Research Report

Omar Ramadan – C00286349

Cybercrime and IT Security

Supervisor: Mark Cummins

Table Of Contents:

Abstract:

This project presents the design and implementation of a static analysis tool for Android applications distributed via the Google Play Store. The tool addresses the gap between privacy policies and user comprehension by leveraging machine learning and natural language processing to analyse privacy policy documents.

The system will employ a fine-tuned Large Language Model (MobileLLaMA 2.7B) trained on publicly available datasets such as OPP-115 Corpus. The web-based interface allows for users to submit application identifiers such as name and developer and receive plain-language summarisations of data collection practices, third-party sharing policies, and permission mismatches.

The research portion will evaluate 100 popular Android applications across 10 categories and will show the gaps between applications stated privacy policies and their actual APK permissions. This will reveal how applications that do not explicitly state what they are doing with user data are being downloaded by thousands of users worldwide, highlighting the amount of information these applications hold over users and potentially profiting from. This will take place after the static APK analyser has been built and will be posted on the web-application alongside the tool for users to read and become more aware of.

This project contributes to making users aware of online privacy, compliance with privacy regulations through the implementation of PIA's, and establishes a re-usable tool for end user usability.

Introduction:

My motivation for developing this tool stems from an RTE PrimeTime investigation broadcast on 18 September 2025, which revealed that precise location data from tens of

thousands of Irish smartphones was available for purchase from data brokers operating in the digital advertising industry (McDonald & Heffernan, 2025).

This begs the question of what exactly is tracking these individuals and using the data for profit by selling it.

Applications on the Google Play Store all have privacy policies and terms and conditions policies that users are required to read and accept before using the application, however most users just accept these policies without reading them to access the application/service. This means there could be seemingly "harmless" applications on the Google Play Store that are actively storing user data against their knowledge and sharing this information with third parties for profit.

Users are not informed of the risks of their location data being readily available on the internet and lack the care to carefully read policies laid out before them by companies/services. That is where I gained the motivation to build a web-application that inexperienced users can use to see what applications are taking from them in terms of information. This can help empower users to care about their online privacy and reduce the number of victims of these apps.

Overview of areas, technologies or topics researched:

Large Language Models (LLM) for Privacy Policy Analysis

Large Language Models (LLMs) are transformer-based neural networks trained on massive text corpora to understand and generate human language. Recent architectures just like GPT, LLaMA and BERT have demonstrated remarkable capabilities in semantic fine-grained understanding.

Transformer Architecture

The attention mechanism enables models to weigh contextual relationships between words regardless of distance in text. For privacy policies, this allows the model to connect "we collect location data" with clauses 30 paragraphs later stating "data may be shared with third-party advertisers (Vaswani et al., 2017.)

Model Selection: MobileLLaMA 2.7B

I selected this model due to its efficient inference on low-end components. On my current setup with an RTX 2060 super, it can reach process 30-40 tokens/seconds. It has strong performance on legal text understanding, making it ideal for my use case. It has quantization support, which is the process of shrinking LLMs to consumer less memory, require less storage space and make them more energy efficient (Valanzuela A., 2025.)

Fine-Tuning Methodology

I am utilising multiple training datasets to train the model. OPP-115 Corpus is an annotated dataset of website privacy policies for natural language processing. It is the basis of the model and will be the second dataset the model is trained off due to its annotations specifying data practices in text, making it easier for the LLM to train off it. This will take an estimated 50 hours on the RTX 2060 super and is the most important dataset as it will teach the model to extract accurate necessary information from legal texts.

Instruction-Tuning Format

I must construct instruction-pairs for the LLM to learn off. The instruction pair format is:

- {
  "instruction": "Identify data retention policies",
  "context": "[privacy policy text]",
  "response": "The policy states data is retained for 5 years..."
  }

These instruction pairs are critically important for training LLMs as they must be fine-tuned to transform the raw language model into exactly what type of tool you require. The model's ability to generate text and follow specific human commands will be achieved through the prompt-response engineering process, teaching it to understand and execute instructions like summarizing, translating, or interpreting accurately. They also allow the me to explicitly train the model to do exactly what I need without straying from its original purpose or providing unwanted information.

Natural Language Processing (NLP) for Legal Text

Natural Language Processing is a vital technology enabler for automated privacy policy analysis, enabling the transformation of unstructured, free-text legal documents into machine-readable structured data. Legal texts, such as privacy policies and terms of service agreements, exhibit deliberate ambiguity devised to retain operational flexibility. Three core NLP techniques have been investigated in this work for privacy policy analytics; sentiment and vagueness detection for linguistic clarity and cross-reference analysis for the detection of internal contradictions.

Named Entity Recognition (NER)

NER is considered a foundational NLP task where the system recognises and categorises predefined categories of entities in unstructured text. Within the domain of privacy policies, the extraction of structured privacy practices from narrative legal documents is based on NER. Noticeable different from general-domain NET, its main objective being to identify persons, organizations and location, domain-specific NER for privacy requires

recognising specific entities that are critical for an understanding of data handling practices.

Sentiment and Vagueness Detection

Vagueness in privacy policies represents a deliberate linguistic strategy that creates flexibility for data controllers while undermining the user's ability to provide informed consent. A formal theory introduced on vagueness develops a taxonomy of vague terms through empirical content analysis of 15 privacy policies. Investigation shows that vagueness significantly impacts privacy risk perception: as statement vagueness increases, users are less likely to share personal information, but users paradoxically accept vague policies without reading them due to perceived transaction costs (Bhatia et al., 2016).

Modal Verbs are one of the main indicators of linguistic certainty levels, highlighting the contrast between definitive statements ("We collect your location data") and modal qualifications ("We may collect your location data") brings in uncertainty that masks real data practices. O'Neill's study on sentential modality in legal language places a distinction between deontic modalities (obligations, prohibitions, permissions expressed as "must", "shall", "may") and epistemic modalities (certainty levels expressed through "will", "might", "could"). His approach on the topic resulted in an 87% classification rate for modal certainty in financial regulatory documents, proving that modal verb patterns give significant signals to automated vagueness detection (O'Neill et al., 2017).

Hedging in a linguistic sense involves the use of words or phrases to convey uncertainty, tentativeness, or lack of commitment and is a fundamental element of privacy policy language. Hedge phrases include peacock expressions that indicate something is "very likely," "generally", or "typically"; weasel words such as "some believe" and "it is understood that"; and evidential markers that indicate the reliability of an information source.

Del Alamo et al. (2022) conducted a systematic mapping study on automated privacy policy analysis and identified vagueness detection as a key research priority, indicating that ambiguous language undermines the transparency function of privacy policies. The identification of quantitative vagueness relies on machine learning classifiers trained on human-annotated texts to predict scores for vagueness. Tang et al. (2025) showed that Large Language Models, especially GPT-4, outperform traditional symbolic and statistical NLP method in identifying vague privacy practices with a query consistency of 89.5% on repeated queries (Tang et al., 2025). Therefore, in these cases, transformer-based models have a better contextual understanding for capturing semantic vagueness than rule-based systems could ever achieve.

Cross-Reference Analysis: Detecting Internal Contradictions

Internal Contradictions in privacy policies represent a level of policy information deficiency that confuses users. Andow et al.'s (2019) PolicyLint system introduced formal modelling of contradictions and providing algorithmic techniques to detect contradictory policy statements. PolicyLink also defines logical contradictions as statements that make opposing claims about identical data practices. For example, a policy states "We do not collect users' personal information" in its introduction, followed by descriptions of personal data collection in later sections constitutes a direct logical contradiction (Andow et al., 2019)

Accurate contradiction detection requires complex negation handling. When an NLP system tries to find contradictions, it must correctly understand which parts of a sentence are being negated (denied/reversed) and which are not. There are several negation techniques employed by organizations when building their Terms of Service and Privacy Policy agreements, such as:

- **Partial Negation:**
  "We do not share your personal information with third parties for marketing purposes."

  What exactly is being negated is unclear, and the ambiguity of the statement creates two different meanings:
  - The personal data is not shared for marketing purposes (narrow scope)
  - Data is being shared with third parties outside marketing purposes (broad scope)

- **Double Negatives:**
  "We cannot guarantee that your data will not be access by unauthorized parties."

  This actually means data *might* be accessed by unauthorized parties. A naive NLP system could see both "cannot" and "not" and get confused.

- **Implicit Negation:**
  "Your data remains confidential within our organization."

The statement only implies no external data sharing without explicit words stating it such as “not” or “never”. Detecting these contradictions require understanding the implications of the statement.

- **Negation in Subordinate Clauses:**
  “We do not collect personal data unless you provide consent.”

  The negation (“we do not collect”) had an exemption (“unless you provide consent”). So, the policy does collect data in some circumstances but only when the user consents. If another section goes on to say, “we collect your email address when you sign up”, is that a contradiction? Only if sign-up does not involve consent.

- **Lexical Negation:**
  “We prohibit third-party access to your data.”
  “We prevent unauthorized sharing.”
  “Your data is inaccessible to external parties.”

  Words like “prohibit,” “prevent,” and “inaccessible” contain negation within them, no explicit “not” appears. The system must recognize these as negative statements also.

