2018 IEEE 26th International Requirements Engineering Conference

Mining Android App Descriptions for Permission Requirements Recommendation

Xueqing Liu, Yue Leng
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, IL, USA
xliu93,yueleng2@illinois.edu

Wei Yang
Department of Computer Science
University of Texas Dallas
Richardson, TX, USA
weiyang.utd@gmail.com

Chengxiang Zhai, Tao Xie
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, IL, USA
czhai,taoxie@illinois.edu

***Abstract*—During the development or maintenance of an Android app, the app developer needs to determine the app's security and privacy requirements such as permission requirements. Permission requirements include two folds: (1) what permissions (i.e., access to sensitive resources, e.g., location or contact list) the app needs to request, and (2) how to explain the reason of permission usages to users.**

In this paper, we focus on the multiple challenges that developers face when creating the explanations for permission usages. We propose a novel framework, CLAP, that mines potential explanations from the descriptions of similar apps. CLAP leverages information retrieval and text summarization techniques to find frequent permission usages. We evaluate CLAP on a large dataset containing 1.4 million Android apps. The evaluation results show that CLAP outperforms existing state-of-the-art approaches, and has great promise to assist developers for permission requirements discovery.

***Index Terms*—Security requirement, Android permission, natural language processing**

I. Introduction

Security and privacy on mobile devices has been a challenging task [1]–[6]. Recently user privacy gathered new attentions following the Facebook-Cambridge Analytica data scandal [7]. The current solution for user privacy protection on the Android platform mainly relies on a permission mechanism, i.e., apps have to request permissions before getting access to sensitive resources. Unfortunately, previous work [2] finds that apps frequently request more permissions than the apps need. To reduce users' concerns toward those *over-privileged apps* [1], [2] and improve the users' understanding of permission usages [8], [9], one effective approach is to give the users warnings by showing natural language explanations [4]. For instance, WHYPER [10] uses app description sentences to explain permissions; Android and iOS also launched their features of runtime permission explanations in 2015 and 2012, respectively.

Permission explanations are short sentences that state the purpose of using a permission. Permission explanations are written by Android developers [11]; within our knowledge, there exists no previous work on studying the steps of multi-stakeholder elicitation [12] or requirements specification [13] for writing such sentences. Without these steps, can we rely solely on developers' decisions to explain permissions? Although there exist many good examples of app explanations, it is unclear whether explanations provided by developers are interpretable from an average user's perspective. In particular, three major challenges can reduce the interpretability of an explanation sentence. (1) *Technical Jargons*. Due to the domain knowledge owned by the developers but not the average users, the developers' explanations sometimes contain technical jargons/logics hard for the average users to understand. For example, app *GeoTimer Lite* explains the location permission as for "*geofence*" [14]; however, the average users may not know the meaning of geofence, not to say why geofence requires the location permission [15]. (2) *Optimal Length*. If the explanation is too short, it is likely ambiguous (e.g., in Figure 1, it is unclear whether "*store locator*" refers to a locator outside or inside the store); on the other hand, if the explanation is long and wordy, users may choose to skip it. It can be challenging for the developers alone to make the decision on the length/degree of detailedness. (3) *Rare*

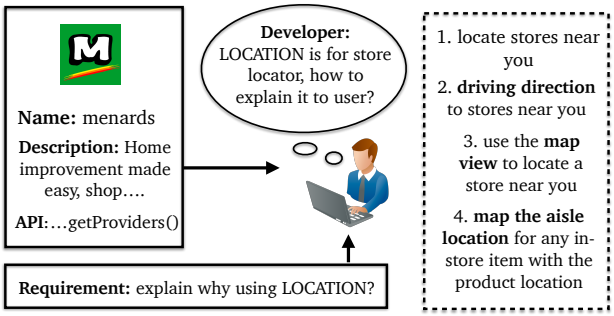

Fig. 1: An example showing how CLAP assists developers with permission requirements, with the dashed rectangle showing sentences recommended by CLAP.

Permission Usage. Although it is relatively easy to explain commonly acknowledged permission usages, e.g., the location permission in a GPS app, it becomes much more challenging to *clearly* explain rare permission usages.

After identifying difficulties in explaining permissions, we propose the first study on the requirements specification/discovery of permission explanations, and we call it the process of *permission requirements discovery*. In particular, we build a recommender system, which recommends a list of potential requirements for the permission explanation (i.e., sentences from similar apps' descriptions[1]) so that developers could refer to the list for improving the interpretability of their explanations. In Figure 1, we illustrate how our system helps the developer of an app discover the requirements. First, by observing sentence 2 and sentence 4, the developer finds the current explanation "*store locator*" ambiguous, and then explicitly specifies indoor/outdoor; second, by observing the keyword "*map*" in sentence 3, the developer is reminded of the map feature and adds it to the explanation; finally, by observing sentence 4, the developer discovers a new feature, i.e., indoor locator, to be added to the app.

Because our recommender system leverages similar apps' descriptions, we name it CLAP, which is the abbreviation for **C**o**L**laborative **A**pp **P**ermission recommendation. CLAP uses the following four-step process to recommend a list of candidate sentences. First, based on information from the current app (the current app's title, description, permissions, or category), CLAP leverages a text retrieval technique to rank every app from the dataset (Section II). Second, for every top-ranked app, CLAP goes through every sentence in its description text and assesses whether the sentence explains the target permission (Section III-B). CLAP further processes matched sentences so that each sentence contains only one explanation (Section III-A). Third, CLAP aggregates text information of the top-K similar apps, and uses the aggregated word values to re-rank the candidate sentences found in the previous step (Section IV). Finally, for top re-ranked sentences, CLAP post-processes the sentences to remove duplications and to improve their interpretability (Section V).

We evaluate CLAP's performance (Section VI) on a large dataset consisting of 1.4 million Android apps. First, we examine the relevance of recommended sentences. To evaluate the relevance, we extract the purpose-explaining sentences from 916 apps as the gold standard sentences, and compare CLAP-recommended sentences with the gold-standard sentences. The evaluation results show that CLAP has a high relevance score compared with existing state-of-the-art approaches [10]. Second, we conduct a qualitative study on specific examples, to observe to what extent the CLAP results can help with the interpretability. The study results show that CLAP can effectively recommend candidate sentences that are concise, convey specific purposes, and support a diverse choice of re-phrasing for the same purpose. These characteristics show great promise of CLAP in helping developers find more interpretable explanations and bridging the knowledge gap between different stakeholders' viewpoints.

[1]Alternatively, we can also use privacy documents and runtime permission messages. However, both data sources are much more scarce than app descriptions. As a result, we choose to use app descriptions. However, the two data resources are both applicable to the CLAP framework.

