Understanding Privacy Awareness in Android App Descriptions Using Deep Learning

Johannes Feichtner
Graz University of Technology
Secure Information Technology Center – Austria (A-SIT)

Stefan Gruber
Graz University of Technology

ABSTRACT

Permissions are a key factor in Android to protect users' privacy. As it is often not obvious why applications require certain permissions, developer-provided descriptions in Google Play and third-party markets should explain to users how sensitive data is processed. Reliably recognizing whether app descriptions cover permission usage is challenging due to the lack of enforced quality standards and a variety of ways developers can express privacy-related facts.

We introduce a machine learning-based approach to identify critical discrepancies between developer-described app behavior and permission usage. By combining state-of-the-art techniques in natural language processing (NLP) and deep learning, we design a convolutional neural network (CNN) for text classification that captures the relevance of words and phrases in app descriptions in relation to the usage of *dangerous permissions*. Our system predicts the likelihood that an app requires certain permissions and can warn about descriptions in which the requested access to sensitive user data and system features is textually not represented.

We evaluate our solution on 77,000 real-world app descriptions and find that we can identify individual groups of *dangerous permissions* with a precision between 71% and 93%. To highlight the impact of individual words and phrases, we employ a model explanation algorithm and demonstrate that our technique can successfully bridge the semantic gap between described app functionality and its access to security- and privacy-sensitive resources.

KEYWORDS

Android, Machine Learning, Description, Permission, NLP, CNN

ACM Reference Format:
Johannes Feichtner and Stefan Gruber. 2020. Understanding Privacy Awareness in Android App Descriptions Using Deep Learning. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3374664.3375730

1 INTRODUCTION

Google Play and third-party markets enable Android users to choose from a vast amount and diversity of apps. For a better overview, markets organize apps in categories, outline their purpose in a one-sentence summary, and upon selection provide a more extensive description, screenshots, and the set of permissions an app requests to be granted. Based on this metadata, users make a decision on whether an app seems trustworthy enough to install and use it.

To meet users' expectations and to set adequate boundaries of behaviors, Android protects privacy-critical device functionality. Whenever an app requires access to the camera, microphone, or other sensitive features, users are requested to grant the according permission. Since Android 6, permissions have been reorganized into groups, whereas *dangerous permissions* comprise all those with access to critical device features and users' personal data.

Recent studies [22, 25] point out that many users cannot assess the security risk associated with granting permissions. Although it may seem intuitive that a messenger application requires access to a user's contacts, it may not be obvious why the same app also requests the permission to fetch the current GPS position. The situation is aggravated by the fact that malware and privacy-invasive applications also infamously claim more permissions than their functionality warrants [4, 20]. Despite recurring attempts to improve how meaning and purpose of permissions are presented to users, permission requests often remain incomprehensible.

Making access to sensitive user data and device features transparent to users is of utmost importance to prevent unknowing or unconscious leaks of personal data. Therefore, the descriptions of Android apps should imply the usage of dangerous permissions and highlight how and why they are required. Unfortunately, in practice, descriptions are often minimal, inaccurate, and lack statements about security-related aspects overall. Hence, the motivating questions in this paper are: *(1) How does an app description reflect privacy-related permission usage?* and *(2) What is the relevance of individual words, word groups, and phrases?* By proposing a system that helps to answer these questions, we aim to assess and improve privacy awareness in app descriptions.

The results of prior work in this direction confirm the existence of correlations between permission usage and individual word combinations [19, 23]. A viable approach to relate them would be to apply established NLP techniques for sentence analysis. However, the arbitrariness of real-world app descriptions and the lack of enforced quality standards impede a conclusive extraction of semantic information. The constantly evolving nature of smartphone apps with frequently changing descriptions also make it cumbersome to constrain text-permission-relationships to a limited set of semantically similar vocabulary. The key challenge, thus, is to find an approach that works reliably with noisy, real-world app descriptions and can map privacy-related text to permission usage.

We propose an innovative method to identify semantic correlations between text and permission usage within real-world Android

CODASPY '20, March 16–18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3375730

applications. Our incentive is to develop a system[1] that can infer permission usage from the functionality described in text fragments. Therefore, we design a deep neural network to reliably assess and grade the need for permissions by analyzing noisy, potentially incomplete, and inaccurate description texts. For each permission, a score is delivered that indicates the likelihood that a certain permission might be required. To remediate the black box character usually associated with deep learning, we require that our solution provides an insight into which words are causative for individual scores. Comparing obtained predictions with the set of actually requested permissions enables users, developers, and markets to draw conclusions about privacy-relevant aspects in app descriptions.

Contributions. Our key contributions are as follows:

- *Linking permission usage to app descriptions.* We design a convolutional neural network (CNN) to identify sample-based correlations between parts of the description text and the permission groups an app requests. Our solution overcomes various limitations present in existing research and can work effectively with unlabeled, potentially flawed descriptions of real-world Android applications.
- *Extracting semantic knowledge from app descriptions.* We study the expressiveness of textual properties in app descriptions and employ an embedding model to capture semantic links between words, word groups, and phrases. Within our evaluation, we compare the performance of two state-of-the-art NLP techniques word2vec [18] and GloVe [21] in their role as an input preprocessor for our deep neural network.
- *Evaluation.* We train, validate, and test our neural network with more than 77,000 descriptions and sets of permission usage from Google Play. We evaluate the prediction quality for each group of *dangerous permissions*. In a case study, we demonstrate that our system can successfully identify text fragments that point to individual permissions, identify missing permission explanations, and also reveal non-intuitive links between permissions and text phrases.
- *Explaining Predictions.* To assess the quality of our network and to avoid incomprehensible black box predictions, we employ the model explaining algorithm LIME [24]. We calculate a score for each word that shows a significance for the output. Visualized as heatmaps, our solution can highlight the influence of particular words in description texts with regard to a predicted permission group.

Outline. In Section 2, we discuss related work. Section 3 explains selected aspects of NLP and neural networks. Section 4 introduces our solution to assess the occurrence of permission-related explanations within app descriptions. In Section 5, we explain how we preprocess tuples of description text and permission sets. Followed by that, Section 6 elaborates on the convolution neural network we propose. We evaluate our solution on real-world apps and present a case study in Section 7. In Section 8, we conclude this work.

2 RELATED WORK

Aligning permission usage with the alleged functionality of Android apps has become a growing field of research. In the following, we present related work and point out differences to our solution.

[1]Our implementation is available at: https://github.com/sg10/android-privacy

Android Permissions. To validate whether permissions are indeed used within apps, Pandita et al. [19] parse descriptions using an NLP parser and build semantic graphs of API calls associated with permissions. Based on the names of classes and methods, keywords are identified and compared with tokens in the semantic graph. A similar approach is presented by Qu et al. [23], who perform a sentence analysis, identify textual patterns based on frequency analysis, and correlate them with permissions. The output can finally be used to annotate privacy-relevant sentences in the description. Wang et al. [30] propose an assistance system to determine the minimum set of permissions required according to keywords in the app description. Likewise, Taylor et al. [26] show a framework to suggest functionally-similar but less privacy-invasive permissions.

Code Analysis. In a combination of static code inspection and text analysis, Watanabe et al. [31] present a keyword-based technique to correlate access to privacy-relevant resources with app descriptions. With a focus on potential abuse of sensitive APIs, Gorla et al. [10] derive app clusters based on pre-labeled description topics. Related to that, the approach of Gao et al. [8] infers expectable permissions by applying statistical correlation coefficients after mining topics from descriptions using NLP techniques and Latent Dirichlet Allocation (LDA). Similar to our solution, the authors address the usage of *dangerous permissions* and derive a score for each group to assess whether a declared permission has a close relevance with the app's presumed functionalities. As topics are derived from words that frequently occur together, the absence of trigger keywords and the lack of further contextual information may provoke false negatives.

Designed to generate descriptions for security-relevant implementation parts, the approach of Zhang et al. [34] evaluates a graph-based app representation. To recognize critical app behavior at runtime, Diamantaris et al. [6] hook calls to privacy-relevant APIs and inspect security-critical parameter values via backtracking.

