THESIS

HARNESSING LARGE LANGUAGE MODELS FOR PERMISSION FIDELITY ANALYSIS FROM ANDROID APPLICATION DESCRIPTIONS

Submitted by

Yunik Tamrakar

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
Co-Advisor: Ritwik Banerjee

Sudipto Ghosh
Steve Simske

ABSTRACT

HARNESSING LARGE LANGUAGE MODELS FOR PERMISSION FIDELITY ANALYSIS FROM ANDROID APPLICATION DESCRIPTIONS

Android applications are very popular these days and as of mid-2024 there are over 2 million applications in the Google Play Store. With such a large number of applications available for download, the threat of privacy leakage increases considerably, primarily due to the users' limited knowledge in distinguishing the necessary app permissions. This makes accurate and consistent checking of the permissions collected by the applications necessary to ensure the protection of the user's privacy. Studies have indicated that inferring permissions from app descriptions is an effective way to determine whether the collected permissions are necessary or not. Previous research in the permission inference space has explored techniques such as keyword-based matching, Natural Language Processing methods (including part-of-speech tagging and named entity recognition), as well as deep learning based approaches using Recurrent Neural Networks. However, app descriptions are often vague and may omit details to meet sentence length restrictions, resulting in suboptimal performance of these models. This limitation motivated our choice of large language models (LLMs), as their advanced contextual understanding and ability to infer implicit information can directly address the weaknesses observed in previous approaches. In this work, we explore various LLM architectures for the permission inference task and provide a detailed comparison across various models. We evaluate both zero-shot learning and fine-tuning based approaches, demonstrating that fine-tuned models can achieve state-of-the-art performance. Additionally, by employing targeted generative AI based training data augmentation techniques, we show that these fine-tuned models can significantly outperform baseline methods. Furthermore, we illustrate the potential of leveraging paraphrasing to boost fine-tuned performance by over 50 percent, all while using only a very small number of annotated samples—a rarity for LLMs.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents for their continuous support over the years. If it wasn't for their sacrifices, I wouldn't be here in this position. I would also like to thank my advisor Dr. Indrakshi Ray for giving me the opportunity to work on this technically challenging and relevant project and for her keen insight and guidance during my time writing this document. I would also like to thank my co-advisor Dr. Ritwik Banerjee for the sheer expertise and guidance throughout the project. His guidance and technical knowledge in the field of Natural Language Processing helped me learn and implement several strategies which are instrumental to the topics discussed in this document. I am also extremely grateful to my committee members Dr. Sudipto Ghosh and Dr. Steve Simske for agreeing to serve on my committee and providing valuable feedback. I also owe a great deal of gratitude to the Computer Science department for giving me the opportunity to be a part of their prestigious graduate program and for giving me the opportunity to grow as a scholar as well as a person. This project was a collaborative effort between students at Colorado State and Stony Brook University and thus, special thanks goes to my collaborators from both universities.

DEDICATION

I would like to dedicate this thesis to my grandfather. Wish you were here

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Mobile applications collect and use a large volume and variety of data from the users [1] [2]. This can include sensitive information from the users such as their name, location, social security number, device IDs, IP addresses - all information that can be potentially linked to the identity of the users. This data which can be linked to the identity of an individual is known as Personal Identifiable Information (PII). Mobile Health (mHealth) applications can collect an extended set of PIIs such as age, heartbeat, respiration rate, blood glucose levels and so on, depending on the domain of the application.

The general expectation is that the privacy and usage documents such as the app descriptions, privacy policy text for mobile applications discloses all the dangerous[1] app permissions that the app requests from the user. These permissions when granted by the user allow the apps to access the user's PII [3] and thus, this introduces vulnerability to privacy leakage problems, as demonstrated by the Facebook-Cambridge Analytica data scandal in 2018, which affected over 80 million people [4]. Furthermore, a study conducted at Berkeley [5] across hundreds of users has shown that 83 percent of the participants did not pay attention to the permissions prompted during installation and 42 percent of the participants were basically unaware about the existence of these permissions. Similarly, only 3 percent of a total of 308 android users were able to answer the questions pertaining to the scope and implications of permissions. This underscores the importance of devising robust, automated methods for inferring app permissions from an application's privacy and usage documents with the goal of automating or assisting the user during the permission fidelity analysis. [2]

[1] Higher risk permissions enable the app to access sensitive information. Traditionally, these would be granted by users before installation, but with Android 6.0 and later, the users can opt to grant these permissions at runtime.

[2] Permission fidelity can be defined as the consistency between the permissions inferred from app descriptions, privacy policies, etc versus the actual declared permissions and a justification of whether the collected permissions are necessary or not

One source of inferring the permissions necessary for an application would be the app's description available on the publisher's site (Play store for android apps.) These app descriptions are often the first point of contact for users, providing a concise overview of an app's features and purpose, which heavily influences their initial impressions and download decisions. Thus, it is important that the descriptions are comprehensive enough to give the user enough accurate insights about the dangerous permissions the app collects. In order to check for the description-to-permissions-fidelity, the first step would be to ensure that the mechanism for inferring permissions from the description is robust enough. This work provides a comprehensive evaluation of various techniques for this inference problem.