This paper makes the following three main contributions:

- We make the first attempt to study the problem of permission requirements discovery, with a focus on explaining an app's permission to users.
- We propose a novel CLAP framework for addressing the formulated problem by leveraging similar apps' permission-explaining sentences.
- We evaluate CLAP on a large dataset and show that CLAP effectively provides highly relevant explaining sentences, showing great promise of CLAP as an assistant for requirements discovery of app-permission explanations.

II. Similar-App Ranker

For the first step of the CLAP framework, we design a similar-app ranker to find apps (which also use the target permission) that are the most similar to the current app.

We define the similarity score between the current app Q and candidate app D on the permission P as the linear interpolation of scores in four components, i.e., the pairwise similarities between Q and D's descriptions, titles, permissions, and categories:

$$\begin{aligned} sim(Q,D,P) &= (\lambda_1 sim_{desc}(Q,D) \\ &\quad +\lambda_2 sim_{title}(Q,D) + \lambda_3 sim_{perm}(Q,D) \\ &\quad +\lambda_4 sim_{cate}(Q,D)) \end{aligned} \quad (1)$$

where the coefficients λ_i's control the importance of each component. Next, we describe the definitions of each similarity component.

A. Description Similarity

To model the similarity between two descriptions, we use Okapi BM25 [16], In contrast, previous work [17] uses the topic modeling technique to capture the similarity between app descriptions. The reason why we choose to use a retrieval model for app descriptions is that app descriptions are usually longer texts (on average an app description contains 135 words). For long texts, the topic modeling technique would bring two apps together even if they only remotely belong to the same topic (instead of closely related, e.g., email apps and SMS apps are "similar" by the topic modeling technique, although they clearly have different functionalities). On the other hand, text retrieval models capture more discriminativeness between the descriptions, so they are more suitable for our problem.

To model the text similarity using BM25, we further capture both the unigrams and bigrams from the description text. We stem the description texts before turning them into unigrams and bigrams. In addition to stemming, we also carry out the following pre-processing steps, which are standard pre-processing techniques in text retrieval tasks. These standard techniques improve the ranking performance by enhancing the discriminativeness of each app description.

Stop-word Removal. We remove regular English stop words from Python's nltk stop words list [18], e.g. "*the*" and "*a*." Meanwhile, words such as "*Android*," "*application*," and "*version*" should also be treated as stop words, because they can appear in any app. We identify a complete list of 294 words. We create the list by empirically scanning through the top frequent words, and then manually annotating whether each word can appear in any app, regardless of the context. The list can be found on our project website [19].

Background-sentence Removal. A mobile-app description usually contains some sentences that explain common issues, e.g., "*fixed bug in version 1.7*." Same as stop words, such sentences are "stop sentences", which do not help explain the unique functionality of the app. As a result, we implement a remover of common background sentences for mobile apps using 53 regular expressions. Same as the creation of stop words, the creation of regular expressions is based on the empirical judgment on whether a sentence can appear in any app, e.g., `.*version\s+\d.*` detects whether a sentence describes a version number. The list of regular expressions can be found on our project website [19].

After the preceding pre-processing steps, we obtain the BM25 scores between the current app Q and every candidate app D in the dataset. To make the description similarity comparable to other similarity components, we normalize the BM25 scores with the maximum BM25 score over all the candidates before plugging the normalized score into Equation 1.

B. Title Similarity

An app's description usually offers the most information to capture its similarities with other apps [17], but if CLAP uses only the descriptions, sometimes it is difficult to retrieve accurate results, due to the noisy components in descriptions that are not fully cleaned in pre-processing[2]. To this end, app titles can serve as a complement to descriptions in modeling app similarities.

One challenge in modeling the title similarity is the vocabulary gap between similar words, e.g., "*alarm*" and "*wake up clock*," mainly because titles are short texts (on average a title contains 2.8 words). As a result, we use a different technique to model the title similarity. We leverage word embedding vectors [20] (GoogleNews-neg300 [21]) for bridging the vocabulary gap. For each pair of apps Q and D, we define their title similarity as the average cosine similarity between each word $w_1 \in Q$ and each word $w_2 \in D$. To avoid over-matching unrelated word pairs, we empirically cut the cosine similarities at 0.4 and set them to 0 if their original scores are less than 0.4.

C. Permission Similarity

Because app permissions are categorical data, we model the permission similarity as the Jaccard distance between the two permission lists. The reason why we incorporate the permission similarity is based on the observation that an app's permissions can reflect its functionality. For example, emergency contact apps usually use `READ_CONTACTS` and `ACCESS_FINE_LOCATION` at the same time, and the usage of location permission distinguishes these apps from other contact apps.

Previous work [17] leverages security-sensitive APIs to model the similarity between apps. Security-sensitive APIs are a finer-grained version of Android permissions. Although APIs carry more information than the permissions, it is also more challenging to model the API similarity. The challenge comes from the fact that developers often use different APIs to achieve the same functionality (e.g., a Stack Overflow post [22] shows several different techniques to obtain user location), and use the same API to achieve different functionalities. As a result, we model only the permission-level similarity and leave the exploration of API similarity for future work.

D. Category Similarity

Finally, we capture the category similarity between the two apps. The reason for using the category information is that we observe multiple cases where using only the descriptions is ambiguous. In some cases, the category information can help clarify the apps' functionalities. For example, we find two apps whose descriptions are close to each other, and yet one app is a cooking app for cookie recipe while the other app is a business app for selling cookies. We represent each category as a TF-IDF vector, which comes from words that appear in the descriptions of apps in the category. The similarity between Q and D is defined as the cosine similarity between the two vectors.

III. Identifying Permission-Explaining Sentences

After retrieving similar apps of the current app Q, the next step of CLAP is to identify permission-explaining sentences among those similar apps' descriptions.

Previous work such as WHYPER [10] addresses this problem (of identifying permission-explaining sentences) by matching sentences from the app description against frequent words in the permission's API documents. WHYPER uses only the *entire* description sentences to explain the permission. In our problem, however, using the entire sentences can be ineffective. The reason for such ineffectiveness is that we are using *other* apps' sentences to explain the current app. An entire sentence from another app sometimes contains redundant information: while a part of the sentence matches the current app's purpose, the other part does not match it. For example, the sentence "*save the recording as a ringtone and share it with your friends*" describes the usages of two permissions: `RECORD_AUDIO` and `READ_CONTACTS`, whereas the current app uses only the first permission. If we use the entire sentence to explain the current app, the second part is irrelevant, whereas if we discard the entire sentence, the relevant part is also discarded. In such cases, if we break the original sentence into shorter units, the first part will contain only the relevant

[2]For example, many app descriptions contain SEO words, which may not be strictly relevant to app functionality.

information. CLAP leverages this methodology to break the original sentence into shorter ones so that some of them are more relevant than the original sentence. We describe this process in Section III-A.

