descriptions based on the frequency. We apply it on our test set by reusing the chosen patterns and integrating 11 permissions into 8 permission groups.

4) ACODE [31] is a variant of the keyword-based method. They used a two-stage filter to distinguish relevant and

TABLE 6. Evaluation results of different methods.

Permission Group	ROC-AUC					PR-AUC				
	AC-Net	Key-Based	WHYPER	AutoCog	ACODE	AC-Net	Key-Based	WHYPER	AutoCog	ACODE
STORAGE	**0.94276**	0.66981	0.55930	0.63028	-	**0.65568**	0.40069	0.32521	0.34742	-
CONTACTS	**0.97197**	0.72572	0.60710	0.68446	0.76576	**0.74657**	0.41641	0.28047	0.41521	0.43078
LOCATION	**0.98376**	0.88762	0.60198	0.81737	0.88072	**0.77496**	0.68715	0.45168	0.51815	0.69954
CAMERA	**0.98249**	0.85640	0.57520	0.84025	0.78580	**0.75821**	0.52267	0.43630	0.49002	0.46576
MICROPHONE	**0.96293**	0.80436	0.59735	0.79669	0.79002	0.49665	0.44078	0.24771	0.40081	**0.51334**
SMS	**0.98991**	0.91227	0.65296	0.78944	0.82176	**0.83454**	0.61593	0.36846	0.40644	0.58006
CALL_LOG	**0.99332**	0.63199	0.53645	-	-	**0.70502**	0.20625	0.41683	-	-
PHONE	**0.99140**	0.93148	0.74290	-	-	**0.62455**	0.55226	0.37226	-	-
CALENDAR	**0.99469**	0.98121	0.65954	0.89737	0.95011	**0.84444**	0.71125	0.28444	0.50611	0.71236
SETTINGS	**0.95104**	0.55398	0.58193	0.65403	0.57119	**0.43251**	0.15872	0.21638	0.31488	0.18696
TASKS	**0.94972**	0.64903	0.53768	-	0.66514	**0.48659**	0.27280	0.17903	-	0.24657
AVERAGE	**0.97400**	0.78217	0.60476	0.76374	0.77881	**0.66907**	0.45317	0.32534	0.42488	0.47942

TABLE 7. Two kinds of keywords for permission groups.

Permission Groups	Key-Based	ACODE
STORAGE	storage, save, download sdcard, delete	-
CONTACTS	contact, number, address account, sync	sms, call, contact grab, google, youtube
LOCATION	location, gps, map position, distance	gps, location, map
CAMERA	camera, photo, scan picture, video	camera, scan, photo
MICROPHONE	record, audio, voice capture, microphone	recording, voice, record
SMS	sms, text, message send, receive	sms, message, sent incoming
CALL_LOG	call log, number, phone digit, history	-
PHONE	call, number, dial block, phone	-
CALENDAR	calendar, event, agenda date, attendee	calendar, reminder meeting
SETTINGS	setting, backgroud, ring wallpaper, ringtone	alarm, ring, bluetooth
TASKS	running, task, process kill, activity	lock, security, task kill, manager

non-relevant document for relevance weight calculation and extract the keywords that have the largest relevance weights. Their public results of Top 3 keywords are directly utilized, shown in the column "ACODE" of Table 7.

The results are provided in Table 6. It can be observed that neither of the keyword-based method and ACODE is comparable with AC-Net under the ROC-AUC measure because of their inherent limitations such as words confusing and the lack of semantic reasoning. For the PR-AUC values, AC-Net surpasses the other approaches on most of the permissions, except for permission group MICROPHONE evaluated by ACOED. Further, it can be seen that keyword-based method and ACODE get higher values of ROC-AUC and PR-AUC on the permission groups of LOCATION and CALENDAR than performance on other permissions. It can be explained that these three permission groups have fewer but better keywords to identify the ***permission sentence***.