Overall the research presented in PolicyLint found that prior negation detection approaches failed on 28.2% of privacy policies due to inadequate handling of these complex structures (Andow et al., 2019).

This is important for my tool, as incorrect negation handling will produce false positives (detecting non-existing contradiction) and false negatives (missing real contradictions). My tool addresses this issue as training MobileLLaMA to understand concepts like Contextual Understanding, Domain Training and Semantic Reasoning, and shows why sophisticated Natural Language Processing is necessary rather than simple keyword matching. Accurately determining what a privacy-policy is promising requires understanding complex linguistic structures, including negation.

Analysing the different modalities of verbs, vagueness, linguistic hedging, and studies on quantitively vague measurements are vital to the researching of Privacy Policy linguistics

and patterns and forming appropriate training methods targeted at exposing these ambiguous policies that attempt to undermine the user.

Static APK Analysis:

Static analysis of Android APKs is key for checking app security and privacy without executing the program. Instead of dynamically analysing the application, requiring it to be run, static analysis involves inspecting the code files, settings, and built-in assets to spot any risks early on, such as unnecessary permissions being requested or data misuse (Laburity, 2024). This way we can match what the app says it does with what its code actually allows.

The Android Package Kit (APK) works similar to a ZIP file, holding everything it needs to run an app on Android devices. The app behaviour can be gathered from the contents of the code and can reveal any bad practices the application partakes in. Pre-existing tools such as Androguard can break down these code segments step by step through an open-source python Framework specifically designed for reverse engineering Android applications.

Android Package Structure

An APK file follows a clear layout set by Android where every part has its own job (StackOverflow, 2013). Due to this setup, security checks become easier as researchers can pinpoint exactly where certain details sit in the applications code.

AndroidManifest.xml - Application Configuration and Permissions

The AndroidManifest.xml file acts as the main setup guide for the app and holds key details that shape how it runs and stays secure (Appdome, 2023). It is stored in binary format inside the APK and specifies the following:

- The package name, which is a distinct application ID, such as “com.example.healthtracker” that systems use universally to identify the software.
- The version information, the current build version of the application and is used to track updates.
- The permissions declaration, a detailed list of private data the application requests from the user that it wants to use, such as location data or contact information.
- App components and activities such as main user interface screens, service tasks that run in the background and broadcast receivers that respond when system events occur, or the user triggers an action. Content providers manage data cross applications by enabling shared storage use.

- The minimum Software Development Kit (SDK) version or lowest Android API level the application supports, determining compatible devices and available system features.
- The hardware requirements laid out by the application, such as a camera, GPS, NFC, or telephony abilities.

The AndroidManifest.xml file acts as the main reference for requested permission in terms of privacy review, making it easy to view the stated access rights in contrast with what is being noted in the privacy policies and flagging them automatically. Certain permissions can reveal early on how the application plans to interact with system features. This file is quite important for my tool to analyse APK's, as it clearly outlines the permissions and operations of the application, allowing for easy cross analysis with Terms and Conditions and Privacy policies.

Classes.dex - Dalvik Executable Bytecode

The classes.dex file holds the applications compiled code as Dalvik bytecode. Most android apps use Java or Kotlin, but their source is initially turned into Java bytecode by compilers, after which it is converted into DEX form, designed for limited mobile resources. DEX files hold the app logic, UI controls, network scripts, data handling routines, and structural commands governing execution order. Because the DEX structure initially supported only up to 65,536 referenced methods, bigger apps often split into several DEX units labelled by number like classes2.dex or classes3.dex (Appdome, 2023). Bytecode analysis enables the identification of:

- Sensitive API calls, methods for accessing location services, contacts, camera, or network resources.
- Cryptographic systems that are using encryption methods to create keys while relying on protected data storage techniques.
- Obfuscation methods such as scrambling code using Proguard or similar software to change class and method names, making analysis harder through renaming tricks.
- Any hardcoded credentials such as API keys, authentication tokens or database passwords embedded directly in code.

In a study published by (Na, G. et al. 2019) it was shown that inside the ART system, which was used from Android 5.0 and onward, DEX files still existed within Optimized Ahead-of-Time (OAT) compiled apps, keeping direct links between bytecode and native instructions. Due to this, older static analysis methods designed for Dalvik continue to work on current Android releases (Na et al., 2019). Researchers continue to point out the downside of

revealing bytecode that is opening security risks, since attackers may inspect it and alter or skip key app logic. This can be used to infect inherently non-malicious applications, creating major problems for developers.

Resources/ and assets/ - Embedded Application Assets

The “resources/” directory contains compiled assets used by the code such as:

- XML Layouts: UI screen definitions specifying visual components displaying.
- Drawable Assets: images, icons and graphics
- String Resources: Text strings for internalization
- Colour Definition: Applying colour schemes
- Styles and Themes

The “assets/” directory holds the non-processed original files used by the application during execution. Some multi-platform tools such as Cordova and React Native use his directory for keeping runnable components, such as JS scripts inside (Appdome, 2023).

Privacy Implications – Embedded resources may inadvertently leak sensitive information:

Sensitive developer information may be in the applications resources, which may accidentally be leaked. Some of these resources are:

- Hardcoded URLs: Backend API endpoints revealing sensitive application infrastructure.
- API Keys: External service credentials stored within config files or string assets may provide malicious users unwarranted access to protected information.
- Debug information: Internal server URLs, testing areas or private user guides
- Data samples containing Personally Identifiable Information (PII): Test accounts, emails or geolocation entries used in testing could have been overlooked and left behind post deployment

Approov (2025), alongside OWASP MASWE-0005, highlights the dangers of API keys left behind in mobile apps, where tools using static analysis can examine sensitive string files or asset folders to determine possible leaks (Approov, 2025; OWASP, 2025). The study conducted by Kaushik, S. et al. (2024) reviewed 5135 android packages and found 2142 stored credentials across 2115 apps, showing how common this flaw remains.

Lib/ - Native Libraries and Third-Party SDKs:

The “lib/” folder contains the compiled low level code files in ELF format, usually written in C or C++, build for specific CPU types like ARM or x86 through the Android Native Development Kit (NDK) (Appdome, 2023). These code files fulfil a variety of roles:

- Performance heavy tasks such as rendering graphics, running cryptographic algorithms, or encoding videos are usually written in low-level code for efficiency to compensate for the high processing power required due to complex calculations in their operations. While boosting efficiency, this raises maintenance required with more debugging where such features are implemented and deployed regularly.
- Third-party SDKs often come with built-in features, where advertising networks, analytics platforms, crash reporting tools, and social media integration libraries frequently include their own native components.
- Some developers utilise code obfuscation, shifting critical application functions to native modules which makes analysis harder as unpacking binary code is more difficult compared to reversing Java classes (Approov, 2025)

Security Analysis Challenges: Native libraries make analysis harder as demonstrated when Sanna and team (2024) studied risks across more than 100,000 android apps; it was discovered nearly 40% used native code, with several including known flaws for widely used libraries. The library versions and risky functions were spotted using pattern matching and mapped the findings to existing vulnerability ratings to calculate app-level risk indicators.

Third-Party Library Detection: Identifying external SDKs inside applications matters for privacy checks as components of these SDKs can collect user data without the main developer always knowing, LibRadar is a tool created by Backes es al. (2017) that uses class structures to detect these third-party tools in android software, making it harder to evade detection with methods such as re-naming or folder reorganisation. Since it examines class arrangement, it correctly identifies library versions even after ProGuard scrambling occurs.

LibRadar collected 29,279 signatures of Android libraries and was gathered by examining more than a million applications on the Google Play Store. Since it covers so widely, ad tools like “Google AdMob” or “Unity ADs” along with analytics services such as Firebase and Mixpanel, these tools are easily spotted by researchers where social features and tracking modules show up clearly. Finding these external features from SDKs allows comparison between what the app claims in its policy; for instance, using Facebook SDK without naming Facebook in policy documents can count as a disclosure violation.

META-INF/ - Code Signing and Integrity Verification:

The META-INF directory contains digital proofs for checking if an application is genuine and unchanged (Appdome, 2023), with notable contents being:

- MANIFEST.MF - Contains sha-256 digests of all the files in the APK, checking if the application is genuine and unchanged (Appdome, 2023)
- CERT.SF: Contains signatures for manifest entries. The differences between this file and the MANIFEST.MF file is instead of the SHA digest value for each source file being the digest (hash) of the binary value in the source file, the digest value of a given source file is the hash of the three lines in the manifest file for the source file.
- CERT.RSA or CERT.EC: The Developers public key certificate.

Changing anything within an APK no matter how small will break/change the signature, requiring it to be signed again with the digital signature before Android allows the user to install the APK (Appdome, 2023). Attackers cannot easily insert harmful code into real apps and spread copies due to this, as the signature has changed and no longer matches the developers public key certificate.

Resources.arsc - Resource Table:

This binary file links code to translated texts, such as appropriate region-specific language strings and locale-dependant layouts (Appdome, 2023) When checking privacy, checking this file can expose hidden language options or regional functions not stated in official policies. This can be due to some applications requiring region-specific behaviour not mentioned in privacy policies, highlighting a potential issue with transparency and compliance regarding data privacy.