A. Breaking Sentences into Individual Purposes

To break a sentence into shorter ones, we leverage the Stanford PCFG parser [23] to parse each sentence s into a tree T. In particular, we extract its sub-sentences based on two main observations. First, following the aforementioned example, if the sentence contains conjunction(s), we split it at the conjunction(s), and then extract the sub-sentences. Second, as discussed in previous work [10], [24], permission usages can usually be captured by short verb phrases, e.g., "*create QR code from contact*," "*assign contact ringtone*." Therefore, we also extract the verb phrases in the sentence.

After the split, CLAP adds both the original sentence and the shorter sentences into a candidate sentence set, which is then passed on to the next step for identifying permission-explaining sentences. We intend to include as many candidate sentences as possible to boost the quality of the finally chosen ones. Therefore, when we traverse the parsing tree T, we keep all the verb phrases; e.g., if one verb phrase is embedded in another, we include both of them in the candidate set.

We summarize our candidate-sentence generator in Algorithm 1 for a clearer view, where $s(n)$ denotes the phrase (in sentence s) corresponding to node n.

Algorithm 1: Constructing Candidate Set

Input : Sentence s and its tree structure T obtained from constituent parsing [23];
Output: Candidate sentences S from s;

```
S ← ∅;
S ← S ∪ {s};  // add the original sentence
for node n in T do
    if n = VP then
        S ← S ∪ {s(n)};      // add verb phrase
    end
    if n = CC then
        for node n_0 in n.parent.children and
          n_0 != CC do
            S ← S ∪ {s(n_0)}; // break conjuncts
        end
    end
end
```

B. Matching Permission-Explaining Sentences

Using Keyword Matching. After obtaining the candidate sentence set from the preceding step, we use a pre-defined set of rules to match each candidate sentence, and keep only those sentences that address the target permission. More specifically, the pre-defined set of rules include keywords and POS tags [25]. The reason why we leverage the POS tags is to disambiguate between a word's senses based on its tag. For example, when the word "*contact*" is used as a noun, it usually refers to phone contacts, so it explains `READ_CONTACTS`, whereas if it is used as a verb, e.g., "*contact us through email*," it does not explain `READ_CONTACTS`. The pre-defined keywords and POS tags set can be found on our project website [19].

Using WHYPER to Match Sentences. Alternatively, we can use WHYPER in this step. The reason why we use the keyword matching is for a low time cost and for real-time processing. WHYPER traverses the entire dependency parsing graph. This step makes WHYPER run at least 100 times slower than the keyword matching. Meanwhile, the size of our data dictates that we need to process tens of millions of sentences for each permission. As a result, we use keyword matching to speed up this step. We plan to support WHYPER in future extensions of CLAP.

After the preceding steps, we discard apps that CLAP has not identified any sentences from.

IV. Ranking Candidate Explaining Sentences

After the preceding steps, CLAP obtains similar apps and candidate permission-explaining sentences. Next, CLAP ranks the candidate sentences and recommends the top sentences to the developer.

Why Ranking Sentences? After obtaining explaining sentences, a straightforward technique for recommending sentences is the greedy technique, i.e., scanning through the app list top-down and extracting the first 5 sentences. However, this simple technique makes mistakes for the following two reasons. First, due to the noise in the data, the retrieved similar apps inevitably contain false positive ones[3]. As a result, it is very likely for the greedy technique to select sentences from a mismatched app; sentences from mismatched apps usually discuss different purposes. Second, even if an app is correctly matched, it may still use the same permission for a different purpose. For example, an alarm app may use `ACCESS_FINE_LOCATION` for weather report and advertisement at the same time.

Ranking Candidate Sentences with Majority-Voting. Because the greedy technique could easily recommend false positive sentences, CLAP adopts an alternative technique: it builds a large set of candidate sentences by breaking and matching the sentences in the top-K apps (i.e., the preceding steps in Section II-Section III), and it then leverages a ranking function to recommend the top-ranked sentences from the candidates. The top-ranked sentences are expected to be more likely the true permission usage. But we do not know the true permission usage; so how to design the ranking function? To answer this question, we get the inspiration from the *majority-voting* principle [28]. In particular, the more frequent an explanation is seen in the data (i.e., the similar apps' explanations), the more likely this explanation is widely accepted

[3]After exploring three retrieval techniques: BM25 [16], language model [26], and vector space model [27], we find that all the techniques generate false positive results. Such results are due to noisy components in the app descriptions, e.g., SEO words that are sometimes irrelevant to the primary app functionality.

by peer developers; as a result, the more likely this sentence is describing the true permission usage.

To adopt the majority-voting principle, we need to find out how frequent each explanation is, or how many votes each sentence receives. The votes should not be based on a sentence's exact-matching frequency in the dataset; a sentence may have appeared only once, and yet its purpose is repeated many times in other sentences. That is to say, votes should reflect the *semantic frequency* of the stated purpose. We can estimate the semantic frequency of a sentence by first estimating the semantic frequencies of its words, and then averaging them to get score of the sentence.

Semantic Frequency of a Word. We may use a word's term frequency to represent its semantic frequency (in the dataset); but if so, the top-ranked words would be non-discriminative, even after removing stop words. For example, the top-3 most frequent words for `READ_CONTACTS` are "*contact*," "*contacts*," and "*read*."

If these words are used to recommend the sentence, they would likely recommend sentences such as "*to read contacts*," which does not address any specific purpose. As a result, we build a discriminative word-voting function by leveraging the *inverse document frequency* (IDF [29]) and text summarization techniques.

We compute the votes for each word with the following two-step process. First, we apply a text summarization algorithm [30] to turn each app description into a ⟨word, weight⟩ vector, and compute the average vector over all the top-K similar apps. Second, for each ⟨word, weight⟩ pair in the average vector, we multiply the word's weight by its IDF value in the dataset. The resulting vector represents the votes that each word receives. The text summarization algorithm is TextRank [30], which is a graph-based algorithm based-on PageRank [31]. TextRank takes a document as input, and outputs a ⟨word, weight⟩ vector by leveraging the affinity of word pairs.

The weight associated with each word represents how much the word connects with other words, or how important it is to the document. After obtaining the TextRank scores, we further normalize the weights so that the weights from different apps are comparable to each other. In summary, the votes for a word are defined as:

$$votes(w) = IDF(w) \times \frac{1}{K} \sum_{k=1}^{K} \frac{TextRank(w, D_k)}{\max_{w' \in V} TextRank(w', D_k)} \quad (2)$$

where V is the vocabulary set and D_k represents the k-th similar app retrieved by our app ranker (Section II). Some examples of the top-ranked words are shown in Table IV. We can see that the most voted words are often strongly related to the true permission usage.

Semantic Frequency of a Sentence. The votes for each sentence s are the average over the votes for each word:

$$votes(s) = \frac{1}{|s|} \sum_{w \in s} votes(w)$$

V. Postprocessing Permission-Explaining Sentences

Finally, CLAP post-processes the most voted sentences from the preceding steps. The post-processing includes the following two steps.

