IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 June 2022, accepted 9 August 2022, date of publication 18 August 2022, date of current version 29 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3199882

RESEARCH ARTICLE

PermPress: Machine Learning-Based Pipeline to Evaluate Permissions in App Privacy Policies

MUHAMMAD SAJIDUR RAHMAN[1], PIROUZ NAGHAVI[1], (Graduate Student Member, IEEE), BLAS KOJUSNER[1], SADIA AFROZ[2], BYRON WILLIAMS[1], SARA RAMPAZZI[1], (Member, IEEE), AND VINCENT BINDSCHAEDLER[1], (Member, IEEE)

[1]Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32603, USA
[2]Avast Software, Emeryville, CA 94608, USA

Corresponding author: Muhammad Sajidur Rahman (rahmanm@ufl.edu)

ABSTRACT Privacy laws and app stores (e.g., Google Play Store) require mobile apps to have transparent privacy policies to disclose sensitive actions and data collection, such as accessing the phonebook, camera, storage, GPS, and microphone. However, many mobile apps do not accurately disclose their sensitive data access that requires sensitive ('dangerous') permissions. Thus, analyzing discrepancies between apps' permissions and privacy policies facilitates the identification of compliance issues upon which privacy regulators and marketplace operators can act. In this paper, we propose *PermPress* – an automated machine-learning system to evaluate an Android app's permission-completeness, i.e., whether its privacy policy matches its dangerous permissions. *PermPress* combines machine learning techniques with human annotation of privacy policies to establish whether app policies contain permission-relevant information. *PermPress* leverages MPP-270, an annotated policy corpus, for establishing a gold standard dataset of permission completeness. This corpus shows that only 31% of apps disclose all dangerous permissions in privacy policies. By leveraging the annotated dataset and machine learning techniques, *PermPress* achieves an AUC score of 0.92 in predicting the permission-completeness of apps. A large-scale evaluation of 164, 156 Android apps shows that, on average, 7% of apps do not disclose more than half of their declared dangerous permissions in privacy policies, whereas 60% of apps omit to disclose at least one dangerous permission-related data collection in privacy policies. Our investigation uncovers the non-transparent state of app privacy policies and highlights the need to standardize app privacy policies' compliance and completeness checking process.

INDEX TERMS Privacy policy, android apps, data privacy, NLP, machine learning, annotated dataset.

I. INTRODUCTION

Users need transparent and truthful app privacy policies to understand what sensitive data apps collect and how the apps may use that sensitive data. Unfortunately, mobile app privacy policies are often vague and incomplete despite data privacy laws requiring mobile apps to have transparent and trustworthy privacy policies. For example, in 2013, the Federal Trade Commission (FTC) fined social media app company Path Inc. for using vague, slippery language in the app's privacy policy to describe its data collection, especially when the app accesses a user's phonebook [1]. FTC found the language in Path's privacy policy, for example – "We automatically collect certain information…such as your Internet Protocol (IP) address, your operating system…" – to be an incomplete and inadequate illustration of data access and collection by the app [2]. In 2021, Ireland's Data Protection Commission fined WhatsApp Inc., a messaging company, a record fine of €225 million for failing to be transparent in its privacy policy about how it handled personal information and shared it with first/third parties [3].

Since 2012, mobile app marketplaces (e.g., Android Play Store, iOS App Store) have started mandating app publishers to list links of privacy policies both in the marketplace app listing pages and inside apps [4]. The Android platform specifically requires apps that request any sensitive or

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio J. R. Neves.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

dangerous permissions, such as contact list, camera, storage, location, or microphone, to have a transparent privacy policy. However, marketplaces only check whether the app has listed a 'URL' of a privacy policy but do not verify its content, i.e., whether the privacy policy fully discloses the app's permission-based sensitive data collection. Analyzing a privacy policy is challenging for two reasons: First, understanding/extracting useful information from privacy policies is difficult because privacy policies are written in natural language with vague, complex, and lengthy texts [5]. Second, app developers often look for workarounds, such as using privacy policy templates/generator services to generate boilerplate privacy policies to expedite app publishing [6]. As these boilerplate privacy policies are often generic and are not necessarily written for a particular app, they may not conform to requirements set by privacy laws [7]. Evaluating discrepancies between apps' requested permissions and privacy policies enables regulators and marketplace operators to identify apps' potential compliance issues earlier than later from unexpected privacy breaches.

A. RESEARCH QUESTIONS

We seek to investigate if an app's privacy policy is *permission-complete*. A privacy policy is permission-complete if it discloses the processing of sensitive data and performing sensitive actions related to dangerous Android permissions. In other words, the policy is permission-complete if it matches the dangerous permissions that the app declares in the manifest file. We introduce *PermPress* – an automated machine-learning (ML) system for checking the **Perm**ission-**Pr**ivacy Complet**e**ne**ss** of Android apps. The following research questions guide the design, implementation, and use cases of *PermPress*:

- *RQ1: Do app privacy policies disclose sensitive information collection via dangerous Android permissions?* (§ IV). Prior works focus on either web privacy policies (e.g., [8], [9]) or on identifying first- and third-party data collections (e.g., [10], [11], [12]). There exists no systematic investigation into whether app privacy policies contain meaningful and relevant information about all of their declared permissions. Answering RQ1 helps fill this gap and advance understanding of the completeness of app policies.
- *RQ2: Can we apply machine learning techniques to infer dangerous permissions from app privacy policies?* (§ V). Prior works [10], [11], [12], [13], [14], [15], [16] often take an overly narrow focus on a subset of Android permissions, such as those related to accessing location or device identifiers, while ignoring other (potentially harder to predict from the privacy policy) permissions (e.g., accessing camera or phonebook). Thus, it is unclear how best to leverage machine learning models for permission inference from app policies and what combination of techniques yields the most accurate model.
- *RQ3: Can we calculate an app's permission-completeness score and predict it?* (§ VI). We emphasize that accurately predicting if a privacy policy is complete and transparent (including where mismatches happen) allows users (e.g., privacy regulators, marketplace operators, app publishers/developers) to make thoughtful privacy decisions about an app. For example, marketplace operators can use an automated system like *PermPress* to identify discrepancies in requested permissions and policy before publishing an app, thus forcing app publishers to comply with privacy laws and marketplace policies. Moreover, *PermPress* complements prior app policy analysis works (e.g., [10], [11]) by measuring the incompleteness of app policies in an automated way.

B. KEY CONTRIBUTIONS

We leverage *PermPress*, which consists of a two-phase machine learning-based pipeline to infer permission-completeness based on app privacy policies and manifest files to answer the research questions. In the first phase, we apply the natural language processing (NLP) technique with supervised machine learning models to predict an app's dangerous permissions from its privacy policy. In the second phase, we leverage the trained models and their predictions from phase-1 to predict a permission-completeness label accurately, i.e., a (mis)match between an app's declared permissions and privacy policy. To create an accuracy baseline for *PermPress*, we develop a gold-standard policy corpus named '**M**apping between **P**ermission and **P**rivacy' (**MPP-270**). This annotated corpus establishes whether privacy policies contain relevant information about apps accessing dangerous permissions to collect sensitive data. We find that only 31% of apps in the MPP-270 corpus contain complete and relevant information about all dangerous permissions declared in their manifest file. Outside of the 31%, all apps disclose at least one dangerous permission-related data collection in privacy policies. Permission-relevant information about `CALENDAR`, `SMS`, and `CONTACTS` are rarely found ($<50\%$) in privacy policies, whereas `PERSISTENTID` and `LOCATION` are frequently mentioned ($\geq 90\%$). By leveraging the gold-standard dataset and machine learning techniques, *PermPress* achieves an AUC score of 0.92 to predict the permission-completeness for an app. *PermPress* conducts a large-scale evaluation of 164,156 Android apps and finds that, on average, 7% of apps across all app categories do not disclose more than half of their declared dangerous permissions in privacy policies. 60% of apps omit to disclose at least one dangerous permission-related data collection in privacy policies. Using LIME [17] (§ V-C), we investigate prediction behavior of machine learning models (FastText and logistic regression). This exploration with model explainability provides insights about cases when privacy policies do not contain meaningful information about the target class (i.e., permission) and when multiple apps share the same policy, irrespective of differing permissions in manifest files.

We believe that *PermPress* holds promise in uncovering the non-transparent state of app privacy policies and automating the compliance checking process for end-users like privacy regulators, marketplace operators, and app publishers/

developers. We are the first to systematically evaluate app privacy policies and identify the relevant features that explain Android apps' dangerous permission-based data collection from privacy policies. By combining human annotation with state-of-the-art machine learning and NLP techniques to extract features from natural language privacy policies, *PermPress* can facilitate 'Usable Privacy' research to model end-user privacy preferences.

In summary, we make the following contributions:

1) **MPP-270 corpus** (§ IV). This paper performs the first systematic mapping study between the app privacy policy and Android permissions by manually annotating 270 Android app policies. It is the only annotated app policy corpus that maps all dangerous permissions to privacy policies. The MPP-270 corpus and the machine learning dataset consisting of 164,156 apps' metadata are open-source [18] for further research.
2) **Model benchmarking and explainability** (§ V). This paper systematically examines the information richness of privacy policies by applying three different machine learning models (logistic regression, FastText, BERT) and comparing these models' predictive performance. We find that models with a sophisticated word- and context-embedding techniques outperform feature-based embedding, even when positive samples are the minority. This paper also leverages LIME (§ V-C) to explain model predictions.
3) **Machine learning-based pipeline** (§ VI). This paper proposes *PermPress* – a scalable machine learning-based pipeline to automatically infer an app's permission-completeness label based on the privacy policy and app manifest file. Our analysis has found broad evidence of privacy non-compliance across all permission groups and Android apps from all categories.

II. BACKGROUND

In this section, we briefly describe sensitive data in the context of smartphones, the Android permission model, and the difference between a web privacy policy and an app privacy policy.

A. PRIVATE DATA IN MOBILE DEVICES

Data in smartphones can be classified based on information type and data source [19]. Based on either the *information* type or the *source*, a piece of information on smartphones can be deemed private if it can be exploited to identify or profile a user. Such data may include device ID, contacts, pictures, and location coordinates. Additionally, seemingly non-sensitive data can be mined and analyzed to infer personal information, which could turn out to be sensitive and private.

B. ANDROID PERMISSION MODEL

In Android, Java API Frameworks form the building blocks for the app developers to create Android apps by simplifying the reuse of core, modular system components, and services. App permissions help support user privacy by protecting an app's access to (i) *restricted data*, such as system state and a user's contact information, and (ii) *restricted actions*, such as connecting to a paired device and recording audio. Based on the scope of restricted data and restricted actions, Android categorizes permissions into different types, including (i) install-time permissions, (ii) runtime permissions, and (iii) special permissions [20].

Install-time permissions (granted at install-time of apps) allow apps to access data and actions beyond the app's sandbox without posing risks to the user's privacy and other apps' operation. Examples are access to the internet or phone vibration.

Runtime permissions ('dangerous permissions') grant an app additional access to restricted data to allow the app to perform restricted actions that more substantially affect the system and other apps. Runtime permissions are assigned with the 'dangerous' protection level, as runtime permissions access private user data (e.g., location, phonebook) and access to the device's sensor- and function-rich hardware (e.g., microphone, camera). In this paper, we categorize 30 permission APIs designated as 'dangerous' into ten functionally related groups (see Table 1).

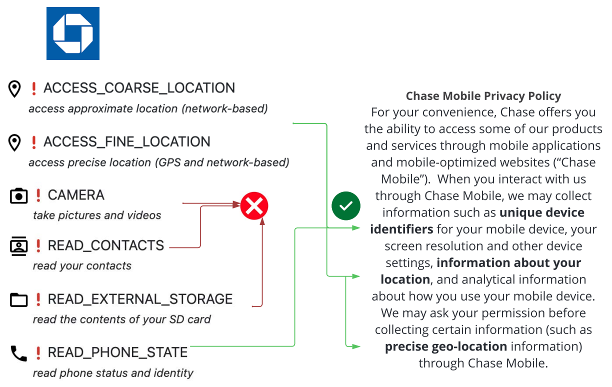
Special permissions refer to specialized app operations that the Android platform and equipment manufacturers (OEMs) can only define. Special permissions are assigned with the 'appop' protection level. This paper only focuses on app policy transparency about disclosing dangerous-permission-related sensitive data access and collection. If not specified, 'permission' refers to 'dangerous permission' in rest of this paper.

C. PRIVACY POLICIES

Privacy laws such as GDPR [21], CalOPPA [22], CCPA [23], and COPPA [24] require websites to disclose in web privacy policies types of personal information (e.g., names, email addresses, shipping addresses, payment card info) websites collect from their visitors/users. In addition to this list of personal information collection, privacy laws also mandate websites to disclose in privacy policies any 'technical data' collected via tracking technologies such as cookies, IP addresses, pixels, and browsing history. In the app privacy policy, the app must disclose personal information collection, similar to websites. In addition, the app must disclose if it accesses any sensitive data or resources (e.g., geolocation data from GPS, photo/media gallery, device ID, biometrics data) on the device by invoking any dangerous permission APIs. The app must also disclose data collection and sharing practices with third-party tools (e.g., ads and analytics libraries) used during the app development [25], [26]. The capability of smartphone apps to access sensitive OS-privileged APIs (i.e., *dangerous permissions*) to operate on a rich set of sensitive data increasingly raises privacy concerns. National and international privacy regulations (e.g., California Consumer Privacy Act [4], European General Data Protection [27]) thus require app privacy policy to be transparent and complete regarding sensitive data collection and sharing.

III. METHODOLOGY

An app's privacy policy is defined as permission-complete if the policy discloses the processing of sensitive data and performing sensitive actions related to dangerous Android permissions. In other words, the policy is permission-complete if it *matches* the dangerous permissions that the app declares in the manifest file.