Permission Analysis using Androguard

Androguard is an open-source Python tool specifically designed for the analysis and modification of Android applications (Androguard Github, 2025). This platform delivered flexible static analysis features non-dependant on paid licenses and is more geared towards scholarly work over more traditional commercial solutions. It interprets the app manifests, breaks down DEX code, linking permissions to API calls which, is a key aspect when assessing how private information might be exposed and enables direct interaction with APK components.

Androids Permission Model Overview:

The android permission model follows the principle of least privilege, only allowing permissions that are explicitly requested in the AndroidManifest.xml file and that the user has consented for the application to use (Android Developers, 2025). These permissions

are classified by risk, shaping the approval process and the potential security implications of granting the requested permissions.

Permission Protection Levels:

Android uses four main security levels for applications (GeeksForGeeks, 2021):

1. **Normal Permissions (Protection Level: normal)**

Minor risk is carried from granting normal permissions as they interact with system areas not tied to personal information. For instance:

- Web access: connection to networks is required for nearly every application
- Accessing the networks state: Query network connection status to communicate with dependencies.
- Bluetooth access: Bluetooth pairing and communication
- Vibrate: control the devices vibration motor for notifications or alarms.

Standard permissions are mostly automatically approved by the system with the no user action required. Alerts are not shown by the system for permissions, and once installed, individuals cannot remove those access rights (Droidcon, 2024). Basic permissions do not demand detailed explanations in privacy notices, but full documentation should cover network usage and related functions regarding full data protection transparency.

2. **Dangerous Permissions (Protection level: dangerous)**

Dangerous permissions let applications reach personal information or change key phone functions, risking data privacy (GeeksForGeeks, 2021). On Android, these permissions are sorted into clusters, such as location, contacts, camera or storage, but allowing one can unlock others in the same set and varies by operating system version. A few instances of risky permissions include:

- Location: potentially grant location accuracy from 5-50 meters range as “ACCESS_FINE_LOCATION” can be turned on. “ACCESS_BACKGROUND_INFORMATION” allows the application to access location information even when it is not actively in use (in the background).
- Contacts: Can allow for the application to access a list of accounts (e.g., email accounts and social media accounts) that are stored in the Android Account Manager service on the device.
- Camera: Gives direct access to the camera hardware, which can include the flash function and access to the microphone. Permission must be requested from the user at runtime on Android version 6.0 and higher.

- Microphone: Allowing the application to access the device's microphone to record audio from its surroundings. Permission must be requested from the user at runtime.
- Storage: Can allow the application to read or write files on the shared drive. Has been deprecated since Android 10/11 and is bypassed by using modern APIs.

With Android 6.0 (API level 23) or newer, risky permissions require direct approval from the user through on-the-spot prompts (Droidcon, 2024). Individuals can withdraw the permissions access later in the settings, where the app should verify what current permissions are being allowed.

Checking these risky permissions first is common practice, as under GDPR rules from 2018, companies must clearly state what personal information they gather, how the information will be used and why, so when an application requests precise location access while its policy refers to just rough location data the mismatch breaks compliance standards, potentially revealing further application bad-practices.

3. **Signature Permissions (Protection Level: signature)**

Signature-level permissions let applications interact solely if they share the same signing certificate as the application defining the permission (GeeksForGeeks, 2021). Because of this setup developers can control communications between different but related applications while blocking unrelated external ones. These permissions are set by the Android OS and are limited to apps signed with the maker's official certificate (Droidcon, 2024), with one instance being access to core device controls such as:

- Creating accessibility services
- Implementing password manager auto filling
- Installing other applications.

Third-party applications cannot ger signature-based permissions for system resources as they do not have the systems signing key (StackOverflow, 2014). Therefore, these permissions will not be focused on for policy checking as they hardly show up in application manifests.

4. **Special Permissions (Protection Level: appop)**

Specific access rights control highly delicate actions that require users to manually enable them via device settings (GeeksForGeeks, 2021) with some examples being:

- System alert windows that show floating layers above applications using window permissions.
- Modifying system settings
- Triggering the installation of APK files
- Accessing application usage statistics

Special permissions are not available through regular pop-up windows and users need to enable them manually in the settings. Due to the risk of intruding on personal data, these access levels must be stated in privacy notices, explaining when and why they are being used.

Dataset Construction and Model Training

The creations of a strong language model for analysing privacy policies depends on well-labelled structured data. I am going to be describing some of the datasets researched/used in the model training and how instruction-based training samples were built, along with how Quantized Low-Rank Adaptation (QLoRA) is applied and assists in faster tuning.

Privacy Policy Datasets

Automated analysis of privacy policies relies on taking note of how legal experts interpret data handling statements. Categorizing privacy texts demands understanding legal aspects of the works and all associated terminology alongside compliance rules or other terminology associated to sectors. There is open source developed datasets tailored to this.

OPP-115 Corpus (ACL 2016)

The OPP-115 Corpus serves as a core resource for studying privacy policies and is a collection of 115 privacy policies that have been manually annotated to highlight certain data handling practices (Usable Privacy, 2023). It amounts to 267 pages along with 23,000 detailed labels on data practices and 128,000 tags marking specific attributes of those practices (Wilson, S et al. 2016)

The full collection was created by three law graduates, guaranteeing skilled understanding of the legal terms and data practices being described (Usable Privacy, 2023). The labelling method organizes content into ten types of data handling activities, making it better than simple grouping, with the types of data handling being:

1. **First Party Collection/Use:** Practices describing data collection by the website itself.

2. **Third Party Sharing/Collection:** Disclosures about data sharing with external entities.
3. **User Choice/Control:** The tools given to the user to handle their own information.
4. User Access, Edit and Deletion: Mechanisms for users to access or modify their data.
5. **Data Retention:** How long collected data is stored.
6. **Data Security:** Measures taken to protect user information.
7. **Policy Change:** Procedures for notifying users of policy modifications.
8. **Do Not Track:** Response to user "Do Not Track" signals.
9. **International and Specific Audiences:** Policies regarding children, EU users, or other specific groups.
10. **Other:** Miscellaneous privacy-related statements.

In every group, labels specify practice attributes including personal information types collected, purposes of collection, user types affected and third-party entities (Wilson et al., 2016). This layered labelling system supports broad sorting and helps find key practice groups and specifics on what data is being taken and why.

Something to be considered is the OPP-115 dataset includes text snippets with labels, but the full versions of the privacy policies must still be gathered online. To get entire documents for deeper review, I can implement automated URL extraction and site crawling techniques to acquire the privacy policies.

Wang et al. (2025) looked at the differences in labelling impact Machine Learning results with OPP-115 and discovered better agreements leads to stronger performance in nearly all cases. F1-scores rose, especially for First Party Collection/Use and Third-Party Sharing/Collection labels (Wang et al., 2025).

The F1 score is a metric for performance for machine learning algorithms, and consists of balancing two things to achieve the average score:

- **Precision:** Tells what portion of correct guesses by the LLM made up all the cases that were labelled positive.
- **Recall:** Shows the portion of true positive the LLM actually detected.

The F1 score is calculated as: F1 = 2 * (precision * recall) / (precision + recall)

It is particularly helpful when class sizes differ greatly and focuses on both precision and recall instead of just improving one.

PolicyQA Dataset

PolicyQA extends OPP-115 into a reading comprehension format, containing 25,017 question-answer pairs curated from the 115 privacy policies (Ahmad et al., 2020). While other datasets label parts of text, this one includes 714 real-user questions covering diverse data practices where answers are pulled directly as exact phrases (Ahmad et al., 2020). This question-answer format works well for instruction tuning as it mirrors how users ask questions and how the system should reply. A few sample queries include:

- "What personal information does the company collect?"
- "Does the website share my data with advertisers?"
- "How can I delete my account?"
- "What security measures protect my data?"

Ahmad et al. (2020) tested neural question-answering questions like BiDAF and BERT using PolicyQA, which boosted results accuracy. Instead of training models on scratch, starting on Stanford Question Answering Dataset (SQuAD) data and then refining further with PolicyQA lead to better results in the study, which is the approach I will be adopting when training my LLM.

PrivacyQA Dataset

PrivacyQA is a dataset comprised of 1,750 queries on app privacy rules, along with more than 3,500 labels from specialists (Ravichander et al., 2019). In contrast to PolicyQA's method of pulling direct answers, judging relevance is more emphasized here where with a question plus a policy of text, experts can decide whether the segment is relevant to answering the question (Ravichander et al., 2019).

The data was split into 27 applications with 1,350 questions, alongside a test section comprised of 8 applications and 400 questions. No document appears in both data parts, which helps ensure proper and fair generalization (Ravichander et al., 2019). According to Ravichander et al. (2019), neural models were observed to be considerably worse than people on PrivacyQA, meaning it is still not fully understood how machines interpret data policies.

The labels in OPP-115 and PrivacyQA match, which in my case can help with sequential training in my scenario. Every query is tagged with applicable OPP-115 classes, making multi-task learning that simultaneously classifies relevance and identifies answer parts feasible; however, I will not be utilizing multi-task learning as my hardware is not sufficiently equipped to handle the excess workload.

