THESIS

PRIVACY THREATS TO MOBILE HEALTH APPS: AN ANALYSIS OF DATA COLLECTION PRACTICES

Submitted by

Charles Ethan Myers

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2025

Master's Committee:

Advisor: Indrakshi Ray

Francisco Ortega
Indrajit Ray
Anura Jayasumana

ABSTRACT

PRIVACY THREATS TO MOBILE HEALTH APPS: AN ANALYSIS OF DATA COLLECTION PRACTICES

Users often install mobile health applications (mHealth apps) to improve their health and lifestyle. mHealth apps collect sensitive personal health related information and may share them with various stakeholders. Many of these mHealth apps that consumers use for their personal lifestyle benefits are not required to be compliant with any regulation, such as the Health Insurance Portability and Accountability Act (HIPAA) or General Data Protection Regulation (GDPR). Our investigation reveals that there is a mismatch between what an app description states about privacy, what permissions it requests from the end user as declared in its manifest file, privacy regulations (GDPR), and what privacy practices are actually enforced by the app.

We provide a formal definition of mHealth apps and discuss an automated approach that uses a pre-trained language model to identify and analyze 13,177 mHealth apps from Google Playstore. We identify the ten most common privacy threats in mHealth apps and map them to GDPR policy violations. Privacy violations pertaining to GDPR include the absence of a consent management system, inconsistent permissions with respect to the app description, and sharing personally identifiable information (PII) without consent. Our analysis reveals that only 4.28% had a consent mechanism, over 88% of app network transmissions shared some form of personally identifiable information (PII) without consent, and nearly 83.7% requested permissions from the users without explaining their use cases. Our research has been successful in building automated tools for detecting privacy violations for some, but not all, of the identified threats.

ACKNOWLEDGEMENTS

First of all, I would like to sincerely thank Indrakshi Ray for serving as my thesis advisor. Without her patience, support, and guidance, this thesis would have never come to fruition. I would also like to thank my mother and twin sister for their support throughout this rather stressful time in my life. Last, this work was supported in part by NIST funding under Award Number 60NANB23D152 and NSF under Award Numbers CNS 2335687, DMS 2123761, CNS 1822118, NIST, ARL, Statnett, AMI, NewPush and Cyber Risk Research.

DEDICATION

This thesis is dedicated to my mother, my twin sister, and the greatest dog of all time, Snowy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Necessity and Challenges

There has been a rise in the use of electronic health care, which is made possible by mobile Android devices. These devices can host a variety of mobile health apps (mHealth apps). With nearly 3.3 billion Android users [1] and a rising number of mHealth apps, it raises questions about non-compliant behaviors these apps exhibit. We focus on mHealth apps as they often collect sensitive personal identifiable information (PII) and other health-related data. Thus, mHealth apps should be examined under more scrutiny than standard apps. To examine these non-compliant behaviors this thesis proposes a set of observable and measurable privacy threats (PTs). Each PT logically maps to privacy regulations. Thus, the logical implication is, if a privacy threat is violated, it implies a violation of privacy regulation. We choose to focus on privacy regulations because they have become widely used across the globe to protect user's privacy. However, the complicated legal language found in these regulations is often difficult to interpret, let alone to implement in a compliant manner. Our PT analysis pipeline can help bridge this gap between legal requirements and measurable compliance.

Although we cast a wide net of PTs, this thesis focuses on measuring a subset of PTs related to data collection. These threats include an absence of consent management, unlawful logging of personal identical information (PII), and inconsistent permission usage. To create our PTs and analyze a large set of mHealth apps we had to answer several questions:

1. Can we create a precise definition for mHealth apps?

2. Can we collect a large number (10,000+) of mHealth apps for analysis in an automated fashion?

3. Is there a data lifecycle model that can be created to show the stages a user's data goes through in mHealth apps?

4. Can we distill novel PTs and create a mapping between privacy regulations and PTs?

5. Can we automate the detection of PTs, and if so to what extent?

6. Can we conduct large-scale experiments specific to app data collection practices?

7. To what extent do mHealth apps practice compliant data collection practices?

1.2 Thesis Organization

In Chapter 2 we cover essential background knowledge, including an overview of Android, Android apps, app analysis techniques, and pertinent privacy law. We discuss previous research in Chapter 3. Chapter 4 describes our Data Lifecycle model. Our categorization of the General Data Protection Regulation (GDPR) into novel GDPR Properties based on our Data Lifecycle is mentioned in Chapter 5. Chapter 6 reviews our Privacy Threat (PT) model. We describe the methods used to create our dataset of 13,177 mHealth apps in Chapter 7. We discuss data collection in Android apps in Chapter 8, including methods and how GDPR defines and enforces data collection. Chapter 9 focuses on PTs related to data collection, with an in-depth technical review of each PT and their analysis results for 13,177 apps. Lastly, Chapter 10, gives this thesis's closing remarks, noting lessons learned and future work.

Chapter 2

Background

This chapter presents the technical background necessary for understanding the subsequent chapters of this thesis and the privacy analysis conducted. We begin with an overview of Android, followed by a discussion of the structure and functionality of Android apps. Next, we will introduce two key analysis techniques relevant to this thesis, static and dynamic analysis. Finally, we conclude the chapter with a review of the General Data Protection Regulation (GDPR) [2]

2.1 Android Overview

Android, a mobile operating system (OS), was introduced in 2008 by Google. Since its introduction, Android has grown in popularity from less than 500 million active users before 2012 to 3.3 billion active users in 2025 [1]. Parallel to its rise in popularity, Android has become one of the most popular mobile OS, with a majority market share of 72.04% in Q4 of 2024 [3]. We focus on Android apps in this thesis, as there are currently billions of active users; ensuring that our findings apply to a large chunk of the world's population.

2.2 Android Applications

Android divides apps into two categories: system apps and third-party apps. System apps are default apps, such as the camera, settings, and contacts [4]. Manufacturers pre-install these apps on the device. Non-system apps (third-party apps) can leverage the functionality provided by system apps. For example, if a third-party app wants to offer SMS messaging features, it can utilize the default SMS app without needing to develop these functionalities from scratch [4].

In this thesis, we focus on third-party apps, i.e., apps downloaded and installed from the Google Play Store. The intrinsic nature of third-party apps prompts questions about how their developers handle privacy. For the remainder of this thesis, we refer to third-party apps as "apps".

2.2.1 Application Description

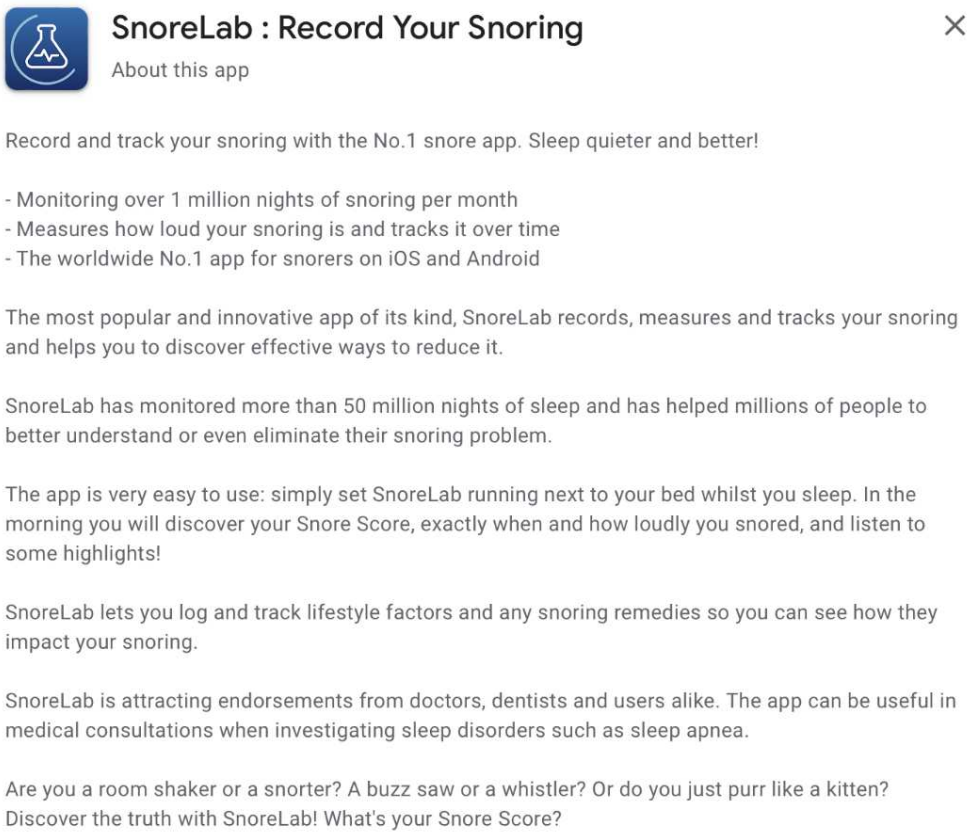

Figure 2.1: Application description for app *com.snorelab.app*

App descriptions are found on the Google Play Store. They act as the first natural language source users leverage to build conceptual models of the types of data the app will collect. Figure 2.1 illustrates an app description for *com.snorelab.app*, a snore tracking app. The description insinuates the app will record your audio, hence the app will need to request various permissions to complete this task. Furthermore, this hints at potential data the app will collect, such as the user's audio, device information, health metrics related to snoring, sleep apnea, etc. Many users will only read the description. This can lead to an incomplete conceptual model due to the incompleteness of privacy practices described in the description. In part of this thesis, we analyze this incompleteness by measuring how accurate or ambiguous the app's description is compared to its actions using Large Language Models (LLM)s.

2.2.2 Application Building Blocks

Apps are distributed using the Android Package Kit (APK) file format. An APK contains all the data necessary for the app to function at runtime [5]. Apps consist of different components that work together as building blocks. Below, we introduce the core building blocks we will use throughout this thesis.

Activities serve as the entry points for user interaction, displaying a single screen of the user interface (UI) [6]. An activity facilitates key interactions between the system and the app, such as tracking which screen the user currently views and enabling interactions between apps, like a messaging app initiating an activity in the system's contact app to retrieve data [6].

Services unlike activities, do not provide a UI. Services are headless and run in the background. Android provides three kinds of services, foreground, background, and bound services. Foreground services perform operations that are noticeable to the user, such as playing music after closing an app [7]. As the name implies, the user is not aware of background services such as syncing data [7]. Bound services connect to different processes, letting the system know there is a dependency. A bound service runs because another app or the system wants to make use of the service [7]. Thus, if there is process X that is bound to process Y, the system knows to keep X running for Y.

Content providers give applications a way to manage their stored data and other stored data on the system. They also enable apps to share data [8]. For example, an app may want to access some data from another running process. Content providers initiate this sharing of data while providing mechanisms to protect data security [8].

2.2.3 Manifest File

The manifest file, *AndroidManifest.xml*, is located at the root of an Android app's source code. It acts as the control center for the app. It specifies building blocks including content providers, broadcast receivers, services, and activities. The permissions an app uses are also declared in the manifest file. The following section explains the use case and importance of Android permissions.

2.2.4 Android Permissions

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="70"
    android:versionName="3.2.5"
    android:compileSdkVersion="33"
    android:compileSdkVersionCodename="13"
    package="com.dnurse"
    platformBuildVersionCode="33"
    platformBuildVersionName="13">
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="33"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.GET_TASKS"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.FLASHLIGHT"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Figure 2.2: *AndroidManifest.xml* file with requested permissions

Android permissions protect access to restricted data such as contact information and actions such as recording audio. Apps request permissions in their *AndroidManifest.xml* to provide key functionality such as sending notifications. The Android platform categorizes permissions as, install-time, special, and run-time. All permissions have a protection level, i.e., their implied potential risk.