ACCESS_COARSE_LOCATION
access approximate location (network-based)
ACCESS_FINE_LOCATION
access precise location (GPS and network-based)
CAMERA
take pictures and videos
READ_CONTACTS
read your contacts
READ_EXTERNAL_STORAGE
read the contents of your SD card
READ_PHONE_STATE
read phone status and identity

Chase Mobile Privacy Policy
For your convenience, Chase offers you the ability to access some of our products and services through mobile applications and mobile-optimized websites ("Chase Mobile"). When you interact with us through Chase Mobile, we may collect information such as **unique device identifiers** for your mobile device, your screen resolution and other device settings, **information about your location**, and analytical information about how you use your mobile device. We may ask your permission before collecting certain information (such as **precise geo-location** information) through Chase Mobile.

FIGURE 1. Example of permission-incompleteness: Three out of six dangerous permissions can be matched to the sensitive data access/collection described in the natural language text of the privacy policy. The remaining dangerous permission-related data collection is not disclosed in the app privacy policy.

For example, consider 'Chase Mobile' (Fig. 1), a mobile financial service app. The app declares six 'dangerous permissions' for its functionality, but only three dangerous permission-related data collection practices (device identifiers, approximate location, and precise location) are disclosed in the privacy policy. The privacy policy does not disclose any data access/collection using `CAMERA`, `CONTACTS`, and `STORAGE` permissions. Presumably, the app needs `CAMERA` permission for mobile check deposits. However, the app privacy policy makes no references to information collected from the phone's camera or how it is processed or shared. Based on the definition of permission-completeness, it can be said that this privacy policy only partially matches with the app's declared dangerous permissions.

After reviewing prior works [10], [11], [12], [13], [14], [15], [16], three significant shortcomings are found: First, the automated analysis approach often looks for sensitive data collection in privacy policies that may not be directly linked with Android's permission-based model of data collection (e.g., data collection during the user registration process, such as email, zip code, city, home address). Second, prior works often narrowly focus on a subset of permissions such as `PERSISTENTID` or `LOCATION` while ignoring permissions – that are potentially harder to predict from the privacy policy – such as accessing the camera or the phonebook (see § IX for details about prior works). It is thus unclear whether app privacy policies are transparent about their complete set of declared, sensitive permissions. Third, in the absence of a way to map between app privacy policy text and Android's permission-based data collection model, it is challenging to identify the cause of failed prediction: the model or the incomplete privacy policy.

In contrast to prior works, *PermPress* combines machine learning techniques with an initial step of policy annotation, where human evaluators annotate a sample of apps to establish whether privacy policies contain meaningful, relevant information about Android apps' sensitive permission-based data collection model. This annotated corpus then serves as a gold standard to evaluate machine learning models' prediction performance (phase-1) and a labeled dataset for training and validating machine learning models to infer permission-completeness (phase-2).

Fig. 2 outlines the three tasks to perform for answering the research questions and to build *PermPress*:

- **Task-1** answers **RQ1** (*do app privacy policies disclose sensitive information collection via dangerous Android permissions?*) by systematically annotating a corpus to establish whether privacy policies contain meaningful, relevant information about apps accessing dangerous permissions to collect sensitive data.
- **Task-2** answers **RQ2** (*can we apply machine learning techniques to infer dangerous permissions from app privacy policies?*) by creating a benchmark of machine learning models to predict permissions from privacy policies.
- **Task-3** answers **RQ3** (*can we calculate an app's permission-completeness score and predict it?*) by building *PermPress* through leveraging results from tasks 1 and 2.

The following sections provide in-depth discussion on the experimental setup for each task (task-1:§ IV, task-2:§ V, task-3:§ VI). Before delving into experimental details, the following is a discussion of the data collection and an exploratory analysis of the dataset.

A. DATASET

A Python-based scraper was implemented to scrape the Google Play Store (USA market) to collect app meta-information (app ID, app rating, number of installs, privacy policy link) from diverse app categories between December 2020 to February 2021. During this time, 500,000 apps' meta-information was collected. Subsequently, the apps' privacy policies were scraped by following the privacy policy links collected in the first phase. Before downloading a document, the scraper checked whether the document contained the phrase 'privacy policy' and had at least ten sentences. The scraper leveraged the NLTK library's [28] parsing and language detection functions to check whether the texts are in the English language. After discarding non-English and unreachable privacy policies, 213,000 app privacy policies were finally collected. Next, a separate scraper started to download the application packages (APKs) for these 213,000 apps from Google PlayStore and finally downloaded 164,156 usable APKs. Using the AndroGuard [29] reverse engineering tool, the downloaded APKs were decompiled to extract

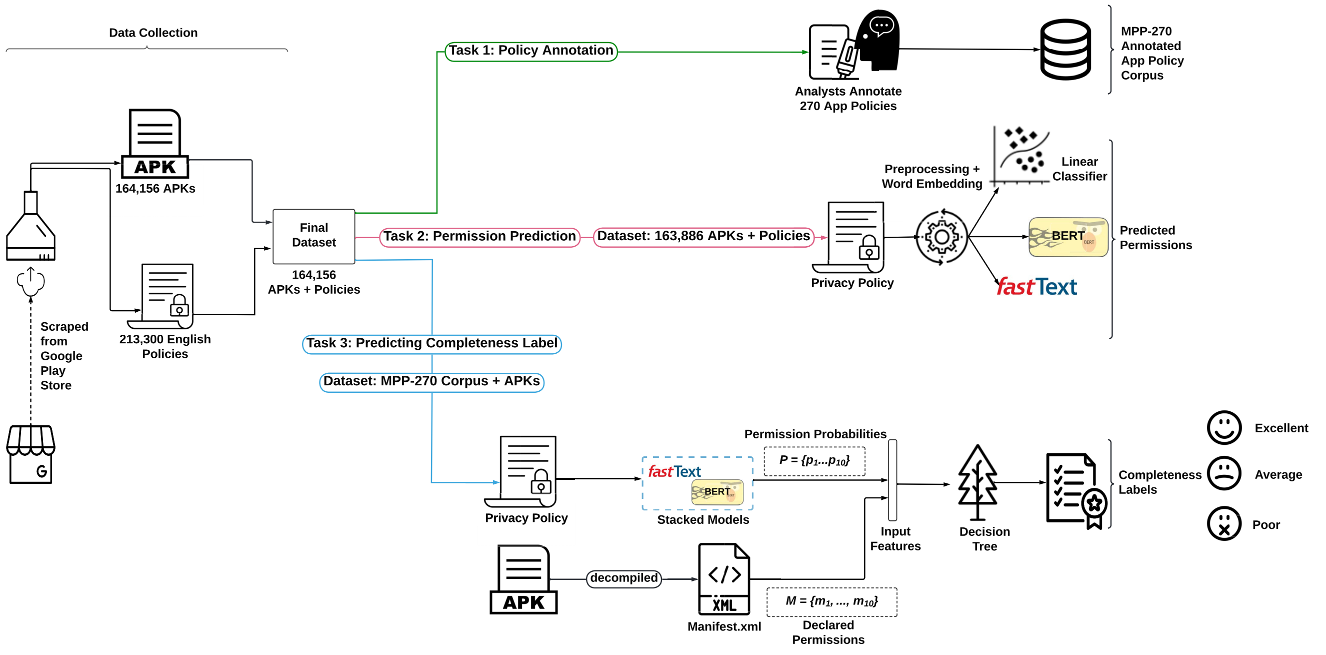


FIGURE 2. **Experimental setup used to investigate permission-completeness of app privacy policies. *PermPress* leverages both the outcomes from tasks 1 and 2 to accurately predict completeness label of an app (task 3), based on the privacy policy and manifest file.**

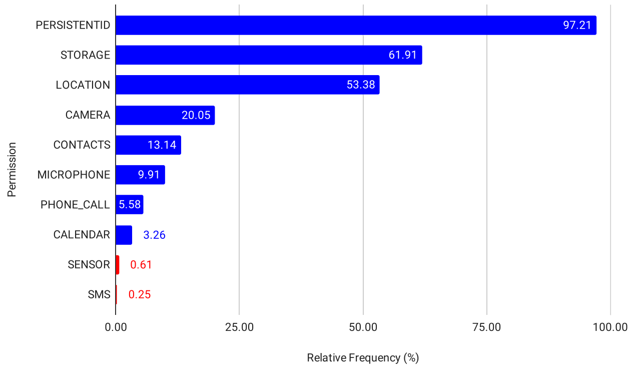


FIGURE 3. **Distribution of dangerous permissions in the final dataset of 164,156 apps.**

declared permissions from the `AndroidManifest.xml` file.

In this paper, we consider the 30 permission APIs which have 'dangerous' protection level in official Android documentation [30]. Functionally related dangerous APIs are grouped together (e.g., all SMS/MMS related APIs into one group). There are ten permission groups: `CALENDAR`, `CAMERA`, `CONTACTS`, `LOCATION`, `MICROPHONE`, `PERSISTENTID`, `PHONE_CALL`, `SENSOR`, `SMS`, and `STORAGE` (Table 1). Unless otherwise stated, any reference related to 'permission' in this paper refers to any of these ten dangerous permission groups. After extracting manifest files, the dangerous permission APIs were mapped to one of the ten dangerous permission groups. The top three dangerous permission groups were `PERSISTENTID` (97%), `STORAGE` (62%), and `LOCATION` (53%), and least accessed permission groups (accessed by $< 5\%$ apps) were `CALENDAR`, `SENSOR`, and `SMS`. The median number of dangerous permissions accessed by apps was 3. Fig. 3 shows the dangerous permission distribution in the collected dataset. In the following sections, if not otherwise mentioned, the final dataset consisted of these 164,156 APKs and their corresponding privacy policies.

B. EXPLORATORY DATA ANALYSIS

After collecting the dataset, a data exploration was conducted to investigate privacy policy-related app metadata. Specifically, the data exploration consisted of exploring web domains where the privacy policies are hosted. The data exploration investigated whether multiple apps point to the same privacy policy URL. Furthermore, the data exploration examined whether apps sharing the same privacy policy URL are similar in manifest files.

1) HOSTING DOMAINS OF PRIVACY POLICIES

First, the domain name was extracted from a privacy policy URL. Next, the number of apps with privacy policy URLs pointing to the same domain name was counted. Fig. 4 shows the top 20 domain names shared and pointed to by several hundred to thousands of apps. The top domain name belongs to Google's free website hosting service, `sites.google.com`, with 17,000 apps (>10% of the dataset) having their privacy policies hosted there. Interestingly, the second top domain (`docs.google.com`) is not a typical website, but Google's online document editing/publishing service

TABLE 1. Table lists 30 dangerous permission APIs categorized in 10 permission groups. All permissions listed here (except those asterisk marked) are marked as having protection level as *dangerous* in the Official Android API Documentation [30]. `ACCESS_WIFI_STATE` **and** `ACCESS_NETWORK_STATE` **have been found by previous efforts to access PII data [31], [32].**

Dangerous Permissions	Permission Group
READ_CALENDAR WRITE_CALENDAR	Calendar
CAMERA	Camera
READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS	Contacts
ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION ACCESS_WIFI_STATE* ACCESS_MEDIA_LOCATION	Location
RECORD_AUDIO	Microphone
READ_PHONE_STATE ACCESS_NETWORK_STATE*	PersistentID
READ_PHONE_NUMBERS CALL_PHONE ANSWER_PHONE_CALLS ADD_VOICEMAIL USE_SIP READ_CALL_LOG WRITE_CALL_LOG PROCESS_OUTGOING_CALLS	Phone_Call
BODY_SENSORS ACTIVITY_RECOGNITION	Sensor
SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS	SMS
READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE	Storage

named Google Docs. Like Google Docs, app privacy policies piggyback on cloud storage and document sharing services, e.g., GitHub repositories, GitHub Gist, Pastebin, and Google's Firebase storage. Fig. 4 also presents five domain names (`freeprivacypolicy`, `iubenda`, `myappterms`, `privacypolicies`, `termsfeed`) that provide services to generate and publish mobile app privacy policies. This variability in hosted domains shows a lack of standardization of privacy policy creation and disclosure. Domains such as Google Docs, GitHub, Pastebin, and Firebase also show that app publishers/developers often choose a sort of "homemade" approach to publishing privacy policies, i.e., the app publishers/developers take any possible workaround to 'host' privacy policies to get their apps published in the Google Play Store.

2) APP PERMISSIONS AND PRIVACY POLICY URLs

First, the apps that share the same privacy policy URL are grouped. Then, the dangerous permission distributions of the grouped apps are plotted. Fig. 5 shows the box-plot of permission distribution of apps sharing the same privacy policy

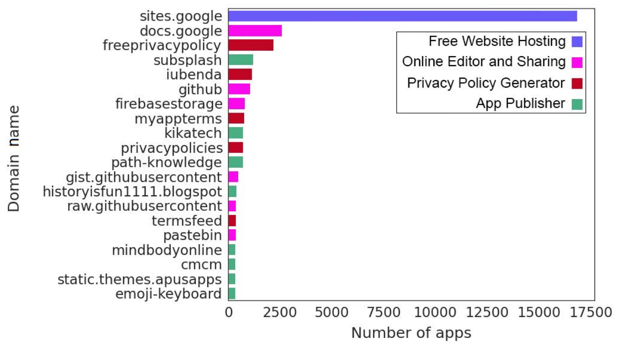


FIGURE 4. Distribution of the top-20 domain names and the number of apps hosting their app privacy policies under these domains. As can be seen from the color code, there are four categories of domains in the top-20 domain names used for hosting app privacy policies. A quarter of these (🟥) domains belong to privacy policy generator services. In contrast, another quarter of the domains (🟪) point to various cloud-based document processing/sharing services. This variability in hosted domains raises questions about the lack of standardization in preparing and publishing app-specific privacy policies.