Similarly, privacy policy texts can be another source via which the list of necessary permissions for any given application can be inferred. However, the policy texts tend to be very dense and can be very difficult for a user to manually parse through and extract any useful information out of it. Furthermore, with the introduction of regulations like the General Data Protection Regulation (GDPR) by the European Union, around 72.6 percent of websites, have taken steps to amend their privacy policies in order to comply with the new standards [6]. The introduction of these regulations has had an interesting impact on the length of the privacy policy texts. It has been reported that the average textual length of the policy texts has gone up by 35.39 percent in countries within the European Union. This trend is observable in the global scene as well where average textual lengths of the policies have been reported to show an increase by about 25.21 percent. [7] These changes have further complicated the reading and comprehension of the already convoluted verbiage of the privacy policy texts. [8] [9] This leads users to clicking on the Agree button without reading through the policies, and thus, this can result in privacy leakage and other issues. [10] Similarly, the new changes can also lead the application developers to resort to privacy policy reuse or delegating to third party services the task of generating privacy policies in order to save time.

1.1 Research Questions

Traditionally, natural language processing (NLP) algorithms have been the standard for solving the linguistic challenge of inferring information from text. But, with the recent boom of large language models (LLMs), which have been trained on tremendously vast corpus of human generated data across the Internet, new possibilities have been unlocked for information extraction (IE) from the semi-structured texts such as the app descriptions and privacy policies.

This work leverages LLMs with various architectures for the problem at hand and seeks to answer the following research questions:

- **RQ1:** How do fine-tuned LLMs compare against the state-of-the-art when it comes to the app-description permission inference performance?
- **RQ2:** How well does zero-shot inference using Generative AI perform on the app description - permission inference task? What is the effect of various LLM parameters on their performance?
- **RQ3:** How well do LLMs fine-tuned on one permissions dataset generalize across other datasets?

During our work, we have also compiled an alternative dataset of 250 app descriptions to permissions and we explore the following research question around this limited dataset:

- **RQ4:** Can we train the LLMs to learn from a limited number of annotated samples? Can data augmentation be leveraged to generate enough high quality samples for the LLM to learn from?

1.2 Thesis Organization and Contributions

In the next chapter, we look at the relevant research that has been conducted in the permission inference / fidelity space and identify existing problems for which our LLM based approach would be a suitable fit. After establishing our motivation, in chapter 3, we provide a comprehensive performance overview of various LLM architectures namely BART, GPT and BERT on the

benchmark dataset. Our results show that BART and various BERT models can achieve (near or at) state-of-the-art performance after fine-tuning. In Chapter 4, we present our study on achieving competitive fine-tuning performance with LLMs while annotating a very limited number of samples. Subsequently, in Chapter 5, we investigate the metrics reported by the models trained on the original AC-Net in greater detail and explore data augmentation techniques to boost the performance of the models beyond the state-of-the-art by a significant margin. This is followed by a generalizability study of our trained models, with the view of assessing their real world utility to an extent. In this chapter, we annotate an alternative dataset to evaluate our fine-tuned models, and our findings support the broad applicability of our approach, where we also show that models trained on balanced data tend to generalize better. Ultimately, we conclude with a brief discussion about some possible limitations of our study and propose some possible improvements as future work.

The first volume of the appendices contains a description of another related study we conducted where we analyze the privacy policy texts of over 10,000 mobile health (mHealth) applications and perform a similarity analysis across this corpus of privacy policies. Our results show that many applications share the same privacy policy, particularly when the applications have been built by the same developer(s). Not only that, we surprisingly discover that even applications built by different developers tend to share the same privacy policies as well due to use of third party privacy policy template generators. The second volume of the appendices contains detailed reporting of all the metrics gathered during our experiments. We also provide links to the fine-tuned models presented in the document and provide initialization details on the random number generator used for the experiments in order to generate our train, validation and test split. This data has been reported to enhance the reproducibility of our work.

Chapter 2

Literature Review

Historically, Natural Language Processing (NLP) techniques have been the obvious choice to solve text classification or information extraction problems. Traditional NLP attacks the text classification problem by dividing it into the feature engineering and model training subtasks. To alleviate the costs associated with manual feature engineering, several deep neural architectures have been proposed over the years. This includes models such as Fast Text Classifier (fastText), textCNN (based on Convolutional Neural Networks), textRNN (based on Recurrent Neural Networks). These models take word embeddings as inputs and return the probability distribution across the classification labels as outputs.

The research problem of bridging the gap between app descriptions and app permissions has been explored in several previous studies. The first published work in this area is WHYPER [11], which uses the permission-semantic model to determine why an app requires a permission. This work aims at bridging the gap between the permissions a user expects the app to collect versus the permissions the app actually collects. The WHYPER model is evaluated on three popularly-used permissions (address book, calendar, and record audio) and a dataset of 581 popular applications (with around 10k natural language sentences). It has been shown that the basic approach of annotating sentences describing sensitive operations pertaining to permissions using keywords is prone to false-positives. As reported by the authors, for instance if we consider the sentences '... displays user contacts, ...' vs '... contact me at abc@xyz.com'. both sentences contain the keyword 'contacts', but only the former refers to a privacy sensitive use case. Similarly, sentences may describe the sensitive operation *reading contacts* without making use of the keyword "contacts". As pointed out by the authors, for the example - "share... with your friends via email, sms..." - this sentence fragment describes the need for reading contacts, but does not explicitly make use of the keyword "contacts" in doing so. To overcome these shortcomings with the keywords based approach, WHYPER makes use of NLP techniques such as Part of Speech (POS) tagging, phrase