Removing Duplicated Sentences. After the sentences are ranked by their votes, some sentences may be duplicated. To ensure the diversity of the resulting sentences, we use the greedy technique to select the first 5 unique sentences and recommend them to the developer.

Adding Direct Mentions of Permissions. Note that one sentence can most clearly explain the target permission when the sentence *explicitly* mentions the permission's name. On the other hand, some sentences contain only *implicit* mentions of the permission usage. For example, the sentence "*send text messages to your contacts*" explicitly mentions the target permission `READ_CONTACTS` while another sentence "*send text messages*" only implicitly mentions the permission. To improve the interpretability of the resulting sentences, CLAP uses a list of pre-defined rules to rewrite an implicit permission-mentioning sentence into an explicit permission-mentioning sentence. For example, "*send text messages*" is rewritten to "*send text message (from/to contact)*." Our evaluations do not rely on the post-processing. However, the post-processing steps intuitively help with the understanding of the resulting sentences. The pre-defined rules used for post-processing can be found on our project website [19].

VI. Evaluation

To assess the effectiveness of CLAP, we design experiments to answer an important research question: to what extent can CLAP help developers with improving the interpretability of explanation sentences?

To answer this research question, we need to first validate the relevance of a recommended sentence to the app's permission purpose. Notice that for assisting the developer in writing explanations, a recommended sentence must first be *relevant* to the current app's permission purpose, i.e., the sentence discusses the same permission purpose as the current app. Otherwise, the sentence would be invalid for helping the developer, wasting the developer's time to read such sentence. To evaluate the relevance of recommended sentences, we conduct quantitative studies using two groups of test collections[4] (Section VI-E and Section VI-F). The first group contains gold-standard permission purposes explicitly annotated by app developers; the second group contains gold-standard sentences annotated by two authors of this paper. After evaluating the relevance, we conduct a qualitative study to inspect the interpretability of example recommended sentences (Section VI-G).

A. Dataset

We use the PlayDrone dataset [32], which is a snapshot of the Google Play store in November 2014. Our

[4]A test collection contains a set of ⟨app, sentence⟩ pairs where the sentence explains the permission usage of the app.

TABLE I: Sizes of our three app-sets and five test collections: Q_{authr}'s, author-annotated explanations; Q_{dev}'s, developer-annotated explanations.

	app-set	Q_{authr}	Q_{dev}
CONTACT	62,147	48	160
RECORD	75,034	48	103
LOCATION	76,528	N/A	564

dataset consists of 1.4 million apps in total. In order to fairly compare with the state-of-the-art technique for permission explanation, i.e., WHYPER [10], we study three permissions [33]: `READ_CONTACTS`, `RECORD_AUDIO`, and `ACCESS_FINE_LOCATION`[5]. We denote the set of apps containing each of the three permissions in a different font: CONTACT, RECORD, and LOCATION. We keep only those apps whose descriptions are in English. We show the sizes of the three app-sets in Table I. Because the original LOCATION app-set is too large (more than 360,000 apps), we sample 21% apps from the original set for efficiency. Column #Apps of Table I shows the sizes of the three app-sets.

B. Extracting Gold-Standard Sentences

When measuring the quality of a recommended sentence, the gold-standard sentence is the ideal explaining sentence to compare with. Strictly speaking, it is difficult to obtain a large-scale gold-standard test collection without soliciting annotations from the developers themselves. However, we are able to obtain a significant number of gold-standard sentences through (1) discovering a small set of apps where the developers have annotated the permission usages, and (2) manually annotating a collection of explaining sentences. We describe the two techniques as below[6].

Developer-Annotated Explanations. In the PlayDrone dataset, we observe that a small number of apps (2‰) have included permission explanations in their app descriptions. For example, app *AlarmMon* [34] appends the following sentences to its main body of description: "*AlarmMon requests access for reasons below...: ... `ACCESS_FINE_LOCATION`: AlarmMon requests access in order to provide the current weather for your location after alarms...*" After observing a significant number of gold-standard sentences annotated by developers, we find that these sentences appear in a clear textual pattern: these sentences are usually located at the end of the app descriptions, with a capitalized permission name followed by a permission-explaining sentence. As a result, we can use regular expressions to automatically extract such sentences from raw description texts (the regular expressions can be found on our project website [19]). We manually inspect a small sample of extracted sentences to double check whether the regular expressions work as expected, and the results of our manual inspection have an average precision of 97%. We use this technique to obtain three test collections for our three permissions, denoted as as Q_{dev}'s. We show the number of ⟨app, gold-standard sentence⟩ pairs in each Q_{dev} in Table I.

Author-Annotated Explanations. Although Q_{dev}'s can reflect permission explanations, there exist length biases in Q_{dev}'s. The average length of app descriptions from Q_{dev}'s (330 words) is 2.4 times that of all app descriptions (135 words). The reason for such difference is that apps that carefully address permission explanations tend to carefully address the entire app description as well. Because CLAP is built on top of text retrieval models, its performance depends on the length of the current app's description. In order to observe CLAP's performance on shorter app descriptions, we follow the evaluation technique from previous work [10] to uniformly sample apps from the entire app-set (for each permission), and then manually annotate the gold-standard sentences. Two authors go through each description sentence, independently annotate the sentences that explain the target permission, and discuss to resolve annotation differences if any. In total, the manual efforts involve annotating ∼2,000 sentences for each test collection. We denote the author-annotated collections as Q_{authr}'s, and show their sizes in Table I[7].

Discussions on the Sizes of Test Collections. The sizes of our test collections range from 48 to 564, which is relatively small. However, it is also almost intractable to obtain larger collections. First, manual annotations on permission explanations require a reasonable amount of domain knowledge in mobile apps and technologies. As a result, these efforts cannot be trivially replaced by crowd-workers' annotations. Second, we also cannot rely on existing tools for automatic annotations. We test state-of-the-art sentence annotation tools in previous work [10], [24]. Unfortunately, these tools have large false positive rates[8], and therefore the annotated sentences by these tools are not clean enough to serve as gold-standard sentences. In total, our five test collections consist of 916 ⟨app, gold-standard sentence⟩ pairs.

C. Evaluation Metrics

To evaluate the relevance of CLAP-recommended sentences to the gold-standard sentence, we define the following metrics.

SAC: Sentence accuracy based on manual judgment. After obtaining sentences recommended by CLAP (and sentences recommended by all baselines), we manually judge the accuracy of the results. For each pair of gold-standard sentence × CLAP-recommended sentence, two authors independently judge whether the sentences in the pair are semantically identical, and discuss to resolve the judgment differences if

[5]The reason for us to choose the three permissions is that the WHYPER tool [10] provides full pipelines for only three permissions. For other permissions, although it is possible to complete the full pipeline with our efforts, the comparison against baselines may not be fair. We plan to include more permissions in future work.

[6]All test collections in this paper can be found on our project website [19].