Clearly, WHYPER and AutoCog could not compete with our approach under both ROC-AUC and PR-AUC measures. We have an assumption that the semantic graph generated from the API documents cannot cover the complete semantic patterns. Moreover, being confined to the fixed semantic patterns, WHYPER is likely applicable to standard

TABLE 8. T-test performance on ROC-AUC.

Permission Groups	AC-Net vs.			
	Key-Based	WHYPER	AutoCog	ACODE
STORAGE	<0.0001	<0.0001	<0.0001	-
CONTACTS	<0.0001	<0.0001	<0.0001	<0.0001
LOCATION	<0.0001	<0.0001	<0.0001	<0.0001
CAMERA	<0.0001	<0.0001	<0.0001	<0.0001
MICROPHONE	<0.0001	<0.0001	<0.0001	<0.0001
SMS	0.0003	<0.0001	<0.0001	<0.0001
CALL_LOG	<0.0001	<0.0001	-	-
PHONE	0.0080	<0.0001	-	-
CALENDAR	**0.2087**	<0.0001	**0.0012**	**0.0327**
SETTINGS	<0.0001	<0.0001	<0.0001	<0.0001
TASKS	<0.0001	<0.0001	<0.0001	<0.0001

description sentences, whereas this type of text occupies a minor proportion in reality. The failure of AutoCog can be in large measure attributed to unsupervised learning which has the drawback of choosing relationships that may not appropriate, leading to many false positives.

To confirm whether the predictions of our AC-Net are significantly better than those provided by another method, we test the statistical significance of the ROC-AUC with Wilcoxon Signed Rank Test [37], [38]. In particular, we test the following Null Hypothesis: "The ROC-AUC provided by the method M_i is significantly less than that of the method M_j.", and set the confidence limit, α, at 0.05. As shown in Table 8, the improvement of our AC-Net over the four benchmarks is significant ($P < 0.001$) in 36 out of 39 cases, which further prove the effectiveness of our AC-Net.

B. RESULTS OF RQ2: THE IMPACT OF DIFFERENT LEARNING MODELS

We also examine the impact of different learning models on prediction effectiveness. We have tried several popular learning models, including TextCNN [39], NBSVM [40], and LR [41], by using their basic implementations. The evaluation results in Table 9 show that all the learning models lead to overall good ROC-AUC and PR-AUC values. In particular, NBSVM is an approach where an SVM is built over Naive Bayes log-count ratios as feature vectors. LR and NBSVM, as strong baselines, are commonly used in text classification by using the representation of the bag of words and a matrix of TF-IDF features. These traditional machine

TABLE 9. Evaluation results of different learning models.

Permission Group	ROC-AUC				PR-AUC			
	TextGRU	TextCNN	NB-SVM	LR	TextGRU	TextCNN	NB-SVM	LR
STORAGE	**0.94276**	0.91781	0.88056	0.86374	**0.65568**	0.58063	0.50954	0.50273
CONTACTS	**0.97197**	0.95570	0.92806	0.92068	**0.74657**	0.64740	0.54706	0.53989
LOCATION	**0.98376**	0.95296	0.91293	0.89233	**0.77496**	0.70215	0.56543	0.51181
CAMERA	**0.98249**	0.95351	0.93617	0.92261	**0.75821**	0.73570	0.64767	0.57175
MICROPHONE	**0.96293**	0.88481	0.88771	0.90359	**0.49665**	0.37146	0.23964	0.26462
SMS	**0.98991**	0.98765	0.98956	0.98916	**0.83454**	0.82384	0.79468	0.76365
CALL_LOG	**0.99332**	0.98317	0.98433	0.97528	**0.70502**	0.57749	0.62930	0.34822
PHONE	**0.99140**	0.98660	0.96408	0.95224	**0.62455**	0.60476	0.52691	0.47819
CALENDAR	**0.99469**	0.99110	0.99319	0.99114	**0.84444**	0.75045	0.70932	0.63841
SETTINGS	**0.95104**	0.88005	0.89285	0.88221	**0.43251**	0.40897	0.36177	0.26689
TASKS	**0.94972**	0.94007	0.92896	0.91224	**0.48659**	0.44741	0.30115	0.31947
AVERAGE	**0.97400**	0.94849	0.93622	0.92775	**0.66907**	0.60457	0.53023	0.47324

TABLE 10. Impact of word embeddings (ROC-AUC).