Princeton-Leuven Longitudinal Corpus

Princeton-Leuven is a different type of dataset, as it does not include any annotated data, but offers more than 1,071,488 English privacy policy versions from 130,620 different sites across 1996-2019 (Amos et al., 2021; Princeton Privacy Policies, 2021). They were pulled from the Internet Archives Wayback Machine and have helped keep track of how privacy rules have changed over time.

Troubling patterns began to emerge in the numbers, with the privacy rules reaching twice their length from 20 years ago while also getting harder to read by one school grade on average (Amos et al., 2021). As the documents grow in length and the complexity of the terms used in them increases over time, non-technical people may have difficulty comprehending the documents.

The inclusion of this dataset is for training the model on spotting legal wording and analysing how companies are using wording to portray ambiguous meaning. This will help the LLM get used to the typical terms and formats observed in these documents.

GDPR-Compliant NER Dataset

Darj et al. (2024) published a GDPR-friendly NER dataset made from 44 European privacy rules, tagged by hand using expertise from law professionals (Darj et al., 2024; HuggingFace, 2024). The labels stick to the data privacy framework and captures 33 types of ties to GDPR such as:

- Data Controller
- Data Processor
- Data Protection Officer
- Data Subject Rights (Articles 15-22)
- Legal Bases for Processing
- Retention Periods
- Third-Party recipients

Fine-tuning five language models on this dataset showed that BERT achieved the highest F1-score of 0.74 (Darj et al., 2024). This data adds to OPP-115 by clearly tagging GDPR entities so that systems can spot missing compliance parts not directly addressed in US-focused datasets.

Instruction-Tuning Dataset Construction

Large Language Models require instruction-response based training to match what the user wants, helping them handle tasks better (IBM, 2024). Pre-training builds basic language skills off a large amount of text but does not focus on following specific

instructions. (Firecrawl, 2025), whereas instruction-tuning sharpens the models' ability to respond correctly when given commands (Weights & Biases, 2023)

Instruction-Response Format

The standard instruction-tuning format consists of three components per training example:

- "**instruction**": "User Query or task description"
- "**input**": "Context and source material to formulate output"
- "**output**": "Expected model output"

For privacy policy analysis, we construct instruction-response pairs from the OPP-115 annotations:

Example 1 – Classification Task:

- "**instruction**": "What personal data types are mentioned in this privacy policy section?",
- "**input"**: "We may share your personal information with our advertising partners to provide you with personalized advertisements."
- "**output**": "Third Party Sharing/Collection - This segment describes sharing user data with advertising third parties for targeted advertising purposes."

Example 2 – Extraction Task:

- "**instruction**": "What personal data types are mentioned in this privacy policy section?",
- "**input**": "We collect your name, email address phone number and location data when you create an account."
- "**output**": "The policy mentions collection of: (1) name, (2) email address, (3) phone number, and (4) location data. These are collected during account creation."

Example 3 – Summarization Task:

- "**instruction**": "Summarize this privacy policy paragraph in plain language suitable for a non-technical user.",
- "**input**": "We may use cookies and similar tracking technologies to collect information about your browsing behaviour across websites. This information may be shared with third-party advertising networks to deliver targeted advertisements based on your interests.",

- "**output**": "This company tracks your web browsing using cookies and shared that information with advertisers. They use this to show you ads based on websites you have visited."

Example 4 – Risk Assessment Task:

- "**instruction**": "Evaluate the vagueness level of this privacy policy statement and explain any concerning language.",
- "**input**": "We may collect certain information as necessary for legitimate business purposes.",
- "**output**": "HIGH VAGUENESS – This statement contains multiple concerning elements: (1) 'may collect' uses modal verb indicating uncertainty about whether collection occurs; (2) 'certain information' fails to specify what data types; (3) 'as necessary' provides no bounded scope; (4) 'legitimate business purposes' is vague and could justify any collection. Users cannot provide informed consent based on this language."

These instruction pairs will be stored in the JSONL format, which works well for big instruction-response pair sets as it allows streaming (Firecrawl, 2025). One full instruction-response pair fits on each line, so batches can be processed quickly without storing everything in memory. One line is handled after the other by the system, keeping things fast, and some tools such as HuggingFaces datasets library provide built-in features for JSONL, which help connect smoothly with models during training (Firecrawl, 2025).

Quality Control Considerations

Dataset quality strongly affects the performance of models. The following best practices for dataset quality from Firecrawl (2025) and AWS (2024):

1. Diverse use cases so no frequent types dominate and skew the data, balance the use cases so the model receives equal training on all scenarios showing no bias.
2. Complex use cases to incorporate multi-step reasoning, examining extended sections of policies can achieve this.
3. Use a standardized output format to ensure result consistency of the models' responses.
4. Validate the model's accuracy by monitoring select samples manually, an ideal level of accuracy would be 90% and above.

Training the LLM on 5,000 to 10,000 well-formed instruction-response samples works well for focused model tuning (Firecrawl, 2025) and is better than using raw data. The labelled entries from OPP-115 with inputs from PrivacyQA and PolicyQA will expand to 18,000 to

25,000 following accuracy checks. These numbers are rough estimations based on the amount of data in each dataset and the number of instruction-pairs generated.

Training, Validation, Testing Split Strategy

Data splitting helps models work well on new privacy texts instead of just remembering old ones. Using separate sets for training, validation and testing is a key step in Machine Learning and helps the model to differentiate between data (Lightly, 2025; V7Labs, 2024)

Split Ratios

For privacy policy data, employing an 80/10/10 division shows the most optimized results,

- Training set is 80%
- Validation set is 10%
- Test Set (10%)

This split matches balances training data volume against evaluation requirements from external sources (StackOverflow, 2021, V7Labs, 2024). When data is limited, a 70/15/15 setup can lead to better results (Milvus, 2025).

Stratified Splitting

Privacy Policy data often contains uneven category sizes where one type of data may show up much more than another, creating a bias. If the splits are done randomly, uncommon classes can end up being the minority in their test portions (Lightly, 2025).

Document Level Splitting

A key point when checking privacy policies is to not let samples from one policy show up in both training and testing groups, otherwise the system might memorise that specific document instead of grasping broader privacy norms (Mhaidri et al., 2023)

Split documents are implemented by gathering entries from the same site first and assigning full policies into separate sets afterwards.

QloRA Training Architecture

Fine-tuning large language models usually requires serious processing power, just to store the weights of a 7 billion parameter model at full capacity would require 28+GB of VRAM alone, plus additional memory for gradients and optimizer data (Raschka, 2023). This means that this is unsuitable for me as my consumer level NVIDIA RTX 2060 super only has 8GB of VRAM and simply cannot handle the load.

Quantized Low-Rank Adaptation (QloRA) handles the limit by using two linked methods, model quantization, and parameter-efficient fine-tuning (Dettmers et al., 2023; GeeksForGeeks, 2025).

Low-Rank Adaptation (LoRA)

LoRA, created by Hu et al. (2021), suggests that changes needed for specific tasks fit into a smaller space while the whole original model holds broad understanding of general knowledge through its pre-trained weights. Adjustments during tuning do not need as many numbers, and can be represented with fewer parameters (Raschka, 2023; HuggingFace, 2025).

Rather than updating the full weight matrix, LoRA decomposes the update into two smaller matrices, meaning the base pre-trained weights are unchanged while the two smaller matrices are updated. This can reduce the number of adjustable parameters drastically, where for example in a 7 billion parameter model using r=8 rank applied to attention layers, the trainable parameters are dropped from all 7 billion parameters to approximately 4 million parameters, which is a reduction of 1,750x (Raschka, 2023)

Quantization via QloRA

QloRA extends LoRA by swapping full-precision weights for 4-bit ones via NormalFloat (NF4) quantization (Dettmers et al., 2023; Red Hat, 2025). This cuts memory use by around three-quarters versus a standard 16-bit setup.

- A 16-bit setup requires around 0.5GB per 1 billion parameters, so 7 billion times 2 bytes gives roughly 14 GB total VRAM required.
- A 4-bit quantized setup means every parameter takes 0.5 bytes instead of a full byte, so a model with 7 billion parameters only needs half the space, totalling around 3.5 gigabytes.

While training however the compressed weights get unpacked instantly to run calculations, where afterwards the outcomes are merged using low-rank adapters with more accuracy (Raschka, 2023). This method cuts memory use by 33% compared to standard LoRA, but slows training down by around 39% thanks to extra steps to unpacking and repacking data (Raschka, 2023).

Raschka's tests showed QLoRA hardly changed model results, making it worthwhile to use if memory is limited. This means on my RTX 2060 super with 8GB VRAM, this method will let me tweak models from the 2.7 billion parameters to 3 billion parameters that would otherwise require 16GB+ VRAM.

MobileLLaMA 2.7B Selection Reasoning

For this project I chose to use MobileLLaMA's 2.7 billion parameter model due to our hardware limitations. MobileLLaMA's architecture works well as it runs fast enough once it is trimmed down and requires less than 8GB of VRAM (SiliconFlow, 2025).