Install-time permissions access limited restricted data or actions [9]. Install-time permissions have two protection levels, Normal and Signature. Normal permissions are lower-risk and are granted by default during installation. Signature permissions are only granted when the app requesting the permission is signed with the same certificate as the app that declared the permission [9]. A number of install-time permissions requests are shown in Figure 2.2: *INTERNET*, *ACCESS_WIFI_STATE*, *VIBRATE*, *WAKE_LOCK*, and *FLASHLIGHT*.

Run-time permissions are given a Dangerous protection level as they access sensitive data or hardware [9]. Users manually grant dangerous permission requests by accepting or declining a run-time prompt. The manifest file shown in Figure 2.2, requests a number of run-time permissions such as: *READ_EXTERNAL_STORAGE*, and *WRITE_EXTERNAL_STORAGE*.

Special permissions differ from run-time and install-time permissions, as they protect access to sensitive system resources and are not directly tied to user privacy but rather system security and integrity [9]. Some examples of special permissions are scheduling exact alarms and accessing all storage data [9]. Special permissions can only be granted in system settings under the special app access page [9].

Android permissions control access to restricted data and actions in apps [9], as such, part of this thesis focuses on threats to privacy due to permissions. In the next section, we will explain the two primary techniques used to analyze Android apps.

2.3 Static and Dynamic Analysis

In this section, we provide an overview of the two primary techniques for analyzing Android apps: static and dynamic analysis.

Static analysis examines the source code of an application without executing it. Researchers have used static analysis for various tasks, such as security analysis, malware detection, and automated test case generation [10]. This approach benefits from its simplicity and scalability. However, it suffers from over-approximation due to analysis of all application code, even dead code; consequently, it tends to produce false positives [10].

Dynamic analysis executes an app's source code to inspect its runtime behavior. UI fuzzers such as, Exerciser Monkey [11], are testing tools that generate streams of pseudo-random user actions to help identify potential bugs and vulnerabilities in applications. Each set of actions represents a single path, and an app can have many possible paths. Dynamic analysis underestimates the number of explored paths due to its non-deterministic behavior [10].

There are downsides to both static and dynamic analysis, however, Li et al., 2017, [10] showed that combining static and dynamic analysis into a hybrid approach produces better results. Therefore, in this thesis, we use both static and dynamic analysis techniques.

2.4 General Data Protection Regulation

The General Data Protection Regulation (GDPR) is a law passed by the European Union (EU) in 2016 and began to be enforced in 2018 [2]. It aims to set a standard to protect the privacy and personal data of individuals. It sets strict guidelines on how companies can collect, process, and share personal data, thereby giving users more control over their own data [2]. Any company that is found to violate the GDPR's guidelines can receive significant fines, including the maximum fine of €20 million or 4% of the company's annual turnover [2]. In the context of mHealth apps, adherence to the GDPR is crucial, as they collect vast amounts of PII. To better understand the GDPR in the context of mHealth apps, we relate the key entities in the GDPR to an IoT system.

- *Data Subject* is a real person who produces data using a system. The data produced by the data subject is called Personal Data [2]. In IoT systems, users are data subjects.

- *Personal Data* can identify a real person, directly or indirectly. Examples of personal data are an identifier such as a name, an identification number, location data, an online identifier, or one or more factors specific to the physical, physiological, genetic, mental, economic, cultural, or social identity of that natural person [2].

- *Controller* is a person, public authority, agency, or other body, which, alone or jointly with others, determines the purposes and means of the processing of personal data [2]. In IoT systems, app developers are controllers.

- *Processor* refers to a person, public authority, agency, or other body that processes personal data on behalf of the controller. In IoT systems, third parties act as processors, because they process personal data to provide a service such as Google AdMob [12] and Analytics [13].

- *Processing* means any operation or set of operations that are performed on personal data, whether or not by automated means, such as collecting, recording, organizing, structuring, storing, etc.

- *Recipient* means a natural or legal person, public authority, agency or another body, to which the personal data are disclosed, whether a third party or not. In IoT systems, a third party is a recipient.

In this chapter, we covered essential background knowledge. The next chapter will cover the related works pertaining to this thesis, their shortcomings, and how we aim to bridge research gaps.

Chapter 3

Related Works

This chapter overviews past research investigating data collection, with a focus on issues such as data leakage, excessive data collection, and the alignment between natural language descriptions and an app's actions.

Data leakages are flows of user data to third parties. Fan et al, 2020, [14], and Ardalani et al., 2023, [15] investigated data leakage by conducting Graphical User Interface (GUI) analysis and taint analysis. First, GUI analysis extracts XML files containing UI components such as forms, buttons, and other fields where users enter data. Taint analysis then tracks the data's path from its source (the GUI) to its destination (network data, logs, or local storage). However, these studies have limitations such as Fan et al. [14] not specifically addressing mHealth apps and both authors use static analysis techniques only. This thesis complements [14, 15] by implementing dynamic analysis, focusing on mHealth apps' predefined device and user identifiers rather than user-entered data through an app's UI.

In contrast, there have been studies that detect predefined device and user identifiers from data sources such as network packets and logs using both static and dynamic analysis. Nguyen et al., 2022, [16], and Fangwei et al., 2020, [17] used dynamic analysis to examine device and user identifiers (e.g., device ID, advertising ID) within network files and logs. Nguyen et al., 2021 [18] tested explicit consent by opening apps and not interacting with them, seeing if apps sent any PII to third parties by inspecting collected network packets through Mitmproxy [19] and Frida [20]. Mitmproxy allows a system certificate to be placed on an Android phone and intercepted by a proxy server, acting as a man in the middle [19]. Frida bypasses SSL pinning [20], allowing the apps to believe the MITM CA cert is trustworthy. If the app is found to be transmitting PII it violates consent as the user did not consent explicitly. Nguyen et al., 2022 [16] built upon their previous work by extending Droidbot [21] to enable dynamic analysis while collecting application screenshots and network packets, analyzing and clustering screenshots for privacy-related content,

and testing various modes of opting in and out of consent. Fangwei et. al., 2020, [17] also examined network traffic, but in apps targeted at children. The authors found that many apps transmitted personal data to third parties, suggesting that privacy rules are not being enforced [17]. Due to [16] extensive research and platform architecture, we decided to leverage parts of Nguyen et al., [16, 18] code and complement it by extending the detectable privacy languages using ChatGPT. Furthermore, we added post-processing for several apps that failed to run and data analysis.

Pandita et al., 2013 [22], proposed WHYPER, a framework that utilizes Natural Language Processing (NLP) techniques to identify sentences that describe the need for particular permissions in an application description. The authors evaluated the effectiveness of this approach on over 1,000 applications, specifically testing for three permissions. WHYPER achieved an ROC-AUC of 0.60. However, it has limitations such as requiring access to source code, depending on external tools, being resource-intensive, and has limited coverage of permissions.

Building upon WHYPER, Qu et al., 2014 [23], proposed a framework called AutoCog, which also uses an NLP approach to analyze descriptions of an app's permissions and compare them to its actual behavior and requested permissions. The authors evaluated this approach on 20,000 Android apps, and found that AutoCog outperformed WHYPER. The authors extended the analysis to 11 dangerous permissions. The authors achieved an average precision of 92.6% and an average recall of 92.0%. The limitations of AutoCog stem from its unsupervised learning approach can lead to false positives by choosing inappropriate relationships.

Feng et al., 2019 [24] proposed a deep learning framework AC-NET. This differs from previous research as they use deep learning and create their own annotated dataset consisting of 11 labels mapped to a total of 16 Android Permissions (see Table 9.5). The authors evaluated their model to find it outperformed previous research, receiving a 0.97 ROC-AUC. However, the dataset has an extensive label imbalance which leads to poor F1 scores. We improve upon [22–24] by using a transformer-based solution along with augmenting AC-NETs dataset, leading to state-of-the-art evaluation results.

Despite previous research, there has yet to be a large-scale analysis that has examined data collection practices in mHealth apps while considering multiple data and natural language sources and their compliance with the GDPR. This work seeks to bridge that gap by providing a holistic view of apps data collection and GDPR compliance via PT analysis. In the next chapter, we will introduce our Data Lifecycle model.

Chapter 4

Data Lifecycle

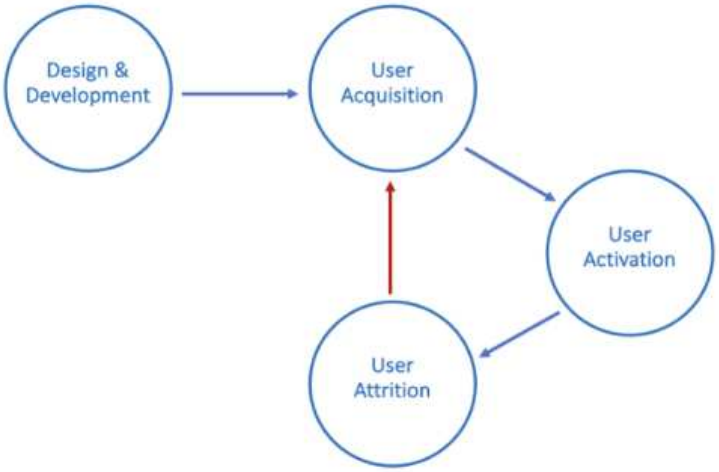

Figure 4.1: Phases of the Data Lifecycle in which the red line represents users that have been re-acquired. This part of the app data lifecycle doesn't apply to every user and can be omitted for many

This chapter presents the lifecycle a user's data goes through when using an app. We use a Data Lifecycle model, shown in Figure 4.1, to better understand when a user's data may be at risk. This lifecycle is composed of five phases that describe a user's relationship with a given app from installation to when the app is deleted. Below we describe each phase.

- *Design & Development* is a pre-user phase in which an app is actively being designed, programmed, connected to a configured database, and outfitted with a specific set of security and privacy practices, before being published to an app store for consumers to use.

- *User Acquisition* starts when an app is actively acquiring users, having them consent to their terms, and providing them with information about how the app is collecting, processing, and using their data, as well as what rights the user has.

- *User Activation* reflects the entire period of time in which a user is actively using an app and having their data collected, processed, and at times, transferred. During this period, as an app is updated - meaning its software or terms of service are changed - users should actively be kept up to date and asked to consent to the changes where applicable. If an app suffers any data breaches, this should also be reported to users at this time too.

- *User Attrition* is the post-user phase in which a user has deleted an app and is no longer having their data collected or processed. In our analysis and according to GDPR guidelines, there are specific ways the data collected up until this point should be handled.

- *User Re-Acquisition* shown by the red line in Figure 4.1, is when a user that has previously left an app, then decides to rejoin the platform. When this happens, if a user's data is destroyed, they may have to create a new account. However, depending on the terms of a given app, some users may be able to use a previous account.

The following chapter will describe our methods to categorize the GDPR based on our Data Lifecycle model. Doing so enables this thesis to focus on data collection practices by relating to specific phases of the Data Lifecycle, i.e., *User Acquisition* and *User Activation*, to GDPR stipulations.

Chapter 5

GDPR Properties

We performed a systemic review of the General Data Protection Regulation (GDPR) [2]. First, we reviewed articles and recitals of the GDPR to categorize them based on our Data Lifecycle model. Doing this allows us to map certain articles, sub-articles, and recitals to specific parts of the data lifecycle; naturally creating groupings of similar GDPR articles. Each grouping is referred to as a GDPR Property. We have a total of 10 GDPR properties. Next, we will describe each GDPR Property and its related GDPR stipulations.