URL. Similar to the previous analysis with hosted domains of app privacy policies, it is observed that a few hundred apps share a single boilerplate privacy policy generated by a privacy policy template service, `myappterms`. Besides using boilerplate policy files, apps also plagiarize and list other apps' privacy policy URLs as their own. For example, Unity3d is a popular SDK to build AR/VR/gaming features in Android/iOS apps and is not an app by itself. Yet, Unity3d's privacy policy URL [1] is listed as privacy policy by 246 apps. Further investigation found that 9 out of 246 apps are published by 'Unity Technologies ApS,' the official app publisher of Unity3d; the remaining 237 apps are various gaming apps and are not officially related to Unity3d (based on app publishers' websites). This phenomenon of privacy policy plagiarism and using boilerplate templates raises questions about the authenticity of app privacy policies.

Key Takeaways

- The majority of app privacy policies are not hosted on app publishers' websites. Rather, app publishers host privacy policies using free domains of privacy policy generators and free website/content publishing services, raising questions about the lack of standardization in preparing and publishing app-specific privacy policies.
- Listing other apps' privacy policies as their own is common in Android apps. While reusing a common, boilerplate privacy policy may seem natural, it raises questions about the authenticity of the privacy policy and disclosure of app-specific sensitive data collection.

IV. TASK 1: POLICY ANNOTATION

In this section, we provide details about app privacy policy annotation task to establish whether app policies disclose meaningful, relevant information about the dangerous

[1]Unity3d Privacy Policy:https://unity3d.com/legal/privacy-policy

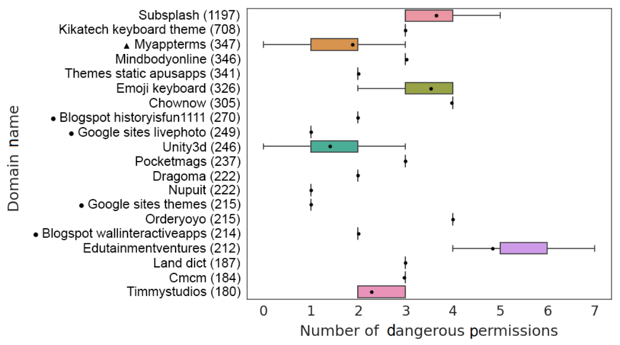


FIGURE 5. **Box-plot of dangerous permission distribution in apps that share the same privacy policy URL. Y-axis shows the domain name of a single privacy policy URL. Next to the domain name, the number of apps pointed to this particular URL has been mentioned. Besides URLs linked with Myappterms, Google sites, and Blogspot, the rest belong to different app publishers. Subsplash, an app publisher, has published only 100 apps, yet almost 1000 apps plagiarize its privacy policy. The privacy policy URL pointed at MyAppTerms (▲) has been shared by 347 apps with requested permissions ranging from 1 to 3. The privacy policy URLs hosted at Google sites and Blogspot (•) are examples of apps that may be published by the same publishers (no variation in the number of requested permissions).**

permissions declared in app manifest files (see § III and Fig. 2). Following is the methodology to create the MPP-270 corpus to map privacy policy text snippets with relevant permission information. We also discuss the findings and results of the policy annotation task.

A. MPP-270: ANNOTATION PROTOCOL

Previous works on policy annotations primarily focus on web privacy policies, not app-specific ones (e.g., [8], [9]). Prior works also focus on first- and third-party data collections rather than sensitive permission-based data collection (e.g., [11]). No labeled dataset maps between app permission and privacy policy. As privacy policies are written in natural language and are notorious for vague, complex, and lengthy texts [5], it is challenging to understand the proper meaning or extract useful information from the app policies. To fill this gap and advance understanding of the completeness of app privacy policy, the MPP-270 policy corpus is created by annotating the privacy policies of 270 unique Android apps. These 270 apps are sampled by ranking the dataset of 164,156 apps based on the total number of app installs and user ratings and then selecting the top 270 apps. It is also ensured that the selected apps cover all 13 app categories present in the larger dataset.

We opt against crowdsourcing policy annotation task due to the ambiguous nature of privacy policies and the required expertise. Instead, following prior work ((Zimmeck *et al.* [11] used two annotators), two of the authors with experience in Android app development and data privacy annotated the 270 privacy policies. Initially, the annotators randomly selected and annotated 10 privacy policies. Afterwards, both annotators met and compared their annotations to discuss their observations. Based on their observations, the following annotation protocol was devised:

- An annotator would only annotate sentence(s)/text phrases from the app privacy policy that could logically relate to any of these dangerous permissions: `CALENDAR`, `CAMERA`, `CONTACTS`, `LOCATION`, `MICROPHONE`, `PERSISTENTID`, `PHONE_CALL`, `SENSOR`, `SMS`, and `STORAGE`. The annotator would use the permission name as annotation labels.
- To avoid annotating misleading sensitive data collection, the annotator would only focus on those sensitive data collection practices which can be linked to an app's declared *dangerous permission(s) APIs* found in the `AndroidManifest.xml` file. The annotator would use Table 1 to select appropriate permission group for the app.
- The annotator would differentiate between information collected as part of the user registration process (e.g., PII) versus sensitive information collected while using the app. The annotator would not link any sensitive data collected from user registration (e.g., name, email, zip code, shipping address) to any dangerous permissions unless explicitly mentioned.
- Unlike prior work [11], the annotator would not distinguish who collected data (first/third-party), rather focused on whether sensitive data relevant to dangerous permissions was accessed/collected.
- When the app privacy policy does not mention accessing dangerous permissions explicitly, but rather mentions collecting certain data, the annotator would consult with official Android documentation to verify whether the app needs access to a particular dangerous permission to collect those sensitive data.
- In the privacy policy, an app often describes in length various app features or user interactions that an app user could perform to get services. In cases when an app does not explicitly disclose sensitive data collection but instead describes user interactions with the app, the annotator would evaluate those interactions, based on Android documentation, to check whether the interactions could lead the user to grant consent to dangerous permission(s). For example, suppose a camera app declares `READ_STORAGE` permission and talks about photo upload functionality in the privacy policy. In that case, the annotator could link the user interaction of uploading photo to `STORAGE` permission.

After establishing the protocol, each annotator annotated the remaining 260 app policies individually. It took a month for the annotators to complete the annotation. Finally, Cohen's Kappa κ score [33] was calculated for each of the ten labels (i.e., permission groups) to measure inter-rater reliability between the two annotators. In the following sections, MPP-270 is referred to denote the annotated corpus of 270 policies.

B. RQ1: ANNOTATION RESULTS

Table 2 shows the inter-rater agreement of the annotators. The average Cohen's Kappa score is 0.86, which shows strong agreement between the two raters. The lowest κ score

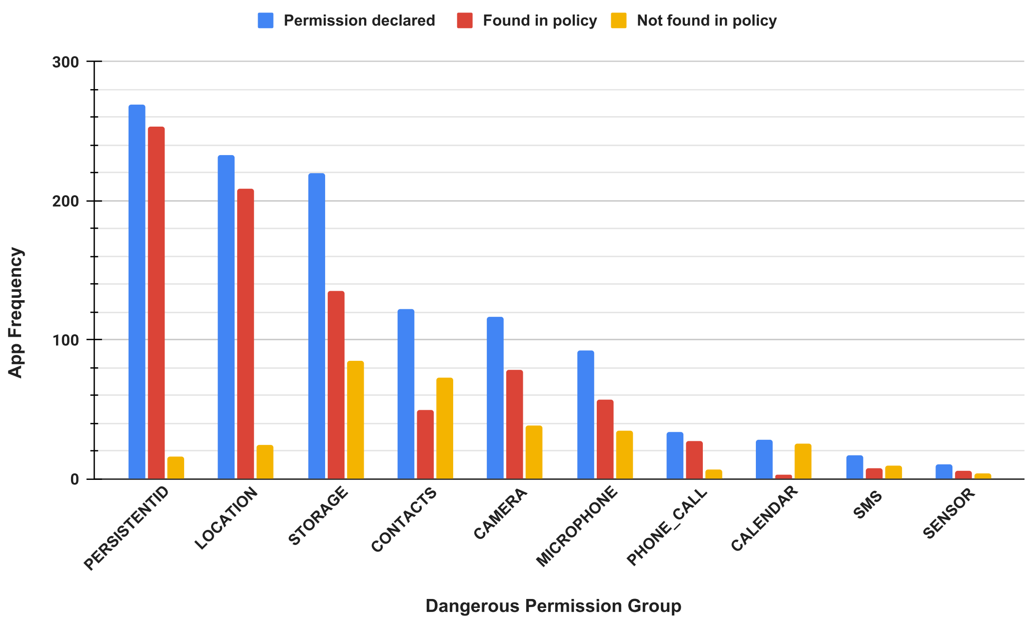


FIGURE 6. Distribution of dangerous permissions and annotations in the MPP-270 policy corpus. Data collection related to PERSISTENTID and LOCATION permissions were mentioned in 97% and 92% of the app policies, respectively. By contrast, CALENDAR was found the least (3 out of 28 apps). CONTACTS and SMS permissions related data access and collection were found in less than 50% of the apps' policies.

TABLE 2. Inter-rater agreement during annotation. On average, there is a 95% agreement between the two annotators. The average Cohen's Kappa is 0.86, which indicates a strong agreement between raters/coders during the annotation task.

Annotation Labels	% Agreement	Cohen's Kappa	No. Agreements	No. Disagreements
PERSISTENTID	97.40	0.68	263	7
STORAGE	85.19	0.77	230	40
LOCATION	97.41	0.93	263	7
CONTACTS	91.85	0.86	248	22
CALENDAR	98.15	0.90	265	5
SMS	97.78	0.82	264	6
SENSOR	99.26	0.90	268	2
PHONE_CALL	97.41	0.89	263	7
MICROPHONE	95.56	0.91	258	12
CAMERA	94.81	0.91	256	14

was 0.68 for PERSISTENTID and κ score $\geq$ 0.90 was achieved for LOCATION, MICROPHONE, CAMERA, and CALENDAR permissions. A low κ score in PERSISTENTID shows the relative difficulty in linking sensitive data to appropriate permissions.

The annotation finds that only 84 out of 270 apps (31% of apps) have privacy policies that contain complete and relevant information about all dangerous permissions declared in apps' manifest files. The annotation shows that PERSISTENTID and LOCATION related data collection is frequently mentioned in privacy policies. Out of 269 apps with PERSISTENTID permission, 94% of them mentioned collecting PERSISTENTID related information (e.g., IMEI, SIM card number). Out of 233 apps with LOCATION permission, 90% of apps mentioned LOCATION-related data collection in privacy policies. Despite their frequent occurrence, it is challenging to link sensitive data collection to appropriate permissions. For example, not all types of 'device identifiers' can be semantically equivalent to what sensitive data type PERSISTENTID API represents in reality. Using PERSISTENTID API, an app can access sensitive device identifiers such as SIM serial number or IMEI. Despite this fact, data types such as IP address, cookies, and trackers are often found to be referred to as 'device identifiers' in the app policies, which is semantically and pragmatically incorrect for PERSISTENTID. This challenge in linking the semantic meaning of an API to its Android-specific pragmatics is reflected in a lower κ score for PERSISTENTID. Besides this challenge in semantics, because of vague language in privacy policies, it is not always obvious how an app collects sensitive data in reality. To illustrate this, consider the following sentence from an app's policy:

"When you register for the Service, you provide us with your mobile phone number, which we use as your account identifier."

This sentence does not clarify how the app acquires the phone number: through user registration of filling a form or through requesting PHONE_CALL permission. Although this particular app declares PHONE_CALL permission, human annotators did not annotate this for disclosing PHONE_CALL permission due to the vague policy language. Some of the permissions are also harder to find in privacy policies. For example, apps with CALENDAR permission rarely describe about accessing/using calendar data (only three out of 28 apps disclose this). SMS and CONTACTS are two other permissions that were found infrequently in privacy policies - SMS being discussed in 47% (8 in 17 apps) of policies and CONTACTS in 40% (49 out of 122 apps). Interestingly, SENSOR was found in 60% of privacy policies, even though less than 5% apps declared SENSOR permission. Unlike permissions such as STORAGE, which were rarely explicitly mentioned in privacy policies and thus are challenging to identify, SENSOR permission was easier to identify in privacy policies because of using distinct keywords (e.g., 'sensor data'). Fig. 6 shows the distribution of app permissions declared in manifest files and their corresponding relevant information being disclosed (or not disclosed) in privacy policies.

Key Takeaways

- Dangerous permissions are almost never mentioned by their name in the app policies.
- Privacy policies rarely disclose permissions by their semantic meaning or by the sensitive data/actions the permissions represent in terms of Android API pragmatics.
- PERSISTENTID and LOCATION are the top frequently-requested permissions by apps, and these permissions are disclosed more clearly than other permissions in the app policies.
- CALENDAR, CONTACTS, and SMS permissions are requested by apps, yet these permissions are harder to find in privacy policies than other permissions.
- The annotated corpus is released [18] for future research.

V. TASK 2: INFERRING DANGEROUS PERMISSIONS

This section describes the experimental setup and results for task-2 – predicting app permissions from privacy policies (Fig. 2). Given the variability in privacy policy language in describing sensitive data collection (see § IV), the goal for this task is to understand how to leverage machine learning models for inferring permissions based on app policies and what combination of techniques yields the most accurate model.

Dataset: From the initial dataset of 164,156, the 270 apps used in MPP-270 corpus (see § IV) are held out. The remaining dataset of 163,886 apps (APKs and their app policies) are used for the following machine learning task.