and clause parsing, typed dependencies and Named Entity Recognition (NER). They make use of First Order Logic (FOL) representations of an app description sentence and semantic graphs of Android permissions to identify whether a permission is necessary for an app. Whyper identifies app description sentences that describe the need for permissions with an average precision of 82.8 percent and an average recall of 81.5 percent. However, since Whyper is heavily reliant on the underlying Stanford parser [12], it tends to perform poorly when the complexity of the app description sentences increase. Whyper also operates on only three permission categories and does not work with permission categories including Location and Phone Identifiers, which have been shown by studies [13] [14] to be vulnerable to privacy leakages. It is also limited by its semantic knowledge base which is inferred by extracting natural language keywords and their synonyms from the Android API documents. WHYPER's methods would also run into problems with certain permissions which do not have an associated API (e.g: RECEIVE BOOT COMPLETED permission). Also, not all textual patterns associated with permissions can be extracted from the API docs. For example: text such as *scan barcode* refers to the use of the camera permission, but this information cannot be derived from the API documents.

To address the shortcomings of the previous approach, another method called Autocog [15] was introduced in 2014. This model also makes use of the Stanford NLP Parser to identify grammatical structure of the sentences and output typed dependencies (or semantic heierarchies of sentences, i.e. how different parts of sentences depend on each other). Autocog also uses the Explicit Semantic Analysis (ESA) algorithm to compute the semantic relatedness of the texts. This approach uses big document corpuses such as Wikipedia as its knowledge base and has a much wider range of semantic information compared to the API documents used in WHYPER. The authors compute the semantic relatedness between two inputs by using the cosine distances between the vector representations of the input. To enhance the accuracy of Autocog, the authors design a learning-based algorithm by analyzing the descriptions and permissions of a large dataset of applications to measure how closely a noun-phrase based governor-dependent pair is related to a permission. Autocog

achieves significant improvements over WHYPER but is ultimately, limited by the capabilities of the Stanford parser it leverages, which has been known to perform poorly for complex sentences.

A more recent work in this area [16] was the first to leverage deep learning to predict permissions from app descriptions to achieve state-of-the-art performance in the task. This study introduces a novel learning mechanism called TextGRU which is a variant of the Recurrent Neural Network (RNN) architecture [17] with the Gated Recurrent Unit (GRU) [18]. The model takes in pre-processed app description sentences as inputs and outputs the probability that the app description belongs to a given permission category. The evaluation results show that AC-Net significantly outperforms Whyper, Autocog and other keyword-based methodologies, therefore, establishing itself as the state-of-the-art for the problem space. This paper also contributes the publicly available AC-Net dataset of over 25k app description sentences across all 11 permission categories.

Permission Group	ROC-AUC					PR-AUC				
	AC-Net	Key-Based	WHYPER	AutoCog	ACODE	AC-Net	Key-Based	WHYPER	AutoCog	ACODE
STORAGE	**0.94276**	0.66981	0.55930	0.63028	-	**0.65568**	0.40069	0.32521	0.34742	-
CONTACTS	**0.97197**	0.72572	0.60710	0.68446	0.76576	**0.74657**	0.41641	0.28047	0.41521	0.43078
LOCATION	**0.98376**	0.88762	0.60198	0.81737	0.88072	**0.77496**	0.68715	0.45168	0.51815	0.69954
CAMERA	**0.98249**	0.85640	0.57520	0.84025	0.78580	**0.75821**	0.52267	0.43630	0.49002	0.46576
MICROPHONE	**0.96293**	0.80436	0.59735	0.79669	0.79002	0.49665	0.44078	0.24771	0.40081	**0.51334**
SMS	**0.98991**	0.91227	0.65296	0.78944	0.82176	**0.83454**	0.61593	0.36846	0.40644	0.58006
CALL_LOG	**0.99332**	0.63199	0.53645	-	-	**0.70502**	0.20625	0.41683	-	-
PHONE	**0.99140**	0.93148	0.74290	-	-	**0.62455**	0.55226	0.37226	-	-
CALENDAR	**0.99469**	0.98121	0.65954	0.89737	0.95011	**0.84444**	0.71125	0.28444	0.50611	0.71236
SETTINGS	**0.95104**	0.55398	0.58193	0.65403	0.57119	**0.43251**	0.15872	0.21638	0.31488	0.18696
TASKS	**0.94972**	0.64903	0.53768	-	0.66514	**0.48659**	0.27280	0.17903	-	0.24657
AVERAGE	**0.97400**	0.78217	0.60476	0.76374	0.77881	**0.66907**	0.45317	0.32534	0.42488	0.47942

Figure 2.1: Performance Comparison of various methods on the AC-Net dataset

However, it is important to note that AC-Net still has sub-par scores pertaining to the Precision-Recall-Area Under the Curve. This suggests imbalance between the precision and recall metrics indicating either a high number of false positives or false negatives. This is not surprising given the vagueness and complexity of many application descriptions. This is where our study seeks to incorporate LLMs into the problem space. LLMs have been trained on a vast corpus of text and make great use of the attention mechanism which has enabled them to exhibit advanced contextual understanding and the ability to infer implicit information from complex or vague text.