Permission Groups	AC-Net	crawl	GloVe	All-1
STORAGE	**0.94276**	0.93194	0.91867	0.89550
CONTACTS	**0.97197**	0.96299	0.94065	0.93319
LOCATION	**0.98376**	0.96583	0.91029	0.94124
CAMERA	**0.98249**	0.97884	0.97335	0.95647
MICROPHONE	**0.96293**	0.93436	0.85689	0.86772
SMS	**0.98991**	0.98785	0.98141	0.98160
CALL_LOG	0.99332	0.99372	**0.99433**	0.98929
PHONE	**0.99140**	0.98977	0.98683	0.97833
CALENDAR	**0.99469**	0.99376	0.99277	0.98416
SETTINGS	**0.95104**	0.90612	0.89651	0.89445
TASKS	**0.94972**	0.93832	0.93770	0.89840
AVERAGE	**0.97400**	0.96214	0.94449	0.93821

learning models work slightly worse in our particular settings. For deep learning models, TextGRU outperforms TextCNN, since TextGRU makes use of sequential information with the output being depended on the previous computations, which captures information about what has been calculated so far. Besides, TextGRU requires fewer parameters catering to less training data to generalize.

C. RESULTS OF RQ3: THE IMPACT OF WORD EMBEDDINGS

We process the word embeddings in four different ways including using the pre-trained word embeddings of our corpus, GloVe [22], crawl [23], and all one's initialization (ALL-1) that generates tensors initialized to 1 and updates weight during training.

Table 10 shows the result under the ROC-AUC measure. Accordingly, it could be found that the AC-Net with our pre-trained word embeddings performs well on most of the permissions with high ROC-AUC values. Instead of training weight only by input sentences in Embedding layer, using pre-trained word embeddings can capture richer external semantic information. What's more, our domain-specific word embeddings fully retain the rich semantics from app descriptions rather than external corpus.

D. RESULTS OF RQ4: CROSS-DATASET EVALUATION

We validate the robustness of AC-Net model by using the public external labeled dataset [16]. This dataset consists of a set of annotations for ***permission sentences*** as well as the outputs of WHYPER on three permissions. Thus, we can directly compare the performance of the two frameworks. We utilize these sentences as the input and their labels as the validation set to evaluate the performance of our model. The metric is ROC-AUC as before.

TABLE 11. Robustness on external labeled dataset (ROC-AUC).

Permission	AC-Net	Key-Based	WHYPER
READ_CALENDAR	**0.96164**	0.85298	0.91628
RECORD_AUDIO	**0.95414**	0.79843	0.88896
READ_CONTACT	**0.91519**	0.87374	0.89269

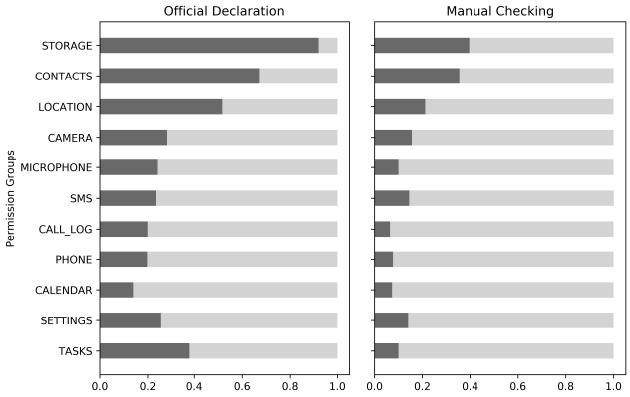

FIGURE 6. Proportions of apps that declare a permission and ones actually refer to the permission after manual checking.