[7]Due to significant manual efforts needed in the annotations, we construct only CONTACT$_{authr}$ and RECORD$_{authr}$ without constructing LOCATION$_{authr}$ for the work in this paper.

[8]We evaluate false positive (FP) rates of WHYPER [35] and AutoCog [24] on the WHYPER benchmark. WHYPER has a 20% FP rate on the `READ_CONTACTS` app-set and 21% FP rate on the `RECORD_AUDIO` app-set. AutoCog has a 33% FP rate on the `READ_CONTACTS` app-set.

TABLE II: The quantitative evaluation results of text-similarity scores: JI (average Jaccard index) and WES (average word-embedding similarity). The highest score among the four approaches is displayed in bold, and the second highest score is displayed with a †. We also show the p-values of T-tests between the highest score and second highest score, and the p-value is shown in bold if it is significant (less than 0.05). The parameter settings here are $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$, top-K=500.

		CONTACT$_{dev}$			RECORD$_{dev}$			LOCATION$_{dev}$			CONTACT$_{authr}$			RECORD$_{authr}$		
		top1	top3	top5	top1	top3	top5	top1	top3	top5	top1	top3	top5	top1	top3	top5
JI	T+K	0.015	0.015	0.014	0.054	0.052	0.054	0.019†	0.019†	0.019†	0.065†	0.061†	0.061†	0.064	0.069	0.069
	T+W	0.023†	0.026†	0.026†	**0.092**	0.087†	0.086†	\	\	\	0.058	0.059	0.055	0.118†	0.107†	0.108†
	R+K	0.013	0.008	0.008	0.042	0.044	0.043	0.014	0.012	0.012	0.042	0.037	0.043	0.090	0.082	0.084
	CLAP	**0.032**	**0.036**	**0.037**	0.091†	**0.105**	**0.103**	**0.027**	**0.025**	**0.023**	**0.186**	**0.170**	**0.152**	**0.133**	**0.147**	**0.129**
	p	0.18	0.07	**0.03**	\	0.16	0.15	**0.04**	**0.03**	**0.03**	**6e-4**	**7e-5**	**1e-4**	0.065	0.06	0.27
WES	T+K	0.012	0.013	0.012	0.041	0.040	0.040	0.014†	0.014†	0.014†	0.040†	0.040†	0.039†	0.033	0.040	0.040
	T+W	0.016†	0.018†	0.019†	0.061†	0.060†	0.060†	\	\	\	0.038	0.039	0.036	0.056†	0.051†	0.050†
	R+K	0.012	0.010	0.010	0.039	0.035	0.038	0.010	0.010	0.010	0.025	0.027	0.031	0.045	0.041	0.043
	CLAP	**0.031**	**0.033**	**0.033**	**0.079**	**0.084**	**0.081**	**0.025**	**0.023**	**0.021**	**0.114**	**0.107**	**0.097**	**0.070**	**0.076**	**0.068**
	p	**3e-4**	**2e-4**	**5e-4**	0.11	**9e-3**	**9e-3**	**6e-5**	**3e-6**	**5e-7**	**1e-5**	**5e-7**	**2e-6**	0.28	**4e-3**	**0.02**

any[9]. This step gives rise to $2 \times 48 \times 4 \times 5 = 1,920$ sentence-pair labels.

AAC: App accuracy based on manual judgment. In addition to the sentence accuracy, we also evaluate the accuracy of the app where the recommended sentence comes from. The reason to evaluate the app accuracy is that the developer may want to further make sure that the retrieved apps share the same functionality with the current app. For each pair of ⟨retrieved app, the current app⟩, two authors independently judge whether the apps in the pair share the same functionality, and discuss to resolve judgment differences if any. This step gives rise to $2 \times 48 \times 4 \times 5 = 1,920$ app-pair labels[10].

JI: Average Jaccard index [36]. We propose to use an automatic evaluation metric. The average Jaccard index measures the average word-token overlap between a recommended sentence and the gold-standard sentence. We remove stop words in both sentences to reduce the matching of non-informative words.

WES: Average word-embedding similarity. The average Jaccard index measures only the word-token overlaps. To better capture the semantic similarity, we propose to use another automatic metric, the average cosine distance between word embedding representations of the two sentences [21], in short as WES. WES shares the same formulation as the title-similarity function in Section II-B. More precisely,

$$WES(s_r, s_g) = \frac{1}{|s_r|}\frac{1}{|s_g|}\sum_{w_1 \in s_r, w_2 \in s_g} sparse\_cos(w_1, w_2)$$

where s_r and s_g are the recommended sentence and the gold-standard sentence, respectively. $sparse\_cos$ is set to the word2vec similarity (between w_1 and w_2) if the word2vec similarity is larger than 0.4; otherwise, $sparse\_cos$ is set to 0.

For each metric, we report the overall average scores over the top-1, top-3, and top-5 recommended sentences.

[9]For example, if gold-standard sentence s_1 = "*this app uses your contacts permission for contact suggestion*," recommended sentence s_2 = "*to automatically suggest contact*," and s_3 = "*to read contacts*," we judge s_2 as relevant and s_3 as non-relevant.

[10]For example, for app a_1 = "*group sms*," a_2 = "*group message*," and a_3 = "*sms template*," we judge the app a_2 as relevant and a_3 as non-relevant.

D. Alternative Approaches Under Comparison

Because no previous work has focused on the same setting as our problem, we cannot compare CLAP's performance with an end-to-end approach that entirely comes from any previous work. However, we can build baseline approaches by following intuitive strategies to assemble state-of-the-art approaches as below.

Top Similar apps + Permission Keywords (T+K). For the first baseline approach, we go through the same process for ranking apps (Section II) and matching permission-explaining sentences (Section III-B). However, instead of breaking and ranking sentences, this baseline approach scans through the original description sentences top-down and greedily recommends the first 5 sentences matched by our keyword matcher (Section III-B).

Top Similar apps + WHYPER (T+W). This alternative approach follows the same pipeline as T + K, except that the sentence matching is through WHYPER [10] instead of our keyword matcher.

Random Similar apps + Keywords (R+K). This alternative approach follows the same pipeline as T + K, except that the sentence selection is not through the greedy way. Instead, the recommended sentences are randomly sampled from all the original sentences matched by our keyword matcher.

E. Automatic Quantitative Evaluation: Text-Similarity Scores

For the first step of the quantitative study, we examine the automatic evaluation metrics JI and WES on the five test collections (including 916 gold-standard sentences). In Table II, we report the average JI and WES over the top-1, top-3, and top-5 sentences recommended by CLAP and the three baselines. To configure the parameter settings for the study, we empirically set the top-K in the majority voting (Section IV) to 500; we empirically set $\lambda_1 = \lambda_2 = 0.4$ and $\lambda_3 = \lambda_4 = 0.1$ in the similar-app ranker (Equation 1), where the λ_i's are shared by all the four approaches. The reason for us to set larger weights on the titles and descriptions than on the permissions and categories is that the titles and descriptions have more discriminative power than the permissions and categories.