Machine Learning. Kong et al. [13] and Wu et al. [33] propose to predict security-related app behavior by supplying the words of user reviews to Support Vector Machines (SVM). To infer how specific permissions are used in code, Wang et al. [28] apply text analysis and different supervised classifiers on a manually labeled set of 622 apps. They perform taint tracking using a customized version of TaintDroid [7] and recently extended [29] their approach to also handle obfuscated code as it is often [32] found in Android apps. McLaughlin et al. [17] interpret source code analysis as a form of textual processing and design a convolutional neural network (CNN) that captures semantic information from opcodes in Dalvik bytecode to detect malware. Also targeted at finding malware sequences in Android apps, Huang et al. [11] propose to organize bytecode as images and learn them in CNNs. Both approaches highlight the efficiency of CNNs to detect contextual patterns in a large amount of training data.

Comparison with our work. Among all related research, the work of Qu et al. [23] comes closest to this paper. However, instead of correlating single permissions to frequently occurring text fragments, we perform a contextual text analysis using state-of-the-art techniques word2vec [18] and GloVe [21] and, as introduced with Android 6, focus on groups of *dangerous permissions*. By applying convolutional filters on words, word groups, and phrases, our

solution can capture and model inherent dependencies between text fragments and permissions. In addition, our work is the first attempt to extend deep learning techniques to predict app permissions based on description sentences. As related work on Android malware detection using CNNs exhibited promising results, our solution confirms that this type of deep neural network is also very well-suited to infer *dangerous permissions* from real-world app descriptions with high precision.

3 BACKGROUND

In this section, we present background information about word embeddings, CNNs, and the model explanation algorithm LIME.

3.1 Word Embeddings

Vector space models, in particular word embedding models, support NLP tasks with memory-efficient and arithmetically meaningful word representations. In the following, we briefly introduce the idea behind embeddings and present the approaches we rely upon in our work, namely Word2vec and GloVe.

A trivial form of word representation for machine learning tasks is *one-hot-encoding*: a vector of zeros, where one cell is set to 1, with the index corresponding to a word. This sparse encoding has two disadvantages. The memory allocated for a full document sample scales both with the dictionary size and the input length. Systems that subsequently process these one-hot vectors necessitate operations and internal parameters for the whole dictionary, despite most input vector entries being zero. Additionally, one-hot encoding does not utilize the semantic relationship between words, e.g., "ship" to "ships" is as dissimilar as "cat" to "car". Statistical neural models that learn the probabilistic distribution of words have been proposed already 20 years ago to tackle this curse of dimensionality issue [5]. By transforming one-hot vectors to data points (embeddings) in a continuous vector space, the dictionary can be processed much more memory-efficient and allows for arithmetic operations.

3.1.1 Word2vec. Mikolov et al. [18] proposed an embedding technique that calculates the probabilistic distribution of words via skip-grams. Based on a single word, the estimator predicts the words to most likely occur around it. The model is designed so that at one point in the chain of calculations, a low-dimensional vector contains all the information to make a prediction. This dense vector is then stored as the embedding for the corresponding input word.

The estimator model itself is a neural network and requires a large amount of training data. Depending on the used training set, properties of the embeddings may vary and, thus, numerous pre-trained word embeddings can be used out-of-the-box as dictionaries[2]. Models are typically trained until the average prediction quality of the surrounding words plateaus.

3.1.2 GloVe. In contrast to Word2vec, GloVe [21] is a count-based embedding model. While Word2vec uses a predictive approach, GloVe creates word embeddings based on co-occurrence: a large matrix X is used where each cell $X_{i,j}$ indicates how often word i occurs in the local context of word j (n-gram). In order to model the occurrence probabilities as dense vectors, Pennington et al. propose a simple weighted least squares regression model instead

[2]https://code.google.com/archive/p/word2vec/

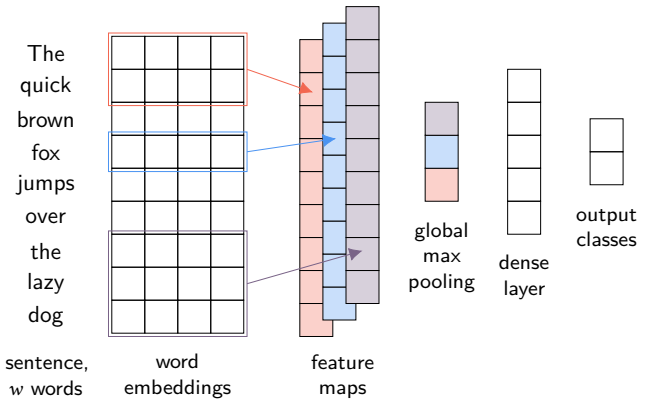

Figure 1: CNN for text classification by Kim et al. [12].

of a neural network. According to the authors, their embedding model performs similar to Word2vec in general, but also show that it can outperform predictive models for specific tasks. As there is no general rule to decide whether to employ a count-based or a probability-based embedding model, the performance of both has to be evaluated with regard to a specific task.

3.2 Convolutional Neural Networks

Fully-connected, dense neural networks are inherently limited by their processing capability. For dense layers, each neuron within a layer is connected to every neuron of the adjacent layers. This dense connectivity assumes that the input data follows a constant structure, where input features always have a fixed global position. For certain domains, like image, text, or speech, utilizing local properties makes more sense. Convolutional neural networks (CNN) [14] offer a solution to address these shortcomings.

The architectural changes between fully-connected networks and CNNs are motivated to achieve three properties [9, p. 329–339]:

(1) **Sparse interactions**: A filter with a kernel smaller than the whole input is windowing the input data. This leads to fewer parameters in memory and also less computing operations.
(2) **Parameter sharing**: A learned pattern can be used for multiple inputs; it is not tied to a single one.
(3) **Equivariant representations**: Although the detection of patterns is not strictly limited to input regions, e.g., image position, a translation is still detectable in the output.

In the concept of CNNs, several *filters* are applied to the input to learn spatial properties, e.g., specific small arches for an image object. The convolution operations of all filters are intermediately stored in *feature maps*. A *pooling layer* acts as a subsampling layer for the feature maps. It effectively reduces the number of parameters by taking the max or average value of several feature map entries. In many state-of-the-art CNNs, multiple combinations of filter kernels / pooling layers are stacked on top of each other, leading to a deep convolutional architecture. One or more *dense layers* commonly flatten the pooled values and calculate the ultimate classification or regression output.

Although originally developed for image processing, CNNs have shown to be useful for NLP as well. Text processing has intuitively been tackled by recurrent neural networks (RNN), which have

been designed to process sequences of data. Due to their recursive structure, RNN training cannot be parallelized as efficiently as for CNNs, leading to longer training times. Moreover, CNNs do well in tasks where positional features have to be extracted, which is a useful characteristic of human language as well.

CNN architectures for text classification have been proposed both on character-level [35] and sentence-level [12]. In our work, we extend the sentence classification network by Kim et al. that is depicted in Figure 1. First, input words are translated to pre-trained embeddings and concatenated to form a matrix. Then 1-dimensional convolutions are calculated on the input matrix and the per-filter output for the whole input is stored in a feature map. The maximum value per feature map (global max pooling) is passed on to a dense layer. As a network output, a multi-class classification is learned.

The model uses different filter sizes in order to learn the impact of a single word as well as short word combinations. In practice, there are often multiple filters of one kernel size, so that numerous words or combinations can be captured. Word embeddings allows for better generalization of the filter kernels. The subsequent max-pooling layer finds high-impact input features and global max-pooling allows the dense layer to make a classification based on the whole sentence by combining single words from local contexts.

3.3 Model Explanation

The essence of machine learning (ML) is finding patterns via function approximations in a given set of data. The inherent underlying problem is that based on the model output only, it is not always obvious why models make certain predictions. As a remedy, in the recent past, model explanation algorithms, such as LIME [24] and SHAP [16] were developed to better understand how decisions are made. In this paper, we leverage LIME to find words in app descriptions that were causative for the prediction of permissions.