It is evident that AC-Net outperforms WHYPER on the external data under the ROC-AUC measure shown in Table 11. Our model has a better effect and robustness, which can maintain relatively high performance when applying to different kinds of datasets.

VI. DISCUSSION

AC-Net model solves the problem of assessing the consistency from a new point of view aims at learning real correspondences between the semantic of descriptions and the permissions usage. We notice that most of the declared permissions are not much well described in their descriptions from the official store (*i.e.*, Google Play). Figure 6 illustrates the fractions of apps that relate a permission from the official declaration as well as the ones from manual checking.

Furthermore, for apps overall consistency, human readers could only find 14% of apps whose official descriptions reveal all that declared permissions. Thus, it is inappropriate to use the corresponding relationship between app description and declared permissions directly for pattern extraction as AutoCog does, because some common patterns would be wrongly selected under frequency-based measurement. In this paper, we use the manually checked dataset as ground truth to train the classification model, thereby ensuring the correctness of the consistency.

A. THREATS TO VALIDITY

1) INTERNAL VALIDITY

With regards to the internal validity, the informativeness of descriptions can impact the permission assessment. To reduce the threat, we choose the comprehensible permissions that developers should provide reasons in the description following the latest writing tips [42], [43] and manually annotate the sentence in our experiment.

TABLE 12. Efficiency comparison of different methods.

Time (s)	AC-Net	Key-based	WHYPER	AutoCog	ACODE
Training	45.8	0	0	-	0
Predicting	0.001	$2.6 \times e^{-5}$	1.59	1.06	$1.0 \times e^{-5}$

Then, as shown in previous studies [44], [45], deep neural models can consume much processing time practically in spite of the good performance. Among the comparison methods in answering *RQ1*, we compare their efficiency in training and predicting on the same machine with Intel(R) CoreTM i7-5820k CPU (3.3GHz, 12 cores) and 128GB RAM. As shown in Table 12, AC-Net consumes relatively less time (0.001s) for predicting the possible permissions of one sentence. The training process can be implemented offline, and will not influence the online permission prediction in practice.

Finally, there are some limitations in applying our model practically. On one hand, it may raise false alerts for non-related or shallow app descriptions which cannot detail all behaviors [12]. Although these apps declaring with redundant permissions are not always designed for malicious purposes, the alert can remind developers to modify and improve app descriptions. On the other hand, malicious apps with feigned descriptions can escape discovery based on our model. Predicting the permissions usage based only on the description does have this drawback. As discussed in previous work, however, even ordinary users are more likely to find obviously abnormal function publicity in natural language descriptions [14]. In addition, This work is only a small part of the field and should be complemented by other detecting methods such as code analysis to jointly promote the sound development of mobile apps.

2) EXTERNAL VALIDITY

First, the 69,713 subject apps in this paper represent a small proportion of apps on the whole app markets. To mitigate this threat, we select popular apps from different categories and declaring different permissions to guarantee the representativeness.

Second, the description-to-permission ground truth was established by manual inspection. To alleviate the threat of human assessment, we ensure that each sentence is individually checked and reached a consensus by at least two participants.

VII. RELATED WORK

Privacy and security of mobile app have always been a research hotspot. Sajjad *et al.* [46] summarized the previous works on adaptive security and privacy for mobile computing and highlighted the future challenges of research including self-protection for mobile devices, user-driven privacy decisions, and context-aware security. Our work assists users to know more about app privacy. Specifically, we predict the declared permissions of an app to help users make privacy-related decisions. Different from the iOS platform where Apple reviews every app for the privacy data access, Android transfers the responsibility to the end users for permission checking [47]. Many researchers have carried out dynamic analysis of suspicious permissions and run-time behaviors on the Android platform [48], [49]. In the static analysis, text-based features, such as app descriptions [14], [16], [31], privacy policy [11], [12], API documents [50], app reviews [51], and source code as text [52], from app markets, are commonly-adopted for predicting the actual app permission usage [53], [54]. In the following, we first illustrate the previous work on text analysis for mobile security and then generally review typical models for text classification.