Result Analysis. To observe CLAP's performance, for each setting in Table II: ⟨test collection, top-K, metric⟩, we highlight

TABLE III: CLAP's WES results of excluding app descriptions (denoted by "-desc"), excluding titles (denoted by "-title"), and including all four components (denoted by "all")

	-desc	-title	all
CONTACT$_{dev}$	0.026	**0.037**	0.033
RECORD$_{dev}$	0.035	0.077	**0.081**
LOCATION$_{dev}$	0.015	**0.024**	0.023

the approach with the highest score (marked in bold) and second highest score (marked with †). We conduct statistical significance tests, i.e., T-tests [37], between the two scores. We display the p-values of the T-tests. A p-value is highlighted in bold if it shows statistical significance (i.e., p-value less than 0.05). We can observe that CLAP has the highest score over all the settings except for ⟨RECORD$_{dev}$, JI⟩. We can also observe that the majority of T-test results are significant. The three least significant settings are JI in CONTACTS$_{dev}$, RECORD$_{authr}$, and RECORD$_{authr}$. In general, CLAP performs better in WES than JI. Because WES captures external knowledge with word embedding vectors while JI captures only the word-token overlaps, WES models the semantic relevance between the recommended sentences more closely.

On the other hand, when comparing the scores across different top-K values, we can observe that the p-values of the top-5 scores are slightly more robust than those of the top-1 scores. This difference can be explained by the fact that each of the top-5 scores is the average over 5 scores while each of the top-1 scores is an individual score.

Among the three baselines, T + W performs better than T + K, indicating that WHYPER performs better than our keyword matching technique (Section III-B). T + K performs better than R + K, indicating that sentences from the top similar apps are more relevant than those from random similar apps.

Effects of CLAP's Parameters. To study the effects that CLAP's parameters have on its performance, we conduct two experiments where we vary the parameters (λ_i and top-K) and examine how the results change with these parameters.

λ_is: λ_is determine the importance of each component in the similar-app ranker. We study two variants of λ_is (while fixing the top-K): (1) excluding app descriptions; (2) excluding titles. In Table III, we show CLAP's performance in these two settings. We can see that excluding the descriptions always hurts the performance, while excluding the titles can improve the performance. This result indicates that app descriptions are more important than app titles for ranking similar apps.

Top-K: the top-K determines how many similar apps to use for the majority voting. We study the effects of varying the top-K value while keeping the λ_is fixed. We plot CLAP's performance in Figure 2. We can see that the overall WES scores are relatively stable; for location data, the scores slightly increase as the top-K increases.

Summary. The main difference between CLAP and the baseline approaches is that CLAP (1) breaks the sentences into shorter ones; (2) ranks the sentences through majority voting. This result indicates that the two heuristic strategies are effective in improving the relevance of the resulting sentences.

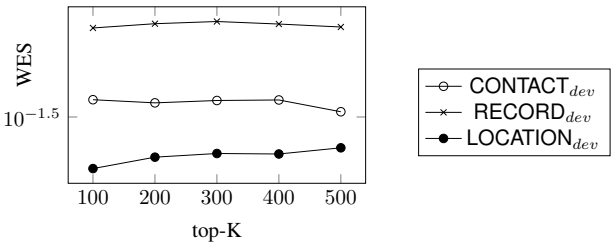

Fig. 2: CLAP's WES results across different K values

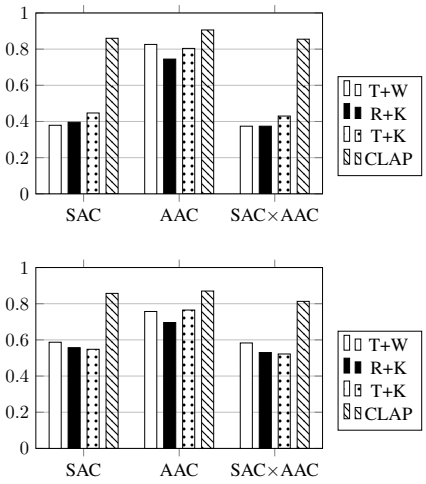

Fig. 3: The quantitative evaluation results of manually-judged accuracy: bar plots show the average accuracy of top-5 results in each of the four approaches. The upper plot shows results on CONTACT$_{authr}$; the lower plot shows results on RECORD$_{authr}$; T-test between the highest and second highest scores in each group are 9e-7, 0.03, 9e-6 (upper) and 4e-6, 0.04, 1e-4 (lower). Parameter settings are $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$, top-K=20.

F. Quantitative Evaluation: Manually-Judged Accuracy

For the second step of the quantitative study, we conduct a manual evaluation on the sentence accuracy (SAC) and app accuracy (AAC). This step is for obtaining more interpretable metrics (accuracy) than JI and WES. The SAC/AAC scores reflect how high percent of the top resulting sentences/apps are relevant. Because SAC/AAC scores come from human judgment, they also more precisely capture the semantic relevance than JI and WES. In Figure 3, we plot the SAC and AAC of the four approaches over the top-5 recommended results. We also plot the average SAC×AAC, which reflects how high percent of ⟨app, sentence⟩ pairs (among top-5 results) contain both a relevant sentence and a relevant app. Here the parameters are fixed to $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$ and top-K = 20.

Results Analysis. Figure 3 shows that CLAP has significantly better performance in all the three metrics. Given the results from Table II, the SAC results are expectable; however, the AAC results are surprising. This serendipity comes from the fact that the baselines (T + K and T + W) follow the greedy technique of recommending the most similar apps, while sometimes those apps turn out to be less similar than the apps recommended by CLAP. Such result might indicate that CLAP has the potential to discover even more relevant

TABLE IV: Example sentences recommended by CLAP

	current app (Q)	CLAP-recommended sentences	$votes(w)$
CONTACT$_{dev}$	• *app name*: lazy **love** • *app description*: lazy **love** allows you to **send message**s to your friends and **love**d ones so you don't forget to **send** to who matters... • *ground truth*: automatically **send** SMS to contacts at scheduled time	• to **send** a scheduled **message** (from/to phone contacts); • can set the time to **send message** (from/to phone contacts) or email • typed in or **select**ed from contacts; • randomly **select**s a **message** (from/to phone contacts) and person from your list to **send** a **message**	**love** **send** **message** feel **text** **select** set
RECORD$_{dev}$	• *app name*: build doc • *app description*: builddoc is an easy-touse project based photo documentation application that allows you to capture field issues and **assign** and mange team member's **task**se ... • *ground truth*: to record voice and audio **notes**	• creating audio **notes** using the device microphone (to record voice); • use your own (recorded) voice to create audio **note**; • record voice **note**s to explain expenses; • compose text **note**s using (recorded) speech to text and voice commands; • capture photo of a book and record yourself reading it to your child;	project **task** upload manage **assign** **note** edit
LOCATION$_{dev}$	• *app name*: menards • *app description*: home improvement made easy, **shop** departments, and more. buy in app or find products at your closest **store**... • *ground truth*: to provide local **store** information and directions from your location	• plus find a **store** near you; • use the map view to locate **store**s near you; • to find a location near you; • search and discover different products from **store**s near you; • map the aisle location of any instock **item** with the product locator;	order reorder **store** **shop** **item** special pickup

apps.