LIME is a method proposed to give local, model-agnostic explanations. Model agnosticism means that the ML model is treated as a black box, i.e., LIME does not know about the inner workings of the model. Locality means that it works based on a particular sample which is slightly varied to obtain new samples, one by one. These samples form a local context around the original sample. The impact of each modification causes different model outputs. In the end, all differences are aggregated to show which properties of the input are most influential for the original sample's prediction.

LIME has an implementation in Python that modifies text or image samples systematically, feeds them to an arbitrary trained ML model, and visualizes the impacts[3].

In general, model interpretability is only a loosely defined concept [15]. There is no formal verification approach to prove that a neural network always works as it is designed. The claim that linear models are more comfortable to explain than deep neural models has not yet been proven mathematically–it is merely easier for humans to see causality. Although deep ML models have yielded astonishing results recently, their complex inner structure requires additional effort to build trust in them. Despite that the field is in its early stages, current explanation methods like LIME and SHAP can give useful information about a model's operating principles when we analyze app descriptions, permissions, and app internals.

[3]https://github.com/marcotcr/lime

4 SYSTEM OVERVIEW

We design a deep neural network to find relationships between text and permissions within a large dataset of real-world Android applications. Our goal is to develop a system that can process a given description text in human language and based on this text, deliver a score for each permission. A better score corresponds to a higher likelihood that an app requests a certain permission. In order to remediate the "black box" that is usually associated with deep learning, we require that our solution provides an insight into which parts of description texts are causative for individual scores. A comparison of obtained predictions with the set of actually requested permissions enables users, developers, and markets to draw conclusions about privacy-relevant aspects in app descriptions.

The primary functionality of our system can be split into two parts: In the **training** phase, our network learns correlations between parts of description texts and actual permissions of apps. Using backpropagation, our convolutional neural network adjusts its internal parameters to predict the app's permissions correctly. Training continues until the performance plateaus or decreases, i.e., the quality of our permission scores does not improve any further. The model giving the best performance in training is then used for testing. In the **prediction** phase, our system receives app descriptions not seen during training. As depicted in Figure 2, the trained network outputs scores indicating whether a given description reveals hints about particular permissions. To estimate which words in the descriptions have an impact on permission scores, we leverage the model explanation algorithm LIME (see Section 6.3).

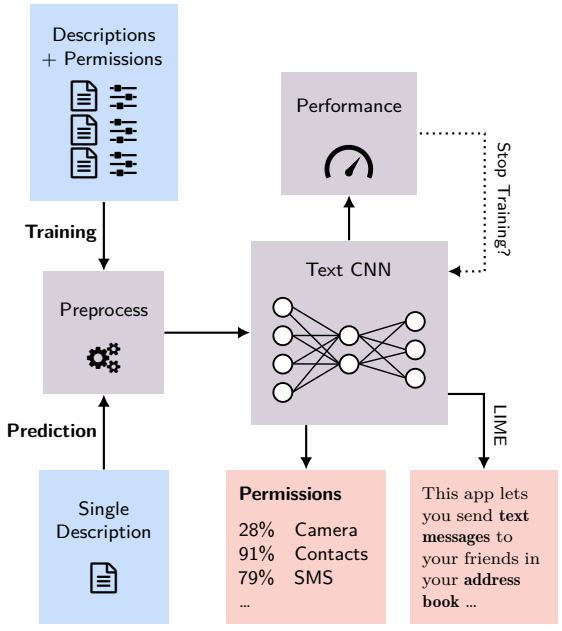

Figure 2: Training on combinations of app descriptions and permissions to eventually predict permissions based on the text only.

5 FEATURE PREPROCESSING

Before training a neural network, it is crucial to prepare the data for efficient learning. In the following, we cover the preprocessing steps that are applied to developer-provided app descriptions and the permission sets after extracting them from Android applications.

5.1 Descriptions

For our network to model the relation of individual words in the description with permissions, we first need to find a way to transform descriptions consisting of character arrays into tokens. In order to capture co-occurrences of words, we have to preserve the word order and cannot use a bag-of-words approach. With the use of a pre-trained word embedding model, we translate the tokens to embedding vectors using a trial-and-error lookup strategy and, thereby, receive dense matrices of a fixed size for each description.

The pre-trained embedding model consists of key/value dictionaries, with the key being a word or word combination, and the value being the corresponding embedding vector. Many embedding models are not only trained on alpha-numeric tokens, but also contain entities like hyphenated expressions, web addresses, or brands and trademarks (including special characters).

The tokenization strategy, thus, has to adapt to the available dictionary keys. We show our approach to this problem in Figure 3. Initially, we strip the description of all formatting done with HTML tags. Thereby, we receive a raw description string. For splitting the string into tokens, we analyze the character set of our pre-trained embeddings. A set of all particular characters available in the dictionary's keys is created, which we refer to as the alphabet. The keys, i.e., the words and word groups, mainly consist of alphanumeric ASCII characters but also include non-alphanumeric and Unicode characters. After obtaining the full alphabet, we use its inverse as a delimiter for tokenization. Consequently, the raw string is split by every character that does not occur in the alphabet, which leaves us with an initial set of tokens.

After acquiring this set of tokens, we initiate the lookup process. For each token in the initial set, we perform the following steps: First, we check whether the full token exists in the dictionary. If it does, we use the embedding e_i corresponding to the entry (word) at dictionary position i. If not, we attempt to clean and split it. Then, we remove the leading and trailing non-alphanumeric characters and perform another lookup. In case of another dictionary-miss, we split the cleaned token into subtokens, with all remaining non-alphanumeric as delimiters. We perform separate lookups for all subtokens. Our goal is to match as many tokens against the pre-trained embedding as possible.

A limitation of the token number, i.e., the description text length to be processed by the network, is necessary. The reason is that the convolutional neural network can only handle a fixed-size input. Descriptions practically have arbitrary length, and thus, an upper limit has to be set. We determine the maximum description length empirically and define a value that allows for 95% of descriptions to be processed as until the last token. For texts larger than that, we truncate them, and for the ones shorter, we add padding tokens. Ultimately, we obtain a matrix of constant size with the row indices corresponding to the description tokens, and each row containing an embedding vector from the pre-trained word embedding model.

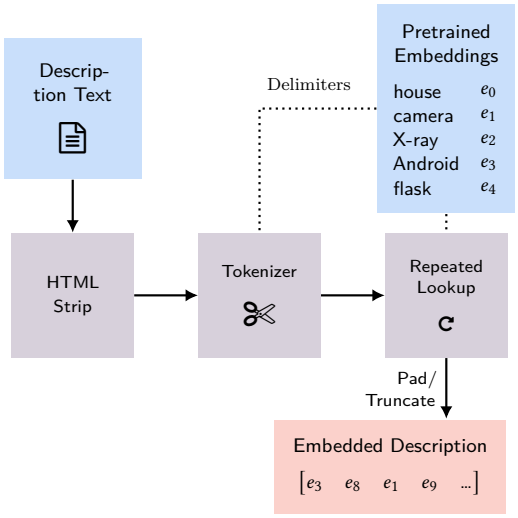

Figure 3: Iterative tokenization and embedding lookup for each word contained in the description of an Android app.

The matrix is a transformed representation of an app's description that can be used as an input for a machine learning model.

5.2 Permissions

Introduced with Android 6, the protection level[4] of *dangerous permissions* comprises all API calls with access to private information and sensitive device features. Considering their critical role and high impact on users' privacy, we focus on permissions included in this group and disregard other protection levels. As apps declare requested permissions by means of listing predefined string identifiers, we need to represent these values in a way that a neural network can process them.

Every Android application includes a *AndroidManifest.xml* file that denotes requested permissions within `<uses-permission>` tags. We extract all permission identifiers, filter for *dangerous permissions* and assign them into groups[5]. This leaves us with nine permission groups: `CALENDAR`, `CALL_LOG`, `CAMERA`, `CONTACTS`, `LOCATION`, `MICROPHONE`, `PHONE`, `SMS`, and `STORAGE`.