Data Privacy Infrastructure: According to Article 25(1) of the GDPR, controllers have the responsibility that personal data shall be protected through state-of-the-art technology design such as pseudonymization and data minimization [2]. Subsections (2) and (3) further emphasize that controllers are responsible for ensuring only necessary personal data being processed with a clear and defined purpose [2].

Consent Management: Consent is the lawful basis for all processing, collection, and sharing of personal data in the GDPR. Article 6(1)(a) of the GDPR specifies that processing personal data is generally prohibited unless the data subject has consented to the processing [2]. Article 7 and Recital 32 of the GDPR provide the conditions for lawful consent, i.e., it must be freely given, specific, informed, and unambiguous. In addition, consent should cover all processing activities carried out and must be given via written, or electronic means [2]. All data collection, processing, and sharing must begin by obtaining lawful consent from the data subject per the above obligations.

Data Collection: Article 5(1)(a) and 13 of the GDPR states the controller must ensure processing of personal data follows, lawfulness, transparency, and fairness [2]. The controller, as specified in Article 5(1)(b)(c) is required to collect personal data for a specific, explicit, and legitimate purpose, i.e., *purpose limitation*. Furthermore, the controller must follow *data minimisation* and is obligated to inform the user of the type of data collected, and for how long this data will be stored [2].

Data Retention: Article 5(1)(e) and Recital 65 explains that personal data must not be kept for longer than needed, and controllers must justify the length of time personal data is kept. Storage limitations, which are the periodical reviews of data, that are erased or anonymized when no longer needed, should be enforced. However, personal data may be kept longer for purposes of public interest such as archiving, research, or statistical purposes [2].

Data Processing: Article 4(1)(8) of the GDPR states, that in controller-processor relations, a processor can process personal data on behalf of a controller [2]. Article 28(1) and 4(1)(11) state that the data subject's first contact is with the controller. Therefore, the controller is responsible for establishing the lawful basis for data processing and ensuring the processor processes personal data following the instructions of the controller [2]. Furthermore, state-of-the-art encryption and other privacy protection techniques are required based on Article 32 of the GDPR. In our IoT system, the processor would be the 'cloud' or a 'third party' depending on circumstances such as, the data being sent to a cloud database and being processed there or sold/given to another company/country for their processing. The controller is responsible for maintaining a record of processing activities. Furthermore, each processor needs to maintain a record of all processing activities done on behalf of the controller. The records need to be readily available [2].

Data Sharing and Transfer: transfer rules, as outlined in Article 44 of the GDPR, apply when the recipient is separate from the controller and processor. Restricted transfers are when the data is being sent to a location outside of its home country. Restricted transfers can happen when the recipient location is safe to transfer under an adequacy decision made in accordance with Article 45(1) [2]. An adequacy decision is a decision adopted by the GDPR, which establishes that a third country (that is, a country not bound by the GDPR) or international organization ensures an adequate level of protection of personal data. Such a decision takes into account the country's domestic law, its supervisory authorities, and international commitments it has entered into as detailed in Article 45 [2]. If an adequacy decision is not made then the restricted transfer must be covered by safeguards, outlined in Article 46(1) [2]. These safeguards include legal documents between a developer and a third party describing the security measures and data transfer rules, as

described in Article 46(2)(a)(b) and Article 43(1)(a)(b), with some additional considerations under Article 46(4). If the recipient's country is not safe to transfer data to (no 'adequacy decision' made) and there are no appropriate safeguards, the transfer can be sent if it is covered by exceptions such as if the transfer is necessary for legal claims, allowed by Article 49(1) [2].

Data Auditing (Rights of Data Subjects): The data subject has the right, under Article 12(1) of the GDPR, to obtain from the controller confirmation whether or not their data is processed [2]. If so, the data subject has the right to request data being processed, as described in Article 12(2) [2]. The controller is required to provide the requested information promptly (within a month), specified in Article 12(3) [2]. Suitable reasons for extensions to the timeframe are given under Article 12(4) [2]. Article 13(1)(2) specifies, at the time personal data is collected from the data subject, the controller must provide information such as identity and contact details of the controller [2]. Article 28(3)(h) Processors are required to make audits available, showing they are compliant with all obligations [2].

Data Rectification: The data subject has the right, under Article 16 and Recital 65 of the GDPR, to request the data controller to rectify inaccurate personal data concerning him without undue delay. Taking into account the purposes of the processing, the data subject has the right to complete incomplete personal data, including using a supplementary declaration [2].

Data Deletion (Right of Erasure): Recitals 65 and 66 describe the data subject has the right to be forgotten when data retention infringes on the data subject's rights. Data subjects under Article 17(1), have the right to obtain personal data from the controller and the controller has obligations to erase personal data where the data is no longer necessary for the original processing purposes [2]. Furthermore, the data subject may request erasure when they have withdrawn their consent and there is no other legal basis for the processing, specified in Article 17(1) [2]. When data is erased under Article 17(1), the controller must notify any involved parties as noted in Article 19 [2].

Each GDPR Property is mapped to a phase of our data lifecycle model shown in Table 5.1. The creation of GDPR Properties allows us to create mappings from these properties to privacy threats, logically connecting complicated legal language to measurable privacy threats.

Table 5.1: Mapping GDPR Properties to Data Lifecycle phases

Life Cycle	**GDPR Property**
Design & Development	Data Privacy Infrastructure
User Acquisition	Consent Management
	Data Auditing
User Activation	Data Processing
	Data Collection
	Data Sharing and Transer
User Attrition	Data Retention
	Data Rectification
	Data Deletion
User Re-Acquisition	Data Deletion

In this chapter, we reviewed our methods to categorize the GDPR into GDPR Properties based on our Data Lifecycle model. We explained each property and its related GDPR stipulations. Finally, we showed the mappings between our GDPR Properties and the phases in our Data Lifecycle model. In the remainder of this thesis, we focus on data collection practices in mHealth apps, thus we analyze the lifecycle phases *User Acquisition* and *User Activation* and the GDPR Properties *Data Collection* and *Consent Management*. We consider *Consent Management* as part of data collection because the GDPR requires a lawful basis for data collection obtained via consent. In the next chapter, we describe privacy threats to mHealth apps and their mappings to GDPR Properties.

Chapter 6

Privacy Threat Model

This chapter establishes a set of threats to privacy in mHealth apps. Each privacy threat (PT) encapsulates common violations in apps and by default the GDPR stipulations they violate. This enables us to analyze PTs in real-world environments and assess compliance quantitatively. Through a rigorous review of the GDPR, literature, and manual analysis, we formed precise definitions for 10 PTs, 6 of which can be precisely measured in an automated fashion. We will open this chapter by defining what a PT is, explaining manual versus automated PTs, and finally the mappings between our GDPR Properties and PTs.

Defining threats to privacy in mHealth apps is difficult, as there has yet to be a clear taxonomy of what threats to privacy exist in mHealth apps. We use a three-stage approach to define what PTs are, which includes GDPR categorization into properties, early app analysis, and literature comparisons. First, based on our review of the GDPR and subsequent creation of GDPR Properties, in Chapter 5, we had clear stages of the Data Lifecycle to focus on, i.e., *User Activation* and *User Acquisition*. Our initial app analysis was performed via manual exploration, including reviewing apps' source code and using security tools such as MobSF [25]. We found many security and privacy issues that helped form our novel PTs. Next, we compared these to PTs reported in existing studies. If a PT we had from the early app analysis was unique, we kept it as a new PT. Any PT that existed in literature was kept, if and only if we could provide a novel perspective and/or implementation. In the end, we had 10 PTs. In the next sections, we give an overview of what we define as our manual privacy threats (PTMs) and automated privacy threats (PTAs).

6.1 Manual Privacy Threats

Table 6.1 shows our PTMs, privacy threats that we were unable to automate. There are various reasons why, such as automation being technically infeasible, and limitations in technologies that were unable to replace the human in the loop. Although we were unable to automate these for

Table 6.1: Manual Privacy Threats

Manual Privacy Threat (PTM) #	Name
PTM1	Absence of Consent Management
PTM2	Inconsistent Permissions
PTM3	Vacant Permissions
PTM4	Database Misconfiguration
PTM5	Data Storage Location
PTM6	Unlawful or Excessive Data Collection
PTM7	Insecure Data Transfer
PTM8	Unlawful Data Logging
PTM9	Unlawful Data Sharing
PTM10	Absence of Auditing Techniques

large-scale analysis, these can still be used for a smaller subset of apps if we wish to explore a specific domain such as data auditing. Thereby enabling us to delve deeper, albeit manually. Below, we give a brief description for each PTM.

- *Absence of Consent Management*: obtaining consent that is freely given, informed and specific, and unambiguous, is the foundation for lawful data collection and processing under the GDPR. Violations occur when apps are found to collect personal data without consent.

- *Inconsistent Permissions*: permissions help protect the privacy and security of users by limiting access to sensitive data or actions. App descriptions should be clear, accurate, and aligned with the permissions the app requests. Misleading users about the app's functionality and the permissions it requires leads to GDPR violations.

- *Vacant Permissions*: unused requested Android permissions inherently are not dangerous, but malicious apps can leverage the vulnerabilities in the Android system or the privileged abilities exposed by apps to escalate their permission or collect PII without user consent.

- *Data Storage Location*: under the GDPR, sending user data to a location outside approved regions, like outside the EU, without an adequacy decision is a serious violation of the GDPR.

- *Database Misconfiguration*: the GDPR emphasizes privacy by design and default. However, if Firebase DB is left unlocked or poorly configured, anyone could access it, putting user data at risk and violating the GDPR.

- *Unlawful or Excessive Data Collection*: an app collects personal data outside the scope of the app's use case, described in the app's natural language. As a result, personal data is collected without the user's consent, leading to GDPR violations.

- *Insecure Data Transfer*: data that is unencrypted traveling over the network is vulnerable to intercept. Thus, passing data unencrypted is a risk. The GDPR outlines requirements for secure data transfer. Sending data in clear text violates these requirements.

- *Unlawful Data Logging*: when apps improperly log PII and device identifiers in the Android shared logging system can lead to sensitive data being exposed across the system without user consent.

- *Unlawful Data Sharing*: the GDPR outlines specific rules for sharing user data. Violations of these rules occur when, for example, an app sends a user's advertising ID to advertising domains without consent or an app covertly sends data to an adversarial country.

- *Absence of Auditing Techniques*: either the app does not support auditing and/or allows auditing such that an adversary while doing auditing can learn something about other users.

In the next subsection, we will discuss privacy threats that we checked automatically.

6.2 Automated Privacy Threats

To test a large amount of apps it is crucial to check PTs in an automated fashion. Doing so enables us to test our dataset of 13,177 apps in a reasonable amount of time. Table 6.2, shows the PTAs that we automated, thus renamed as PTAX. Table 6.3 displays our final list of PTAs. In subsequent chapters of this thesis, we will focus only on PTAs that relate to data collection. In the following section, we describe the mapping between our PTAs and GDPR Properties.

Table 6.2: Comparison of Manual and Automated Privacy Threats

Manual Privacy Threat #	Automated Privacy Threat #	Name
PTM1	PTA1	Absence of Consent Management
PTM2	PTA2	Inconsistent Permissions
PTM5	PTA3	Data Storage Location
PTM4	PTA4	Database Misconfiguration
PTM8	PTA5	Unlawful Data Logging
PTM9	PTA6	Unlawful Data Sharing

Table 6.3: Automated Privacy Threats

Automated Privacy Threat #	Name
PTA1	Absence of Consent Management
PTA2	Inconsistent Permissions
PTA3	Data Storage Location
PTA4	Database Misconfiguration
PTA5	Unlawful Data Logging
PTA6	Unlawful Data Sharing

6.3 Mapping GDPR Properties and Privacy Threats

Creating a logical mapping between our GDPR Properties and PTs enables us to relate specific GDPR articles and recitals directly to privacy threats. Thus, if *snorelab.app.com* violates PTA1 (absence of consent management), in turn, it would violate GDPR Property *Consent Management* and Article 6(1)(a) and Recital 32 of the GDPR [2]. Thus, our privacy threat measurements directly reflect non-compliant app behavior. Figure 6.4 describes the mappings between our PTAs and GDPR Properties.