Chapter 3

Permission Inference from Application Description

App descriptions contain information about various aspects of an application such as its general problem domain, features, subscription model and so on. From a privacy compliance and security standpoint, a regular user can use the app description to infer the potential permissions being requested. However, it has been shown that most users do not possess the discretion or professional knowledge needed to derive useful security-related information from these descriptions. This is where the outstanding semantic information extraction capabilities of LLMs can be leveraged to assist the users.

3.1 Problem Characterization

We characterize the permission inference problem as a multi-label binary classification problem. The input to the LLM during fine-tuning are sentences from the app-descriptions compiled in the AC-Net dataset, which to our knowledge, is the most comprehensive and widely used dataset for the permission inference task [16]. For each sentence, the LLM is then tasked with assigning a 0 (false) or a 1 (true) value for each of the permission categories labeled in the dataset. There are a total of 11 permission labels, which have been listed in Table 3.1. The permission labels correspond to the permission groups depicted in the table and each sentence can be assigned to multiple permission categories. The 11 permission groups consist of 16 permission categories in total which have been grouped according to Google's official grouping strategy for dangerous permissions. These permission categories have been derived from the AC-Net paper which lists the top 13 out of 24 dangerous dangerous permissions with the highest user-concerns [19]. Besides the dangerous permissions, the signature permission - WRITE SETTINGS has also been included in the list as it allows the app to read or write system settings. Also, following previous studies [20] [15], normal permissions like GET_TASKS and KILL_BACKGROUND_PROCESS have

also been included as these permissions cannot be revoked after installation, potentially conflicting with user preferences.

Table 3.1: Permission Groups and their corresponding Permissions.

Permission Groups	**Permission**
STORAGE	WRITE_EXTERNAL_STORAGE
	GET_ACCOUNTS
CONTACTS	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_LOGS
PHONE	CALL_PHONE
CALENDAR	READ_CALENDAR
SETTINGS	WRITE_SETTINGS
TASKS	GET_TASKS
	KILL_BACKGROUND_PROCESS

3.2 AC-Net Dataset

We use the AC-Net dataset in order to fine tune and evaluate our models. The dataset consists of 24726 sentences from 1415 applications. Out of these, 4984 sentences have been manually labeled as sentences containing at least one permission (referred to as **permission sentence** by the

authors). This is around 20.2 percent of descriptions compiled in the dataset. For the annotations, 14 participants (CS students) were employed. For each sentence, at least two students participated in the annotations and more students were added to cases where there were conflicts between the original annotators. The authors do not report any sort of inter-annotator agreement in their published study.

3.3 Metrics

In order to assess the performance of various LLMs on the permission inference task, we make use of the following metrics:

Macro-F1

The F1 score is a popular metric to calculate the performance of a classification model. The F1 score is defined as:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Where the Precision metric is the ratio of the true positive instances to the total predicted positive instances and the Recall metric is the ratio of correctly predicted positive instances to the total actual positive instances. The micro f1 score metric works well for the binary classification problem, but with multi-class classification problems, we will need to compute the f1 score for each class. After calculating f1 scores for individual classes, we will need a method to average them. Macro f1 is an unweighted method of performing this average which is given by:

$$\text{Macro-}F_1 = \frac{1}{N} \sum_{i=1}^{N} \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Where:

N is the number of classes,

Precision_i and Recall_i are the precision and recall for class i.

This method treats all classes equally regardless of their support values. One downside with this approach is that it is not ideal for imbalanced datasets as if the classification model has sub-par performance for a class with low support, then the overall f1 score will suffer significantly despite it performing well for classes with higher support.

Weighted-F1

The datasets we use for evaluation are significantly imbalanced, so to account for the imbalance, we also record the weighted F1 metric where the impact of any individual class on the final F1 score will be proportional to its support, that is to say, if a model does well on a class with higher support, then it will witness a significant positive boost on its overall F1. On the other hand, if it does poorly on a class with low support, then the overall F1 will not be penalized as much. The formula for weighted-f1 is given by:

$$\text{Weighted-}F_1 = \sum_{i=1}^{N} \frac{n_i}{N_{\text{total}}} \cdot \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Where:

N is the number of classes,

n_i is the number of instances in class i,

N_{total} is the total number of instances across all classes,

Precision_i and Recall_i are the precision and recall for class i.

AUC-ROC

AUC-ROC score (Area Under the Curve - Receiver Operating Characteristic) score is a metric that represents how well a model can distinguish between the positive and negative classes [21], across various classification thresholds. For this, we plot the True positive rate (TPR) as well as the False positive rate (FPR) for the various thresholds. The AUC-ROC score can range from 0 to 1 where 1 represents perfect performance, 0.5 represents random guessing and 0 represents the worst measure of separability - this is where the model is inverting the negative class as positive and vice versa. This is computed in python using the scikit-learn library. For a classification problem with N prediction labels, we can compute the overall AUC-ROC using the One-Versus-All (OVA) approach that is, say for three labels X, Y, Z, one curve would represent X classified against Y and Z, another curve would represent Y classified against X and Z and the third curve would be Z classified against X and Y.