The nine permission groups are subsequently transformed to multi-hot encoded vectors. For each app, we check whether it requests one or more single permissions within a group. If it does, then the corresponding column of the vector is set to 1, and 0 otherwise. As a result, we obtain a 1-by-9 binary vector that portrays the privacy-critical permissions an app makes use of. We employ these vectors as targets within our machine learning model.

6 MODEL CONSTRUCTION

We propose a convolutional neural network (CNN) to predict the likelihood that an Android app requires a certain permission. By applying different filters, we capture the local properties of individual words, word groups, and phrases and apply them within a multi-label classification task, in which each label corresponds to a permission group. In this section, we highlight the advantages of

[4]https://developer.android.com/guide/topics/permissions/overview
[5]https://developer.android.com/reference/android/Manifest.permission.html

CNNs for our problem and present the architecture of our network with regard to the set of chosen filters, layers, and hyperparameters.

Unlike recurrent architectures, e.g., LSTM cells, CNNs are faster to train and more straightforward to interpret. The convolutional architecture captures the impact of single words, word groups, and phrases. If we only wanted to examine the existence of certain words or set of words, machine learning would not be required. However, CNNs do not only find local patterns but also interpret their co-occurrence. In our model, we leverage this feature to determine whether certain words occur together and assess how relevant each of them is regarding the use of certain permissions.

Moreover, the concept of CNN filters allows for generalization. As we use word embeddings, the 1-d filters of our network do not remember single words. Instead, a particular filter ideally captures groups of words with similar properties in the word embedding subspace, e.g., the semantically related words "photo", "picture", and their plurals. These properties fostered our decision to use CNNs for this multi-label text classification problem.

6.1 Architecture

As a basis, we employ the text CNN proposed by Kim et al.[12] and augment it for our task. Figure 4 presents our full neural network in detail. The network input is a $d \times m$ matrix, where d stands for the embedding size and m for the maximum number of tokens (embeddings) we process per description. The network then applies 1-d convolutions with different kernel sizes: 1, 2, and 3, i.e., filters that can cover one to three words at a time. For each of these three sizes, we allocate 1024 filters to be trained. Each of the three filter bundles is subsequently reduced through a global max-pooling layer. We are left with $3x1024$ max-values, which are concatenated so they can be processed by the successive 1-d dense layers. For regularization, we apply a dropout of 0.2 after each of the two dense layers, meaning 20% of the weight connections are chosen randomly and set to zero in each training iteration. The output layer, consisting of nine neurons that match the nine permission groups, finally returns values between 0 and 1 for each of them. Table 1 summarizes our choice of additional hyperparameters. To find good parameters for the number of layers, dense neurons, and dropout we used grid search. We chose a final architecture based on the best-measured performance.

Table 1: Hyperparameters used within our CNN for the perception of Android app permissions.

Filters	Kernels: 1x1, 1x2, 1x3 (1024 each) Padding: Same Stride: 1
Hidden Dense Layers	5000 Neurons, 2500 Neurons
Optimizer	Adam, $\eta = 0.0001$
Batch Size	32
Weight Initialization	Glorot (Uniform)
Loss Function	Binary Cross-Entropy
Hidden Activations	ReLU
Final Activations	Sigmoid
Early Stopping	Patience: 6 epochs Delta: 2% F_β-score (macro)

6.2 Model Training

Designed to maximize accuracy and reduce error, our network performs best when the number of samples in each class are about equal. However, in real-world Android apps permission usage is not equally distributed. As there are in practice, e.g., by far more apps that request access to a user's location than to the microphone, class imbalance occurs. As a countermeasure, we weigh the loss function and, as subsequently described, scale individual performance metrics. Moreover, we split our dataset into a training, validation, and test set. This allows us to adopt early stopping and to find a good final training state that prevents both over- and underfitting.

6.2.1 Class Imbalance. In our system, the unequal distribution of labels among apps results from differences in permission usage. The grouped permissions of an app, as well as their co-occurrence, heavily depend on the intended app purpose. Some groups tend to occur together more often. Additionally, some permissions are required by fewer apps than others. As we have nine network outputs, i.e., permission groups, which can occur in any combination, the impact of class imbalances is even higher than in other domains.

We aim to partially even out the imbalance by using class weights, as shown in Equation 1. To this end, we first need to find the actual distribution. We add up the binary 1-by-9 permission vectors $\mathbf{p}_i$ of all samples to obtain a sum vector $\mathbf{s}$. We divide the inverse of the sum vector by the total number of samples. We also normalize the vector via dividing by 9, so that the non-weighted loss and the class-weighted loss can be compared directly. We receive a 1-by-9 vector $\mathbf{c}$ that counter-balances the overall class distribution. It is multiplied with the actual value of the loss (error value) that is back-propagated during training for every single output. Consequently, for a class with lower sample support, the extent of the error is increased and decreased for higher support. Although this balancing mechanism increases the importance of particular samples over others, it reduces the magnitude of class-dependent overfitting.

$$\mathbf{p}_i = \begin{bmatrix} p_{i,0} & p_{i,1} & \dots & p_{i,8} \end{bmatrix} \qquad \text{for } 0 \leq i < N_{\text{samples}}$$
$$\mathbf{s} = \sum_{N_{\text{samples}}} \mathbf{p}_i \qquad \mathbf{c} = \frac{N_{\text{samples}}}{\mathbf{s}} \cdot \frac{1}{9} \tag{1}$$

As a fundamental requirement, our system must be able to work with descriptions of real-world Android applications. This implies that parts of the texts may not contain sufficient information for each permission group. For these samples, humans cannot identify permission usage in the description, and thus, the machine learning model should not attempt to find meaningless correlations. This obstacle, in general, is challenging to tackle, as the training set does not contain particular information of a description's quality. We at least want to address this problem through a weighed F-score and use the β parameter. We set it to 0.5, which causes the precision to be treated as more important than the recall. Thereby, we prefer models that make more confident predictions.

While the previously described class-weighing approach focuses on the loss function, i.e., the back-propagation of the error during training, class imbalance must be taken into account for performance metrics as well. We use precision, recall, and F_β-score to measure the performance for each class. For the sake of comparing

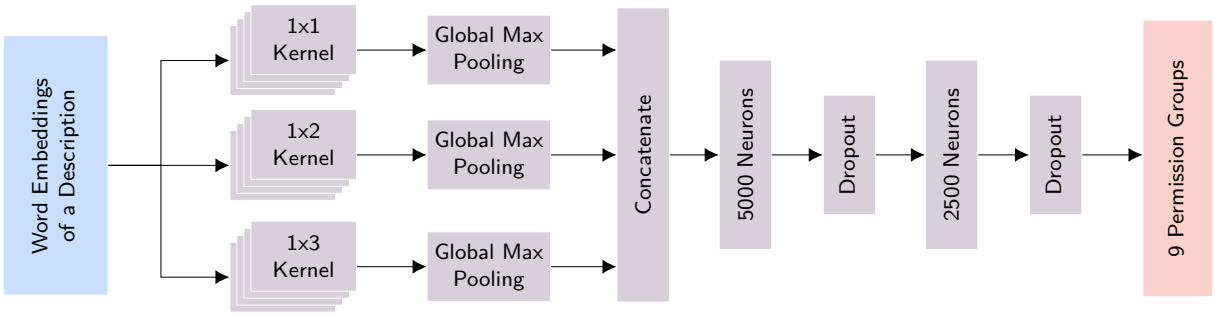

Figure 4: CNN architecture to process (embedded) descriptions and get the permission likelihood.

different architectures and training states, a single numeric value for a trained model is required. Thus, we calculate macro and micro averages over all classes. For comparison, we use the macro F_β-score s it factors in the class imbalance through the support of true positives per class.