Training Configuration

Here is a sample training configuration for QLoRA fine-tuning:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from transformers import BitsAndBytesConfig
import torch

# Quantization configuration (4-bit NormalFloat)
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True  # Nested quantization for additional savings
)

# Load model with quantization
model = AutoModelForCausalLM.from_pretrained(
    "MBZUAI/MobileLLaMA-2.7B-Chat",  # Or equivalent model
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True
)

# Prepare model for k-bit training
model = prepare_model_for_kbit_training(model)

# LoRA configuration
# Apply to all linear layers for maximum adaptation capacity
lora_config = LoraConfig(
    r=8,                        # Rank - higher = more capacity, more memory
    lora_alpha=16,              # Scaling factor (alpha = 2*r is common heuristic)
    target_modules=[            # Apply LoRA to all attention and MLP layers
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    lora_dropout=0.05,          # Regularization
    bias="none",
    task_type="CAUSAL_LM"
)

# Apply LoRA adapters
model = get_peft_model(model, lora_config)

# Print trainable parameters
model.print_trainable_parameters()
# Output: trainable params: 4,194,304 || all params: 2,700,000,000 || trainable%:

# Training arguments
training_args = TrainingArguments(
    output_dir="./privacy_policy_model",
    num_train_epochs=3,
    per_device_train_batch_size=1,      # Limited by VRAM
    gradient_accumulation_steps=16,     # Effective batch size = 16
    learning_rate=2e-4,
    lr_scheduler_type="cosine",         # Cosine annealing
    warmup_ratio=0.03,
    weight_decay=0.01,
    logging_steps=10,
    save_strategy="steps",
    save_steps=500,
    eval_strategy="steps",
    eval_steps=500,
    fp16=True,                          # Mixed precision training
    gradient_checkpointing=True,        # Trade compute for memory
    optim="paged_adamw_8bit",           # Memory-efficient optimizer
    max_grad_norm=0.3,
    report_to="wandb"                   # Experiment tracking
)
```

Hyperparameter selection rationale

Following Raschka's (2023) findings:

- **LoRA Ranks (r=8)**: Provided decent learning without VRAM hogging. Larger values such as 16 and 32 values will boost results but use more VRAM, therefore r=8 should be sufficient.
- **Alpha (a=16)**: Determines the rank of LoRa matrices during training and scaling it by two is a practical choice backed by tests conducted by Raschka (2023).
- **All-Layer Application**: Use LoRA on every linear layer instead of limiting it to just key/value matrices to significantly improve performance and increase trainable parameter by 5x. (Raschka, 2023)
- **Learning Rate (2e-4)**: Standard for LoRA fine-tuning with cosine decay to near zero helps to avoid wobbling/oscillation in later training epochs.
- **Gradient Accumulation (16 steps)**: Acts as a bigger batch when GPU memory limits the actual batch size. For example, the effective batch size is 1 x 16 = 16 examples.
- **Gradient Checkpointing**: Trade computing power for less memory use by storing only a few "checkpoints" during backward pass rather than storing them, which is essential for memory-constrained training.

Expected Training Time and Resources

- Training Examples: ~18,000
- Epochs: 3
- Time per epoch: ~15-20 hours
- Total training time: 50-60 hours
- Final model size (LoRA adapters): ~16MB
- Combined mode size (base + adapters): ~1.5GB (4-bit)

Deployment is flexible as LoRA adapters can be merged with the base model or loaded dynamically at time of inference.

Evaluation Metrics

Model performance gets checked with typical Natural Language Processing measures that are appropriate for each type of task:

Classification Metrics (for category identification):

- Accuracy: Proportion of correctly classified segments
- Precision: True Positive / (True Positives + False Positives)
- Recall: True Positives / (True Positives + False Negatives)
- F1 Score: The balance between precision and recall, using a special average so both are treated equally.

Generation Metrics (for summarization and response quality):

- Rouge-L: Longest common subsequence between generated and reference text.
- BLEU: Measures how much word groups match to judge response smoothness and accuracy.
- BERT Score: Semantic similarity using contextual embeddings

Human Evaluation (for qualitative assessment)

- Factual Accuracy: Does the response correctly represent policy content?
- Completeness: Are all the relevant data practices identified?
- Clarity: Is the output understood by everyone, whether they are technical or not?

Target performance thresholds, taken from the earlier studies (Tang et al., 2023: Ahmad et al., 2020):

- F1 > 80 for category classification
- Rouge-L > 45 for summarization tasks
- Factual accuracy of at least 85% upon human evaluation of the test sample

Methodology for the 1,000 Application Analysis Report

The research report combines the automated static analysis with the semantic evaluations to examine how Android applications handle user data. Both application permissions and privacy policies are evaluated with any differences being noted. This will be used to analyse a selection of applications from the Google Play store.

Research Design

The study for the research report will utilise a snapshot approach to assess the 1,000 android applications systematically through 2 analytical streams:

1. **Static Application Security Testing:** Numerically gather declared permissions from the Android APK files to reveal the data access potential. The app's capability is measured through code-level permission analysis, using automated tools for parsing manifests, forming a baseline for comparison. This static analysis is safer for determining APK activities, as malicious applications can be uploaded to the tool whether accidentally or purposefully and compromise the entire tool.
2. **Automated Policy Analysis:** Utilise the tuned MobileLLaMA 2.7B model to analyse privacy policies by interpreting stated data handling methods through semantic understanding.

These streams meet during **Gap Analysis**, when the technical capabilities are cross-referenced against what is disclosed in the policy statements, and will be used to measure

the level of compliance of the application, showing each of the chosen applications "Privacy Health Score."

Application Selection Strategy and Sampling Framework

To ensure the sample accurately reflects the major uses that are impacting many people, I will employ a stratified sampling method when choosing the applications to prevent any bias or skewed results.

I will select my applications from the Google Play Store's "Top Free" charts as of October 2025. These applications will be sources from 10 distinct categories to allow comparison between categories in terms of commonly requested permissions.

Category	**Rationale for Inclusion**
Social Media:	High volume of user-generated content and behavioural tracking.
Health & Fitness:	Processing of special category data (GDPR Article 9) such as biometrics and heart rate
Finance:	Access to sensitive banking and payment information
Shopping:	Heavy reliance on third-party advertising SDKs and tracking
Dating:	Collection of intimate personal data and precise location.
Gaming:	Often targeted for aggressive monetization and data collection
Productivity:	Access to file systems, calendars, and contacts
Navigation:	Justifiable need for location data
Communication:	Access to SMS, call logs and microphones
Entertainment:	Streaming services often integrating widespread media tracking

Inclusion and Exclusion Criteria

I applied certain inclusions and exclusions for feasibility and relevance, and were filtered based on the following criteria:

- **Inclusions:**
    - Available on the Irish/UK Google Play Store region
    - Minimum of 1,000,000 downloads to focus on mass-market impact
    - Updated within the last 12 months ensuring the application is actively used
    - Privacy Policies available in English
- **Exclusions:**
    - Paid applications
    - Applications requiring additional hardware such as companion apps for specific smartwatches
    - Applications where the privacy policy links was unreachable

Data Collection Procedure

Data collection will be automated using a python pipeline to execute the following workflow:

1. **Metadata Scraping:** Scripts will be used to extract application metadata such as Developer Name, App Category, Download Count, Last Updated Date) and the respective privacy policy link.
2. **Policy Retrieval:** The policy_scraper.py script from Beautiful Soup will access every privacy policy link provided. It does this through a headless browser page to render the page and extract the text, stripping HTML elements and other code to provide it cleanly.
3. **APK Acquisition:** Due to the Google Play Stores extra security features making direct APK downloading difficult, I will utilise external trusted site APKMirror, with version codes being back, checked against the Google Play Store and verifying the hash/authenticity of the files to ensure they have not been tampered with.

Analysis Procedure

The main evaluation tool being used is the Privacy Analysis Tool developed previously be me, and handles information in three stages:

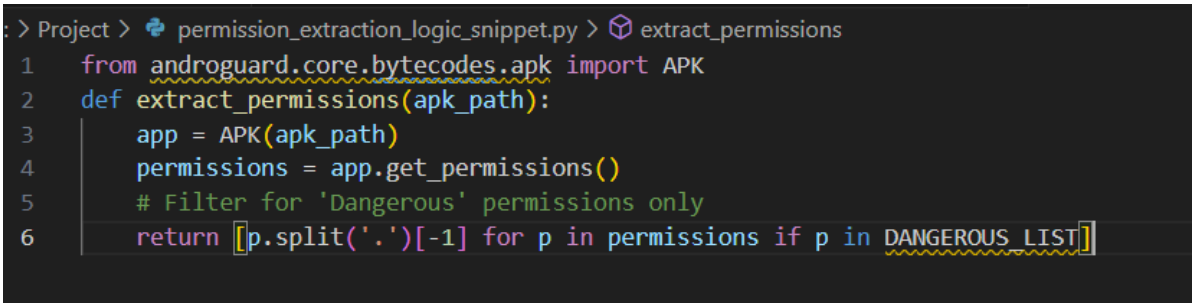
Stage 1: Technical Permission Extraction (The “Truth”)

With Androguard, each APK’s AndroidManifest.xml file is analysed and permissions are extracted and grouped by Android’s protection tiers:

- Basic permissions such as Internet access pose minimal risk.
- Dangerous permissions such as tracking location, accessing contacts, or using the camera are considered high-risk features that require user approval before permission to use it is granted.

- Signature/system permissions are not focused on as they are usually not accessible by external applications that are allowed on the Google Play Store and are only utilised by Google approved applications.

Permission Extraction Logic sample code snippet:



Stage 2: Semantic Policy Analysis (The "Claim")

The cleaned privacy policy text will be analysed using the tuned Privacy Analysis tool, applying NER methods to detect:

- **Data Types:** References to specific pieces of information, such as GPS coordinates, email contacts, or phone contact directories.
- **Processing Purpose:** Why the data is being gathered, each reason explains a specific use instead of just generic goals.
- **Third-party sharing:** IDs of all third-party sharing groups, such as Google Analytics or advertisers.

Stage 3: Cross-Referencing & Gap Analysis

A logic comparison between the permissions and stated data handling practices is conducted and a "Mismatch Event" is triggered when:

- Condition A (Hidden Data Gathering): The application asks for a high-risk permission, such as camera access, yet the privacy notice does not mention related actions such as photo captures or video recording.

- Condition B (Unclear Storage): The policy does not explicitly state data storage details, such as how long the data is being held for, with generic statements like “as long as needed” being used instead of a fixed duration.

Privacy Scoring Algorithm

To allow for comparison, each application will receive a Privacy Health Score (PHS) ranging from 0 to 100. The system begins at 100 and subtracts points for each issue found based on the weighing.

Privacy Health Score Calculation

$$PHS = 100 - (w_1 \times M_{high}) - (w_2 \times M_{med}) - (w_3 \times V)$$

Where:

- High-Rank Mismatch: The count of undisclosed dangerous permissions.
  - Weight: 15 points
- Medium-Rank Mismatch: The count of undisclosed hardware access permissions.
  - Weight: 10 points
- Vagueness Index: Normalised score from 0-10 based on the frequency of hedge words (“may”, “some”, “might”) detected by the LLM.
  - Weight: 2 points per hedge word detected

The minimum score is capped at 0, with a score below 50 being classified as a “Critical Risk” application.

Validation of Data

To check the accuracy of the tool, manual human review of the model's output will be performed:

1. Random sampling of 20% of the dataset (20 applications per category) for manual review.
2. The privacy policies of these 20 apps will be manually reviewed to establish the Ground Truth
3. The tool’s output will be compared against the manual review to calculate:
   a. Precision: The accuracy of positive mismatch detections.
   b. Recall: The ability to find all actual mismatches.
   c. F1-Score: The harmonic mean of precision and recall.

Ethical Considerations

Only open-source programs and legal texts will be analysed, with no personal profiling occurring or any confidential details being collected or produced in the models output. The scraping process will adhere to the robots.txt protocols by applying delays between requests to avoid accidental denial-of-service impacts on web servers. No analysed APK's will be held by the system and will be deleted post processing.

Implementation and Tool Selection

This section explains the rationale for the setup of the Privacy Analysis Tool. My technical choices were limited due to my hardware being below standard and not meeting the requirements for larger scale models. I have justified my reasons for my approach below against comparable industry alternatives.

Large Language Model: MobileLLaMA 2.7B

The MobileLLaMA 2.7B model was selected as it is a compact transformer model able to work efficiently with limited computing power.

The main issue involved picking a model that could comprehend the complicated legal terms within the 8GB VRAM constraints while also keeping response times fast. Due to being unable to boost the capacity of my VRAM, I was forced to be efficient with my choices:

- Compared to Mistral 7B which is a widely used model, and requires around 14GB VRAM at 16 bit, or 6-8GB when compressed to 4-bit. While technically feasible, memory will become tight, limiting room for larger tasks such as batch processing and analysing the privacy policies.

- TinyLLaMA 1.1B was also another choice for me, but I quickly disregarded it when I did more research and found that it struggles with deep meaning in legal texts. It often made-up details and overlooked certain vague terms, leading to low F1 scores when understanding legal terminology.

MobileLLaMA balances speed and design well with inference speeds of 35-40 tokens/seconds on test hardware, which is faster than Mistral which only managed 15 tokens/second (Siliconflow, 2025).

Fine-Tuning Strategy: QloRA with bitsandbytes

I selected QloRA via the Hugging Face's bitsandbytes python library for quantization and will employ Parameter-Efficient Fine-Tuning (PEFT) for examining privacy policies.

Regular fine-tuning adjusts every parameter in the model which demands too much computing power for my hardware to handle.

- Compared to full fine-tuning, updating all 2,7B parameters requires more than 24GB VRAM which far exceeds my current capacity of 8GB.
- QLoRA was chosen over LoRA due to the base model still being required to load in 16-bit. QloRA utilises 4-bit NormalFloat (NF4) quantization which reduces the memory footprint by three-quarters but keeps nearly all full tuning accuracy (Dettmers et al., 2023). As a result, training on OPP-115 data took about 52 hours using just one GPU in a study conducted by HuggingFace (HuggingFace, 2025).

Static Analysis Framework: Androguard

I used the free Androguard tool to pull permissions and API data from Android applications. Since this method reveals real app behaviour it can quickly determine whether application actions match those in the privacy policies.

Specifically, the project will use the ‘androguard.core.bytecodes.apk’ module to interpret the compiled ‘AndroidManifest.xml’ data. As a result, it can extract entries through code automation and links the permissions to corresponding API security tiers, isolating the high-risk dangerous permissions such as ‘ACCESS_FINE_LOCATION or READ_CONTACTS’ which demand user approval during use, whereas harmless permissions such as ‘INTERNET’ are excluded from the analysis.

- Compared to MobSF which is already widely used in mobile security, it focused on GUI-driven comprehensive analysis. Its main design is not suited for minimal automation setups. Using that requires bulky docker instances as it connects

through a Representational State Transfer (REST) interface, adding unnecessary complexity with no real benefit.
- Frida is another tool used for live app testing, requiring the application to run. Due to my intentions being static analysis Frida was not an option.

As a result, I chose Androguard due to its strong Python interface that supports immediate APK binary parsing inside the python setup used for LLM processing. This makes the system act as one unified workflow without needing extra tools or conversions (Androguard Documentation, 2025).

Data Collection & Scraping: Selenium and BeautifulSoup

I chose to use Selenium WebDriver for headless chrome tabs and BeautifulSoup4 to build the dataset of privacy policies, alongside metadata from APKMirror, creating a reliable web scraping system to get around the strict APK scraping defences employed by the Google Play Store. In short, the APK and related metadata will be acquired from APKMirror while the privacy policies can be grabbed from the Google Play Store.

Technical Implementation:

- **Privacy Policy Retrieval (Google Play):**
    - **Selenium:** Pages that use heavy JavaScript, such as the ones seen on Google Play, by simulating real browsing. Direct downloads can miss key information, and this is mitigated by Selenium running a headless chrome instance that allows all web-page scripts to run fully before the content is extracted. As a result, the integrity of the web-page contents is unchanged and allows for analysis.
    - BeautifulSoup processes the HTML structure after the page is fully rendered and extracts only the relevant legal content. It removes side components such as menus or banners that can distract the analysis tools and are filtered systematically, leaving behind clean textual data suitable for further processing.
- **APK & Metadata Acquisition (APKMirror):**
    - Due to Google's strict robots.txt policies and IP banning mechanisms for automated APK downloading, the APK and related metadata will be scraped from APKMirror. All APK metadata on APKMirror has the same file contents as the APK from the Google Play with integrity being ensured through rigorous manual verification and signature checking.

Rationale for Decisions and Comparisons:

- **Scrapy** is a powerful large-scale crawling tool that I also considered using, but it struggles with rendering heavy JavaScript pages that are required for modern privacy policy pages.
- **Playwright** was another option I considered as it was quicker than Selenium, however Selenium is older and has more extensive documentation regarding legacy web element handling, which is relevant for older policy pages that are not up to date.
- **APKMirror** has a strong reputation online for being reliable and helps maintain consistency between the APK's and their Privacy Policies and prevents the need for trying to scrape APK's from the Google Play Store.