Table 6.4: GDPR Properties mapped to Automated Privacy Threats

GDPR Properties	Automated Privacy Threats
Data Privacy Infrastructure	PTA1, PTA2, PTA3, PTA4, PTA5, PTA6
Consent Management	PTA1
Data Collection	PTA2
Data Processing	PTA5
Data Sharing and Transfer	PTA6

In this chapter, we presented our Privacy Threat Model, which included a set of manual and automated PTs, and their mappings to our GDPR Properties described in Chapter 5. Creating our PT Model and associated mappings enables us to analyze non-compliant behaviors in mHealth apps and directly relate our findings to GDPR and the Data lifecycle. Furthermore, PTAs allow for a large-scale analysis of thousands of mHealth apps. This thesis focuses on the Data Lifecycle phases *User Acquisition* and *User Activation* mapped to our GDPR Properties *Data Collection* and *Consent Management*, and their subsequent PTAs (PTA1, PTA2). Doing so enables us to analyze data collection practices in mHealth apps. The following chapter will describe how we created our dataset of 13,177 mHealth apps to be used in this thesis including, how we define a mHealth app, dataset annotation, inter-rater agreement, fine-tuning experiments, and finally model inference to vet apps as non-mHealth or mHealth.

Chapter 7

Dataset Creation

This chapter outlines our process for vetting mHealth apps. This process was necessary as previous research relied on Google Play Store's categories to vet mHealth apps. We propose a fine-tuned model that can classify mHealth apps via the apps' descriptions, effectively bridging this research gap. This chapter begins by describing our formal definition of mHealth apps. Next, we describe the methods used to create our annotated dataset: dataset collection, annotation guidelines, and inter-rater agreement. Finally, our fine-tuning process and results are discussed, followed by using our fine-tuned model to vet 500,000 apps, resulting in a dataset of 13,177 tested mHealth apps.

7.1 Defining Mobile Health Apps

Previous research relied on Google Play Stores categorizations of mHealth apps, such as *Health & Fitness*, and *Medical* apps [14, 15, 26]. However, many apps in these categories do not fit our definition of a mHealth app. For example, food delivery services can be categorized as medical apps because they can deliver prescriptions and over-the-counter medicines.

Whereas previous research relied on Google Play Store's categorizes, Jin et al., 2022, [27] used transformers to vet IoT apps. Following the approach described by Jin et al., we first created a formal definition for mHealth apps. The World Health Organization (WHO) defines mHealth as mobile applications and wearable devices used in healthcare [28]. Furthermore, the software used in health-related services on mobile devices are mHealth apps. The main feature of mHealth apps is to enable patients to take control of their health. These key characteristics include health tracking and care support, health data privacy, data transfer and communications with providers, and behavior change techniques [28].

Based on the broad definition from the WHO, we propose a fine-grained categorization of mHealth apps:

- Health Monitoring: apps that can collect and track health-related data, such as heart rate, blood pressure, physical activity, and sleep patterns.
- Disease Management: apps that are used to track chronic conditions such as asthma, diabetes, and hypertension. These apps can include symptom tracking, dietary guidance, exercise programs, etc.
- Telemedicine and remote consultations: apps that allow users to connect with providers remotely. These apps can use video calls, and secure SMS to help support and diagnose symptoms.
- Medication Management: apps that track users' medication dosages and schedules. They may also provide information on drug interactions and information.

7.1.1 Dataset Collection

The first step in creating our manually annotated dataset was to collect 500,000 app package names i.e., their unique identifier. We then used an Application Programming Interface (API) to collect app descriptions.

We leveraged AndroZoo [29] to collect app package names. AndroZoo is a collection of 24,785,205 apps [29]. AndroZoo collects apps from the Google Play Store and third-party sources. In our case, we only used apps collected from the Google Play Store. We first downloaded AndroZoo's list of apps in CSV format. AndroZoo updates the CSV every night (before 6am Luxembourg/Paris time) [29], ensuring we collected the most recent apps. We collected the first 500,000 apps' package names. The next step was to collect their descriptions from the Google Play Store.

To collect app descriptions, we use the Python API package *Google-Play-Scraper* [30]. For each app's package name we programmatically collect the app descriptions and add it to a pandas data frame. We employ minimal data preprocessing to keep semantic context. Thus, the app's descriptions were converted to lowercase, and any non-readable characters were removed such as HTML. Creating this dataset laid the foundation for annotation and fine-tuning our mHealth

app classifier; enabling the collection of a large set of mHealth apps to analyze in this thesis. The following subsection describes how we use this dataset to create an annotated dataset using Krippendorff's alpha to measure inter-rater agreement.

7.2 Inter-Rater Agreement

In this subsection, we cover the methods used to create our annotated dataset, including annotation guidelines and inter-rater agreement.

Given our formalized definition of mHealth apps, we created two batches of 100 apps and their descriptions and manually annotated them for a binary classification problem; is the selected app an mHealth app or a non-mHealth app? First, we will present the annotation guidelines and annotators:

- 0: Indicates a non-mHealth app
- 1: Indicates a mHealth app
- 2: Indicates an ambiguous app that is not a "pure" mHealth app, but could have medical information stored and used in certain circumstances.

Two undergraduates were selected as annotators. We had an initial discussion with the two annotators, to explain our definition of mHealth apps, and give examples of app descriptions that fit this scope. We gave the annotators the first batch of 100 apps to annotate. After the initial 100 apps were annotated, the Krippendorff's alpha between the two annotators was calculated.

We leverage the implementation of Krippendorff's Algorithm for bootstrapping the Krippendorff's alpha coefficient [31], implemented by Proutskova et al., 2017 [32]. Given that Krippendorff's alpha lacks a known a priori distribution, creating confidence intervals from alpha estimates requires the generation of bootstrap replicates. The chosen implementation allows for the generation of these replicates.

Table 7.1 illustrates the results from the first batch, showing that the two annotators demonstrated a moderate inter-rater agreement, achieving a Mean Alpha of 0.880. However, a closer

Table 7.1: Batch 1, Krippendorff's Alpha Metrics

Metric	Value
Lower limit of the bootstrapped confidence interval	0.781
Upper limit of the bootstrapped confidence interval	0.960
Mean Alpha	0.880

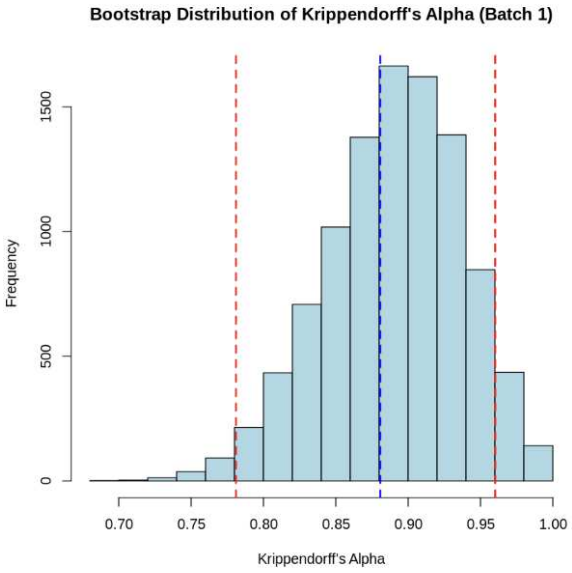

Figure 7.1: Bootstrap Distribution of Krippendorff's Alpha (Batch 1)

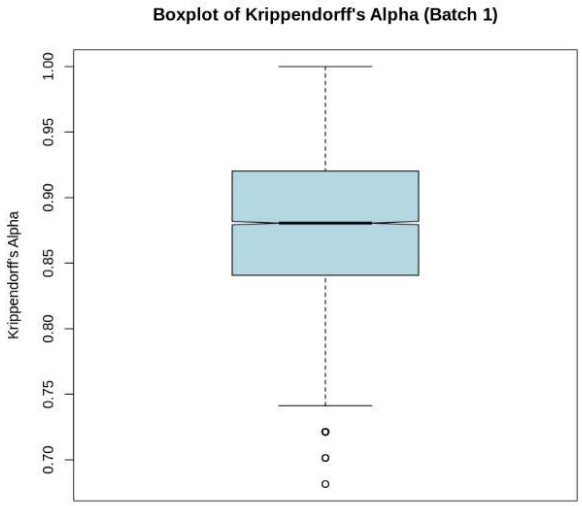

Figure 7.2: Box plot of Krippendorff's Alpha (Batch 1)

look at Figure 7.1 and Figure 7.2 reveals a spread in the confidence intervals, contributing to the overall variability around the mean alpha. Despite this, there remains a notable level of consistency and reliability between the two raters. To further assess the inter-rater agreement, we had the annotators classify the second batch of 100 apps.

Table 7.2: Batch 2, Krippendorff's Alpha Metrics

Metric	Value
Lower limit of the bootstrapped confidence interval	0.901
Upper limit of the bootstrapped confidence interval	1.00
Mean Alpha	0.961

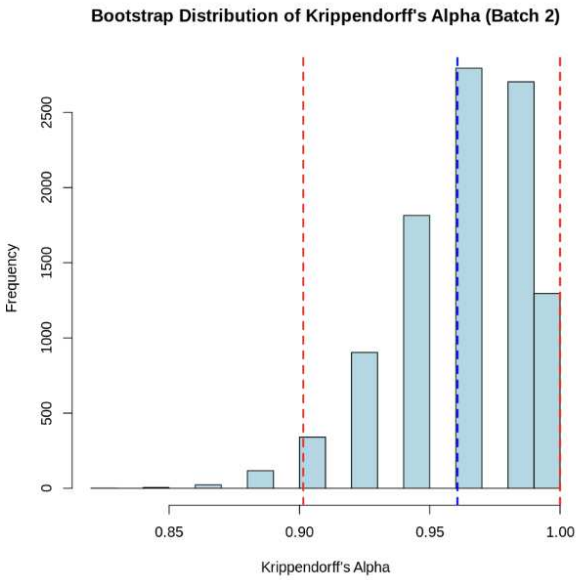

Figure 7.3: Bootstrap Distribution of Krippendorff's Alpha (Batch 2)

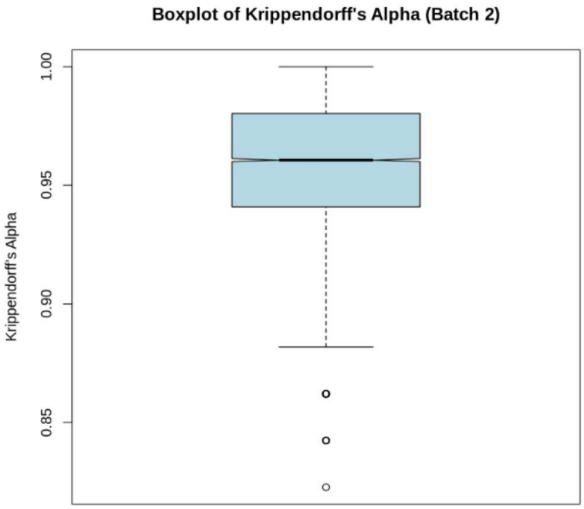

Figure 7.4: Box plot of Krippendorff's Alpha (Batch 2)

Based on Table 7.2, the mean alpha has increased significantly to 0.961. This shows a substantial level of agreement between the two raters. Notably, the upper limit of the confidence interval reaching 1 in the second batch suggests perfect agreement. Figure 7.3 and Figure 7.4 show a relatively narrow spread, centered around a high mean alpha, meaning high consistency and reliability among raters for Batch 2.