6.2.2 Overfitting. As a measure to prevent overfitting, we apply early stopping using a distinct set of training, validation, and test data. Most samples are put into the training set, which is used for back-propagating the error. The performance on the validation set is calculated after every epoch to assess how well the model generalizes. The test set is not used during training. All set affinities are decided randomly, and the order within a set is permuted after every epoch as well. In our solution, we aim to find a good epoch count dynamically by continuously measuring training performance. If it decreases, training continues for six more epochs if another local minimum can be found later. If this is not the case, the weights are restored to the previously best-performing state. This weight configuration also represents the final state of the trained network.

6.3 Explaining Predictions

Besides obtaining the likelihood that certain permission groups are represented within app descriptions, we are interested in the words leading to the predicted permission scores. In order to interpret how our CNN makes a prediction, we use the model explainer LIME [24] to find relevant parts of the input. We calculate a value for each word that shows a significance for the output. Ultimately, we get nine heatmaps for the description, one for each permission group.

LIME treats a machine learning classifier as a black box and evaluates variations of a particular input sample. To use the description of an Android app as input, we first split it into single tokens. The LIME algorithm then generates similar test samples: for each of them, random tokens are removed from the original sample. The impact of each change is then accumulated per output label, i.e., permission group. LIME needs two parameters: (1) the number of features to be varied, which we set to the number of tokens a sample has and (2) the number of random samples to be generated, whereby a higher number computes more combinations. We empirically decided for 100 times as a trade-off between computation time and result accuracy. The algorithm can then calculate the token-related significance of the model's decisions. In case a token occurs multiple times, LIME removes all occurrences. This unique-feature treatment makes sense in our case because of the global pooling layer that merges the information coming from all CNN filters. Eventually, we are left with several impact scores, a list for each permission group, with a sub-list consisting of values for each word in the description regarding that group.

6.3.1 Weighing the Impacts of Words. The impact scores generated by LIME are negative and positive decimal numbers. Depending on the exact model and input, the minimum and maximum scores can have arbitrary absolute values, e.g., -3.7 for a negative impact of a single token on the result and 2.4 for a positive impact. A negative impact means that the removal of a word causes a more confident prediction. Thus, we need to set a lower boundary and normalize the LIME values. Additionally, we need to take the actual prediction of our model into account. For each permission group, our neural network outputs sigmoid values between 0 and 1 that refer to the confidence of a decision. For example, if one class output is close to 0.3, i.e., a confidence of only 30% that an app requires a particular permission, the normalized LIME score also has to be weighed lower in comparison to, e.g., 90%.

$$\widetilde{L}_{m,j} = \frac{L_{m,j}}{\max_k \left(L_{m,k}\right)} \qquad \text{for k ... } 0 \leq j < N$$
$$h_{m,j} = \widetilde{L}_{m,j} \cdot p_m \tag{2}$$

Consequently, we apply a normalization and scaling formula, as shown in Equation 2. For each output label m we have the estimated probability y_m, and N tokens in the description. LIME outputs the impact scores $L_{m,j}$ for each token j and each permission m. First, we zero-out all negative values and then divide each of these impact scores by the maximum value for that sample and label m. Then, these values $\widetilde{L}_{m,j}$ are scaled by the probability y_m for permission m. The result is a heat value $h_{m,j}$ for each word between 0 and the prediction for that permission m.

Hence, $h_{m,j}$ can be used as a heatmap over the text. It shows the relative impact each token has on the CNN's prediction for a specific permission group. A shortcoming of LIME is, however, that it only alters individual, single words. Therefore, we cannot evaluate our filter kernels that cover two and three tokens. Despite this downside, LIME can still give meaningful information about why the model decides for a particular permission group probability. In Section 7.3, we demonstrate how heatmaps can effectively distinguish between good and bad correlations learned by our neural network.

7 EVALUATION

The goal of this evaluation is twofold. First, we investigate the performance of our neural network with real-world Android apps. Second, applying our solution on a hand-picked set of applications, we underline the quality of permission predictions and demonstrate how the impact of individual words can be assessed (see Section 7.3).

7.1 Dataset

For the training and evaluation of our network model, we require a pre-trained word embedding model and sample apps.

Table 2: Subsets of Android apps used as network input.

Apps crawled	115,294 apps
English descriptions	81,803 apps
20+ embeddable tokens	77,758 apps
Training and validation set	76,758 apps
Test set	1,000 apps

7.1.1 Android Applications. We evaluate our approach using real-world applications from the PlayDrone dataset [27]. We opted for this repository of apps as it does not only feature raw app archives but also provides metadata from Google Play, including the app description, category, and download count.

We downloaded 115,294 Android apps and corresponding metadata from the PlayDrone dataset. Subsequently, we filtered apps based on their text embeddability: After preprocessing each description, it had to consist of at least 20 embeddings from the pre-trained model to be used further on. This boundary was set to reduce the potential impact of insignificant samples on the training process.

As highlighted in Table 2, we split the resulting set of 77,758 apps into three subsets. The smallest one, the test set, includes 1,000 randomly chosen apps that are not used during training. The remaining apps are added to the training set, and 20% of them are randomly picked to be also part of the to the validation set. This partitioning scheme is required to prevent overfitting of our machine learning model and to ensure meaningful predictions.

7.1.2 Word Embeddings. As our approach requires descriptions to be provided as word embedding vectors, for performance reasons we leverage a pre-trained model that covers a large set of words in English language. Among the two most common architectures for word embeddings are word2vec [18] and GloVe [21]. The authors of both techniques provide pre-trained models[6][7] that include the text corpora of Wikipedia, news articles, and crawled websites. As summarized in Table 3, both models exhibit substantial differences that might have an influence on the representation of words in our neural network. Before employing word-vector pairs for tokenization (preprocessing) and word representation, we, thus, perform a qualitative comparison between word2vec and GloVe.

We assess the performance of both embeddings models by applying them on all crawled app descriptions. After tokenization and repeated lookups, we find that there is no significant difference between word2vec and GloVe when it comes to harnessing pre-trained embeddings. However, as GloVe yields marginally better values for precision, recall, and F_β-score, we proceed with GloVe and do not consider pre-trained word2vec embeddings any further.

[6] https://code.google.com/archive/p/word2vec/
[7] https://nlp.stanford.edu/projects/glove/

Table 3: Embedding model properties and performance comparison when applied on the descriptions of 81,803 apps. Scores are macro-averaged over all labels and five folds.

	Word2vec	GloVe
Architecture	Predictive	Co-occurrence
Text corpora trained on	Wikipedia, UMBC, and statmt.org	Wikipedia
Total words in corpus	≈ 16 billion	≈ 5 billion
Output embeddings	999,994	400,000
Characters	Lowercase	Lowercase
Precision	76%	81%
Recall	54%	55%
F_β-score	70%	77%

7.2 Results

We evaluated the performance of our CNN with the descriptions and sets of requested permissions for a total of 77,758 apps across training, validation and test set. The test set results, comprising 1,000 apps, can be seen in Table 4, whereby the particular values are averaged independently across the five folds. The last column shows the support, i.e., the number of true positives the model captured. As mentioned, the varying support for each permission group is influenced by the unequal distribution of permission groups in apps. The recall column altogether has lower values compared to the precision column. This can be explained by the fact that we do not have labeled samples but also many low-quality descriptions. Moreover, some descriptions miss permission-related phrases overall, preventing our model from predicting a positive result.

The precision values range between 71% and 93%. The macro precision average is at 77%, and certain permission groups perform significantly better or worse than that. `EXTERNAL_STORAGE` has the highest precision, concurrently with the highest support. While `CAMERA`, `PHONE`, and `CALL_LOG` are on the lower end, `CALENDAR` is worst due to its rare usage. As the scores suggest that some permissions are more clearly identifiable than others, our case study takes a closer look at this assumption and the reasoning of our network.

Table 4: Test set performance, 5-fold average.