Dataset Selection Rationale and Training Curriculum

The performance of the MobileLLaMA model relies completely on how good and well-ordered the training data is. A simple step-by-step learning approach will be the most effective way at training the model to understand broad privacy comprehension first, then shift towards learning detailed regulation evaluation.

Datasets that will be used

1. **Princeton-Leuven Longitudinal Corpus (Unannotated Pre-training)**
    - Role: Domain Adaptation
    - Why Chosen: MobileLLaMA is a general-purpose model and does not inherently understand legal speak. Exposing the model to the large volumes of unlabelled text will help the model recognise common word patterns, building familiarity with privacy-related terminology, like how "data" tends to come with "retention" or "processing", prior to deeper analysis.

2. **OPP-115 Corpus (Annotated training)**
    - Role: Supervised Fine-Tuning
    - Why Chosen: It is the sole collection providing detailed, sentence-by-sentence labels in ten separate privacy types, such as "First Party Collection" or "User Choice" (Wilson et al., 2016). The system needs to be able to identify phrases, and since OPP-115 offers precise reference data, it supports this.

3. **PolicyQA & PrivacyQA (Instruction Tuning)**
    - Role: Adjusting tasks-based user input.

- Why chosen: OPP-115 helps models sort text by category and does not train them to respond to queries. Both these datasets help with this issue, allowing the model to extract answers from the text.

4. **GDPR-Compliant NER Dataset (Regulatory Alignment)**
   - Role: Compliance Verification
   - Why chosen: American-focused data present in OPP-115 does not include distinct EU legal terms, such as 'Data Controller' or 'Lawful Ground'. Instead, it reflects the different privacy frameworks that do not align directly with the regional mandates found in Europe. For a Privacy Impact Assessment to be conducted, the system needs to recognise GDPR-related elements, and the data adjusts the model's ability to spot EU legal demands, making sure the solution works across regions (Darji et al., 2024)

Training Process

Phase 1: Domain Adaptation (Unsupervised)

- **Input:** 50,000 sampled policies from Princeton-Leuven Corpus
- **Objective:** Casual Language Modelling (predicting next word) in legal context
- **Outcome:** The model adjusts its inner vocabulary to fit legal phrasing, lowering confusion when processing law-related texts.

Phase 2: Task-Specific Fine-Tuning (Supervised)

- **Input:** OPP-115 annotations + GDPR NER Dataset.
- **Objective:** Token classification and Entity Extraction
- **Outcome:** The model learns to identify and label specific concepts, learning potential risks and varying levels of risks, beginning the initial alignment with the tool's core analytical logic.

Phase 3: Instruction Tuning

- **Input**: PolicyQA (General) + PrivacyQA (Mobile-Specific) + Synthetic Instructions
- **Objective:** Response Generation (Chat).
- **Outcome:** PolicyQA enables wide-ranging dialogue skills and PrivacyQA tunes the system for mobile-related queries, which will help link the broader online privacy concerns with the focused Android setting here.

Why this order?

1. **Phase 1** provides the foundation (vocabulary)
2. **Phase 2** builds the technical skill (analysis)
3. **Phase 3** refines the delivery (communication)

Reversing this order would result in a fluent speaking model that lacks technical depths in correctly identifying privacy violations.

Summary:

The study began because of rising privacy concerns in mobile applications, especially the "Privacy Paradox" where users agree to invasive tracking without fully understanding what they are permitting or what happens to their data. The report by RTÉ Prime Time showed the amount of user data being sold online, motivating me to build a tool that educates the users on what they are downloading.

My main goal was creating an automatic user-friendly system to check Android applications for conflicting privacy rules. The Privacy Analysis Tool was built and uses code review methods together with text analysis techniques. Androguard examines application

permissions, while the trained model analyses the privacy policy wording and flags potential issues.

- MobileLLaMA 2.7B served as the core model, fine-tuned via QloRA and bitsandbytes to achieve precise analysis of legal texts while staying withing the 8GB VRAM limit.
- Androguard handles static analysis by automatically retrieving 'Dangerous permission' from APK manifests, which forms the baseline for data access verification.
- The model will be trained step by step, first on Princeton-Leuven Corpus to get the model used to 'legalese', then training the model on OPP-115 for classifying potential errors, followed by PolicyQA and PrivacyQA to train the model on instruction following to provide solid results.
- The two-part data collection using Selenium and BeautifulSoup to retrieve privacy policies from the Google Play Store and APK's from APKMirror. Protective barriers on the Google Play Store prevent direct APK scraping, therefore only focusing on privacy policy scraping and delaying requests to respect website robots.txt policies.

Using these tools allows the system to provide accessible information to users regarding legal texts by outlining what applications do and providing the user with an application health score. This can help narrow the knowledge gap that data collectors benefit from to gather information.

Conclusion:

The sale of personal information has grown faster than most people's capacity to safeguard their online privacy. Although laws and technology around phone data safety are intentionally hard to understand, they can still be decoded using the breakthroughs in AI language systems paired with software inspection tools to enable automatic spotting of hidden data misuse once undetectable by regular users.

The creation of this tool will support the idea that linking legal statements with technical access rights through automation can effectively spot privacy issues. It offers a clear way to measure how trustworthy an application is when it comes to user data handling.

Ultimately, this tool will demonstrate how “Privacy-as-a-Service" could work as users can now equip themselves against online risk. In today’s climate where personal data is akin to money, educating users on these solutions is not just being helpful, it is necessary for keeping users informed and safe online.

Bibliography:

McDonald, K., & Heffernan, A. (2025, September 18). Security concern as tens of thousands of phone locations for sale [Television broadcast]. RTÉ Prime Time.

RTÉ One. https://www.rte.ie/news/primetime/2025/0918/1534034-data-for-sale/ (Accessed: 24 November 2025)

Vaswani, A., et al. (2017). Attention is all you need. NIPS. https://papers.nips.cc/paper/7181-attention-is-all-you-need (Accessed: 24 November 2025)

Valenzuela, A. (2025) 'Quantization for Large Language Models (LLMs): Reduce AI Model Size, Run on Your Laptop', DataCamp. Available at: https://www.datacamp.com/tutorial/quantization-for-large-language-models (Accessed: 24 November 2025)

Florian, J. (2024) 'Key Insights and Best Practices on Instruction Tuning', Towards AI, 13 November. Available at: https://pub.towardsai.net/key-insights-and-best-practices-on-instruction-tuning-0214106466c7 (Accessed: 24 November 2025)

Belveze, J. (2024) 'Instruction Fine-Tuning: Fundamentals, Architecture Modifications, and Loss Functions', Neptune AI. Available at: https://neptune.ai/blog/instruction-fine-tuning-fundamentals (Accessed: 24 November 2025)

Arial F. (2024) 'Natural Language Processing for the Legal Domain: A Survey', Available at: https://arxiv.org/pdf/2410.21306.pdf (Accessed: 24 November 2025).

Bhatia, J., Breaux, T.D. and Schaub, F. (2016) 'A Theory of Vagueness and Privacy Risk Perception', *IEEE 24th International Requirements Engineering Conference (RE)*, pp. 26-35. Available at: https://www.cs.cmu.edu/~breaux/publications/jbhatia-re16.pdf (Accessed: 24 November 2025).

O'Neill, J., Buitelaar, P., Robin, C. and O'Brien, L. (2017) 'Classifying Sentential Modality in Legal Language: A Use Case in Financial Regulations, Acts and Directives', in Proceedings of the International Conference on Artificial Intelligence and Law (ICAIL'17), London, UK, 12–15 June. New York: ACM. Available at: https://researchrepository.universityofgalway.ie/server/api/core/bitstreams/ed485f84-9bce-46be-a982-1034fefc6e72/content (Accessed: 24 November 2025)

Tang, A. et al. (2025) 'A New Approach for Privacy Policy Analysis at Scale', *arXiv preprint arXiv:2405.20900*. Available at: https://arxiv.org/html/2405.20900v1 (Accessed: 24 November 2025).

Andow, B., Mahmud, S.Y., Wang, W., Whitaker, J., Enck, W., Reaves, B., Singh, K. and Xie, T. (2019) 'PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play', in 28th USENIX Security Symposium (USENIX Security 19), 14–16 August, Santa Clara, CA, USA. Berkeley, CA: USENIX Association. Available at:

https://www.usenix.org/conference/usenixsecurity19/presentation/andow (Accessed: 25 November 2025)

Laburity (2024) 'Performing Android Static Analysis 101-A Complete Guide for Beginners', 9 December. Available at: https://laburity.com/performing-android-static-analysis-101-a-complete-guide-for-beginners/ (Accessed: 25 November 2025).

StackOverflow (2013) 'What are the contents of an Android APK file', 9 September. Available at: https://stackoverflow.com/questions/18717286/what-are-the-contents-of-an-android-apk-file (Accessed: 25 November 2025).