Following batch 2, the annotators proceeded to annotate until they reached 5,000 apps, which was processed resulting in 2,989 apps used in fine-tuning. This decision was made, considering that Jin et al., 2022 [27], required around 7000 total samples to fine-tune their model in a reasonable amount of time. The next subsections explain the rationale behind our chosen pre-trained models and hyperparameters used in the fine-tuning experiment.

7.3 Selected Models

We tested a variety of pre-trained language models, including BERT [33], DeBERTa [34] RoBERTa [35], and DistilBERT [36], for their well-established performance in text classification tasks. Each model was loaded in its *base* and *uncased* versions to ensure a fair and consistent

comparison across models, as it represents a standard configuration. Additionally, *uncased* aligns with our preprocessing approach, where all text is converted to lowercase and general text cleaning tasks such as removing HTML tags.

7.4 Selected Learning Rates

To determine the learning rates for fine-tuning, we considered insights from Sun et al., 2020, [37] emphasizing the important role of the learning rate in preventing *Catastrophic Forgetting*, where the pre-trained knowledge is lost during the acquisition of new knowledge. Sun et al., revealed that reducing the learning rate, such as using 2e-5, enables BERT to overcome Catastrophic Forgetting. The authors also found that employing a more aggressive learning rate, like 4e-4, resulted in the training set failing to converge [37]. Furthermore, Devlin et al., 2019, [33] recommended using learning rates of 5e-5, 4e-5, 3e-5, and 2e-5 for fine-tuning, across all GLUE tasks. Considering this, we choose a diverse range of learning rates of 1e-5 to 3e-5 to mitigate the *Catastrophic Forgetting* and convergence problems.

7.5 Selected Batch Sizes

Sun et al., 2020, [37] utilized a batch size of 32 for fine-tuning. This was influenced by BERT's maximum sequence length. Considering that each word in the app description undergoes encoding into a floating-point vector of 768 dimensions, and given the specified maximum sequence length of 512, both batch sizes of 32 and 16 are deemed okay to use. Devlin et al., 2018, [33] recommended batch sizes falling within the range of 16 to 32. In line with these recommendations, we opted to conduct our experiment with batch sizes of 16 and 32. It is noteworthy that GPU memory constraints were not a limiting factor in our scenario, providing the flexibility to select a batch size of 32 without concerns about fitting into GPU memory.

7.6 Selected Epochs

Sun et al., 2020, [37] fine-tuned with just 4 epochs with early stopping. Devlin et al., 2018, [33] used just 3 epochs to fine-tune over GLUE tasks. Based on previous work, it appears that the number of epochs used in fine-tuning is generally small, often with a maximum of 5. Considering this, we used 3 and 5 epochs to assess whether the additional 2 epochs would yield meaningful changes in results.

7.7 Selected Optimizer

We opted for the Adafactor optimizer which is an alternative optimizer that dynamically adapts the learning rates for each parameter. Adafactor's performance has been observed to be robust across various tasks [38]. This decision was influenced by strong recommendations from HuggingFace [38].

7.8 Fine-tuning Experiment

To fine-tune a variety of models with our selected hyperparameters, we utilized the Falcon High-Performance Computing (HPC) cluster provided by CSU. This decision was made to address the computational resources required for running a large set of models, as executing such tasks on Colab or a personal computer would be impractical. A custom slurm script was used to schedule the fine-tuning job with the following options. We use one node, as there are no parallel tasks. As a result, the number of tasks across the single node is 1. We allocate one CPU per task, as fine-tuning involves GPU-heavy computations rather than CPU processing. We set the memory for our node to 4g, enough but low enough to avoid over-requesting resources. We selected an NVIDIA A100 GPU with 80GB of memory and a time limit of 24 hours for the job. In total, we tested 48 combinations of the above models and hyperparameters. In the next subsection, we explain the results of the fine-tuning experiment.

Table 7.3: Experiment evaluation results from top 5 models

Model	Accuracy	F1	Learning rate	Epochs	Batch size
deberta-base	0.946	0.916	0.0002	3	16
roberta-base	0.941	0.909	0.0002	3	32
deberta-base	0.939	0.906	0.0002	3	32
deberta-base	0.939	0.905	0.0003	5	32
deberta-base	0.939	0.905	0.0002	5	32

7.9 Fine-tuning Results

In this section, we will present the results for all models and identify the best-performing models and hyperparameters.Table 7.3 demonstrate that *deberta-base* appears multiple times among the top performers, showing that it had very good performance across different hyperparameter settings. The learning rates (0.0002 and 0.0003) demonstrate that moderate values contribute to better performance, and confirm previous research that extreme values may not be necessary and may be detrimental. Models with batch sizes of 16 and 32 are prominent, aligning with common practices for fine-tuning pre-trained BERT models. Higher batch sizes seem to provide good results without memory concerns. The chosen epochs (3 and 5) show consistent performance, suggesting that the models converge well within this range. Both 3 and 5 epochs performed well indicating that additional epochs are unnecessary. This aligns with findings in the literature [33, 37]. The top model and hyperparameters are microsoft/deberta-base with a batch size of 16, 3 epochs, and a learning rate of 0.0002. The resulting evaluation F1 score and accuracy shows that among the experimented models *deberta-base* led to the highest F1 score and accuracy. It suggests that it's specific hyperparameters significantly contribute to its performance.

The training results associated with the best model show the following trends. In the earlier epochs (steps 0-100), the model achieves high precision and recall, resulting in a high F1 score and accuracy on the training set. However, as the training progresses, there is a slight decrease in performance on the training set. This suggests potential overfitting. However, despite the overfitting in later epochs, the F1 score and accuracy remain relatively high and stable, indicating that the model was performing well on the training data. Beyond a certain point (after 300 steps), there

is a small improvement in F1 score and accuracy, suggesting that the model has reached a performance plateau. The model consistently achieves high precision and recall throughout training, contributing to the overall high F1 score. Overall, the training results suggest that the selected hyperparameters contribute to a well-performing model. Furthermore, the evaluation results show that despite potential overfitting, the model generalized well to unseen data. The next section describes how we use our fine-tuned model to create a set of mHealth apps to analyze in this thesis.

7.10 Model Inference

To utilize our fine-tuned model to make new predictions we created an AppProcessor Python class. This class iterated over the apps' descriptions and tokenized them. After, it passes the tokenized app description to our fine-tuned model. We used the common prediction threshold of 0.5. In the end, 13,177 mHealth were vetted and tested.

We now have a dataset of 13,177 mHealth apps to be used in this thesis. The following chapter gives the background information necessary to understand data collection methods in mhealth apps.

Chapter 8

Data Collection

This chapter provides essential background on data collection used by Android apps. We examine methods used to collect data in Android apps and identify relevant GDPR stipulations.

8.1 Data Collection Methods in Android

Apps collect data from various sources, including user-entered data, device and sensor-derived data, system APIs, and third-party APIs/SDKs. The following provides a brief overview of these data collection sources in Android.

- *Permissions* control what data apps can access from the Android device. Most data collection starts with requesting permission, which apps then use to query information through Android APIs. For example, an app may request permission to access your location. Using Android APIs like LocationManager, the app can query your geographical location through system location services [39].

- *System APIs* allows apps to access system-level functions and resources. These APIs enable apps to gather device and user data [40].

- *Third-Party APIs and SDKs* provide communication and tools that enable data collection from external sources. These often provide additional functionality and insights such as analytics [13].

- *User-Entered Data Collection* is when a user interacts with an app, they may input personal information into the app's GUI.

- *Device and Sensor Data Collection* occur when apps query the Android device for device-related information, such as the advertising ID. This data differs from user-entered data

because, unlike inputs visible to the user like a GUI prompt or a permission request for location services, Apps can collect device data programmatically without the user's knowledge. Furthermore, Android devices contain various sensors, such as GPS and accelerometers, that collect environmental and contextual information about the user [41].

This thesis examines permissions and device data, with a focus on derived data like network traffic packets, and logs. In the following section, we introduce previous research in this area, the limitations, and how our work bridges the gap in current research.

8.2 Privacy Regulations

Under the GDPR before data collection can be started, the controller is required to obtain consent from the data subject. Then, during data collection, the controller must comply with specific GDPR requirements to ensure lawful collection and processing. The following key GDPR articles outline these requirements.

- GDPR Article 6(1)(a) "*the data subject has given consent to the processing of his or her personal data for one or more specific purposes.*" [2]

- GDPR Article 7 "*Where processing is based on consent, the controller shall be able to demonstrate that the data subject has consented to processing of his or her personal data. 1) If the data subject's consent is given in the context of a written declaration which also concerns other matters, the request for consent shall be presented in a manner which is clearly distinguishable from the other matters, in an intelligible and easily accessible form, using clear and plain language. 2) Any part of such a declaration which constitutes an infringement of this Regulation shall not be binding. 1) The data subject shall have the right to withdraw his or her consent at any time. 2) The withdrawal of consent shall not affect the lawfulness of processing based on consent before its withdrawal. 3) Prior to giving consent, the data subject shall be informed thereof. 4) It shall be as easy to withdraw as to give consent. When assessing whether consent is freely given, utmost account shall be taken*

of whether, inter alia, the performance of a contract, including the provision of a service, is conditional on consent to the processing of personal data that is not necessary for the performance of that contract." [2]

- GDPR Recital 32 "*Consent should be given by a clear affirmative act establishing a freely given, specific, informed and unambiguous indication of the data subject's agreement to the processing of personal data relating to him or her, such as by a written statement, including by electronic means, or an oral statement. 2) This could include ticking a box when visiting an internet website, choosing technical settings for information society services or another statement or conduct which clearly indicates in this context the data subject's acceptance of the proposed processing of his or her personal data. 3) Silence, pre-ticked boxes or inactivity should not therefore constitute consent. 4) Consent should cover all processing activities carried out for the same purpose or purposes. 5) When the processing has multiple purposes, consent should be given for all of them. 6) If the data subject's consent is to be given following a request by electronic means, the request must be clear, concise and not unnecessarily disruptive to the use of the service for which it is provided.*" [2]

- GDPR Article 5(1)(a) "*processed lawfully, fairly and in a transparent manner about the data subject (lawfulness, fairness, and transparency).*" [2].

- GDPR Article 5(1)(b) "*collected for specified, explicit, and legitimate purposes and not further processed in a manner that is incompatible with those purposes; further processing for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes shall, in accordance with Article 89(1), not be considered to be incompatible with the initial purposes ('purpose limitation').*" [2].

- GDPR Article 5(1)(c) "*adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed ('data minimisation').*" [2].

- GDPR Article 32(1) "*Taking into account the state of the art, the costs of implementation and the nature, scope, context and purposes of processing as well as the risk of varying*

likelihood and severity for the rights and freedoms of natural persons, the controller and the processor shall implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk, including inter alia as appropriate." [2].

- GDPR Article 13 " *Where personal data relating to a data subject are collected from the data subject, the controller shall, at the time when personal data are obtained, provide the data subject with all of the following information: (a) the identity and the contact details of the controller and, where applicable, of the controller's representative; the (b) contact details of the data protection officer, where applicable ... In addition to the information referred to in paragraph 1, the controller shall, at the time when personal data are obtained, provide the data subject with the following further information necessary to ensure fair and transparent processing...* " [2].