	Precision	Recall	F_β-score	Support
`EXTERNAL_STORAGE`	93%	76%	89%	699
`CALENDAR`	55%	24%	42%	36
`CALL_LOG`	70%	44%	62%	108
`CAMERA`	71%	56%	67%	199
`CONTACTS`	80%	65%	76%	369
`LOCATION`	82%	56%	74%	327
`MICROPHONE`	83%	64%	78%	125
`PHONE`	73%	68%	72%	440
`SMS`	83%	52%	73%	100
Macro Average	77%	56%	70%	
Micro Average	81%	65%	77%	

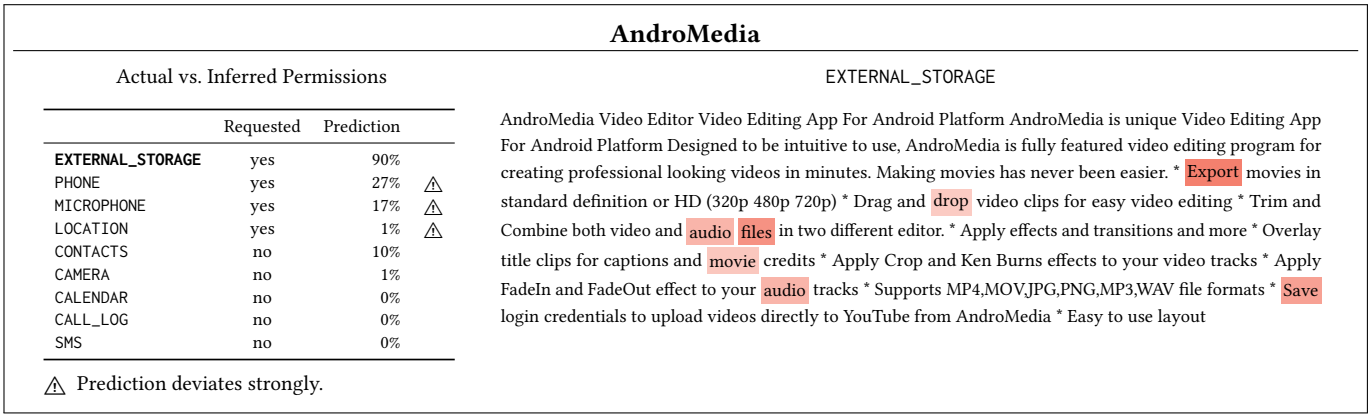

Figure 5: *AndroMedia*: actual vs. inferred permission groups based on the app description. Predictions are explained for the permission group `EXTERNAL_STORAGE` using a LIME heatmap overlay.

Although the dense layer of our CNN makes predictions based on word combinations, frequently occurring words might be indicative for the network's decision process. Assuming that the repetitive occurrence of particular key words has high impact on the prediction of certain permission groups, we apply the model explanation algorithm LIME to identify significant words. Therefore, we gather all words across all app descriptions in our dataset, sort them by how often they occur, by how high their LIME impact sore is and repeat this process for each permission group.

Table 5 presents the most significant tokens per permission group. These words have a comparably high impact for true positives (LIME score >70%) and occur multiple times. The word lists demonstrate that the system indeed captures significant words that make sense for a human reader. Certain correlations are obvious, e.g., the word *calendar* for the `CALENDAR` permission, *map* for the `LOCATION` permission, and *ringtones* for the `PHONE` permission.

Keyword overlaps between groups indicate that writers of app descriptions obviously tend not to distinguish when describing some permission requests. At the same time, it can imply that the implementation of particular functionality typically involves a very precise combination of permissions. Considering the underlying purpose of the `CONTACTS`, `PHONE`, and `CALL_LOG` permissions, such as providing telephone numbers, calling, and storing the record in the call log, a strong inter-relationship seems highly plausible.

Table 5: Recurring words with high impact, according to their LIME score.

`CALENDAR`	calendar
`CALL_LOG`	contact(s), call(s), phone, sms
`CAMERA`	qr, barcode, scanning, photo, flash(light)
`CONTACTS`	contact(s), call(s), ringtone(s), friends, sync
`EXTERNAL_STORAGE`	photo(s), files, sync, mp3, voicemail
`LOCATION`	gps, map(s), near(est), location, weather
`MICROPHONE`	microphone, walkie / talkie, record, voice, calls
`PHONE`	call(s), ringtones, voice, personalized
`SMS`	sms, text, message, call

Our qualitative analysis shows that the reasoning for certain permission groups, such as `CAMERA` and `LOCATION`, can easily be explained due to intuitive links. The good interpretability is supported by frequent textual reflections within these groups. A similar situation can also be observed with the `CALENDAR` and `MICROPHONE` groups, which tend to have a narrower spectrum of use cases and a lower class support, leading to a semantically very unambiguously assigned set of keywords. The `EXTERNAL_STORAGE` permission, in contrast, is located at the other end of this spectrum, as it is the most frequently requested permission and commonly lacks good reflections. Comparable effects exist for the `CONTACTS`, `PHONE`, and `CALL_LOG` permissions that partially share the same trigger words. Correlation-based tendencies of other words are less evident:

- Weather apps tend to require the user's location, which makes *weather* a trigger word for the `LOCATION` permission.
- The `CONTACTS` permission is often used for synchronizing the user's address book with remote services, and also for setting different *ringtones* for single contacts.
- The broadest spectrum of words can be found for the `EXTERNAL_STORAGE` permission. Keywords like *screenshot*, *photos*, or especially *files* are reasonable correlations as writing or reading them mostly requires that permission.

While these less-evident findings are obvious to developers and experienced users, their link to particular permission groups might not be clearly comprehensible for regular users. Certain links between, e.g., *weather* and the `LOCATION` permission exhibit no natural semantic relationship but are entirely the effect of correlation-based learning as it is possible with CNNs.

7.3 Case Study

For a better understanding on how our model operates with real-world descriptions and their qualitative shortcomings, we provide an insight by comparing predicted permissions with actually requested permissions. To obtain knowledge which correlations the model has learned and whether they are meaningful or coincidental, we employ the model explanation algorithm LIME.

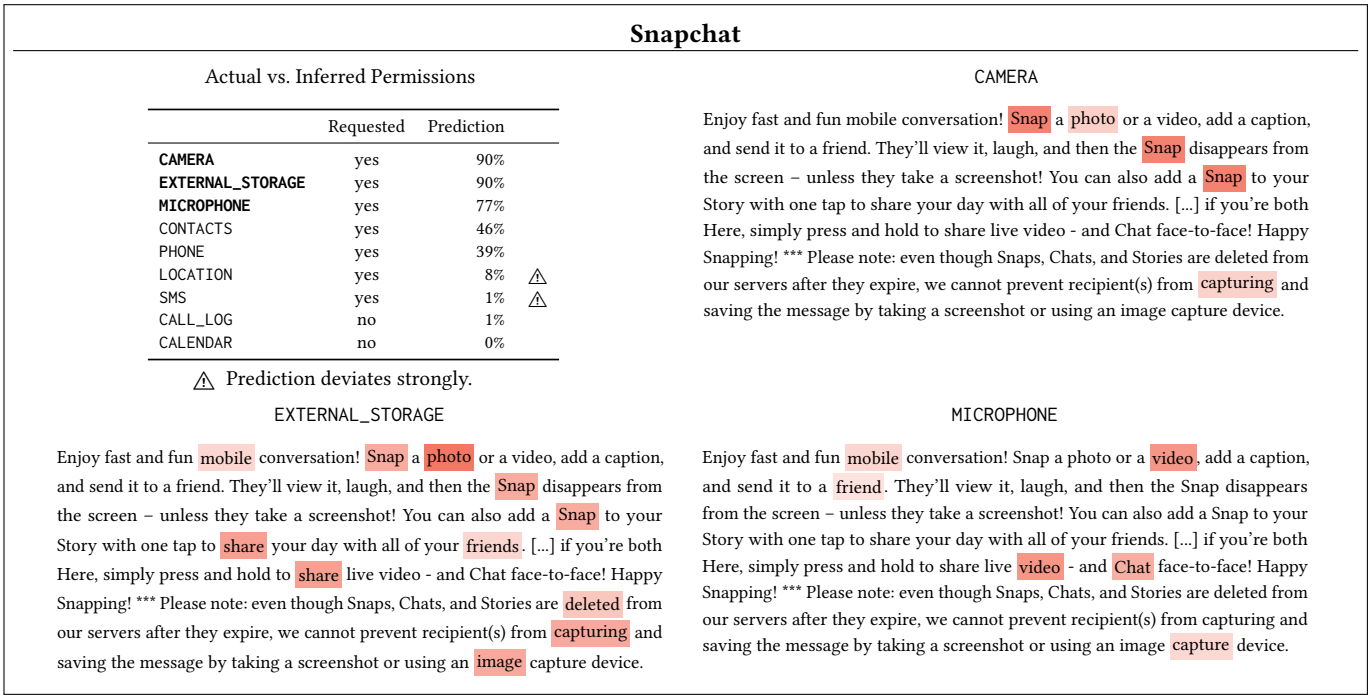

Figure 6: *Snapchat*: actual vs. inferred permission groups based on the app description. Predictions are explained for the permission groups CAMERA, EXTERNAL_STORAGE, and MICROPHONE using LIME heatmap overlays.