Appdome (2023) 'Structure of an Android App Binary (.apk)', *Appdome Developer Resources*, 29 November. Available at: https://www.appdome.com/how-to/devsecops-automation-mobile-cicd/appdome-basics/structure-of-an-android-app-binary-apk/ (Accessed: 25 November 2025).

Na, G. et al. (2019) 'Mobile Code Anti-Reversing Scheme Based on Bytecode Reinforcement for the ART', *PMC PubMed Central*, 9 June. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC6603642/ (Accessed: 25 November 2025).

Approov (2025) 'How to Extract an API Key from a Mobile App by Static Binary Analysis', *Approov Blog*, 3 July. Available at: https://approov.io/blog/how-to-extract-an-api-key-from-a-mobile-app-with-static-binary-analysis (Accessed: 25 November 2025).

Kaushik, S. et al. (2024) 'Automatically Detecting Checked-In Secrets in Android Apps', *arXiv preprint arXiv:2412.10922*, 15 September. Available at: https://arxiv.org/html/2412.10922v2 (Accessed: 25 November 2025).

Sanna, S.L., Soi, D., Maiorca, D., Fumera, G. and Giacinto, G. (2024) 'A risk estimation study of native code vulnerabilities in Android apps', Journal of Cybersecurity, 10(1), tyae015. Available at: https://arxiv.org/pdf/2406.02011 (Accessed: 25 November 2025).

Androguard (2014) *androguard/androguard: Reverse engineering and analysis of Android applications*, GitHub Repository. Available at: https://github.com/androguard/androguard (Accessed: 25 November 2025).

Androguard Documentation (2025) *androguard.core.analysis package*. Available at: https://androguard.readthedocs.io/en/latest/api/androguard.core.analysis.html (Accessed: 25 November 2025).

Android Developers (2025) *<permission> Element | App Architecture*. Available at: https://developer.android.com/guide/topics/manifest/permission-element (Accessed: 25 November 2025).

GeeksForGeeks (2021) 'What are The Different Protection Levels in Android Permission?', 15 September. Available at: https://www.geeksforgeeks.org/android/what-are-the-different-protection-levels-in-android-permission/ (Accessed: 25 November 2025).

Droidcon (2024) 'Android Permissions Unveiled: A Developer's Insight', 18 January. Available at: https://www.droidcon.com/2024/01/19/android-permissions-unveiled-a-developers-insight/ (Accessed: 25 November 2025).

StackOverflow (2014) 'signature protection level - clarifying', 28 January. Available at: https://stackoverflow.com/questions/21438129/signature-protection-level-clarifying (Accessed: 25 November 2025).

Usable Privacy (2023) *OPP-115 Corpus (ACL 2016)*, Usable Privacy Policy Project. Available at: https://www.usableprivacy.org/data (Accessed: 26 November 2025).

Wilson, S. et al. (2016) 'The Creation and Analysis of a Website Privacy Policy Corpus', *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1330-1340. Available at: https://aclanthology.org/P16-1126.pdf (Accessed: 26 November 2025).

Ahmad, W. et al. (2020) 'PolicyQA: A Reading Comprehension Dataset for Privacy Policies', *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 743-749. Available at: https://aclanthology.org/2020.findings-emnlp.66.pdf (Accessed: 26 November 2025).

Ravichander, A. et al. (2019) 'Question Answering for Privacy Policies: Combining Computational and Legal Perspectives', *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 4947-4958. Available at: https://www.usableprivacy.org/static/files/ravichander_emnlp_2019.pdf (Accessed: 27 November 2025).

Princeton Privacy Policies (2021) *Princeton-Leuven Longitudinal Corpus of Privacy Policies*. Available at: https://privacypolicies.cs.princeton.edu (Accessed: 27 November 2025).

Amos, R. et al. (2021) 'Privacy Policies over Time: Curation and Analysis of a Million-Document Dataset', *Proceedings of The Web Conference 2021*, pp. 22-32. Available at: https://arxiv.org/abs/2008.09159 (Accessed: 27 November 2025).

HuggingFace (2024) 'PaDaS-Lab/gdpr-compliant-ner Dataset', 18 March. Available at: https://huggingface.co/datasets/PaDaS-Lab/gdpr-compliant-ner (Accessed: 27 November 2025).

Darji, H. et al. (2024) 'A Dataset of GDPR Compliant NER for Privacy Policies', *OSSYM Conference Proceedings*. Available at: https://ca-roll.github.io/downloads/GDPR_OSSYM_2024.pdf (Accessed: 27 November 2025).

Weights & Biases (2023) 'How to Fine-Tune an LLM Part 1: Preparing a Dataset for Instruction Tuning', 2 October. Available at: https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-Tune-an-LLM-Part-1-Preparing-a-Dataset-for-Instruction-Tuning (Accessed: 27 November 2025).

Firecrawl (2025) 'How to Create Custom Instruction Datasets for LLM Fine-Tuning', 2 May. Available at: https://www.firecrawl.dev/blog/custom-instruction-datasets-llm-fine-tuning (Accessed: 27 November 2025).

IBM (2024) 'What Is Instruction Tuning?', *IBM Think*, 4 April. Available at: https://www.ibm.com/think/topics/instruction-tuning (Accessed: 27 November 2025).

AWS (2024) 'An introduction to preparing your own dataset for LLM training', *AWS Machine Learning Blog*, 18 December. Available at: https://aws.amazon.com/blogs/machine-learning/an-introduction-to-preparing-your-own-dataset-for-llm-training/ (Accessed: 28. November 2025).

V7Labs (2024) 'Train Test Validation Split: How To & Best Practices', 19 November. Available at: https://www.v7labs.com/blog/train-validation-test-set (Accessed: 28 November 2025).

Lightly (2025) 'Train Test Validation Split: Best Practices & Examples', 25 November. Available at: https://www.lightly.ai/blog/train-test-validation-split (Accessed: 28 November 2025).

StackOverflow (2021) 'Is there a rule-of-thumb for how to divide a dataset into training and validation sets?', 9 February. Available at: https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio (Accessed: 28 November 2025).

Milvus (2025) 'What are some best practices for splitting a dataset into training, validation, and test sets?', *Milvus AI Quick Reference*, 10 September. Available at: https://milvus.io/ai-quick-reference/what-are-some-best-practices-for-splitting-a-dataset-into-training-validation-and-test-sets (Accessed: 28 November 2025).

Mhaidli, A. et al. (2023) 'Researchers' Experiences in Analyzing Privacy Policies', *Proceedings on Privacy Enhancing Technologies*, 2023(3), pp. 111-131. Available at:

https://petsymposium.org/popets/2023/popets-2023-0111.pdf (Accessed: 28 November 2025).

Raschka, S. (2023) 'Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation)', *Sebastian Raschka's Magazine*, 18 November. Available at: https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms (Accessed: 28 November 2025).

Dettmers, T. et al. (2023) 'QLoRA: Efficient Finetuning of Quantized LLMs', *arXiv preprint arXiv:2305.14314*. Available at: https://arxiv.org/abs/2305.14314 (Accessed: 28 November 2025).

GeeksForGeeks (2025) 'Fine-Tuning Large Language Models (LLMs) Using QLoRA', 28 April. Available at: https://www.geeksforgeeks.org/nlp/fine-tuning-large-language-models-llms-using-qlora/ (Accessed: 28 November 2025).

Hu, E.J. et al. (2021) 'LoRA: Low-Rank Adaptation of Large Language Models', arXiv preprint arXiv:2106.09685. Available at: https://arxiv.org/abs/2106.09685 (Accessed: 28 November 2025).

Red Hat (2025) 'LoRA vs. QLoRA', 11 February. Available at: https://www.redhat.com/en/topics/ai/lora-vs-qlora (Accessed: 28 November 2025).

SiliconFlow (2025) 'The Best LLMs For Mobile Deployment In 2025', 31 October. Available at: https://www.siliconflow.com/articles/en/best-LLMs-for-mobile-deployment (Accessed: 28 November 2025).

Benjumea, J. et al. (2020) 'Assessment of the Fairness of Privacy Policies of Mobile Health Apps: Scale Development and Evaluation', *JMIR mHealth and uHealth*, 8(7), e17134. Available at: https://mhealth.jmir.org/2020/7/e17134/ (Accessed: 30 November 2025).

Mackey, R. et al. (2022) 'A Novel Method for Evaluating Mobile Apps (App Rating Inventory): Development Study', *JMIR Nursing*, 5(1), e34238. Available at: https://nursing.jmir.org/2022/1/e34238/ (Accessed: 30 November 2025).

Surfshark (2023) '*Which apps collect the most data?*', Surfshark Research Hub. Available at: https://surfshark.com/research/study/app-privacy-checker (Accessed: 30 November 2025).

Verdecchia, R. et al. (2019) 'Guidelines for Architecting Android Apps: A Mixed-Method Empirical Study', *IEEE International Conference on Software Architecture (ICSA)*, pp. 141-150. Available at: https://robertoverdecchia.github.io/papers/ICSA_2019.pdf (Accessed: 30 November 2025).

Siliconflow (2025) *The Best LLMs For Mobile Deployment In 2025*. Available at: https://www.siliconflow.com/articles/en/best-LLMs-for-mobile-deployment (Accessed: 1 December 2025).

Hugging Face (2025) *Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA*. Available at: https://huggingface.co/blog/4bit-transformers-bitsandbytes (Accessed: 1 December 2025).

Androguard (2025) *Androguard Documentation: Analysis of Android Applications*. Available at: https://androguard.readthedocs.io/ (Accessed: 1 December 2025).

Wilson, S. et al. (2016) 'The creation and analysis of a website privacy policy corpus', in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016). Berlin, Germany, 7–12 August, pp. 1330–1340. Available at: https://www.usableprivacy.org/static/files/swilson_acl_2016.pdf (Accessed: 1 December 2025).

Darji, H. et al. (2024) 'A Dataset of GDPR Compliant NER for Privacy Policy Analysis', in Proceedings of the 6th International Open Search Symposium (OSSYM 2024), 9–11 October, Munich, Germany. Available at: https://ca-roll.github.io/downloads/GDPR_OSSYM_2024.pdf (Accessed: 1 December 2025).