PR-AUC

For this metric, we plot the precision and recall value in a single visualization (not the TPR or FPR as in ROC-AUC), across various classification thresholds. PR-AUC is the area under the PR curve. This metric is more useful for imbalanced datasets to show its performance with respect to the positive classes as it focuses mainly on the positive class and cares less about the frequent negative class [22]. A higher PR-AUC indicates that the model is capable of balancing between precision (correctly identifying positive cases) and recall (identifying most of the true positive cases)

Train, Test Split Data Configuration

We train and evaluate our models using the AC-Net dataset. We randomly select 3600 app sentences and allocate that as our test dataset. The remaining app descriptions are then allocated as the training set. Subsequently, the training set is split into training and validation set using a 4:1 ratio for improving the model during training. This exercise is repeated for a total of 10 times and

the average and median metric values are reported. This is the standard dataset splitting procedure we employ, unless stated otherwise.

3.4 Generative AI for permission inference

Generative AI refers to algorithms that are capable of seemingly generating new text, images, audio, etc based on training content [23]. Sufficiently large generative models exhibit a property known as emergence where the model exhibits complex and unpredictable behavior or capabilities due to the model's size, training data, or increase in computational resources. These emergent behaviors are not explicitly applied to the system during the training phase, but occur as a result of the model's ability to adapt and generalize through extensive training over large datasets [3]. This ability to solve general problems makes large generative models suitable candidates for the permission inference task. However, in order for the large language models' capability for inferring permissions to be emergent, it is important that this capability is only present in larger models and not in smaller models [24].

In the following sub-sections, we leverage generative models belonging to distinct transformer families - BART, GPT and evaluate how they perform on the permission inference task under zero-shot prompting configurations as well as under various hyper-parameter configurations (temperature, top-p).

3.4.1 BART

BART stands for Bidirectional and Auto-Regressive Transformers [25]. This class of LLMs uses a transformer architecture with both an encoder and a decoder. While BERT has an encoder-only architecture and GPT has a decoder-only architecture, BART can be seen as a generalization of the two architectures. BART uses span corruption as its pre-training objective which involves corrupting entire spans of documents, rather than mere words and then mapping these corrupted

[3]There are very few comprehensive, publicly available datasets for the permission inference from app description task and thus, it is highly likely that the large language models have never encountered this problem during training, which is one of the prerequisites for satisfying the emergence property

documents to the original ones. As a result, BART has been known to perform well for tasks such as capturing long-range dependencies in sentences, understanding entire documents or long sequences of texts.

For our permission inference task, we use the Bart-Large-MNLI model which is a checkpoint for Bart-Large after being trained on the Multi-Genre Natural Language Inference (MNLI) dataset [26]. This model is also fitted with a classification head making it suitable for zero-shot classification. The model network has over 400 million parameters, making it the largest model we consider for the fine-tuning task.

Zero-shot inference

Zero-shot inference involves conditioning the model to perform a certain task without explicit training on the task and by using only the instructions (prompts) describing the task [27]. In our first experiment, we load Bart-MLNI's zero-shot classification pipeline and then feed our permission labels to the model. Then, for each input sentence, the model performs scoring across each of the 11 permission labels. The scores assigned by the model ranges from 0 to 1 and we use 0.5 as our default binary threshold i.e we treat the score values greater than 0.5 as true (set to 1) and values less than 0.5 as false (set to 0). The experiment is conducted using our standard repeated holdout cross validation approach and we experiment with various binary thresholds. The results are reported in tables Table 3.2, Table 3.3 and Table 3.4

Comparison of Metrics Across Different Thresholds

Table 3.2: BART Zero-Shot: Performance metrics at different thresholds (95% CI)

Threshold	Accuracy	Precision	Recall	Weighted-F1	Macro-F1
0.7	0.6500 ± 0.0268	0.0608 ± 0.0330	0.1233 ± 0.0430	0.0686 ± 0.0262	0.0348 ± 0.0173
0.6	0.5370 ± 0.0354	0.0458 ± 0.0168	0.2055 ± 0.0591	0.0709 ± 0.0240	0.0367 ± 0.0158
0.5	0.4150 ± 0.0351	0.0551 ± 0.0198	0.3286 ± 0.0619	0.0859 ± 0.0241	0.0567 ± 0.0273
0.4	0.2985 ± 0.0268	0.0435 ± 0.0104	0.4356 ± 0.0453	0.0768 ± 0.0161	0.0479 ± 0.0169
0.3	0.2070 ± 0.0175	0.0377 ± 0.0074	0.5104 ± 0.0546	0.0694 ± 0.0130	0.0408 ± 0.0101

Discussion

We can observe that the zero-shot inference performance of Bart-MNLI is not good. All macro f1-scores computed across various probability thresholds are less than 0.1, which indicates very poor performance. It is interesting to note that the precision scores are significantly poor compared to the recall score, indicating that the model is prone to predicting many false positives. Also, the 95 percent confidence interval for various metrics across the 10 experimental runs show a sizable range of values, indicating some instability in the performance of the model.