A. MODEL SETUP

Machine learning is leveraged to scale the analysis by recasting the problem of inferring permissions as binary classification problems. The use of binary classifiers is chosen over multi-label classifiers to check whether machine learning models can learn individual target class-specific (i.e., permission) features from privacy policies, rather than inter-class correlation. Different machine learning models with various architectural complexity, feature selection criteria, and text representation are used to benchmark model performance in predicting app permissions from privacy policy text. Three different machine learning models are trained: feature-based model (logistic regression), word2vec model (FastText [34]) and state-of-the-art BERT [35] (Bidirectional Encoder Representations from Transformers) language model. For model evaluation, AUC, F1, precision, recall, and accuracy scores are calculated for each model type. Ten binary classifiers from each model category (logistic regression, FastText, and BERT) are trained to predict ten permission groups. The dataset of 163,886 apps (holding out the MPP-270 dataset) is split into the train-validation-test sets with a 60:20:20 ratio. The same train-validation-test dataset is used for training and testing all three types of models. As can be seen from Fig. 3, the dataset is highly imbalanced in class labels (i.e., permissions) – permissions such as PERSISTENTID (97%), STORAGE (62%), and LOCATION (53%) belong to the majority class whereas CALENDAR, SENSOR, and SMS belong to minority class (<5% apps). To handle class imbalance and avoid overfitting, appropriate regularization is applied for all types of model training. The ROC-AUC score is used during cross-validation to select optimum hyperparameters.

1) TRAINING LOGISTIC REGRESSION MODELS

CountVectorizer method from Scikit-learn [36] is used to preprocess text data and extract meaningful features for the inference task. Specifically, a text preprocessor is implemented to remove common English stop words, punctuation, numbers, hyperlinks, and HTML tags. This preprocessor is passed to the CountVectorizer to conduct text preprocessing. An English stop word list is built and passed to the CountVectorizer method to remove stop words and rare words. A minimum threshold is set for a word to be included in the vocabulary if the word appears in at least three documents (min_df=3). Lastly, an n-gram range is set up to construct multiple n-gram features (unigram, bigram, and trigram) to be used in the model. To retain the most relevant and unique features for each privacy policy document, the resultant term-frequency vector is fed into a TfIdfTransformer function to compute tf-idf scores. To find the best parameters for the binary classifiers for each permission group, a cross-validated grid-search is conducted over a parameter grid (regularization constant C (0.1, 0.01, 0.001, 1), penalty functions ($l1$ or $l2$, class weights)). ROC-AUC score is used for evaluating the performance of the cross-validated model on the validation set to tackle the class-imbalance issue. For the cross-validation splitting strategy, 5-fold cross-validation is used. After selecting the optimum parameters from grid-search, the models are evaluated on the test dataset to infer permissions based on the full-policy texts.

2) TRAINING FastText MODELS

The FastText model is a multinomial logistic regression classifier employing hierarchical softmax with stochastic gradient descent and a linearly decaying learning rate, which minimizes the model's negative log-likelihood of the class probability output. Note that hierarchical softmax approximates full softmax loss, and hierarchical softmax loss is meant for unbalanced classes. Based on experimentation with the validation set, the default hyperparameters proved appropriate, except for the number of epochs. During hyperparameter tuning, it was observed that the ROC-AUC score on the validation set improved with more epochs. Thus, the number of epochs was increased to 200, which also improved the class-imbalance issue.

3) TRAINING BERT MODELS

The pretrained BERT model [35] is chosen because of its state-of-the-art performance on a variety of text generation and classification applications. To handle class imbalance, cost-sensitive loss function [37] is implemented, which assigns different weights to each class. To illustrate this mathematically, let's assume that the model's output is $\mathbf{z}$ for C no. of classes, where

$$\mathbf{z} = [z_1, z_2, \ldots, z_C]^\top$$

For a sample x with class label y, the softmax cross-entropy loss can be calculated as:

$$\mathbf{CE}_{\text{softmax}}(\mathbf{z}, y) = -\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{C}\exp(z_j)}\right)$$

In this default loss calculation, each sample is treated equally which may cause model over-classifying the larger class(es) due to their higher prior probability. To overcome this issue, a sample-weighted cross-entropy loss is calculated as follows:

$$\mathbf{WCE}_{\text{softmax}}(\mathbf{z}, y) = -\frac{1}{N_{ny}}\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{C}\exp(z_j)}\right)$$

The selected weight for each sample is one over the total number of samples belonging to the class y (i.e., N_{n_y}), which overcomes model biases of correctly classifying only the majority class and misclassifying the minority class. This behavior is evident in the confusion matrix and the F1 score of the minority class on the validation set (see Fig. 11 in Appendix). For model training, a pretrained BERT model from Hugginface Transformer library [38] is used. The selected BERT model has 12 transformer blocks, 768 hidden layer sizes, and 12 self-attention heads with a case-insensitive vocabulary size of 30,522. Ten BERT models are finetuned for each of the ten permissions groups. One limitation of BERT is that the sequence length is limited to 512 tokens. Still, most of the privacy policies in the dataset are much longer (mean, median, and mode of tokens being 2160.54, 1225, and 8637, respectively). Prior work [39] has shown that for longer documents beyond BERT's max sequence length, truncating the document into smaller chunks and training the models on truncated texts (containing key information) can enable classification accuracy as competitive as hierarchical methods. Following this truncation strategy, an attempt is taken to identify text chunks from privacy policies containing only the relevant information. During annotation of the MPP-270 corpus, it is observed that a privacy policy's first few words (1000 characters) are generally introductory and not relevant to data collection. Based on this observation, a heuristic is made to discard the first 1000 characters from the beginning of a privacy policy and only consider the following 512 tokens as inputs for BERT models. After applying this heuristic on MPP-270 and examining the truncated policies, it is found that, after truncation, 97% of the policies contain general app description and data collection relevant information. This truncation strategy is finally adopted for inputs of BERT models during training, validation, and test. A linear layer is added to the BERT output layer for the downstream classification task. For further hyperparameter tuning, the initial learning rate, parameters for the linear learning rate scheduler with a warm-up, self-attention head dropout, and residual dropout are selected according to the results on the validation set. The initial learning rate is selected as $1e^{-4}$ with a batch size of 64. The ROC-AUC score is used during training to select the best-performing model. Training is stopped after a model fails to improve the ROC-AUC score on the validation set for three consecutive epochs.

B. RQ2: PERMISSION INFERENCE RESULTS

The following shows machine learning models' experimental results to predict permissions from app policies. All models are tested on the same test dataset of 32,000 apps and their privacy policies. Table 3 shows a summary of the comparison of model performance. FastText models achieved the highest average AUC score of 0.92, whereas BERT models have an average AUC score of 0.90. Logistic regression models have an average AUC score of 0.75 and an average macro-F1 score of 0.63 with 0.85 weighted-F1 scores. FastText models also achieved the highest macro-F1 (0.82) and weighted-F1 (0.94) scores in the permission inference task.

1) TEST RESULT: LOGISTIC REGRESSION MODELS

The macro AUC score for logistic regression models is 0.75, with an average weighted-F1 score of 0.85 and an average macro-F1 score of 0.63 (Table 3). The (macro) precision scores show that the models have higher false positive rates, especially with cases when class labels are highly imbalanced. For example, `PERSISTENTID` has majority samples belonging to positive class, whereas `SMS` has positive samples belonging to minority class. It is found that logistic regression models generally show high false positive rates with low false negative rates. Table 5 in Appendix presents weighted- precision and recall scores of all logistic regression models.

2) TEST RESULT: FastText MODELS

The AUC, macro-, and weighted-F1 scores of the ten FastText binary classifiers provided in Table 3 demonstrate that the models are learning to determine if the information in the text of a privacy policy indicates whether an app is using dangerous permissions. The lowest AUC scores for `SMS` (0.88) and `PERSISTENTID` (0.89) suggest that learning becomes difficult when the classes (positive or negative) are imbalanced. The macro-F1 score also highlights similar trends that the FastText model's objective function may be sub-optimal when the minority class is very small (e.g., `SMS`, `PERSISTENTID`). Interestingly, the high scores of AUC and macro-F1 for `SENSOR`, which also has a very small positive class, suggest that the privacy policy contains certain text features that allow the model to learn to recognize if an app is requesting access to `SENSOR` permission. This

TABLE 3. Comparison of model performance in the task of inferring dangerous permissions from the app privacy policy. The rows are arranged in descending order of AUC scores for Logistic Regression models. Logistic regression models have an average AUC score of 0.75 and an average macro-F1 score of 0.63 with 0.85 weighted-F1 scores. FastText models achieved the highest average score of 0.92, whereas BERT models have an average AUC score of 0.90. FastText models also achieved the highest macro-F1 (0.82) and weighted-F1 (0.94) scores in the permission inference task.

Dangerous Permission	AUC			Macro F1 Score			Weighted F1 Score		
	Logistic Regression	Fast Text	BERT	Logistic Regression	Fast Text	BERT	Logistic Regression	Fast Text	BERT
SMS	**0.91**	0.88	0.88	0.62	**0.67**	0.54	0.98	**1.00**	0.98
CONTACTS	0.86	**0.93**	0.90	0.73	**0.83**	0.71	**0.93**	**0.93**	0.84
CAMERA	0.84	**0.93**	0.90	0.51	**0.85**	0.76	0.81	**0.90**	0.82
PHONE_CALL	0.84	**0.93**	0.92	0.69	**0.85**	0.61	0.85	**0.97**	0.86
MICROPHONE	0.82	**0.92**	0.90	0.67	**0.83**	0.63	0.80	**0.94**	0.81
CALENDAR	0.81	**0.94**	**0.94**	0.69	**0.88**	0.64	0.75	**0.99**	0.92
STORAGE	0.72	**0.91**	**0.91**	0.72	**0.84**	0.76	0.72	**0.85**	0.77
LOCATION	0.69	**0.92**	0.86	0.70	**0.85**	0.77	0.72	**0.85**	0.77
SENSOR	0.50	**0.93**	0.92	0.50	**0.85**	0.57	1.00	**1.00**	0.97
PERSISTENTID	0.50	**0.89**	0.86	0.49	**0.79**	0.49	0.96	**0.98**	0.81

behavior is likely due to the unique keywords often used in privacy policies to specify access to the `SENSOR` permission group, which is observed during policy annotation. Macro- and weighted- precision, recall and accuracy scores are listed in Appendix (Table 6).

3) TEST RESULT: BERT MODELS

The average AUC score for BERT models is 0.89, making it clear that BERT models are learning to recognize if a privacy policy describes using a particular dangerous permission. Despite constraints on the input sequence length, BERT models are on par in performance with FastText models based on AUC and weighted-F1 score (Table 3). Similar to FastText models, BERT models outperform logistic regression models in accuracy, precision, and recall scores, and have been found effective in handling class-imbalance. For example, both FastText and BERT models appear to learn and detect positive samples, even though the samples are very rare (e.g., apps with `SMS` permissions). Fig. 11 in Appendix presents a side-by-side comparison of FastText and BERT models to predict `MICROPHONE`, `SMS`, and `STORAGE` permissions. Table 7 in Appendix shows accuracy, precision, and recall scores.

C. MODEL EXPLAINABILITY

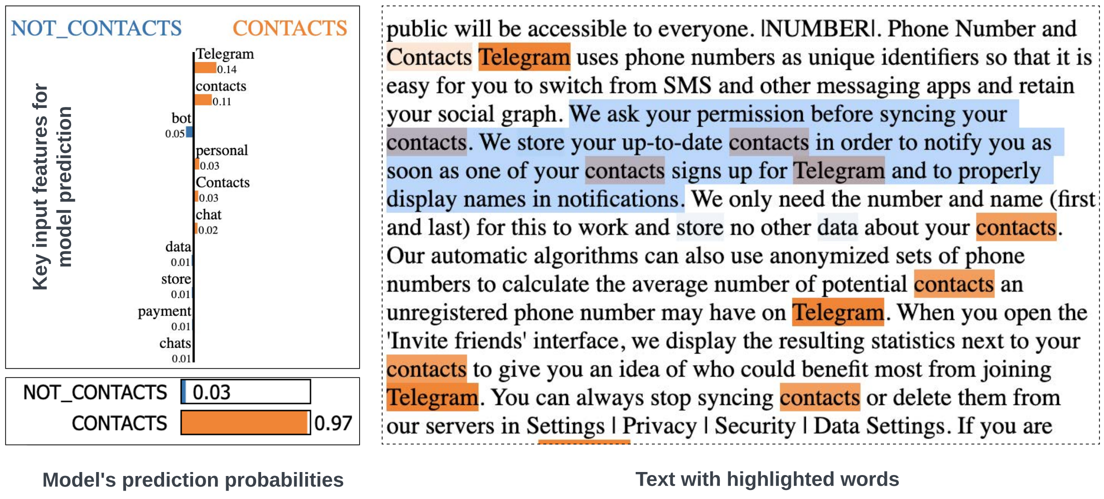
To better assess how the trained model performs with respect to real-world privacy policies, the model's predictions are compared with the annotations made by the human annotators. LIME [17] is employed to explain models' prediction behaviors after randomly selecting three apps from the MPP-270 corpus to examine machine learning models in three cases: the model predictions exactly match with ground truth (true positive), the model predictions contain false positives, and the model predictions contain false negatives. For each case, logistic regression and FastText models are selected. Note that the ground truth for a permission prediction is based on the declared permission in the app manifest file.

1) TELEGRAM MESSENGER – TRUE POSITIVE

Telegram is an instant messaging and video calling app. During policy annotation, Telegram's privacy policy is found entirely transparent and complete with its declared permissions in the manifest file. For model explainability, FastText and logistic regression models trained to predict `CONTACTS` permission are chosen, since `CONTACTS` permission is rarely mentioned in privacy policies. Fig. 7 shows the LIME output for the logistic regression model. It is observed that during this prediction, the dominated text features picked up by the logistic regression model overlap with human annotation. The logistic regression model emphasizes text features such as 'contacts, personal, chat' and predicts `CONTACTS` permission with 96% probability. These features coincide with the sentences annotated by human annotators during the policy annotation task. During FastText model's prediction for `CONTACTS`, it is also observed that the model puts more weight on those text features which overlap with human annotation.