For demonstration purposes, we select three real-world apps and highlight interesting findings: the video editing app *AndroMedia* (see Section 7.3.1), the messaging app *Snapchat* (see Section 7.3.2), and the food delivery app *Lieferheld* (see Section 7.3.3). For the sake of brevity and readability, we reduce the presented description texts and show only those sentences that have been identified as relevant for the prediction of permissions.

We repeatedly apply the LIME algorithm for all predictions and words in a description text and derive scores that indicate the individual relevance of words with regard to a prediction outcome. For the text fragments shown in Figures 5-7, we adjust the background color's opacity of all words, such that the values align with the LIME impact scores. A darker color implies a higher impact on the prediction result. As a result, we can determine single words of descriptions that our system deems to be significant, and see how accurate the prediction is compared to the actual permission.

7.3.1 AndroMedia. Designed for the purpose of video editing, we present the evaluation of this app in Figure 5 and highlight words associated with the prediction of the EXTERNAL_STORAGE permission. We selected this app for our case study as it underlines the capabilities of our system to identify privacy-critical discrepancies between a described app purpose and actual permission usage.

As can be seen in the description, key words, such as *export*, *audio*, *files*, and *save* clearly emphasize the need for the EXTERNAL_STORAGE permission. The app further requests the LOCATION, MICROPHONE, and PHONE permissions but does not explain their usage. There is also no apparent practical reason for video editing apps to require these privacy-critical permissions. Although the app could include additional functionality, the three requested but unmentioned permissions should alarm users. Assuming no malicious intent, the developer might be encouraged to augment the description with an explanation how these permissions are used.

7.3.2 Snapchat. This app is designed as a social messenger that allows sending pictures, text messages, sharing one's location and making calls. In Figure 6, we summarize the permissions the app requests and contrast them with the predictions our system makes. To validate the reasoning of our neural network, we highlight the three permission groups CAMERA, EXTERNAL_STORAGE, and MICROPHONE.

For the CAMERA permission, the words *snap*, *photo*, *video*, and *capturing* show high impact. Similar words are important for the EXTERNAL_STORAGE permission, which also relies on the word *share*. This seems reasonable as file sharing often requires access to the phone's storage. High prediction scores of 90% for these two permission groups are, thus, comprehensible. For the MICROPHONE permission, the score of 77% shows to be widely influenced by the words *chat*, *video*, and *capture*. This value would likely be higher if words like *voice* or *audio* would have occurred in the text. Although requested, the SMS and LOCATION permissions cannot be predicted based on the description. The absence of relevant words justifies the stated predictions of 8% and 1%, respectively. The CONTACTS and PHONE permissions are predicted with a score lower than 50%, due to the obvious lack of textual representation. Summarizing, in the description of *Snapchat*, the presence of significant words for CAMERA, MICROPHONE, and EXTERNAL_STORAGE usage can be confirmed, while the text misses semantic indicators that could explain the remaining subset of permissions.

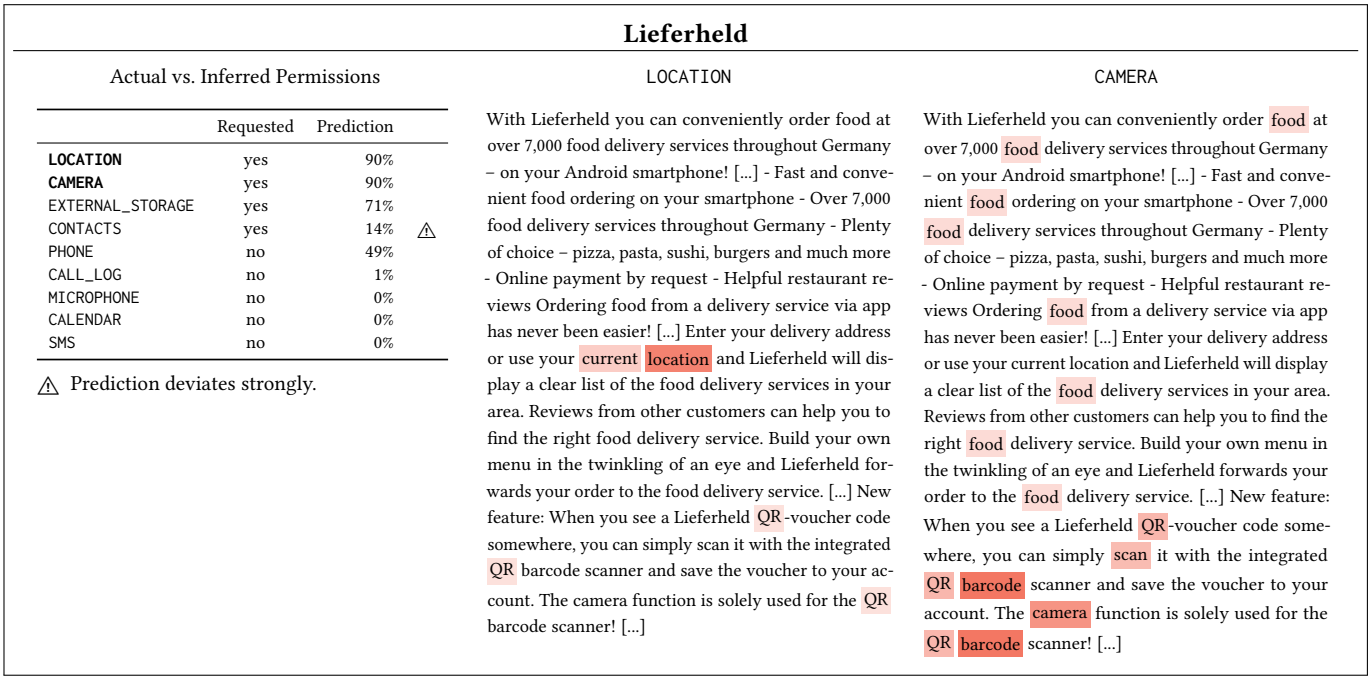

Figure 7: *Lieferheld*: predictions for all permission groups based on the description and LIME heatmap overlay for the `LOCATION` and `CAMERA` permission groups.

7.3.3 Lieferheld. The third app, as depicted in Figure 7, intends to connect users to a food delivery platform. Our system successfully identifies the word *location* in combination with *current* to predict the `LOCATION` permission with a likelihood of 90%. As these words occur next to each other and presumably exist multiple times in the dataset, it is likely that a 1-by-2 or 1-by-3 filter of the CNN has learned this combination. The `CAMERA` permission is mainly triggered by the key words *camera*, *barcode*, and *QR* (*scanner*). This relation reveals that there are many apps in our dataset that mention the words *barcode* and *QR* (*codes*). Most of them tend to require the `CAMERA` permission even more than apps that solely contain the word *camera* in their description. Although the `EXTERNAL_STORAGE` permission was predicted with a confidence of 71%, there are no hints on its usage in the text. We attribute this to the higher co-occurrence likelihood of `EXTERNAL_STORAGE` and `CAMERA`. The `CONTACTS` permission's usage was also not reflected in the description. Summarizing, by our analysis of this app, we can confirm the sufficiently good description of certain permissions but also identify an entire lack for others. Their absence may have security-critical implementations and, thus, developers might be advised to address potential privacy concerns of users.