ROC AUC and PR AUC Scores

Table 3.3: BART Zero-Shot: ROC AUC score for each label (95% CI)

Label	ROC AUC
STORAGE	0.5880 ± 0.0594
CONTACTS	0.6253 ± 0.1100
LOCATION	0.5605 ± 0.0515
CAMERA	0.5467 ± 0.1204
MICROPHONE	0.5238 ± 0.0983
SMS	0.7209 ± 0.1110
CALL_LOG	0.6366 ± 0.3758
PHONE	0.5837 ± 0.1602
CALENDAR	0.4581 ± 0.2215
SETTINGS	0.5078 ± 0.0700
TASKS	0.6029 ± 0.1647

Table 3.4: BART Zero-Shot: PR AUC scores for each label (95% CI)

Label	PR AUC
STORAGE	0.0966 ± 0.0336
CONTACTS	0.1359 ± 0.0990
LOCATION	0.1258 ± 0.0630
CAMERA	0.0669 ± 0.0673
MICROPHONE	0.0360 ± 0.0350
SMS	0.1247 ± 0.1085
CALL_LOG	0.0956 ± 0.1135
PHONE	0.0283 ± 0.0145
CALENDAR	0.0754 ± 0.0817
SETTINGS	0.0304 ± 0.0099
TASKS	0.0418 ± 0.0184

The ROC-AUC scores for most permission labels hover between the 0.5 to 0.6 range which indicates that the model is mostly performing slightly above the level of random guessing while displaying some discriminative ability for labels such as SMS, CONTACTS. The CALL LOG label has a very wide confidence interval, which suggests a high degree of uncertainty when it comes to classification performance for this label. On the other hand, the PR AUC scores are poor and hover around or below 0.1. This indicates an inability to correctly identify positive instances of the permission labels or the ability to identify most of the true positive labels.

Fine Tuning

The zero-shot performance for Bart-MNLI showed poor performance across all permission categories. This indicates that the NLI dataset that the model was originally trained on might have contained training instances different from app descriptions. To potentially improve the performance of the model on this particular classification problem, we need to adapt the model to our task. This can be accomplished by further training or fine-tuning the model on our dataset.

Training

We use BART's default tokenizer for its "facebook/bart-large-mnli" version. For training, we use the following hyperparameter configuration (learning rate=2e-5, batch size=16, num train epochs=5). The experiments were performed on Google Colab, making use of an Nvidia A100 GPU (40 GB RAM) to improve training / inference time.

Table 3.5: BART: Overall Metrics After fine-tuning

Metric	Mean	Median
Accuracy	0.8535	0.8544
Precision	0.7054	0.7062
Recall	0.5814	0.5990
Weighted F1	0.6213	0.6262
Macro F1	0.6152	0.6207

Table 3.6: BART: Per-Class Metrics after fine-tuning on the original AC-Net dataset

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.7027	0.7227	0.5011	0.4574	0.5772	0.5651	0.9486	0.9490	0.6668	0.6780
CONTACTS	0.7706	0.7981	0.5866	0.6056	0.6601	0.6562	0.9733	0.9719	0.7488	0.7415
LOCATION	0.6705	0.6870	0.7604	0.7766	0.7107	0.7248	0.9832	0.9831	0.7815	0.7885
CAMERA	0.7643	0.7612	0.7013	0.6620	0.7293	0.7234	0.9789	0.9769	0.7556	0.7499
MICROPHONE	0.5978	0.6000	0.4484	0.4375	0.4785	0.4848	0.9827	0.9815	0.5854	0.5759
SMS	0.7991	0.8261	0.6988	0.7292	0.7408	0.7246	0.9905	0.9909	0.8174	0.8457
CALL_LOG	0.7445	0.7037	0.5045	0.5000	0.5912	0.6250	0.9964	0.9967	0.7372	0.7454
PHONE	0.5682	0.5556	0.6201	0.7368	0.5562	0.5789	0.9816	0.9762	0.5922	0.6143
CALENDAR	0.7400	0.7447	0.8207	0.8511	0.7775	0.7843	0.9884	0.9895	0.8244	0.8210
SETTINGS	0.5866	0.5490	0.4496	0.4828	0.4968	0.5138	0.9643	0.9630	0.5344	0.5347
TASKS	0.7190	0.6667	0.3345	0.2885	0.4490	0.4412	0.9403	0.9480	0.5654	0.5575