2) PayTM – FALSE NEGATIVE

PayTM is a money transfer and billing app that requests nine out of the ten dangerous permissions. However, during annotation, it is found that the app's privacy policy disclosed only two permission-related data collections (i.e., `PHONE_CALL`, `LOCATION`). One of the undocumented permissions is `SMS`. `SMS` permission is selected to investigate what the machine learning models would predict when permission-relevant information is missing in the privacy policy. During model explainability, the logistic regression model for `SMS` made a false-negative prediction with an 80% probability of not requesting `SMS` permission. This prediction, although incorrect, matches with human evaluation as human annotators did not find any `SMS`-permission relevant information in PayTM's privacy policy. On the other hand, the FastText `SMS` model correctly predicts the app requesting `SMS` permission. Investigating the FastText model's prediction behavior with LIME shows that the features associated with this prediction are irrelevant from a human point of view. For example, FastText correlates features such as 'platform, event, regulation' to predict `SMS` permission, yet these features do not necessarily indicate `SMS` permission from a human point of view.



public will be accessible to everyone. |NUMBER|. Phone Number and Contacts Telegram uses phone numbers as unique identifiers so that it is easy for you to switch from SMS and other messaging apps and retain your social graph. We ask your permission before syncing your contacts. We store your up-to-date contacts in order to notify you as soon as one of your contacts signs up for Telegram and to properly display names in notifications. We only need the number and name (first and last) for this to work and store no other data about your contacts. Our automatic algorithms can also use anonymized sets of phone numbers to calculate the average number of potential contacts an unregistered phone number may have on Telegram. When you open the 'Invite friends' interface, we display the resulting statistics next to your contacts to give you an idea of who could benefit most from joining Telegram. You can always stop syncing contacts or delete them from our servers in Settings | Privacy | Security | Data Settings. If you are

FIGURE 7. Example of LIME output to explain logistic regression model's prediction for `CONTACTS` **permission used in 'Telegram' app. Human annotators annotate those texts highlighted in** blue**, and LIME highlights texts in** orange**. As the figure shows, human annotation overlaps with key features associated with the logistic regression model for predicting** `CONTACTS` **permission.**

3) SOLO LAUNCHER – FALSE POSITIVE

Solo Launcher is a mobile app to personalize the look and feel of phone features. During annotation, it is found that this app's privacy policy contains permission-relevant information not requested by the app (false positive). For example, the privacy policy documents accessing phone book information, yet the app does not declare CONTACTS permission in the manifest file. Using LIME, it is found that both logistic regression and FastText models predict the app 'requesting' CONTACTS permission, based on the app's privacy policy. The models make this false-positive prediction for CONTACTS because of text features such as 'phone calls, users' contact, contacts' present in the privacy policy.

Key Takeaways

- FastText and BERT models perform similarly, but they outperform the logistic regression models. The confusion matrices in Fig. 11 demonstrate that BERT and FastText models are appropriate predictors when class size is heavily imbalanced.
- The obtained results from FastText, BERT, and logistic regression models indicate that the privacy policies contain noisy or no information concerning the permissions labels in certain cases. This can be due to the privacy policies missing relevant information about the target class and multiple apps with different permissions labels sharing the same privacy policy.

VI. TASK 3: PERMISSION-COMPLETENESS LABEL FOR APP PRIVACY POLICY

This section presents the formal definition of the permission-completeness score and presents how *PermPress* leverages MPP-270 dataset (§ IV) and machine learning models trained for permission prediction (§ V) to infer permission-completeness of an app's privacy policy (Fig. 2).

A. PermPress: MACHINE LEARNING-BASED PIPELINE TO CHECK PERMISSION-COMPLETENESS

1) PERMISSION-COMPLETENESS SCORE

An app's *permission-completeness score* is defined by Jaccard's similarity score. Specifically, the permission-completeness score is calculated as

$$\text{permission-completeness score} = |A \cap B|/|A \cup B|$$

where A is the set of dangerous permissions declared in app manifest file and B is the set of those permissions that can be intuitively derived from the natural language text of privacy policy. Both sets A and B contain binary values (0: absence and 1: presence of a permission) and both sets have maximum size of ten. The permission-completeness score is guaranteed to be between 0 and 1, where 1 indicates a complete match and 0 indicates a complete mismatch. Based on the score, the *permission-completeness label* for an app is defined: **label 0 (poor)** for score < 0.5, **label 1 (average)** for score < 1 but ≥ 0.5, and **label 2 (excellent)** for score $= 1$.

2) DATASET AND MACHINE LEARNING MODEL

The annotated corpus MPP-270 is leveraged to calculate the permission-completeness score and corresponding labels. The MPP-270 can be considered the gold standard because the privacy policies in the corpus are manually reviewed and mapped with declared permissions in manifest files. After calculating the permission-completeness score for apps in the MPP-270 corpus, the following distribution of completeness labels is observed: 24 apps with label 0, 161 apps with label 1, and 85 apps with label 2. Subsequently, the trained machine learning models from § V are reused and

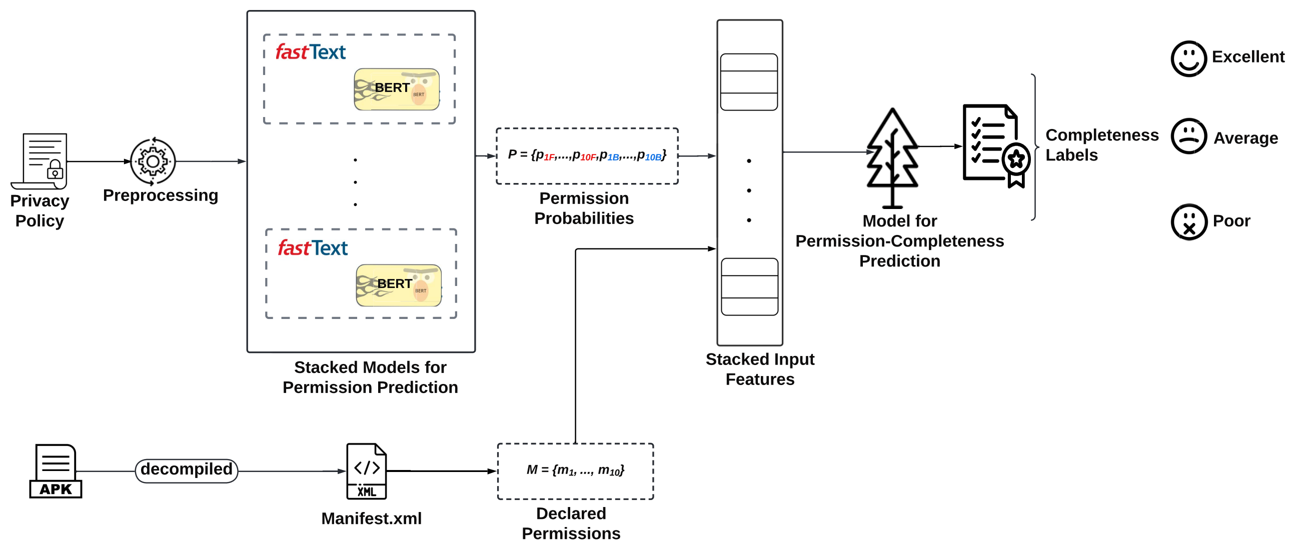


FIGURE 8. Machine learning-based pipeline to infer permission-completeness labels for an app privacy policy. First, binary classifiers (ten BERT and ten FastText models) are used to predict ten dangerous permission groups based on the app privacy policy. Next, a permission vector M is constructed by reverse-engineering the app. The input features are then prepared for predicting the permission-completeness label by concatenating the predicted probability vector P with the declared dangerous permission vector M. The subscripts F and B in P refer to the corresponding prediction probabilities from FastText and BERT models. Based on the input features, the model predicts a permission-completeness label (poor, average, and excellent) for the app's privacy policy.

applied to the MPP-270 dataset to infer the dangerous permission probabilities based on the apps' privacy policies. Permission probabilities from FastText and BERT models are only considered as they outperform logistic regression models in predicting permissions. These predicted probabilities and declared dangerous permissions extracted from manifest files are stacked together to construct feature vectors. Fig. 8 shows the *PermPress* machine learning-based pipeline to infer permission-completeness label.

The feature vector $\boldsymbol{v}$ can be mathematically presented using $\boldsymbol{v} = \langle p_{ij},\ m_j \rangle$ for all $i \in T,\ j \in G$ where,

- G is the set of the ten permissions and T is the set of models (FastText and BERT)
- p_{ij} is the likelihood of an app using a permission belonging to one of the ten permission groups, as predicted by the model, and
- m_j represents binary value (0 or 1) based on whether a corresponding dangerous permission is declared in app manifest file.

3) MODEL TRAINING

Three types of classifiers (logistic regression, random forest, and gradient boosting) were trained with five-fold cross-validation to predict the permission-completeness labels: label 0 (**poor**), label 1 (**average**), and label 2 (**excellent**). Out of the 270 annotated app dataset, 200 apps were randomly sampled for training. Their declared dangerous permissions from app manifest files and class probabilities of those permissions from FastText and BERT models were used as feature vectors (Fig. 8). The remaining 70 apps were used as the test set.

B. RQ3: PERMISSION-COMPLETENESS RESULTS

A grid search was performed to find the optimal classifier. Gradient boosting classifier achieved the highest AUC score of 0.92 with a macro F1-score of 0.78 and a weighted F1-score of 0.83. The following hyperparameters provided the best scores for gradient boosting classifier: number of estimators = 370, learning rate = 0.12, max_depth = 4, and criterion = `MSE`. The predictor's accuracy on the test set is 83%, demonstrating the classifier's performance in accurately predicting permission-completeness labels. Fig. 9 highlights the classifier's ability to distinguish between the three completeness-labels in the presence of class imbalance. Although the majority of the samples in the test set belong to label 1, Fig. 9 shows that a larger portion of samples belonging to label 0 and 2 are classified correctly. For label 2, which corresponds to perfectly complete privacy policies, 75% of the samples are labeled correctly by the classifier on the test set.

Large-scale evaluation. We apply *PermPress* on the 32,000 apps from task-2 (used as test dataset, see details §-V). Only 33% apps are observed to have a perfect completeness-label 2 – meaning their privacy policies are complete ('excellent') and contain relevant information about their declared permissions in manifest files. Approximately 7% of apps have completeness-label 0, referring to their 'poor' condition of privacy policies that these apps do not disclose more than half to none of their dangerous permission-related data collection. The largest group of apps (≈60%) have 'average' privacy policies (completeness-label 1) – they disclose at least half of their dangerous permission-related data collection in privacy policies, but not all of their declared dangerous permissions. From Fig. 10, it is observed that over 50% of

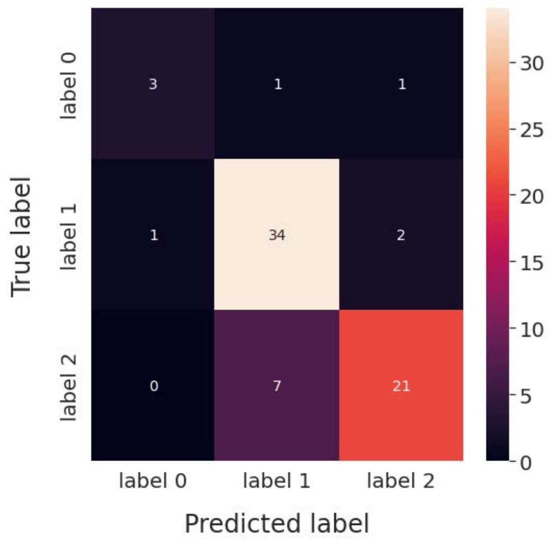


FIGURE 9. Confusion matrix of the Gradient boosting classifier on the 70 apps of the test set. The classifier correctly labels more than half of the samples in each class.

apps across all app categories have 'average' privacy policies, i.e., they do not disclose all of their dangerous permissions in privacy policies, which is concerning. Apps with incomplete privacy policies may receive a monetary penalty and lose customer trust. Moreover, an adversary can hide the sensitive data collection by creating incomplete privacy policies and fooling end-users. Prior works have found privacy non-compliance in a limited set of permission groups (e.g., 41% of 9,050 apps not disclosing location data collection and sharing [10]). The analysis of *PermPress* uncovers broader evidence of privacy non-compliance in all permission groups and apps from all categories. Unlike prior works (e.g., [10], [11]), *PermPress* shows that app policies are incomplete and demonstrates how to measure incompleteness in an automated way.

Key Takeaways

- By combining human annotation with state-of-the-art machine learning and NLP techniques, *PermPress* shows a scalable machine learning-based pipeline to check permission-completeness of an app's privacy policy. *PermPress* achieves AUC score of 0.92 with a macro F1-score of 0.78 and a weighted F1-score of 0.83.
- The majority of apps across app categories have 'average' privacy policies (completeness-label 1) – they disclose at least half of their dangerous permission-related data collection in privacy policies, but omit to disclose at least one dangerous permission in privacy policies.