Based on the related GDPR articles the controller must adhere to several requirements to ensure lawful data collection. Specifically, the controller must: obtain lawful consent, collect minimum data with lawfulness, fairness, and transparency, has a legitimate purpose, stays within the scope of the legitimate purpose, clearly states the data collected and time stored. After reviewing the GDPR, we established a GDPR Property, *Data Collection*, that encompasses all stipulations related to data collection requirements:

Apps must collect minimum data with lawfulness, fairness, and transparency. The app developer must obtain lawful consent from the user before any data collection is initiated. The app developer is obligated to inform the user of the type of data collected, the purpose, and for how long this data will be stored.

This chapter established the foundation for understanding the various methods of data collection methods in Android apps such as device and sensor data vectors. We also highlight why these practices matter and examine their relationship to the GDPR by identifying related articles and recitals. In the following chapter, we will explore specific PTAs related to data collection i.e., PTA1, and PTA2.

Chapter 9

Privacy Threats Related to Data Collection

In the previous chapter, we described the data collection methods in Android, the limitations of previous research in this area, and our classification of GDPR articles relating to data collection. This chapter will give foundational knowledge of PTs relating to data collection in mHealth apps. First, we outline the PTs that map to our GDPR Property *Data Collection* and *Consent Management*. Next, we present a technical review of the PTs relating to data collection.

9.1 Privacy Threats Mapped to Data Collection

Table 9.1 presents our mapping between PTAs and GDPR Properties. PTA1 is mapped to both GDPR properties *Data Collection* and *Consent Management*, as consent is the basis for lawful data collection in the GDPR. PTA1 and PTA2 are logically mapped to *Data Collection*. PTA1 analyzes the consent screens shown to the user during dynamic analysis. Obtaining lawful consent is the foundation to collect, process, and share user data. Thus, PTA1 measures this baseline. PTA2 measures inconsistencies between the natural language sources used to inform the user, such as the application description to the requested permissions found inside of the *AndroidManifest.xml*. Under GDPR a user needs to be fully informed of all data that will be collected. If a mHealth app has an ambiguous description or privacy policy it is not clear to the user what data will be collected. Furthermore, if the app is found to use a large number of dangerous permissions this can be used to collect data that the user is not informed of. Thus, the natural language sources presented to the user should reflect all data collected by the app. The following sections will give an in-depth technical overview of the two PTAs that relate to data collection (PTA1 and PTA2) and their analysis results on our dataset of 13,177 mHealth apps.

Table 9.1: GDPR Properties and Associated Automated Privacy Threats

GDPR Properties	Automated Privacy Threats
Data privacy infrastructure	PTA1, PTA2, PTA3, PTA4, PTA5, PTA6
Consent Management	PTA1
Data Collection	PTA2
Data Processing	PTA5
Data Sharing and Transfer	PTA6

9.2 PTA1 - Absence of Consent Management

The GDPR emphasizes obtaining lawful consent before data collection, processing, and sharing. mHealth apps collect extremely sensitive PII, thus the user has inherent trust in these apps. However, non-compliant consent screens and subsequent data collection can infringe on the user's rights, violating the GDPR. Furthermore, the app developers could receive monetary fines enforced by the GDPR. Thus, it is important for mHealth apps to lawfully obtain consent. First, we will describe how the GDPR defines consent and the necessary steps to obtain lawful consent. Next, an in-depth review of our methodology to detect consent violations, and how results for 13,177 apps will be discussed.

Article 7 and recital 32 of the GDPR define the conditions for lawful consent. Specifically, consent must be freely given, specific, informed, and unambiguous.

- *Freely given*: The user should give consent voluntarily and no pressure or influence placed upon the user [2].
- *Informed and specific*: The data subject must be informed of the controller that will be conducting data collection and processing, what data will be used, and the purpose of the processing activities. Furthermore, the data subject must be able to withdraw consent as easily as it was obtained [2].
- *Unambiguous*: Consent needs to be obtained through a statement or an affirmative action. Consent can not be implied, it must be obtained through opt-in, a declaration, or an active motion. [2].

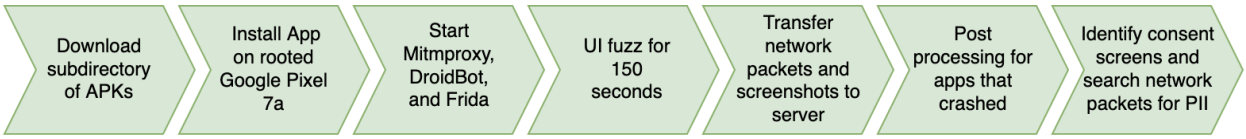

Figure 9.1: PTA1 analysis process

PTA1's overall analysis process is shown in Figure 9.1. The first step of PTA1 is to create a CSV of APKs and their paths using our script *create_PTA1_apklist.py*. First, we downloaded a zip of APKs from our OneDrive storage. These are roughly 7 gigabytes each, containing roughly 200 apps for each 53 zip files. After the file is downloaded, it is unzipped using the Linux tool *unzip*.*create_PTA1_apklist.py* then iterates over the unzipped APKs and constructs a CSV with the APK package name and its path on the server. After the CSV is constructed it is used as input for the next step.

Our next script *network-analysis_m.py*, receives the list CSV of APKs and starts a custom instance of Driodbot [16]. We begin by iterating over the APKs. For each APK we install it onto a rooted Google Pixel 7a running Android 14. We wait 12 seconds and use ADB to capture a screenshot of the first screen shown to the user. Under GDPR the controller needs to obtain lawful consent before data collection and processing activities. Thus, it is our intuition the first screen shown to the user is to obtain lawful consent. After collecting screenshots we collect network packets generated by the app using Mitmproxy's command line version, *mitmdump*.

The PTA1's analysis pipeline had various errors, some of which are limitations of Mitmproxy and Frida tool's *Objection*. First, we found many apps failed to start after installation. Therefore, a screenshot of the app's home screen was captured, nullifying any other results. Second, many of the captured PNGs were corrupted. We use the Scriptable Image Processing System (SIPS) to repair the images by rereading them and regenerating lost metadata. After repairing the images, we used a Jupiter notebook and a NIVIDA Tesla T4 GPU, to extract text from the captured screenshots. Nguyen et al. used Python Tesseract. However, we found Python Tesseract had poor performance. As a result, we ran a benchmark on EasyOCR [42] and Python Tesseract [43] and TrOCR [44] using the 2019 Scanned receipts OCR and information extraction (SROIE) dataset.

SROIE is a collection of 973 scanned English receipts cropped and bounded, creating 33,626 samples. We begin by loading the SRIOE dataset and sampling 500 rows from the test set. We then load each OCR model and track its GPU usage, time elapsed, and accuracy on the 500 sampled rows. Table 9.2 depicts the performance results. The transformer-based model outperformed all other models with an accuracy of 93.80%. However, it is limited to single text line images [44]. Therefore, we opted to dispel TrOCR base printed from our consideration. In the end, we choose to use EasyOCR as it outperforms Python Tesseract and can extract complex multiple line text effectively.

Table 9.2: Performance metrics for OCR models

Metric	**Value**
EasyOCR Accuracy	37.20%
Tesseract Accuracy	22.20%
TrOCR Base-printed Accuracy	93.80%
TrOCR Small-printed Accuracy	93.40%
EasyOCR Average Time	0.0674s
EasyOCR Average GPU Memory	160.34 MiB
Tesseract Average Time	0.1736s
TrOCR Base-printed Average Time	0.2048s
TrOCR Base-printed Average GPU Memory	26.53 MiB
TrOCR Small-printed Average Time	0.1207s
TrOCR Small-printed Average GPU Memory	19.06 MiB

Before we can extract text to identify privacy-related language we need to vet apps that crashed using a two-pass system. We use *convert-screenshot-to-text-easyOCR.py* and leverage Tesseract for the first pass and EasyOCR for the second pass. In each pass, we attempt to identify occurrences when an app crashes, leading to screenshots of the home page. To detect these errors, we extract the text and use pattern matching to detect errors. For example, “Play Store” text reliably appears on the home screen after an app crash, therefore we use the "Play Store" pattern to detect crashes. Our second pass using EasyOCR helps to reduce false negatives due to Python Tesseract’s poor

performance. As a result, we created a new CSV containing only the apps that worked. After, we can extract the text from the apps that worked using EasyOCR.

After we have parsed our apps that crashed, we can extract privacy-related language from the captured screenshots. To do so, we use *identify-privacy-related-ui.py*, to search the OCR'd text for privacy language using our extended dictionary. This will generate two new directories showing the detection of consent related screens shown to the users.

- *privacy-related-uis*: containing privacy-related screenshots from apps. The naming convention is by appending the APK name to screencap.png, resulting in apk-name_screencap.png.
- *privacy-related-uis-text*: containing extracted privacy-related text. The naming convention is by appending the APK name to .txt, resulting in APK-name.txt.

9.2.1 Analysis Results

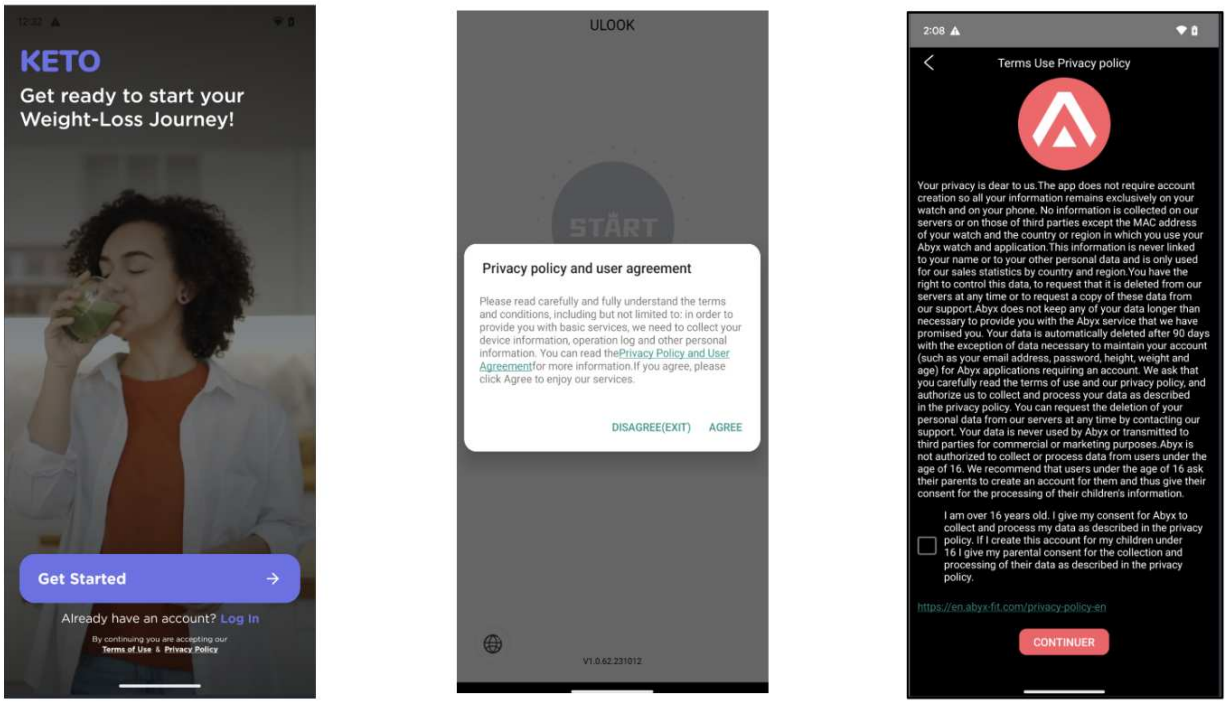

Figure 9.2: Three categories of consent screens shown to the user

Table 9.3: Number of consent screens shown

Consent Screen Shown	Apps
True	255
False	5801

Table 9.4: Consent detected by type of PII for subset of app network transmissions

Type of PII	False (%)	True (%)
AID	90.63%	9.36%
GSF	93.25%	6.74%
EMAIL	88.43%	11.56%
SERIAL	86.44%	13.55%

App developers must obtain lawful consent compliant with Articles 6(1)(a), 7, and Recital 32 of the GDPR. We analyzed 13,010 mHealth apps out of our dataset of 13,177. However, 54.26% (7,060/13010) of apps failed to run due to apps detecting the testing environment or crashing. Worryingly, Table 9.3 shows only 4.28% (255/5950) of mHealth apps showed a consent screen to the user. We have categorized these 255 apps into three groupings: apps that show a privacy policy link, equivalent to no consent screen, apps that show a small self-contained snippet of text, and finally apps that have a scrollable consent dialog screen (see Figure 9.2). 153 apps only showed a link to the users, 65 apps had a self-contained snippet of text, and only 37 apps showed a consent dialog that is a scrollable element, hinting at a potential of full compliance with the GDPR. When analyzing a subset of tested apps' network traffic packets, over 88% of network transmissions shared PII without obtaining consent from the user. For example, Figure 9.4 shows that 90.63% of network transmissions shared the user's Google Advertising ID (AID) without consent.