Fine Tuning Results

Per-Class Metrics

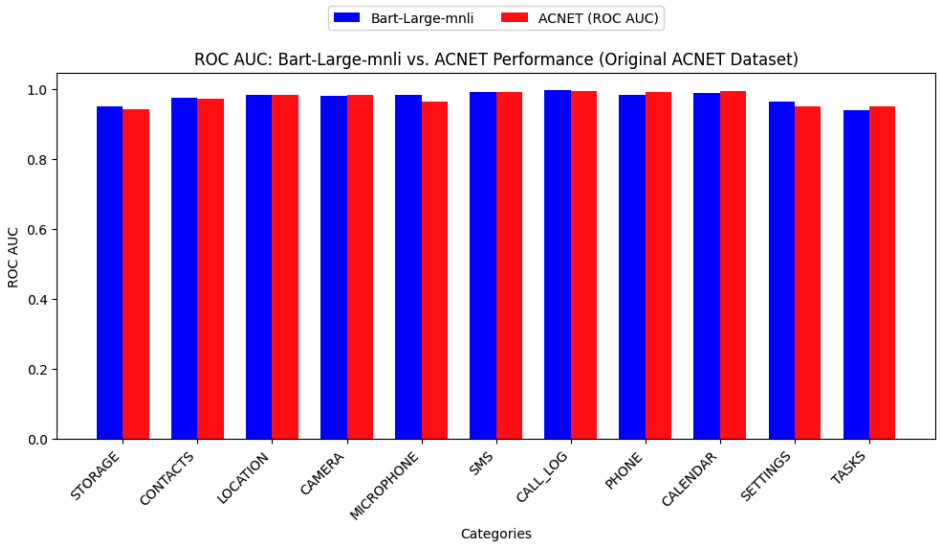


Figure 3.1: ROC-AUC Performance of Bart vs AC-Net

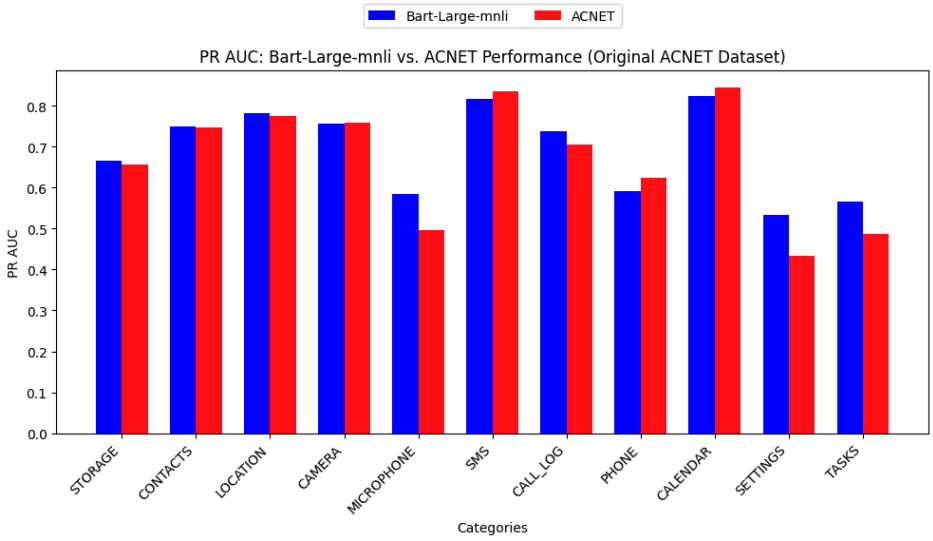


Figure 3.2: PR-AUC Performance of Bart vs AC-Net

Discussion

The per class metrics after fine tuning looks significantly better than those from the zero-shot inference. The model does very well for detecting CALENDAR, SMS, LOCATION and CAMERA labels. Whereas, it struggles the most to make SETTINGS and TASKS label related inferences. We observe a significantly low recall for particularly these label categories indicating that the model misses a large number of positive labels for those categories (false negatives). From the table and graphs above, we show that the fine tuned BART model is comparable to the state-of-the-art (AC-Net). The ROC-AUC and PR-AUC scores for both models are nearly identical. However, BART outperforms AC-Net on the microphone, settings, and tasks labels, suggesting it is better at correctly identifying true instances of these labels. Despite this, both AC-Net and the fine-tuned BART still struggle to predict positive instances for these classes compared to the other labels.

3.4.2 GPT

Another class of Generative AI we use for our experiments is the GPT [28] family. GPT stands for Generative Pre-Trained Transformers, which was introduced in 2018 by OpenAI. GPT was designed to solve the problem of solving a wide variety of tasks despite the dearth of labeled data

specific to these tasks, while requiring minimal changes to the model architecture. To achieve this, GPT first performs the generative pre-training step on a diverse and voluminous corpus of unlabeled text like Wikipedia, followed by discriminative fine-tuning on specific tasks. GPT is based on the decoder-only transformer architecture and uses autoregressive language modeling as its pre-training objective. As per this objective, GPT is tasked to predict the next token in a left-to-right (unidirectional) fashion, while modeling the conditional probability of the next token based on the previous words.

The results have shown GPT outperforming models that are dedicated to achieve high performance on these various tasks. The capabilities of GPT has seen several improvements over the years and the recent models have proven to be very capable - matching human performance in various professional and academic benchmarks [29]. In our experiments, we make use of GPT-4o, which is OpenAI's flagship model - designed to perform well on a wide variety of general tasks [30].

Zero-shot Learning

This step makes no use of task-specific fine tuning whatsoever and utilizes the out-of-the-box inference capabilities of the model. We supply GPT with textual prompts, with descriptions of the permission inference task and also, the input sentences to operate on. We use the default configuration for all other LLM parameters (temperature, nucleus sampling, etc). The original prompt used for our experiments is given below:

```
"Based on the following app description, perform the following
steps:
1. **Identify the Expected Android Permissions**:
- From the app description, determine the set of
Android permissions that the app is likely to require.
- Focus only on permissions relevant to the described
features and functionality.
```

```
- Use the standard Android permission names
(e.g., INTERNET, ACCESS_NETWORK_STATE, WAKE_LOCK).
- Exclude permissions that seem unnecessary or ambiguous.
Here are the list of Android permissions you need to look for:
'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'],
Only return the general permission category
like STORAGE, LOCATION, CONTACTS, etc from the list above.
Also, return the output in the format
(if there are predicted permissions):
        "{\"sentence\": [app_description],
        \"[general_permission_category]\": 1} "
If the sentence appears to have no explicit reference to the given
permissions, then only return the sentence in the format
        "{\"sentence\": [app_description]}"
Please exclude any other details.
```

The prompt listed above instructs GPT to look for the 11 permission categories and then return the output in the specified JSON format. In order to guide GPT towards a conservative approach of

focusing on the more explicit permissions, rather than all implicit ones, we prompt it to focus only on permissions relevant to the described features and functionality. An illustration of the explicit versus the implicit permissions from a given app description has been depicted in Table 3.7.