VII. DISCUSSION

A. SEMANTICALLY-RELEVANT INFORMATION IS OFTEN MISSING

Searching for permission-relevant information in privacy policies is challenging for humans due to the difference in pragmatic meanings of dangerous Android permissions and the natural language text describing the privacy implications of granting these permissions. During policy annotation, it is observed that Android permissions are often not documented directly by their names. Instead, permission-related information is often buried implicitly either in the types of sensitive data collected or in the description of the app's functionalities. As an example of semantic confusion between permission and privacy policy text, 'push notification' in the app privacy policy is unrelated to SMS/MMS permissions (e.g., `RECEIVE_WAP_PUSH`). In Android API pragmatics, push notification is a cloud-based notification messaging system to drive user re-engagement and retention [40]. It does not involve invoking any dangerous permissions. To avoid such ambiguity and be guided by the pragmatic meanings of Android permissions, in this paper, human annotators decide to apply permission labels to those sentences that may indicate an app functionality that can only be logically achieved via accessing dangerous permission. To illustrate this, the annotators decide to apply `STORAGE` labels to sentences that mention' uploading pictures.' This rationale is based upon the pragmatic realization that an app needs access to photo/media storage before giving a user the option to upload pictures. This observation of the subtlety between semantic and pragmatic meanings of Android permissions and privacy policy text is also supported by prior work of Baalous and Poet [16]. The researchers found that upon querying a Sent2Vec model [41] trained on app privacy policies with 'receive wap push' would return semantically similar sentences like "This application may send push-notifications to the user"- which is entirely unrelated to `RECEIVE_WAP_PUSH` permission. The systematic policy annotation task described in § IV reveals that privacy policies rarely disclose permissions by their names, semantic meaning, or by the sensitive data/actions the permissions represent in terms of Android API pragmatics.

B. MACHINE LEARNING MODELS MAY LEARN FROM NOISE

Machine learning models may associate noisy signals to permission labels during inference in the absence of relevant information. The exploratory analysis of privacy policy URLs (§ III-B) corroborate the fact of using recycled (one policy to cover all apps), plagiarized (using someone else's privacy policy as their own), and boilerplate (privacy policy template generators) privacy policies. This near-duplicate training data (i.e., privacy policy text) with varying associated labels (i.e., dangerous permissions accessed by apps) is eventually helping models put more weight on repeated text features in privacy policies.

The model explainability experiment with LIME (§ V-C) also supports that models with complex and robust text-embedding capability (e.g., FastText) could associate features that seem irrelevant to a human evaluator. Still, the model achieves the best-performing precision and recall scores (average macro precision 0.87 and recall 0.79). On the other hand, it is observed that feature-based classifiers like

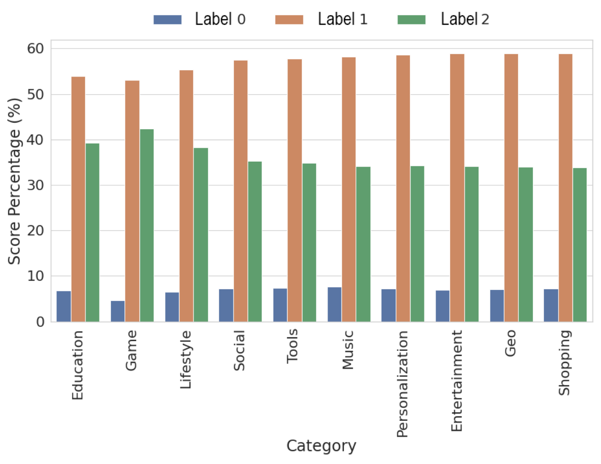


FIGURE 10. Bar chart shows permission-completeness labels (0, 1, and 2) of 32000 apps across ten app categories. The distribution of completeness labels is uniform across app categories: ≈ 33% apps have a perfect completeness label of 2, while the remaining apps are incomplete. Over 50% apps have 'average' privacy policies that do not disclose at least one or more declared permissions. ≈ 7% apps have 'poor' privacy policies, meaning they contain fewer to none of their dangerous permissions.

logistic regression models can associate text features relatable to dangerous permissions from the human standpoint. However, the logistic regression models perform poorly (average micro precision 0.62 and recall 0.75) compared to FastText models. The primary difference is the word-embedding technique between logistic regression and FastText models: logistic regression models use tf-idf weighted features with a limited vocabulary size of 30,000 without considering word order/context. Comparatively, FastText models implement a continuous-bag-of-words model with a word-vectorization process similar to Word2Vec [42], thus capturing more contextual information than logistic regression models. While a large vocabulary with contextual information may help models better infer permission labels, it also restricts models from learning meaningful information. The annotation task shows that permission-relevant snippets constitute only a small fraction of the app privacy policies. It can be hypothesized that model performance might improve if, instead of training models on the full text of app privacy policies, privacy policy snippets can be used as inputs for model training. Privacy policy snippets refer to those portions of the privacy policies that primarily discuss data collection and sharing. The OPP-115 [43] policy taxonomy can be leveraged, for example, to extract privacy policy snippets by identifying those sections of app privacy policies that describe data collection and sharing (e.g., first-party collection and third-party sharing).

C. NEED APP PRIVACY POLICY STANDARDIZATION

The annotation experiment (§ IV) finds that in the absence of a standard guideline to document permission-relevant data collection in the app privacy policies, apps often disclose only the default sensitive data collection (Location, PersistentID) but neglect to disclose other sensitive data access and collection (e.g., face biometrics, calendar data). The MPP-270 corpus shows which permission groups are documented more frequently than others in privacy policies. `PERSISTENTID` and `LOCATION` permission-relevant text snippets are found in 94% and 90% of the app policies in the annotated corpus. `CALENDAR` permission appears the least; only 3 out of 28 apps mention them. `CONTACTS` and `SMS` permission-relevant snippets are in less than 50% of the apps' policies. One significant difference between web and app privacy policies is that the app must disclose all sensitive data along with the usual disclosure of personal information collected. For example, whether the app accesses any sensitive data or resources (e.g., geolocation data from GPS, photo/media gallery, device ID, biometrics data) by invoking any dangerous permission APIs. The app must also disclose data collection and sharing practices with any third-party ads and analytics libraries used for the app development [25], [26]. Prior works on app policy analysis often leverage web privacy policy taxonomy, which does not account for app-specific sensitive data collection/sharing model. It is important to standardize the structure and organization of privacy policy to accommodate the permission-based data collection/sharing model. For example, the existing privacy policy taxonomy can be revised and extended to cover the rich set of sensitive data accessed and collected by apps through its permission-based data collection model.

VIII. LIMITATIONS

For the policy annotation task, 270 popular apps are sampled from the original dataset of 164,156 apps. However, the ranking criteria (number of installs, user rating) could bias the selection as the content and transparency of privacy policies could vary between popular and non-popular apps. Another limitation is that privacy policy annotation is a qualitative process subject to the understanding and interpretation of individual annotators; even privacy experts might disagree on the interpretation of privacy policies [44]. Although the annotators tried to be logical and objective in interpreting app privacy policies, they could have discarded permission-relevant snippets because of the conservative inclusion criteria. The current work focused on examining the transparency of app privacy policies and did not verify whether apps invoked declared permissions through taint analysis. Future work can enhance prediction performance by combining static/dynamic analysis with the proposed pipeline of *PermPress*. In this paper, we choose Android apps for current analysis and methodology, as Android controls the mobile OS market with a 72.92% share [45]. However, the current approach presented here can be further adapted to iOS apps with minor modifications.

IX. RELATED WORK

This section presents and discusses the related work from two dimensions: investigating the permission-driven model and various side-channel/metadata of Android apps to detect privacy leakage in apps and investigating privacy compliance checking between the app privacy policy and app behavior. The section also discusses how this work differs from others.

A. DETECTING PRIVACY LEAKAGE IN APPS

Research in privacy leakage detection can be broadly divided into (i) source code analysis and (ii) meta-information analysis.

1) SOURCE-CODE ANALYSIS

The source code-analysis approach identifies code implementations in apps by instrumenting static, dynamic, or hybrid analysis to identify the invocation of dangerous permission APIs granted to an app. However, the app may not call the APIs/operations exposed by the permissions for its functional purpose, or the app might unnecessarily request dangerous permission when a lower-privileged counterpart of the dangerous permission would still provide the app's required functionality. Source code analysis techniques attempt to identify these 'overprivileged apps,' i.e., apps granted with unused or non-essential higher-privileged permissions. For example, Felt *et al.* built Stowaway [46], a static analysis tool that determines an app's API calls and then maps the API call to system-level permission to check whether the app is overprivileged. Stowaway is one of the earliest overprivileged-app detection tools that systematically curated the list of dangerous permission APIs (Android 2.2) through automatic unit test case generation for API calls, Content Providers, and Intents. Stowaway detected permission over-privilege issues in one-third of 940 Google Play Store apps. To improve Stowaway's coverage of Android APIs and create a version-independent static analysis tool for extracting permission specifications from Android OS, PScout (compatible with Android 4.0) [47] and later Axplorer (compatible up to Android 5.1) [48] were built. These tools have contributed largely to building various permission-overprivileged detection tools. Bartel *et al.* [49] introduced Spark-Android to demonstrate over-privileged apps on the Google Play Store and found that 124 out of the 679 applications declared one or more unused permissions. RNPDroid [50] is a malware detection tool that leverages the machine learning-based clustering technique to analyze permissions declared by apps. It assigns risk factors (high, medium, low, and no risk) to an app, with an accuracy of 97.48%. SensDroid [51] improves the performance of RNPDroid by incorporating features such as Android Intents and permissions in developing a statistical model for malware detection through sensitivity analysis and achieves an accuracy of 98.65%. There exist tools that apply various machine learning techniques along with static or dynamic code analysis to detect Android malware, e.g., MaMaDroid [52], Alde [53], DroidDet [54], SAMADroid [55], AVIS [56], and AndroDialysis [57] (for a complete review of tools and methods to detect Android malware and privacy leakage, see [58], [59]). In contrast to these prior works on machine learning-based malware analysis, *PermPress* provides an end-to-end machine learning system to identify suspicious apps based on reliable detection of incomplete privacy policies.

2) APP METADATA ANALYSIS

Besides permission analysis, prior works also analyze user interfaces to identify sensitive user inputs and detect any private data leakage through these user interfaces or other related side-channels. This line of work includes, for example, SUPOR [60], UIPicker [61], GUILeak [62], Privet [63], IconIntent [64], DeepIntent [65], and side-channel based leakage [31]. This line of research identifies discrepancies between privacy practices described (or omitted) in privacy policies and actual code implementation. Meta-information-based privacy compliance checking complements static analysis with text data (e.g., app description, code documentation, user reviews). The commonly used NLP techniques are first-order logic [66], topic analysis [67], and part-of-speech tagging [68] to characterize whether apps are over-privileged with unnecessary, dangerous permission APIs. Our work builds on the insights of these previous works and complements them by empirically measuring app policies' incompleteness in disclosing dangerous permission-related data collection practices.

B. CHECKING PRIVACY COMPLIANCE OF APPS

App privacy policies are often lengthy and filled with vague and ambiguous language, making it difficult for humans to read and determine the app's data privacy practices. Recent years have seen an increasing interest in usable privacy policy research by combining human annotation with state-of-the-art machine learning and NLP techniques to extract features from natural language privacy policies to check whether apps comply with privacy laws.

1) NLP-BASED PRIVACY POLICY ANALYSIS

For example, Sunayev *et al.* [69] investigated 600 popular mHealth apps and found that only 30% of apps had privacy policies, and two-thirds of these privacy policies did not specifically address the app itself. Shipp and Blasco [70] analyzed 30 women's reproductive-health apps and uncovered that privacy policies often neglect to disclose reproductive-health-related data, though the apps collect such data and transfer them to remote servers. Feal *et al.* [71] studied 46 parental control Android apps and found that 72% of the apps shared personal data with third parties without disclosing this sharing practice in privacy policies. Onik *et al.* [72] found that app publishers and associated owners could track end-users by aggregating private data such as contact number, biometric ID, email, location, and social graph. Slavin *et al.* [14] studied privacy violations of Android apps by developing a privacy-policy-phrase ontology and a collection of mappings from API methods to policy phrases. Using this mapping framework, Slavin *et al.* evaluated 477 apps semi-automatically and found 341 privacy violations when apps did not disclose privacy practices in privacy policies at all or used vague language to disclose the practice. Other tools like PPChecker [12] detect privacy non-compliance issues in android apps and their privacy laws. PPChecker uses hand-coded NLP rules with parts-of-speech tagging to

TABLE 4. Comparison of *PermPress* with state-of-the-art privacy compliance checking tools. Note that the reported average F1 scores reflect the corresponding tool's machine learning performance based on privacy policy-related task. Except for Baalous and Poet [16], these F1 scores do not necessarily correspond to the final result of the tools, for example, the reported F1-score for *PermPress* refers to the permission prediction task (see § V for details). Among these four tools, only *PermPress* covers all ten dangerous permission groups in its permission inference task. *PermPress* also releases the largest annotated corpus (MPP-270) that systematically maps privacy policies to all dangerous permission groups. ML = Machine Learning.

Tool	Platform	Methods	ML Models	Goal for ML Component	Permission Groups	Avg. F1 (%)	Dataset Availability
MAPS [11]	Android	Static analysis, ML	Logistic regression	Detect privacy practice description (data type, party, modality) in privacy policies.	3 (Contacts, Location, PersistentID)	91.0	✓
PPChecker [12]	Android	Static analysis, ML	Hand-coded rules, POS tagging, Max entropy, SVM, Random forest, Naïve Bayes	Identify the sentences related to information collection, usage, retention, and disclosure in privacy policies.	6 (Camera, Calendar, Contacts Microphone, PersistentID, Phone)	90.2	✗
Baalous and Poet [16]	Android	ML	Sent2Vec	Find semantically related phrases to the dangerous permissions in privacy policies.	9 (Calendar, Camera, Contacts, Location, Microphone, Phone, Sensors, SMS, Storage)	20.0	✗
PermPress	**Android**	**Static analysis, ML**	**Logistic regression, FastText, BERT**	**Infer dangerous permissions based on apps' privacy policies.**	**10 (Calendar, Camera, Contacts, Location, Microphone, PersistentID, Phone_Call, Sensors, SMS, Storage)**	**94.0**	✓ [18]

identify first- and third-party data collection practices. *PermPress* is significantly different from PPChecker by demonstrating that permission-related information is not documented explicitly in privacy policies; thus, hand-derived NLP rules are brittle to checking the completeness of privacy policies.