Our analysis results indicate that the majority of app developers seem to disregard consent management in their apps; thus violating our GDPR Property *Consent Management* and by default Articles 6(1)(a) and 7, and Recital 32 of the GDPR. This may be due to a lack of regulation understanding, implementation hurdles, and lack of tools to measure compliance.

9.3 PTA2 - Inconsistent Permissions

Android permissions act as an access control mechanism in Android OS. They aid in protecting sensitive user data and hardware. Apps can request permissions to use within their AndroidManifest.xml file. Android app descriptions can be found on the Google Play store. These natural language descriptions are the main resource users use to build conceptual models of what type of data an app will collect, what permissions it will need to use, etc. Inconsistencies occur when apps are requesting permissions but their descriptions do not mention them, violating our GDPR Property *Data Collection*. These inconsistencies indicate the app is not being clear and transparent about the data it collects, which violates default Article 5(1)(a) and 13 of the GDPR which requires data to be processed in a lawful, fair, and transparent manner. If permissions are not justified by the description of the app it could be collecting data for an unclear purpose which would violate *data minisation* described in Article 5(1)(c) and *purpose limitation* from Article 5(1)(b) of the GDPR.

Fine-tuning Experiment and Results

We fine-tuned 6 pre-trained LLMs, T5 [45], Electra [46], ALBERT [47], DistilBERT [36], DeBERTa [34], and BERT [33]. We trained for 5 epochs, a learning rate of 2e-5, and a batch size of 8, as recommended by Sun et al., 2019 [37]. We used Google Colabs free NVIDIA Tesla T4 GPU and CSU's Falcon HPC cluster with an NVIDIA A100 40 Gb GPU. Google Colab took approximately 1 to 4 hours to fine-tune (resource limits) while the HPC cluster was around 40 minutes. Due to this, we fine-tuned all 6 models on the Falcon HPC cluster. Following previous literature, we used ROC-AUC and F1 as evaluation metrics.

First, we fine-tuned the original AC-Net dataset, however, the results were not great. The evaluation F1 score was 0.63 and the ROC AUC was 0.949. Compared to previous literature, the ROC AUC was decent but the F1 score was too low for our comfort. we proceeded to inspect the data closely. we found there was a clear label imbalance. The minority labels had 3 to 5 times fewer examples as depicted in Figure 1. We resorted to data augmentation to help balance this imbalance.

Table 9.5: AC-Net's [24] annotated dataset labels mapped to Android permissions

Label	Permissions
STORAGE	WRITE_EXTERNAL_STORAGE, READ_EXTERNAL_STORAGE
CONTACTS	GET_ACCOUNTS, READ_CONTACTS, WRITE_CONTACTS
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
CAMERA	CAMERA
MICROPHONE	RECORD_AUDIO
SMS	READ_SMS, SEND_SMS
CALL_LOG	READ_CALL_LOG
PHONE	CALL_PHONE
CALENDAR	READ_CALENDAR
SETTINGS	WRITE_SETTINGS
TASKS	GET_TASKS, KILL_BACKGROUND_PROCESS

Table 9.6: Example sentence with app permissions showing interdependencies between CALL_LOG and PHONE permissions labels

Sentence	CALL_LOG	PHONE	...	TASKS
See call logs and answer the phone ...	1	1	...	0

We opted to use, *NLPAug*, a Python library that provides various data augmentation techniques for NLP tasks [48]. One of the techniques that *NLPAug* supports is called "Insert word by contextual word embeddings using BERT" [48].

This technique uses word embeddings from BERT to insert a new word into a sentence in a way that maintains the contextual meaning of the sentence. To perform this augmentation, *NLPAug* feeds sounding words into BERT to find the most suitable words for augmentation [48].

We produced 6,276 augmented sentences for the minority classes from the training set shown in Figure 9.4. This approach helped to balance the dataset. We will later compare the fine-tuning results with and without data augmentation.

We tried to simply oversample the minority labels but that was tricky as the data is multi-labeled. Merely selecting, say rows with positive instances of CALL_LOG to augment would in turn augment other labels if CALL_LOG was not alone, which is often the case, as permissions have interdependencies., as shown in Table 9.6.

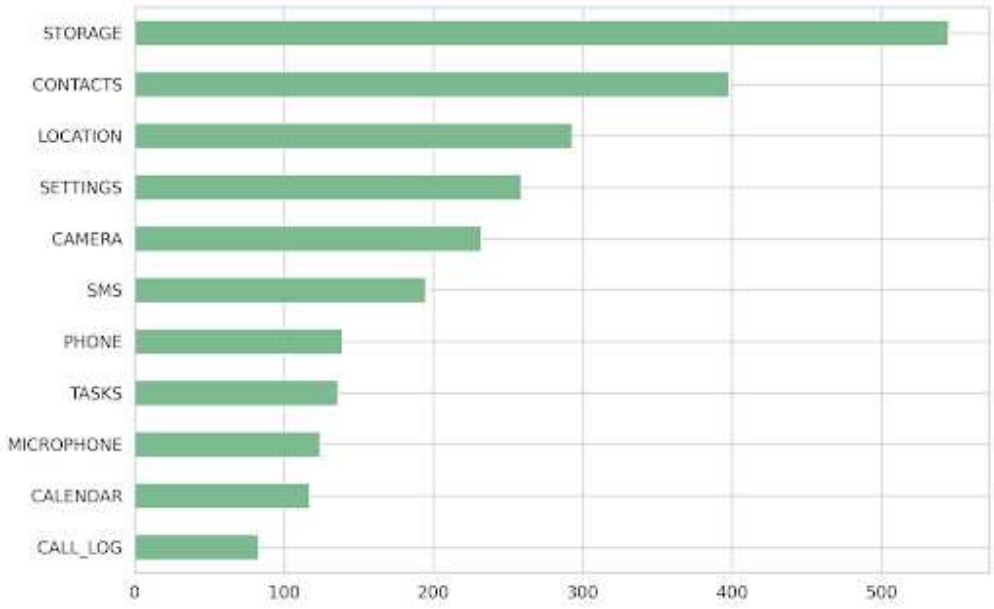

Figure 9.3: Class labels of the training dataset pre-augmentation

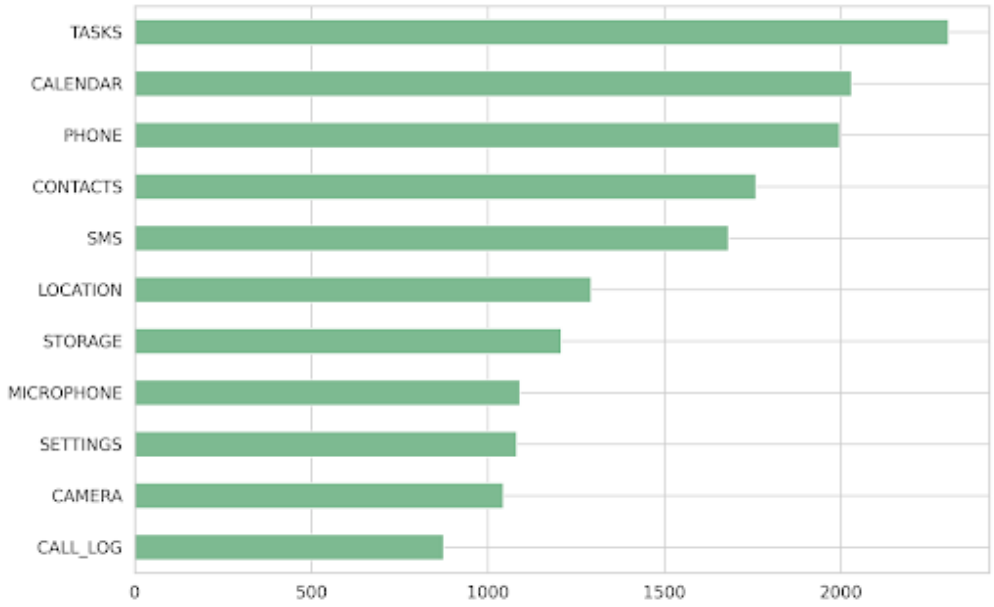

Figure 9.4: Class labels of the training dataset post augmentation

We augmented all rows and weighed each label differently. Weight in this context means how many augmented rows we will create. We weighed the majority labels 1 and the minority labels 2-5. This led to better label balance as shown in Figure 2. Thus, helps to balance the dataset.

Table 9.8 shows the evaluation results for the 6 fine-tuned models. All models performed well based on their ROC AUC scores. However, BERT-based models outperformed models such as T5-small. Furthermore, smaller models such as DistilBERT had similar performance to larger models like BERT. Table 9.7 compares our fine-tuned BERT model to other researchers' approaches. We achieved the highest F1 (0.949) and ROC AUC (0.994) scores, indicating that the transformer and data augmentation approach worked well. Based on the results, we choose to keep the fine-tuned BERT model for inferences. The next section describes our process for analyzing 13,177 apps.

Table 9.7: Comparison of fine-tuned BERT model performance to previous work [22–24]

Model / Paper	**ROC-AUC**	**F1**
BERT fine-tuned	0.99	0.94
AC-NET [24]	0.97	-
Keyword Based [24]	0.78	-
AutoCog [23]	0.76	0.92
WHYPER [22]	0.60	0.82

Table 9.8: Evaluation metrics for top fine-tuned models

Eval Loss	**Eval Model**	**Eval F1**	**Eval ROC AUC**	**Eval Accuracy**	**Epochs**
0.045	t5-small	0.588	0.968	0.852	5
0.034	google/electra-small-discriminator	0.7466	0.985	0.884	5
0.016	microsoft/deberta-v3-base	0.8979	0.995	0.949	5
0.011	albert/albert-base-v2	0.937	0.994	0.971	5
0.013	bert-base-uncased	0.949	0.994	0.978	5
0.01	distilbert-base-uncased	0.941	0.996	0.974	5

PTA2's overall analysis process is shown in Figure 9.5. The first step of PTA2 is to create a CSV of APKs and their paths on the server using the *apk_csv_maker.py*. After we run *permission.py*. This script completes a set of tasks. First, it iterates over the CSV generated by *apk_csv_maker.py*.