A. CONSISTENCY MEASUREMENT BETWEEN APP DESCRIPTIONS AND PERMISSIONS

In this paper, we focus on studying the relevance between app permissions and descriptions of mobile apps, which has already been explored by a few previous studies. Most of them are based on either keyword retrieval or semantic analysis approach.

For bridging the semantic gap between app descriptions and apps' real functionalities, Pandita *et al.* [16] proposed a semantics-based framework, namely WHYPER. They leveraged the first-order-logic (FOL) representation of a sentence in the description and semantic graphs of Android permissions to identify whether a permission is necessary for an app. But WHYPER is limited by its based semantic information which is inferred by extracting natural language keywords in API documents. AutoCog proposed by Qu *et al.* [14] automatically discovers a set of textual patterns correlated with permissions from the app descriptions as well as flag apps that have discrepancies according to these patterns "black or white". Nevertheless, since the textual patterns are selected based on their frequencies in the whole description collection without supervision, AutoCog may raise false alerts when the common patterns for a sensitive permission

do not appear in the descriptions of the apps that request the permission, or the patterns are wrongly selected.

Following WHYPER, Watanabe *et al.* [31] proposed ACODE (Analyzing COde and DEscription), which combines static code analysis and text analysis. ACODE leverages keyword-based techniques to determine whether an app description relates to privacy-sensitive resources, and can be inevitably influenced by polyseme and lack of semantic reasoning. Gorla *et al.* [13] also employed both app descriptions and code information, but their proposed CHABADA approach is more focused on analyzing the abnormal usage of sensitive APIs by comparing apps with the others in the same cluster identified by description topics. To prohibit permission abuse, Wang and Chen [55] proposed an Android semantic permission generator called ASPG to get the minimum set of permissions an app needs according to the app descriptions. Similarly, Taylor and Martinovic [54] implemented a framework called SecuRank to identify and suggest a functionally-similar alternative that used fewer permissions. App descriptions are assisted in generating popular search queries by extracting functional keywords. AUTOREB [51] and PACS [56] focus on assessing the review-to-behavior fidelity of mobile apps. Both of them explore the user review information to predict the risky behaviors, which requires review collection firstly rather than giving suggestions at the beginning of app release and also needs manual intervention to achieve better performance. Yu *et al.* [12] developed a system named TAPVerifier by analyzing four kinds of software artifacts, including privacy policy, bytecode, description, and permission, to conduct cross-verification among them automatically.

Whyper [16] and AutoCog [14] are most close to our work. Different from them, we focus on assessing the permissions of finer-granularity descriptions, *i.e.*, the multiple correspondences between description sentences and permissions. In addition, our work is the first attempt to extend deep learning techniques to predict app permissions based on description sentences. The experimental results further confirm the effectiveness of our work.

B. CLASSIFICATION METHODS IN NLP

Text classification is one of the typical problems in NLP. The traditional classification methods divide the task into two types, *i.e.*, feature engineering and model training. For feature engineering-based studies, most of them use Bag Of Words [57] or Vector Space Model [58] for text representation. For them, the classifiers are typically chosen as Naive Bayes (NB) [59], Logistic regression (LR) [41], [60], [61], or Support Vector Machine (SVM) [62], [63] and its variant Naive Bayes-Support Vector Machine (NBSVM) [40]. To alleviate the manual efforts for feature selection in feature engineering-based work, several kinds of deep neural models for automatic feature extraction have been proposed, such as fast text classifier (fastText) [64], convolutional neural network (TextCNN) [39], and recurrent neural network (TextRNN) [65]. They take word embeddings [20], [66] in the text sequence as the input. The output of the deep model is the probability distribution over category labels.