G. Qualitative Evaluation

We next present our qualitative evaluation on helping developers improve the interpretability of their permission explanations: (1) how interpretable are the sentences recommended by CLAP? (2) to what extent can these sentences help developers discover new permission requirements? Because it is difficult to answer these questions quantitatively, we inspect specific examples of the recommended sentences and examine their interpretability.

Column 3 of Table IV shows the sentences that CLAP recommends for three example apps. The three apps come from CONTACTS$_{dev}$, RECORD$_{dev}$, and LOCATION$_{dev}$, respectively. For each app, Column 2 shows its title, description, and the gold-standard explaining sentence. Column 4 shows the top-voted words (based on Equation IV, Section IV). We show a word in bold if it overlaps with words in the recommended sentences or with the current app's description.

From Table IV, we observe the following three characteristics of the recommended sentences.

Diverse Choices of Phrasing. We observe that the recommended sentences provide various rephrasing, e.g., "*to send a scheduled sms*" vs. "*set the time to send message*", allowing the developer to choose from a diverse vocabulary to improve the explanation. The reason why CLAP can support diverse wording choices is that it removes the duplicated sentences in the post-processing step (Section V).

Detailed Purposes. We observe that the sentences recommended by CLAP usually state concrete and detailed permission purposes. In contrast, the sentences recommended by the baselines often contain examples such as "*to read contacts*," which does not mention any specific purpose. The reason why CLAP can recommend more detailed purposes is that it uses the inverse document frequency (IDF) for word voting (Section IV). The IDF helps select the most meaningful words by demoting common and non-discriminative words [29]. Indeed, we observe that words in Column 4 are good indicators of specific permission purposes.

Concise Sentences. We observe that the sentences recommended by CLAP are usually short and concise. This result is due to the fact that CLAP breaks long sentences into shorter ones. Both the long sentences and the shorter sentences are added to the candidate set (Section III-A); however, it is easier for the shorter sentences to be highly voted, because a long sentence tends to contain infrequent words that some of its sub-sentences do not contain. Because the most voted words are frequent words, the shorter sentences are more likely to receive high votes.

We further conduct a quantitative study on the lengths of the sentences recommended by CLAP and the baselines. We

compute the average and maximum lengths of the recommended sentences over all the five test collections in Table I. We find that the average length of the CLAP-recommended sentences is less than 56% of the second shortest average length (CLAP: 8.1; T + W: 14.6, T + K: 14.3, R + K: 15.6) while the maximum length of the CLAP-recommended sentences is less than 36% of the second shortest maximum length (CLAP: 31, T + W: 174, T + K: 174, R + K: 86). Note that if a recommended sentence is as long as 174 words, it must be difficult for the developer to digest. Because conciseness is an important aspect of interpretability [38], sentences recommended by CLAP effectively improve the worst case of interpretability against the baselines.

VII. Limitations and Future Work

In this section, we discuss the limitations of CLAP and future work.

User Study. One limitation of this work is that we have not had a systematic way to directly evaluate the interpretability of explanation sentences. In future work, we plan to investigate more direct evaluation than our current evaluation. In particular, we plan to measure the interpretability from an *end-user*'s perspective, e.g., investigating the following research questions: how often do explanations confuse average users? are there any general rules that developers could follow to improve the interpretability of permission explanations? how to effectively explain rare permission usages?

Availability of Similar Apps. Because CLAP recommends sentences from similar apps' descriptions, its performance depends on both the availability of similar apps and the quality of similar apps' descriptions. If an app lacks enough similar apps, or if its similar apps are poorly explained, CLAP's performance will decrease. To improve CLAP's performance under such cases, we recommend using a larger dataset to increase the number of well-explained candidate sentences.

Checking Apps' Actual Behaviors. In our current work, we measure the similarity between two apps by leveraging four components: the two apps' descriptions, their titles, their permissions, and their categories. Besides the four components, we can further check the Android API methods invoked by the two apps to observe whether these invoked API methods *indeed* share the same permission purpose. One caveat is that CLAP cannot be used to detect over-privileged permissions; for such permissions, CLAP explains their usages in the same way as for legitimate permissions.

VIII. Related Work

Mining App Store Data for Requirements Engineering. In recent years, the requirements engineering community has shown great interest in mining data from the Google Play app store [39], especially text data [40]–[43]. App store data serves as a bridge between app developers and app users. On one hand, text data from the Play store (e.g., app descriptions, existing user reviews, and ratings) has a broad impact on users' decision-making process (e.g., whether to install an app, purchase an app, or give reviews and rating). On the other hand, such data provides important clues for guiding future development and requirements discovery.

App description data can be used for requirements discovery tasks such as domain analysis [44], e.g., analyzing similar apps to discover their common and varied parts. App review data [43], [45]–[49] contain rich user feedback information such as their sentiments toward existing features [43], future feature requirements [47], and bug reports [48]. Privacy policy data can be mined to assist privacy requirements analysis [40]–[42], [50]–[53].

Explaining Android Permission. Compared with targeted attacks, a more prevalent security issue in Android apps is the over-privileged problem [2], i.e., apps using more permissions than they need. The study results by Felt et al. [3] show that users usually have a difficult time understanding why permissions are used. Lin et al. [4], [5] examine users' expectations toward Android permissions. Their results reveal general security concerns toward permission usages; however, the security concerns can be alleviated by providing a natural language sentence to explain the permission purpose.

Previous work has explored multiple approaches to explain an app's permission, e.g., using the app's description sentences [10], [24], a set of manually-annotated purposes [54], pre-defined text templates [55], or GUI mapping [56]. However, these previous approaches all assume that the permission explanations already exist in the app, and therefore these approaches cannot be used to discover new requirements. Our work fills this gap in the previous work by providing tool supports for recommending new permission requirements.

NLP for App Security. In recent years, NLP techniques are widely applied to various security tasks [17], [53]. CHABADA [17] uses the topic modeling technique and outlier detection techniques to discover potential malware within each app cluster. Slavin et al. [53] construct a knowledge hierarchy that joins security sensitive APIs with natural language concepts to detect violations of textual privacy policies. As follow-up work of WHYPER [10], AutoCog [24] uses the app description to represent the most frequent permission purposes.