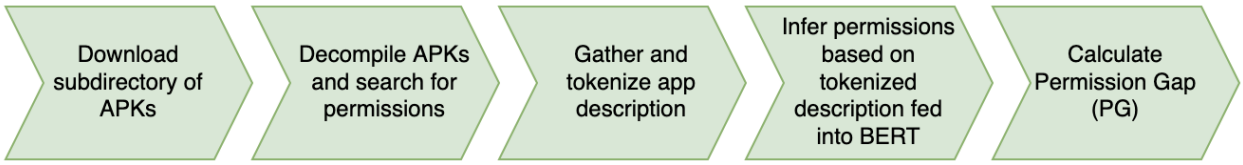

Figure 9.5: PTA2 analysis process

Each APK is decompiled using JADX [49]. After the app is decompiled, requested permissions are searched for within the *AndroidManifest.xml* using regex. However, our fine-tuned model can only predict 11 permission groups. As a result, we filter the requested permissions, keeping the ones we can predict. Next, the application's description is collected by searching a metadata JSON file, generated when the apps were downloaded. After collecting the description, we split it into sentences using the PunktSentenceTokenizer provided by NLTK's *sent_tokinze* function [50]. We then removed HTML tags and converted the text to lowercase. After, the description is tokenized using our fine-tuned BERT's tokenizer. Finally, the tokenized text is passed into the fine-tuned model to make inferences. The raw output of the model is collected and converted to labels using a threshold of 0.5. Next, the Permission Gap (PG), which is the set difference between the filtered requested permissions and predicted permissions, is calculated. A PG can exist if and only if it is not equal to the empty set. An instance of a PG represents the permissions that are requested but not inferred, indicating an inconsistency. The script creates output files. Below, we will describe each.

Table 9.9: App sentence and prediction

App	Sentence	Predictions
com.snorelab.app	Record and track your snoring with the no.1 snore app.	[MICROPHONE]

sentences_with_predictions.csv contains detailed results for each app, including the app's package name, individual sentences from the app's description, and the predicted permissions based on the model's inference. All apps will be shown regardless of whether there was a PG detected (see Table 9.9).

Table 9.10: Permission Gap results

App	Permission Gap
com.snorelab.app	True

permission_description_gap_results.csv provides a summary of PGs for each app, indicating whether there is a PG detected. It includes the app name and a boolean value (True/False) representing the presence of a PG, where True indicates detected and False indicates a PG was not detected (see Table 9.10).

Table 9.11: meta_data.csv containing package name, requested permissions, predicted permissions, and permission gap

App	Requested Permission	Permission Gap	Predicted Permission
com.snorelab.app	[GET_TASKS, RECORD_AUDIO]	[GET_TASKS]	[RECORD_AUDIO]

meta_data.csv contains additional metadata for each app, including the filtered requested permissions, permission gap, predicted permissions by our fine-tuned model, and all requested permissions (omitted due to length) (see Table 9.11).

9.3.1 Analysis Results

Table 9.12: Permission Gaps detection results for 10,031 mHealth apps

Permission Gap Detected	Count
True	8396
False	1635

App developers must ensure their app's description accurately describes the need for the permissions they request. A PG occurs when there is a set difference between an app description's inferred permissions (predicted permissions) and its requested permission (filtered requested permissions). An occurrence of a PG indicates the app is violating Article 5(1)(a-c) of the GDPR.

Table 9.13: 10 permissions within Permission Gaps for 10,031 mHealth apps

Permission	Count
CAMERA	5834
ACCESS_FINE_LOCATION	5027
ACCESS_COARSE_LOCATION	4978
WRITE_EXTERNAL_STORAGE	4923
READ_EXTERNAL_STORAGE	4712
RECORD_AUDIO	4175
READ_CALENDAR	3190
READ_CONTACTS	2576
CALL_PHONE	789
GET_ACCOUNTS	653

We analyzed 10,031 out of our dataset of 13,177 due to decompilation errors and some apps being taken off the Google Play Store before we scraped their descriptions. Table 9.12 shows 83.7% (8396/10031) of mHealth apps having a PG. Furthermore, the most common permissions requested inside PGs are run-time permissions with a dangerous protection level, e.g., *CAMERA* with 6833 instances inside the PG and *ACCESS_FINE_LOCATION* with 5882 instances inside the PG (see Table 9.13). Our analysis results show the majority of app descriptions do not align with their requested permissions. Thus, the majority of apps are violating key GDPR principles including *data minisation*, *purpose limitation*, and Articles 13 and 5(1)(a-c).

Chapter 10

Conclusion

10.1 Lessons Learned

In this thesis, we proposed a novel PT analysis process with results for 13,177 mHealth apps. We achieved this by creating a Data Lifecycle model and mapping it to our categorization of the GDPR into GDPR Properties. We then created novel PTs, focusing on PTs relating to data collection. Our app vetting process vetted 13,177 mHealth applications. To do this, we created a manually annotated dataset of 2,990 non-mHealth and mHealth apps. We used this dataset to fine-tune an LLM to classify mHealth apps based on their description. This allowed us to vet 500,000 apps ending with a dataset of 13,177 mHealth apps. We then analyzed two PTAs (PTA1 and PTA2) on this dataset of 13,177 mHealth apps, inspecting the absence of consent management systems, and inconsistency between app descriptions and requested permissions. Overall what we found was concerning, only 4.28% (255/5801) had a consent management system. Over 88% of apps subsequently collected PII without consent. 83.7% of application descriptions were not consistent with the permissions the app was using. We hope both users and developers can use our PTAs to assess an app's compliance in terms of data collection practices. Making it so, app developers are aware of any non-compliance and are given the chance to implement fixes and to make users aware when their privacy is being violated.

10.2 Future Work

There are several avenues for future work. Consistency checks between collected data and natural language sources can implemented. For example, the consistency between the PII mentioned in the app descriptions to the PII found in collected logs can be assessed. There were quite a lot of apps that crashed during dynamic analysis. Work can be done to improve the current state of dynamic analysis techniques. We have 8 other PTAs that can be used for future analysis focused on

different parts of the Data Lifecycle and their associated GDPR Properties. We noticed similarities between the GDPR and other privacy regulations. Work can be done to extend our GDPR categorization to other regulations. New augmentation techniques such as Vorobev et al., 2023 [51] paraphrasing model *humarin/chatgpt_paraphraser_on_T5_base* can be used to create better augmentations of AC-Net's dataset. Lastly, using our app vetting process, new categories of apps, such as IoT apps, can be analyzed.

Bibliography

[1] Naveen Kumar. Android Usage Statistics 2025: Devices & Market Share, December 2024.

[2] European Parliament and Council of the European Union. Regulation (eu) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). *OJ*, L 199:1–88, 2016-05-04.

[3] Staista. Mobile os market share worldwide 2009-2024. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#statisticContainer. [Accessed 01/25/2025].

[4] Google. Android Platform Architecture. https://developer.android.com/guide/platform. [Accessed 01/31/2025].

[5] Google. Application fundamentals. https://developer.android.com/guide/components/fundamentals. [Accessed 03/13/2025].

[6] Google. Introduction to activities. https://developer.android.com/guide/components/activities/intro-activities. [Accessed 01/31/2025].

[7] Google. Services overview | Background work. https://developer.android.com/develop/background-work/services. [Accessed 01/31/2025].

[8] Google. Content providers | App data and files. https://developer.android.com/guide/topics/providers/content-providers. [Accessed 02/15/2025].

[9] Google. Permissions on Android | Privacy. https://developer.android.com/guide/topics/permissions/overview. [Accessed 02/16/2025].

[10] Li Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88:67–95, August 2017.

[11] Google. UI/Application Exerciser Monkey | Android Studio. https://developer.android.com/studio/test/other-testing-tools/monkey. [Accessed 01/31/2025].

[12] Google. Google AdMob - Earn More With Mobile App Monetization. https://admob.google.com/home/. [Accessed 01/30/2025].

[13] Google. Google Analytics. https://developers.google.com/analytics. [Accessed 01/30/2025].

[14] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. An Empirical Evaluation of GDPR Compliance Violations in Android mHealth Apps, August 2020. arXiv:2008.05864.

[15] Alireza Ardalani, Joseph Antonucci, and Iulian Neamtiu. Towards Precise Detection of Personal Information Leaks in Mobile Health Apps, September 2024. arXiv:2410.00277.

[16] Trung Tin Nguyen, Michael Backes, and Ben Stock. Freely Given Consent? Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 2369–2383, New York, NY, USA, November 2022. Association for Computing Machinery.

[17] Fangwei Zhao, Serge Egelman, H. Weeks, N. Kaciroti, Alison L. Miller, and Jenny S. Radesky. Data Collection Practices of Mobile Applications Played by Preschool-Aged Children. *JAMA pediatrics*, September 2020.

[18] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share First, Ask Later (or Never?) Studying Violations of {GDPR's} Explicit Consent in Android Apps. pages 3667–3684, 2021.

[19] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy, 2010–. [Version 11.1].

[20] Frida A world-class dynamic instrumentation toolkit.

[21] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 23–26. IEEE Press, 2017.

[22] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. {WHYPER}: Towards Automating Risk Assessment of Mobile Applications. pages 527–542, 2013.

[23] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1354–1365, Scottsdale Arizona USA, November 2014. ACM.

[24] Yinglan Feng, Liang Chen, Angyu Zheng, Cuiyun Gao, and Zibin Zheng. AC-Net: Assessing the Consistency of Description and Permission in Android Apps. *IEEE Access*, 7:57829–57842, 2019. Conference Name: IEEE Access.

[25] MobSF is an open-source automated security testing tool for mobile apps.

[26] Gioacchino Tangari, M. Ikram, K. Ijaz, M. KÃćafar, and S. Berkovsky. Mobile health and privacy: cross sectional study. *The BMJ*, 373, June 2021.

[27] Xin Jin, Sunil Manandhar, Kaushal Kafle, Zhiqiang Lin, and Adwait Nadkarni. Understanding IoT Security from a Market-Scale Perspective. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1615–1629, Los Angeles CA USA, November 2022. ACM.

[28] World Health Organization. Mhealth. *SEVENTY-FIRST WORLD HEALTH ASSEMBLY A71/20 Provisional agenda item 12.4*, Mar 2018.

[29] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[30] google-play-scraper: Google-Play-Scraper provides APIs to easily crawl the Google Play Store for Python without any external dependencies!

[31] Klaus Krippendorff. Bivariate Agreement Coefficients for Reliability of Data. *Sociological Methodology*, 2:139–150, 1970. Publisher: [American Sociological Association, Wiley, Sage Publications, Inc.].

[32] Mike Gruszczynski. MikeGruz/kripp.boot, November 2023. original-date: 2013-04-06T21:30:06Z.

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. arXiv:1810.04805.

[34] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with Disentangled Attention, October 2021. arXiv:2006.03654.

[35] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, July 2019. arXiv:1907.11692.

[36] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, March 2020. arXiv:1910.01108.

[37] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to Fine-Tune BERT for Text Classification?, February 2020. arXiv:1905.05583.

[38] Huggingface. Optimization. https://huggingface.co/docs/transformers/main_classes/optimizer_schedules. [Accessed 03/13/2025].

[39] Google. LocationManager | API reference. https://developer.android.com/reference/android/location/LocationManager. [Accessed 03/13/2025].

[40] Google. Android API reference. https://developer.android.com/reference. [Accessed 03/13/2025].

[41] Google. Sensors Overview | Sensors and location. https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview. [Accessed 03/13/2025].

[42] Jaided AI. Easyocr.

[43] Samuel Hoffstaetter. Pytessaract.

[44] Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models, September 2022. arXiv:2109.10282 [cs] version: 5.

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[46] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[47] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[48] Edward Ma. Nlp augmentation. https://github.com/makcedward/nlpaug, 2019.

[49] skylot. jadx. original-date: 2013-03-18T17:08:21Z.

[50] NLTK. nltk.tokenize.sent_tokenize. https://www.nltk.org/api/nltk.tokenize.sent_tokenize.html. [Accessed 02/15/2025].

[51] Maxim Kuznetsov Vladimir Vorobev. A paraphrasing model based on chatgpt paraphrases. 2023.