7.4 Summary

In our evaluation, we examined the practical benefit of our deep neural network from three different angles. First, we showed the metric-based performance of our neural network. Second, we demonstrated the plausibility of learned words for each permission group. Third, we showed single description texts in detail for which our system correctly predicted permission configurations and also found discrepancies between the text and the actual permissions.

Our approach for identifying permission-related phrases has stressed the employment of a dynamic, self-learning model. Current approaches strongly rely on static relations, e.g., the word *camera* has to appear in the description text if an Android SDK method includes the exact same word, or the model training process relies on a limited set of hand-picked description sentences. A quick look at high-impact words for the camera permission group above illustrates the superiority of our approach: The use of pre-trained word embeddings and a large description dataset allow the system to learn a broad range of plausible relations that are very specific, from words like *capturing* over *QR/barcode* and *snap*. Manually labeling all these words and their combinations before training is practically infeasible. Our deep learning approach especially addresses the ever-changing nature of app use cases and their evolving peculiarities.

The aspect that sets our approach apart from current solutions arises from the filter-based convolutional architecture, which finds local properties, i.e., words and word groups, and makes global decisions based on them. Our examples demonstrate that the way a CNN processes text is not only limited to a sentence-based search but takes description-wide information into account. For AndroMedia, trigger words are *export*, *save*, and *files*, but they are be spread across multiple sentences. A system limiting the scope to one sentence cannot interpret this context. Utilizing multiple filter sizes increases the effectiveness by also training on word groups of two and three words. Our evaluation makes clear that crucial information can span over multiple description sentences and that we are capable of capturing these relations.

8 CONCLUSION

Android users often fall for apps with intriguing descriptions and grant them permissions that could adversely impact their privacy. However, cross-checking an app's description and its permission usage is challenging due to the lack of enforced quality standards for descriptions and the nonexistent verification of their content.

In this work, we presented a solution that can reliably assess and grade the need for *dangerous permissions* by analyzing real-world description texts. Based on a convolutional neural network, our approach accurately captures contextual properties in potentially incomplete app descriptions and can reliably predict individual groups of semantically related permissions. We carefully evaluated our approach on more than 77,000 real-world app descriptions and uncovered discrepancies between the actual and predicted usage of permissions with a precision between 71% and 93%. Our solution provides an effective method to assess privacy awareness in descriptions of Android applications.

REFERENCES

[1] [n.d.]. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL.

[2] [n.d.]. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. ACM.

[3] [n.d.]. *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2018, Gothenburg, Sweden, May 27 - 28, 2018*. ACM.

[4] Anshul Arora and Sateesh Kumar Peddoju. 2018. NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions. In *Trust, Security and Privacy in Computing and Communications – TrustCom'18*. IEEE, 808–813.

[5] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A Neural Probabilistic Language Model. In *Neural Information Processing Systems – NIPS'00*. MIT Press, 932–938.

[6] Michalis Diamantaris, Elias P. Papadopoulos, Evangelos P. Markatos, Sotiris Ioannidis, and Jason Polakis. 2019. REAPER: Real-time App Analysis for Augmenting the Android Permission System. In *Conference on Data and Application Security and Privacy – CODASPY'19*. ACM, 37–48.

[7] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick D. McDaniel, and Anmol Sheth. 2014. TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones. *Commun. ACM* 57 (2014), 99–106.

[8] Hongcan Gao, Chenkai Guo, Yanfeng Wu, Naipeng Dong, Xiaolei Hou, Sihan Xu, and Jing Xu. 2019. AutoPer: Automatic Recommender for Runtime-Permission in Android Applications. In *43rd IEEE Annual Computer Software and Applications Conference, COMPSAC 2019, Milwaukee, WI, USA, July 15-19, 2019, Volume 1*. IEEE, 107–116.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[10] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *International Conference on Software Engineering – ICSE'14*. ACM, 1025–1035.

[11] TonTon Hsien-De Huang and Hung-Yu Kao. 2018. R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections. In *Conference on Big Data – BIGDATA'18*. IEEE, 2633–2642.

[12] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification, See [1], 1746–1751.

[13] Deguang Kong, Lei Cen, and Hongxia Jin. 2015. AUTOREB: Automatically Understanding the Review-to-Behavior Fidelity in Android Applications, See [2], 530–541.

[14] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1 (1989), 541–551.

[15] Zachary Chase Lipton. 2016. The Mythos of Model Interpretability. *CoRR* abs/1606.03490 (2016).

[16] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Neural Information Processing Systems – NIPS'17*. 4765–4774.

[17] Niall McLaughlin, Jesús Martínez del Rincón, BooJoong Kang, Suleiman Y. Yerima, Paul C. Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. 2017. Deep Android Malware Detection. In *Conference on Data and Application Security and Privacy – CODASPY'17*. ACM, 301–308.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Neural Information Processing Systems – NIPS'13*. 3111–3119.

[19] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *USENIX Security'13*. USENIX Association, 527–542.

[20] Jungsoo Park, Hojin Chun, and Souhwan Jung. 2018. API and permission-based classification system for Android malware analysis. In *Conference on Information Networking – ICOIN'18*. IEEE, 930–935.

[21] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation, See [1], 1532–1543.

[22] Anthony Peruma, Jeffrey Palmerino, and Daniel E. Krutz. 2018. Investigating user perception and comprehension of Android permission models, See [3], 56–66.

[23] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. 2014. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Conference on Computer and Communications Security – CCS'14*. ACM, 1354–1365.

[24] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Conference on Knowledge Discovery and Data Mining – SIGKDD'16*. ACM, 1135–1144.

[25] Gian Luca Scoccia, Stefano Ruberto, Ivano Malavolta, Marco Autili, and Paola Inverardi. 2018. An investigation into Android run-time permissions from the end users' perspective, See [3], 45–55.

[26] Vincent F. Taylor and Ivan Martinovic. 2016. SecuRank: Starving Permission-Hungry Apps Using Contextual Permission Analysis. In *Security and Privacy in Smartphones & Mobile Devices – SPSM@CCS*. ACM, 43–52.

[27] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A measurement study of google play. In *Measurement and Modeling of Computer Systems – SIGMETRICS'14*. ACM, 221–233.

[28] Haoyu Wang, Jason I. Hong, and Yao Guo. 2015. Using text mining to infer the purpose of permission use in mobile apps. In *Conference on Pervasive and Ubiquitous Computing – UbiComp'15*. ACM, 1107–1118.

[29] Haoyu Wang, Yuanchun Li, Yao Guo, Yuvraj Agarwal, and Jason I. Hong. 2017. Understanding the Purpose of Permission Use in Mobile Apps. *ACM Trans. Inf. Syst.* 35 (2017), 43:1–43:40.

[30] Jiayu Wang and Qigeng Chen. 2014. ASPG: Generating Android Semantic Permissions. In *Conference on Computational Science and Engineering – CSE'14*. IEEE Computer Society, 591–598.

[31] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori. 2015. Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps. In *Symposium On Usable Privacy and Security – SOUPS'15*. USENIX Association, 241–255.

[32] Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. 2018. A Large Scale Investigation of Obfuscation Use in Google Play. In *Annual Computer Security Applications Conference – ACSAC'18*. ACM, 222–235.

[33] Jingzheng Wu, Mutian Yang, and Tianyue Luo. 2017. PACS: Pemission abuse checking system for android applictions based on review mining. In *Conference on Dependable and Secure Computing – DSC'17*. IEEE, 251–258.

[34] Mu Zhang, Yue Duan, Qian Feng, and Heng Yin. 2015. Towards Automatic Generation of Security-Centric Descriptions for Android Apps, See [2], 518–529.

[35] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Neural Information Processing Systems – NIPS'15*. 649–657.