VIII. CONCLUSION AND FUTURE WORK

To bridge the semantic gap and promote mobile app development, we propose a framework AC-Net that combines NLP and deep learning to assess the consistency between mobile app descriptions and permissions. Instead of outputting a simple answer, we focus on the degree of consistency. We evaluate our AC-Net on real-world app descriptions that involve 16 typical permissions. Experimental results show that AC-Net achieves better performance with the average ROC-AUC at 97.4% and the average PR-AUC at 66.9%. In the future work, we will focus on the following topics:

More permissions. In this paper, our study only involves 16 common permissions, but AC-Net can be easily extended over more permissions. Our proposed model can be adaptively trained on descriptions with other permission labels and infer the permissions of newly-arrival descriptions.

Practical implementation. We will encapsulate our well-trained classification model and design an end-to-end tool. Given an app description, the tool can visualize corresponding permissions of each sentence and also global consistency with the app's declared permissions.

Semi-supervised learning. Although AC-Net can achieve considerable results, it still requires a large number of labeled descriptions. Since semi-supervised learning methods generally need relatively fewer training data, we will build an improved classification model.

REFERENCES

[1] P. Faruki *et al.*, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.

[2] M. Rosenberg, N. Confessore, and C. Cadwalladr. *How Trump Consultants Exploited the Facebook Data of Millions*. Accessed: Mar. 17, 2018. [Online]. Available: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html

[3] E. Graham-Harrison and C. Cadwalladr. *Revealed: 50 Million Facebook Profiles Harvested for Cambridge Analytica in Major Data Breach*. Accessed: Mar. 17, 2018. [Online]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

[4] 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.* New York, NY, USA: ACM, 2012, p. 3.

[5] P. Andriotis, M. A. Sasse, and G. Stringhini, "Permissions snapshots: Assessing users' adaptation to the Android runtime permission model," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2016, pp. 1–6.

[6] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft, *Exploring Decision Making With Android's Runtime Permission Dialogs Using in-Context Surveys*. Berkeley, CA, USA: USENIX Association, 2017.

[7] Y. Zhang *et al.*, "Vetting undesirable behaviors in Android apps with permission use analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, 2013, pp. 611–622.

[8] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.

[9] A. Sunyaev, T. Dehling, P. L. Taylor, and K. D. Mandl, "Availability and quality of mobile health app privacy policies," *J. Amer. Med. Inform. Assoc.*, vol. 22, no. e1, pp. e28–e33, 2015.

[10] R. Slavin *et al.*, "Toward a framework for detecting privacy policy violations in Android application code," in *Proc. 38th Int. Conf. Softw. Eng.* New York, NY, USA: ACM, 2016, pp. 25–36.

[11] L. Yu, T. Zhang, X. Luo, and L. Xue, "Autoppg: Towards automatic generation of privacy policy for Android applications," in *Proc. 5th Annu. ACM CCS Workshop Secur. Privacy Smartphones Mobile Devices*. New York, NY, USA: ACM, 2015, pp. 39–50.

[12] 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.

[13] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proc. 36th Int. Conf. Softw. Eng.* New York, NY, USA: ACM, 2014, pp. 1025–1035.

[14] 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.* New York, NY, USA: ACM, 2014, pp. 1354–1365.

[15] A. S. Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt, "Large-scale assessment of mobile notifications," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.* New York, NY, USA: ACM, 2014, pp. 3055–3064.

[16] 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. 1–17.

[17] T. Vidas, N. Christin, and L. Cranor, "Curbing Android permission creep," in *Proc. Web*, vol. 2, 2011, pp. 91–96.

[18] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of Android applications' permissions," in *Proc. IEEE 6th Int. Conf. Softw. Secur. Rel. Companion (SERE-C)*, Jun. 2012, pp. 45–46.