2) ML-BASED PRIVACY POLICY ANALYSIS

Zimmeck *et al.* [10] showed the viability of combining machine learning-based privacy policy analysis with static code analysis of apps. Their results found that 71% of 17,991 apps that accessed sensitive data through dangerous permissions did not have a privacy policy. Verderame *et al.* [13] built 3PDroid to evaluate 5,057 Android apps from Google Play Store and found that 94.5% of apps are non-compliant with the app marketplace privacy requirements, such as missing privacy links inside apps while collecting or sharing sensitive data. Luo *et al.* [15] studied 16,162 apps across six Android marketplaces (Xiaomi, Baidu, 360, Tencent, SnapPeam, Google Play) and found a broad range of privacy non-compliance such as lacking privacy policy links, collecting private data before informing users, requesting unnecessarily dangerous permissions. Zimmeck *et al.* [11] developed MAPS, a privacy compliance checking system that investigated 1,035,853 Android apps and found broad evidence of potential non-compliance. Unsurprisingly, privacy policies are often silent about the data practices of apps, and 12.1% of such apps have at least one location-related potential compliance issue. Zimmeck *et al.* [11] also released the APP-350 policy corpus that labels various data collection practices, including first/third party sharing of location data access patterns and identifier collection. One significant difference between our work and MAPS is that *PermPress* systematically evaluates privacy policies to map between dangerous permission-based data collection and natural language-based privacy policy texts. This systematic evaluation provides a labeled policy corpus MPP-270 to examine the limitations of machine-learning models (due to incomplete privacy policies) and build a pipeline to predict the permission-completeness score. The systematic exploration conducted in our work uncovers broader privacy non-compliance of apps (67%) in multiple permission groups rather than few (e.g., `LOCATION`). Another related work is Baalous and Poet [16], who trained a Sent2Vec model on privacy policies and later queried the model with a string of dangerous permission (e.g., `READ_CONTACTS`) to see if the model could retrieve useful semantic information from privacy policies. The researchers claimed that the model could find semantically relevant permission information from the privacy policy for some of the permissions, such as `CAMERA` and `LOCATION`, but failed in other cases, such as `SENSORS` and `CALENDAR`. In contrast, our current work empirically finds that permission information is often hidden under the description of operational features or functions of an app, and thus finding semantic information about dangerous permissions is a challenging task for machine learning models. Table 4 shows the difference between *PermPress* and current state-of-the-art machine learning based privacy compliance checking tools.

X. CONCLUSION

In this paper, we propose *PermPress*, which is a machine learning-based pipeline to check whether Android apps' privacy policies match their permissions. *PermPress* leverages a novel policy corpus named MPP-270 that allows for (i) identifying text-based features that link between the permission-based model of sensitive data collection to privacy disclosures; (ii) explaining machine learning model predictions by comparing the machine learning model's features with human-annotated features; and (iii) establishing ground truth to build a machine learning-based pipeline to predict permission-completeness labels. The MPP-270 corpus reveals that `CALENDAR`, `SMS`, and `CONTACTS` are rarely documented ($<50\%$) in privacy policies, whereas `PERSISTENTID` and `LOCATION` are frequently mentioned ($\geq 90\%$). By leveraging the MPP-270 corpus and machine learning techniques, *PermPress* achieves an AUC score of 0.92 to predict the permission-completeness for an app. A large-scale evaluation of 164, 156 Android apps shows that, on average, 7% apps do not disclose more than half of their

TABLE 5. Results from Logistic Regression models: high false positive rates, especially in predicting permission groups where class distribution is highly imbalanced (e.g., PERSISTENTID (majority of positive classes), SMS (minority of positive classes)). M = Macro, W = Weighted.

Dangerous Permission	AUC	F1 (W)	F1 (M)	Precision (W)	Precision (M)	Recall (W)	Recall (M)
SENSOR	0.91	0.98	0.62	0.99	0.57	0.97	0.91
PHONE_CALL	0.86	0.93	0.73	0.95	0.68	0.91	0.86
CALENDAR	0.84	0.81	0.51	0.97	0.55	0.72	0.84
MICROPHONE	0.84	0.85	0.69	0.92	0.66	0.82	0.84
CONTACTS	0.82	0.80	0.67	0.90	0.66	0.77	0.82
CAMERA	0.81	0.75	0.69	0.87	0.70	0.73	0.81
LOCATION	0.72	0.72	0.72	0.72	0.72	0.72	0.72
STORAGE	0.69	0.72	0.70	0.73	0.72	0.73	0.69
SMS	0.50	1.00	0.50	1.00	0.50	1.00	0.50
PERSISTENTID	0.50	0.96	0.49	0.94	0.49	0.97	0.50
Avg. Scores	0.75	0.85	0.63	0.90	0.62	0.83	0.75

TABLE 6. Results from FastText models. Similarly, high false positive rates for PERSISTENTID because majority of labels are positive. Also, high false negative rates for SMS because majority of labels are negative. M = Macro, W =Weighted.

Dangerous Permission	AUC	F1 (W)	F1 (M)	Precision (W)	Precision (M)	Recall (W)	Recall (M)	Accuracy
CALENDAR	0.94	0.99	0.88	0.99	0.93	0.99	0.84	0.99
CAMERA	0.93	0.90	0.85	0.90	0.86	0.91	0.83	0.91
PHONE_CALL	0.93	0.97	0.85	0.97	0.89	0.97	0.82	0.97
CONTACTS	0.93	0.93	0.83	0.93	0.86	0.93	0.80	0.93
SENSOR	0.93	1.00	0.85	1.00	0.91	1.00	0.80	1.00
MICROPHONE	0.92	0.94	0.83	0.94	0.87	0.95	0.80	0.95
LOCATION	0.92	0.85	0.85	0.85	0.85	0.85	0.85	0.85
STORAGE	0.91	0.85	0.84	0.85	0.84	0.85	0.84	0.85
PERSISTENTID	0.89	0.98	0.79	0.98	0.88	0.98	0.74	0.98
SMS	0.88	1.00	0.67	1.00	0.85	1.00	0.61	1.00
Avg. Scores	0.92	0.94	0.82	0.94	0.87	0.94	0.79	0.94

TABLE 7. BERT models have comparatively lower precision scores than FastText models, suggesting room for improvement as BERT models have input constraints of maximum sequence length. M = Macro, W = Weighted.

Dangerous Permission	AUC	F1 (W)	F1 (M)	Precision (W)	Precision (M)	Recall (W)	Recall (M)	Accuracy
CALENDAR	0.94	0.92	0.64	0.97	0.60	0.90	0.88	0.90
PHONE_CALL	0.92	0.86	0.61	0.95	0.60	0.81	0.84	0.81
SENSOR	0.92	0.97	0.57	0.99	0.54	0.94	0.87	0.94
MICROPHONE	0.90	0.81	0.63	0.91	0.63	0.76	0.81	0.76
CAMERA	0.90	0.82	0.76	0.73	0.86	0.81	0.82	0.81
CONTACTS	0.90	0.84	0.71	0.89	0.68	0.81	0.82	0.81
SMS	0.88	0.98	0.54	1.00	0.52	0.97	0.75	0.97
LOCATION	0.86	0.77	0.77	0.77	0.77	0.77	0.77	0.77
PERSISTENTID	0.86	0.81	0.49	0.79	0.54	0.71	0.78	0.71
STORAGE	0.85	0.77	0.76	0.77	0.75	0.76	0.76	0.76
Avg. scores	0.89	0.86	0.65	0.88	0.65	0.82	0.81	0.82

dangerous permissions in privacy policies, whereas 60% apps omit to disclose at least one dangerous permission-related data collection in privacy policies. We have released the MPP-270 annotated corpus and the policy corpus of 164, 156 apps for future research. We believe that *PermPress* holds promise in uncovering the incomplete landscape of app privacy policies and automating the compliance checking process for end-users like privacy regulators, marketplace operators, and app publishers/developers.

APPENDIX A
PERMISSION INFERENCE RESULTS USING LOGISTIC REGRESSION, FASTTEXT, AND BERT

See Tables 5–7.

APPENDIX B
COMPARING BERT AND FASTTEXT MODELS: CONFUSION MATRICES

See Fig. 11.

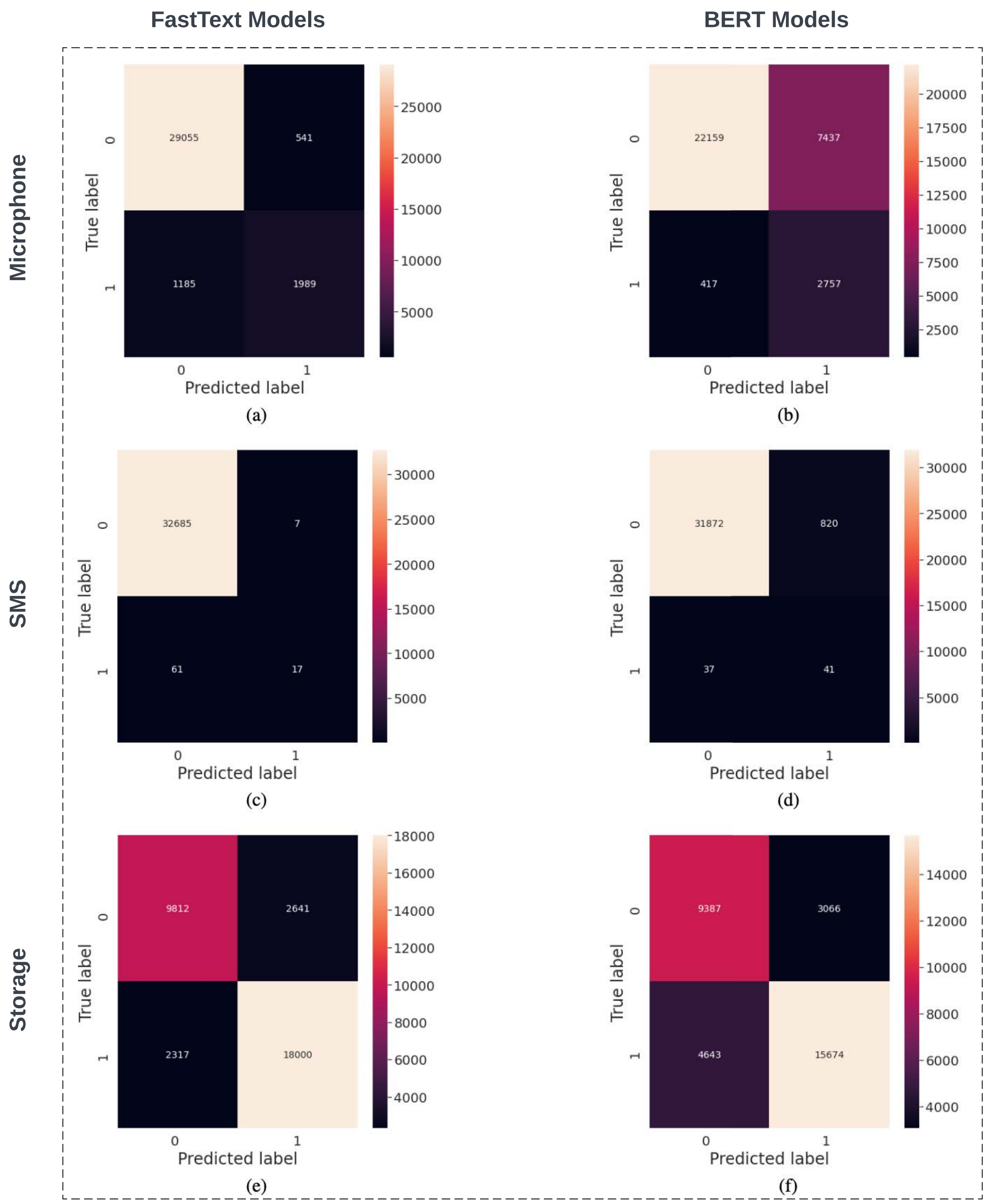


FIGURE 11. Confusion matrices of the FastText and BERT models, showing permission prediction performance on the test set. The left column (a, c, e) shows FastText models, and the right column (b, d, f) shows the BERT models. From the top: the top row (a, b) shows prediction results for `MICROPHONE`, the middle row (c, d) for `SMS`, the bottom row (e, f) for `STORAGE`. Despite the highly imbalanced positive class sizes, both FastText and BERT models learn to recognize true-positive samples. BERT models are on-par in performance with FastText models even after applying input truncation. BERT models' performance shows support for implementing the weighted cross-entropy loss as the appropriate objective function to handle class-imbalance issues.

REFERENCES

[1] Federal Trade Commission. *FTC vs Path | Complaint For Civil Penalties, Permanent Injunction, and Other Relief.* Accessed: Dec. 1, 2021. [Online]. Available: https://www.ftc.gov/sites/default/files/documents/cases/2013/02/130201pathinccmpt.pdf

[2] D. J. Solove and W. Hartzog, "The FTC and the new common law of privacy," *Colum. L. Rev.*, vol. 114, no. 3, p. 583, Apr. 2014.

[3] BBC. (Sep. 2021). *WhatsApp Issued Second-Largest GDPR Fine of 225m—BBC News*. Accessed: Feb. 17, 2022. [Online]. Available: https://www.bbc.com/news/technology-58422465

[4] K. Harris. (Jan. 2013). *Privacy On The Go—Recommendations for the Mobile Ecosystem*. Accessed: Nov. 28, 2020. [Online]. Available: https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/privacy_on_the_go.pdf

[5] A. M. McDonald and L. F. Cranor, "The cost of reading privacy policies," *I/S, A J. Law Policy Inf. Soc.*, vol. 4, no. 3, p. 543, Winter 2009.