IX. Conclusion

In this paper, we conduct the first study on the problem of permission requirements discovery for an Android app. When a developer needs to explain a permission usage in the app description, permission requirements discovery could help the developer find potential ways to improve the interpretability of permission explanations. We have proposed the CLAP framework for recommending permission-explaining sentences from similar apps, based on leveraging consensus among the most similar apps and selecting the sentences that best match the consensus. Our evaluation results have shown that CLAP can recommend sentences that are relevant, concise, include detailed purposes, and provide diverse choices of phrasing.

Acknowledgment. This work was supported in part by NSF CNS-1513939, CNS-1408944, CCF-1409423, and CNS-1564274.

REFERENCES

[1] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones." in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, 2010, pp. 393–407.

[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2011, pp. 627–638.

[3] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Symposium On Usable Privacy and Security*. USENIX Association, 2012, pp. 3–14.

[4] J. Lin, N. M. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the ACM International Conference on Ubiquitous computing*. ACM, 2012, pp. 501–510.

[5] J. Lin, B. Liu, N. M. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Proceedings of the Symposium On Usable Privacy and Security*. USENIX Association, 2014, pp. 199–212.

[6] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2015.

[7] "Facebook and Cambridge Analytica data breach," https://en.wikipedia.org/wiki/Facebook_and_Cambridge_Analytica_data_breach, accessed: 2018-06-29.

[8] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proceedings of the Symposium On Usable Privacy and Security*. ACM, 2012, pp. 1 – 16.

[9] P. G. Kelley, L. F. Cranor, and N. M. Sadeh, "Privacy as part of the app decision-making process," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 3393–3402.

[10] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *Proceedings of the USENIX Security Symposium*. USENIX Association, 2013, pp. 527–542.

[11] J. Tan, K. Nguyen, M. Theodorides, H. Negrn-Arroyo, C. Thompson, S. Egelman, and D. A. Wagner, "The effect of developer-specified explanations for permission requests on smartphone user behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 91–100.

[12] "Requirements elicitation," https://en.wikipedia.org/wiki/Requirements_elicitation, accessed: 2018-06-29.

[13] "Software requirements specification," https://en.wikipedia.org/wiki/Software_requirements_specification, accessed: 2018-06-29.

[14] "GeoTimer Lite," https://androidappsapk.co/detail-geotimer-lite/, accessed: 2018-06-29.

[15] "Set up for geofence monitoring," https://developer.android.com/training/location/geofencing, accessed: 2018-06-29.

[16] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1994, pp. 232–241.

[17] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the International Conference on Software Engineering*. ACM, 2014, pp. 1025–1035.

[18] "NLTK language toolkit," https://www.nltk.org, accessed: 2018-06-29.

[19] "CLAP project website," https://sites.google.com/view/clapprojsite/, accessed: 2018-06-29.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*. Morgan Kaufmann, 2013, pp. 3111–3119.

[21] "word2vec," https://code.google.com/archive/p/word2vec/, accessed: 2018-06-29.

[22] "Stack Overflow: How do I get the current GPS location programmatically in Android?" https://stackoverflow.com/questions/1513485/how-do-i-get-the-current-gps-location-programmatically-in-android, accessed: 2018-06-29.

[23] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. The Association for Computational Linguistics, 2003, pp. 423–430.

[24] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "AutoCog: Measuring the description-to-permission fidelity in Android applications," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1354–1365.

[25] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. The Association for Computational Linguistics, 2003, pp. 173–180.

[26] C. Zhai and J. Lafferty, "Model-based feedback in the language modeling approach to information retrieval," in *Proceedings of the International Conference on Information and knowledge management*. ACM, 2001, pp. 403–410.

[27] G. Salton, A. Wong, and C. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[28] "Majority rule," https://en.wikipedia.org/wiki/Majority_rule, accessed: 2018-06-29.

[29] "Inverse document frequency," https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html, accessed: 2018-06-29.

[30] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. The Association for Computational Linguistics, 2004, pp. 404–411.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Stanford University, Tech. Rep., 1999.

[32] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google Play," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2014, pp. 221–233.

[33] "Permission groups," https://developer.android.com/guide/topics/permissions/overview#perm-groups, accessed: 2018-06-29.

[34] "AlarmMon app," https://play.google.com/store/apps/details?id=com.malangstudio.alarmmon, accessed: 2018-06-29.

[35] "WHYPER dataset," https://sites.google.com/site/whypermission/home/results/, accessed: 2018-06-29.

[36] "Jaccard index," https://en.wikipedia.org/wiki/Jaccard_index, accessed: 2018-06-29.

[37] "Python T-test," https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.ttest_ind.html, accessed: 2018-06-29.

[38] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable decision sets: A joint framework for description and prediction," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1675–1684.

[39] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? A case study on free Android applications," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*. IEEE Computer Society, 2015, pp. 301–310.

[40] A. K. Massey, J. Eisenstein, A. I. Antn, and P. P. Swire, "Automated text mining for requirements analysis of policy documents," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2013, pp. 4–13.

[41] J. Bhatia, T. D. Breaux, and F. Schaub, "Mining privacy goals from privacy policies using hybridized task recomposition," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 3, pp. 1–24, 2016.

[42] M. C. Evans, J. Bhatia, S. Wadkar, and T. D. Breaux, "An evaluation of constituency-based hyponymy extraction from privacy policies," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2017, pp. 312–321.

[43] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2014, pp. 153–162.

[44] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher, "Supporting domain analysis through mining and recommending features from online product listings," *IEEE Transactions on Software Engineering*, vol. 39, no. 12, pp. 1736–1752, 2013.

[45] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proceedings of the Working Conference on Mining Software Repositories*. IEEE Computer Society, 2012, pp. 108–111.
[46] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2013, pp. 125–134.
[47] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2013, pp. 582–591.
[48] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2015, pp. 116–125.
[49] T. Johann, C. Stanik, A. M. A. B., and W. Maalej, "SAFE: A simple approach for feature extraction from app descriptions and app reviews," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2017, pp. 21–30.
[50] A. I. Antón and J. B. Earp, "A requirements taxonomy for reducing web site privacy vulnerabilities," *Requirements Engineering*, vol. 9, no. 3, pp. 169–185, 2004.
[51] J. Bhatia and T. D. Breaux, "A data purpose case study of privacy policies," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2017, pp. 394–399.
[52] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. M. Sadeh, S. M. Bellovin, and J. R. Reidenberg, "Automated analysis of privacy requirements for mobile apps," in *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2017.
[53] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in Android application code," in *Proceedings of the International Conference on Software Engineering*. ACM, 2016, pp. 25–36.
[54] H. Wang, J. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 1107–1118.
[55] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for Android apps," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 518–529.
[56] Y. Li, Y. Guo, and X. Chen, "PERUIM: Understanding mobile application privacy with permission-UI mapping," in *Proceedings of the ACM International Conference on Ubiquitous computing*. ACM, 2016, pp. 682–693.