[19] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," in *Proc. USENIX Security Symp.*, 2015, pp. 499–514.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. (2013). "Efficient estimation of word representations in vector space." [Online]. Available: https://arxiv.org/abs/1301.3781

[21] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Aug. 2011.

[22] 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.

[23] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin. (2017). "Advances in pre-training distributed word representations." [Online]. Available: https://arxiv.org/abs/1712.09405

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 129–136.

[26] K. Cho *et al.* (2014). "Learning phrase representations using rnn encoder-decoder for statistical machine translation." [Online]. Available: https://arxiv.org/abs/1406.1078

[27] W. Yin, K. Kann, M. Yu, and H. Schütze. (2017). "Comparative study of CNN and RNN for natural language processing." [Online]. Available: https://arxiv.org/abs/1702.01923

[28] A. Kumar *et al.*, "Ask me anything: Dynamic memory networks for natural language processing," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1378–1387.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." [Online]. Available: https://arxiv.org/abs/1412.3555

[30] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *Proc. ACL Interact. Poster Demonstration Sessions*. Association for Computational Linguistics, 2004, p. 31.

[31] 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.

[32] D. M. Kline and V. L. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neural Comput. Appl.*, vol. 14, no. 4, pp. 310–318, Dec. 2005.

[33] *App Permission Levels Declared by Google*. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview#normal-dangerous

[34] A. P. Felt, S. Egelman, and D. Wagner, "I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns," in *Proc. 2nd ACM Workshop Secur. Privacy Smartphones Mobile Devices*. New York, NY, USA: ACM, 2012, pp. 33–44.

[35] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[36] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proc. 23rd Int. Conf. Mach. Learn.* New York, NY, USA: ACM, 2006, pp. 233–240.

[37] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[38] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. 1988.

[39] Y. Kim. (2014). "Convolutional neural networks for sentence classification." [Online]. Available: https://arxiv.org/abs/1408.5882

[40] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proc. 50th Annu. Meeting Assoc. Comput. Linguistics, Short Papers*, vol. 2. Association for Computational Linguistics, 2012, pp. 90–94.

[41] D. W. Hosmer, Jr., S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, vol. 398. Hoboken, NJ, USA: Wiley, 2013.

[42] K. Abrosimova. (2016). *Tips on How to Write App Descriptions That Sell Your Product*. [Online]. Available: https://yalantis.com/blog/how-write-app-description-that-markets-itself/

[43] M. A. Team. *5 Tips on Writing an App Description for Better Conversion Rates*. Accessed: Aug. 13, 2018. [Online]. Available: https://www.mobileaction.co/blog/app-analytics/5-tips-writing-an-app-description/

[44] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proc. 11th Joint Meeting Found. Softw. Eng.* New York, NY, USA: ACM, 2017, pp. 49–60.

[45] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Proc. 11th Joint Meeting Found. Softw. Eng.* New York, NY, USA: ACM, 2017, pp. 763–773.

[46] M. Sajjad, A. A. Abbasi, A. Malik, A. B. Altamimi, and I. M. Alseadoon, "Classification and mapping of adaptive security for mobile computing," *IEEE Trans. Emerg. Topics Comput.*, to be published.

[47] Y. Agarwal and M. Hall, "ProtectMyPrivacy: Detecting and mitigating privacy leaks on iOS devices using crowdsourcing," in *Proc. 11th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, Taipei, Taiwan, Jun. 2013, pp. 97–110. doi: 10.1145/2462456.2464460.

[48] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, p. 5, 2014. doi: 10.1145/2619091.

[49] V. Rastogi, Z. Qu, J. McClurg, Y. Cao, and Y. Chen, "Uranine: Real-time privacy leakage monitoring without system modification for Android," in *Proc. 11th Int. Conf. Secur. Privacy Commun. Syst. (SecureComm)*, Dallas, TX, USA, Oct. 2015, pp. 256–276. doi: 10.1007/978-3-319-28865-9_14.