[6] Stack Overflow. *Android—What can I do About a Privacy URL?* Accessed: Nov. 29, 2021. [Online]. Available: https://stackoverflow.com/questions/13597666/what-can-i-do-about-a-privacy-url?

[7] R. Sun and M. Xue, "Quality assessment of online automated privacy policy generators: An empirical study," in *Proc. Eval. Assessment Softw. Eng.*, Apr. 2020, pp. 270–275.

[8] S. Wilson, F. Schaub, F. Liu, K. M. Sathyendra, D. Smullen, S. Zimmeck, R. Ramanath, P. Story, F. Liu, N. Sadeh, and N. A. Smith, "Analyzing privacy policies at scale: From crowdsourcing to automated annotations," *ACM Trans. Web*, vol. 13, no. 1, pp. 1–29, Feb. 2019.

[9] V. B. Kumar, R. Iyengar, N. Nisal, Y. Feng, H. Habib, P. Story, S. Cherivirala, M. Hagan, L. Cranor, S. Wilson, F. Schaub, and N. Sadeh, "Finding a choice in a haystack: Automatic extraction of opt-out statements from privacy policy text," in *Proc. Web Conf.*, Apr. 2020, pp. 1943–1954.

[10] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. M. Bellovin, and J. Reidenberg, "Automated analysis of privacy requirements for mobile apps," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–11.

[11] S. Zimmeck, P. Story, D. Smullen, A. Ravichander, Z. Wang, J. Reidenberg, N. C. Russell, and N. Sadeh, "MAPS: Scaling privacy compliance analysis to a million apps," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 66–86, Jul. 2019.

[12] L. Yu, X. Luo, J. Chen, H. Zhou, T. Zhang, H. Chang, and H. K. N. Leung, "PPChecker: Towards accessing the trustworthiness of Android apps' privacy policies," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 221–242, Feb. 2021.

[13] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, "On the (Un) reliability of privacy policies in Android apps," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–9.

[14] 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 *Proc. 38th Int. Conf. Softw. Eng.*, May 2016, pp. 25–36.

[15] Q. Luo, Y. Yu, J. Liu, and A. Benslimane, "Automatic detection for privacy violations in Android applications," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 6159–6172, Apr. 2022.

[16] R. Baalous and R. Poet, "Utilizing sentence embedding for dangerous permissions detection in Android apps' privacy policies," *Int. J. Inf. Secur. Privacy*, vol. 15, no. 1, pp. 173–189, Jan. 2021.

[17] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 1135–1144.

[18] M. S. Rahman, P. Naghavi, B. Kojusner, S. Afroz, B. Williams, S. Rampazzi, and V. Bindschaedler. (Aug. 2022). *MPP-270: Annotated Policy Corpus to Map Between Permission and Privacy Disclosures*. [Online]. Available: https://sites.google.com/view/permpress/

[19] M. Theoharidou, A. Mylonas, and D. Gritzalis, "A risk assessment method for smartphones," in *Proc. IFIP Int. Inf. Secur. Conf.* Berlin, Germany: Springer, 2012, pp. 443–456.

[20] Android Developers. *Permissions on Android*. Accessed: Aug. 29, 2021. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview

[21] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, vol. 10, 1st Ed. Cham, Switzerland: Springer, 2017, pp. 10–5555.

[22] G. Hadjipetrova and H. G. Poteat, "States are coming to the force of privacy in the digital era," *Landslide*, vol. 6, no. 6, pp. 13–71, Jul./Aug. 2014.

[23] W. Stallings, "Handling of personal information and deidentified, aggregated, and pseudonymized information under the California consumer privacy act," *IEEE Secur. Privacy*, vol. 18, no. 1, pp. 61–64, Jan. 2020.

[24] Federal Trade Commission, "Children's online privacy protection rule ('COPPA')," *Children's Online Privacy Protection Act*, vol. 15, pp. 6501–6505, 1998. Accessed: Dec. 22, 2021. [Online]. Available: https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa

[25] Google PlayStore. (2020). *Prepare Your APP for Review—Play Console Help*. Accessed: May 6, 2021. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9859455?hl=en

[26] TermsFeed. *Do I Need Separate Privacy Policies For My Website and Mobile App?*. Accessed: Jul. 7, 2021. [Online]. Available: https://www.termsfeed.com/blog/separate-privacy-policies-website-app/

[27] C. Castelluccia, S. Guerses, M. Hansen, J. Hoepman, J. van Hoboken, B. Vieira. (2017). *Privacy and Data Protection in Mobile Applications: A study on the App Development Ecosystem and the Technical Implementation of GDPR*. [Online]. Available: https://www.enisa.europa.eu/publications/privacy-and-data-protection-in-mobile-applications/at_ 1149 download/fullReport

[28] S. Bird and E. Loper, "NLTK: The natural language toolkit," in *Proc. ACL Interact. Poster Demonstration Sessions*. Barcelona, Spain: Association Computational Linguistics, 2004, pp. 214–217.

[29] A. Desnos and G. Gueguen. *AndroGuard: Reverse Engineering, Malware and Goodware Analysis of Android Applications*. Accessed: Jun. 24, 2022. [Online]. Available: https://github.com/androguard/androguard

[30] Android Developers. *Android API Reference | Android.Manifest.Permission*. Accessed: Mar. 24, 2021. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[31] J. Reardon, A. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 603–620.

[32] P. Calciati, K. Kuznetsov, A. Gorla, and A. Zeller, "Automatically granted permissions in Android apps: An empirical study on their prevalence and on the potential threats for privacy," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, Jun. 2020, pp. 114–124.

[33] M. L. McHugh, "Interrater reliability: The Kappa statistic," *Biochem. Med.*, vol. 22, no. 3, pp. 276–282, 2012.

[34] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics*, Valencia, Spain, vol. 2, Apr. 2017, pp. 427–431.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[37] H. T. Madabushi, E. Kochkina, and M. Castelle, "Cost-sensitive BERT for generalisable sentence classification on imbalanced data," in *Proc. 2nd Workshop Natural Lang. Process. Internet Freedom, Censorship, Disinformation, Propaganda*. Hong Kong, China: Association Computational Linguistics, 2019, pp. 125–134.

[38] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Davison, "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empirical Methods Natural Lang. Process., Syst. Demonstrations*, 2020, pp. 38–45.

[39] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?" in *Proc. China Nat. Conf. Chin. Comput. Linguistics*. Cham, Switzerland: Springer, 2019, pp. 194–206.

[40] Firebase. *Firebase Cloud Messaging | Firebase Documentation*. Accessed: Nov. 29, 2021. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/

[41] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional N-gram features," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.* Stroudsburg, PA, USA: Association Computational Linguistics, Jun. 2018, pp. 528–540.

[42] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[43] S. Wilson, F. Schaub, A. A. Dara, F. Liu, S. Cherivirala, P. G. Leon, M. S. Andersen, S. Zimmeck, K. M. Sathyendra, N. C. Russell, and T. B. Norton, "The creation and analysis of a website privacy policy corpus," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 1330–1340.

[44] J. R. Reidenberg, T. Breaux, L. F. Cranor, B. French, A. Grannis, J. Graves, F. Liu, A. McDonald, T. Norton, R. Ramanath, N. C. Russell, N. Sadeh, and F. Schaub, "Disagreeable privacy policies: Mismatches between meaning and userss understanding," *Berkeley Tech. LJ*, vol. 30, no. 1, p. 39, Spring 2015.

[45] Statista. (Nov. 2020). *Mobile Operating Systems Market Share Worldwide From Jan. 2012 To October 2020*. Accessed: Dec. 2, 2020. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

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

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

[48] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On demystifying the Android application framework: Re-visiting Android permission specification analysis," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 1101–1118.

[49] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon, "Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing Android," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 617–632, Jun. 2014.

[50] K. Sharma and B. B. Gupta, "Mitigation and risk factor analysis of Android applications," *Comput. Elect. Eng.*, vol. 71, pp. 416–430, Oct. 2018.

[51] G. Shrivastava and P. Kumar, "SensDroid: Analysis for malicious activity risk of Android application," *Multimedia Tools Appl.*, vol. 78, no. 24, pp. 35713–35731, Dec. 2019.

[52] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," *ACM Trans. Priv. Secur.*, vol. 22, no. 2, pp. 1–16, Apr. 2019.

[53] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the Android ecosystem," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1184–1199, May 2020.

[54] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of Android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, Jan. 2018.

[55] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A novel 3-level hybrid malware detection model for Android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[56] H. Kim, T. Cho, G.-J. Ahn, and J. H. Yi, "Risk assessment of mobile applications based on machine learned malware dataset," *Multimedia Tools Appl.*, vol. 77, no. 4, pp. 5027–5042, Feb. 2018.

[57] K. Sharma and B. B. Gupta, "Towards privacy risk analysis in Android applications using machine learning approaches," *Int. J. E-Services Mobile Appl.*, vol. 11, no. 2, pp. 1–21, Apr. 2019.

[58] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[59] G. Shrivastava and P. Kumar, "Android application behavioural analysis for data leakage," *Exp. Syst.*, vol. 38, no. 1, Jan. 2021, Art. no. e12468.

[60] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "SUPOR: Precise and scalable sensitive user input detection for Android apps," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 977–992.

[61] Y. Nan, Z. Yang, M. Yang, S. Zhou, Y. Zhang, G. Gu, X. Wang, and L. Sun, "Identifying user-input privacy in mobile applications at a large scale," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 3, pp. 647–661, Mar. 2017.

[62] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu, "GUILeak: Tracing privacy policy claims on user input data for Android applications," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 37–47.

[63] L. L. Zhang, C.-J. M. Liang, Z. L. Li, Y. Liu, F. Zhao, and E. Chen, "Characterizing privacy risks of mobile apps with sensitivity analysis," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 279–292, Feb. 2018.

[64] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "IconIntent: Automatic identification of sensitive UI widgets based on icon classification for Android apps," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 257–268.

[65] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, and J. Lu, "DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2421–2436.

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

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

[68] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. N. Leung, "Enhancing the description-to-behavior fidelity in Android apps with privacy policy," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 834–854, Sep. 2018.

[69] A. Sunyaev, T. Dehling, P. L. Taylor, and K. D. Mandl, "Availability and quality of mobile health app privacy policies," *J. Amer. Med. Inform. Assoc.*, vol. 22, no. 1, pp. 28–33, Apr. 2015.

[70] L. Shipp and J. Blasco, "How private is your period?: A systematic analysis of menstrual app privacy policies," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 4, pp. 491–510, Oct. 2020.

[71] A. Feal, P. Calciati, N. Vallina-Rodriguez, C. Troncoso, and A. Gorla, "Angel or devil? A privacy study of mobile parental control apps," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 2, pp. 314–335, 2020.

[72] M. M. H. Onik, C.-S. Kim, N.-Y. Lee, and J. Yang, "Personal information classification on aggregated Android application's permissions," *Appl. Sci.*, vol. 9, no. 19, p. 3997, Sep. 2019.

MUHAMMAD SAJIDUR RAHMAN received the M.Sc. degree in computer science from Kansas State University, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Science and Engineering, University of Florida. Previously, he worked as a Software Engineer at Fintech industry for over four years. His research interests include intersection of software security, privacy engineering, threat modeling, NLP/ML, and human-centered computing. See sajid-rahman.com for more details.

PIROUZ NAGHAVI (Graduate Student Member, IEEE) received the B.A.Sc. degree in mechanical engineering from the University of British Columbia, Canada, in 2015, and the M.S. degree in electrical and computer engineering from the University of Washington, USA, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer and Information Science and Engineering, University of Florida, USA. His research interests include static and dynamic analysis of mobile systems, cyber-physical systems security, machine learning in privacy preservation technologies, side-channel information recovery, and security.

BLAS KOJUSNER received the B.Sc. degree in computer science from the University of Florida, in 2020, where he is currently pursuing the M.Sc. degree with the Department of Computer and Information Science and Engineering. His research interests include mobile systems security, reverse engineering, and data privacy.

SADIA AFROZ is currently a Staff AI Scientist at Avast Software and a Principal Investigator at the International Computer Science Institute (ICSI). Her research interests include anti-censorship, anonymity, and adversarial learning. She received the Best Student Paper Award at the 2012 Privacy Enhancing Technology Symposium (PETS), the 2013 Privacy Enhancing Technology (PET) Award for her work on adversarial authorship attribution, and the 2014 ACM SIGSAC Dissertation Award (runner-up). See her https://www1.icsi.berkeley.edu/∼sadia/website for details.

BYRON WILLIAMS received the Ph.D. degree in computer science from the Mississippi State University, in 2009. He is currently an Assistant Professor with the Department of Computer and Information Science and Engineering, University of Florida. His research has focused on the intersection of software engineering and cybersecurity. His current research interests include investigating approaches to secure software development, vulnerability assessment using static and dynamic analysis, and security modeling applying statistical and machine learning techniques.

SARA RAMPAZZI (Member, IEEE) is currently an Assistant Professor at the Department of Computer and Information Science and Engineering, University of Florida. Her research interests include cyber-physical systems security, embedded systems design, modeling, and simulation with applications in healthcare, autonomous systems, and the Internet of Things. She has investigates security and privacy risks and designs defense strategies for trustworthy cyber-physical systems despite emerging attacks. Find out more on `sararampazzi.com`.

VINCENT BINDSCHAEDLER (Member, IEEE) received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, in 2018. He is currently an Assistant Professor with the Department of Computer and Information Science and Engineering, University of Florida. His research interests include data privacy, applied cryptography, and privacy-preserving technologies. His current work focuses on emerging problems at the intersection of machine learning with security and privacy.