[50] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for Android apps," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Denver, CO, USA, Oct. 2015, pp. 518–529.

[51] D. Kong, L. Cen, and H. Jin, "AUTOREB: Automatically understanding the review-to-behavior fidelity in Android applications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Denver, CO, USA, Oct. 2015, pp. 530–541.

[52] 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)*, Osaka, Japan, Sep. 2015, pp. 1107–1118.

[53] E. Kowalczyk, A. M. Memon, and M. B. Cohen, "Piecing together app behavior from multiple artifacts: A case study," in *Proc. 26th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Gaithersbury, MD, USA, Nov. 2015, pp. 438–449.

[54] V. F. Taylor and I. Martinovic, "Securank: Starving permission-hungry apps using contextual permission analysis," in *Proc. 6th Workshop Secur. Privacy Smartphones Mobile Devices*. New York, NY, USA: ACM, 2016, pp. 43–52.

[55] J. Wang and Q. Chen, "ASPG: Generating Android semantic permissions," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 591–598.

[56] J. Wu, M. Yang, and T. Luo, "PACS: Pemission abuse checking system for Android applictions based on review mining," in *Proc. IEEE Conf. Dependable Secure Comput.*, Aug. 2017, pp. 251–258.
[57] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *Proc. Eur. Conf. Comput. Vis.* Springer, 2006, pp. 490–503.
[58] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.
[59] A. McCallum *et al.*, "A comparison of event models for naive bayes text classification," in *Proc. AAAI Workshop Learn. Text Categorization*, 1998, vol. 752, no. 1, pp. 41–48.
[60] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic retrieval based on staged logistic regression," in *Proc. 15th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.* New York, NY, USA: ACM, 1992, pp. 198–210.
[61] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale Bayesian logistic regression for text categorization," *Technometrics*, vol. 49, no. 3, pp. 291–304, 2007.
[62] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proc. ICML*, vol. 99, 1999, pp. 200–209.
[63] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, Nov. 2001.
[64] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. (2016). "Bag of tricks for efficient text classification." [Online]. Available: https://arxiv.org/abs/1607.01759
[65] P. Liu, X. Qiu, and X. Huang. (2016). "Recurrent neural network for text classification with multi-task learning." [Online]. Available: https://arxiv.org/abs/1605.05101
[66] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

YINGLAN FENG received the bachelor's degree in software engineering from Jinan University, Guangzhou, China, in 2017. She is currently pursuing the master's degree in computer technology with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. Her research interests include software engineering, mobile app analysis, natural language processing, and deep neural networks.

LIANG CHEN received the bachelor's and Ph.D. degrees from the Advanced Computing and System Laboratory (CCNT), College of Computer Science and Technology, Zhejiang University, China, in 2015 and 2009, respectively. He is currently a Distinguished Research Fellow with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include service computing, social networks, recommendation systems, and heterogeneous information networks. His research focuses on solving traditional challenges via heterogeneous data sources and data mining techniques.

ANGYU ZHENG received the bachelor's degree in software engineering from South China Normal University, Guangzhou, China, in 2018. She is currently pursuing the master's degree in computer science and technology with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou. Her research interests include recommendation systems, natural language processing, deep learning, and transfer learning.

CUIYUN GAO received the B.Eng. degree from the Department of Communication Engineering, Shanghai University, in 2014, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2018. She is currently a Postdoctoral Fellow. Her research interests include user experience study, natural language processing, and mobile app analysis. She had published 11 publications in conferences in her area of expertise. She served as a Reviewer for the IEEE Access, in 2018, and an external Reviewer for many top-tier conferences and journals.

ZIBIN ZHENG received the Ph.D. degree from The Chinese University of Hong Kong, in 2011. He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include service computing, software engineering, and blockchain. He received the ACM SIGSOFT Distinguished Paper Award from ICSE 2010, the Best Student Paper Award from ICWS 2010, and the IBM Ph.D. Fellowship Award.