Table 3.7: Explicit vs. Implicit Permissions for an Example Social Media App Description

Sentence: Our social media app allows users to share photos, send text messages, and make voice and video calls with friends.	
Explicit Permissions	**Implicit Permissions**
STORAGE	LOCATION (theoretically possible that this app uses the location of the user)
SMS	CONTACTS (theoretically possible that this app uses the phone's contacts list)
PHONE CAMERA MICROPHONE	

It is also important to note that depending on the prior technical knowledge of the annotator, given an app description, different annotators may have different interpretations of the sentences and thus, different interpretations of explicit or implicit permissions. This could be due to the ambiguity and vagueness of the language used in policies and app descriptions, which can also explain why inter-annotator agreements of policy texts tend to be relatively low [31]. Once the prompt has been configured, for each prompt, we concatenate it with the app description and then make an API call to OpenAI's Chat Completions endpoint using a Spring boot server that we operate locally. We do not make use of any batch processing and these calls happen at a per sentence basis synchronously.

After we have the json responses for all our app descriptions, we compare the responses to the ground truth labels and compute the performance metrics as reported in the section below.

Results

Table 3.8: GPT: Per-Class Performance Metrics when evaluated on the original AC-Net dataset

Class	Precision	Recall	F1 Score	Support
STORAGE	0.7124	0.5441	0.6170	487
CONTACTS	0.8995	0.4802	0.6262	354
LOCATION	0.9281	0.8516	0.8883	364
CAMERA	0.9486	0.7571	0.8421	317
MICROPHONE	0.7458	0.5301	0.6197	83
SMS	0.8889	0.7419	0.8088	248
CALL_LOG	0.8529	0.4328	0.5743	67
PHONE	0.8405	0.5905	0.6937	232
CALENDAR	0.7917	0.8636	0.8261	44
SETTINGS	0.7250	0.1362	0.2292	213
TASKS	0.8182	0.2455	0.3776	110

Table 3.9: GPT-4o: Overall Performance Metrics when evaluated on the original AC-Net dataset

Metric	Precision	Recall	F1 Score
Macro Average	0.8320	0.5613	0.6457
Weighted Average	0.8407	0.5848	0.6711

Discussion

Under zero-shot setting, GPT achieves far better performance than Bart-MNLI. GPT excels at classifying LOCATION, CAMERA, SMS and CALENDAR labels On the other hand, it struggles to classifiy SETTINGS, TASKS labels. This observation is very similar to BART's performance as well, where it struggled to classify the SETTINGS and TASKS label. A reason for this could be that the app descriptions pertaining to these permission labels may not be as obvious as for the other permissions.

Also, the ROC-AUC, PR-AUC metrics have not been reported for GPT as OpenAI does not return the probabilities for all 11 of the permission labels. Since the stated metrics require the probability thresholds to be available, computing these metrics with GPT is not feasible.

Fine-tuning

We do not perform fine-tuning of the GPT-4o model for this experiment as the cost of fine-tuning an enterprise model like gpt on a large dataset would be prohibitive. Instead, we looking into other low-resource methods to potentially tune the performance of GPT.

LLM parameters

GPT supports various configurable parameters that can be leveraged to affect the output produced by the models. We investigate the effect of two of the most popular GPT parameters on the inference performance:

Temperature

Sampling temperature is used to control the randomness of the output returned by GPT. According to OpenAI, a low temperature value like 0.2 results in more concise, deterministic responses, whereas a higher temperature value of around 1.0 or beyond would result in more random and creative responses.

Table 3.10: GPT: Comparison of Outputs for Different Temperature Values

Prompt	**Temperature = 0.1**	**Temperature = 0.9**
Describe a serene sunset by the ocean.	The sun sets over the ocean, creating a calm and peaceful scene.	As the golden orb descended into the endless horizon, the sky ignited with streaks of crimson and amber, reflecting brilliantly on the serene ocean below.

For this experiment, we have curated a 300 sentence subset of our original dataset to evaluate GPT-4o's performance under temperature settings: 0.1, 0.4, 0.7, 1.0 and 1.5.

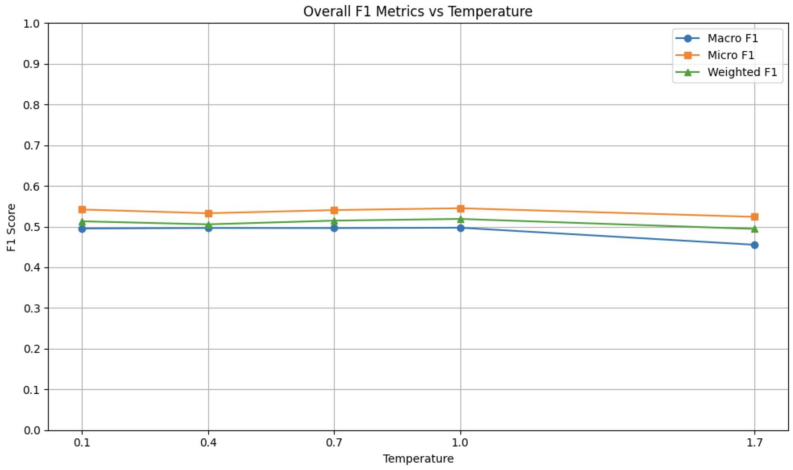


Figure 3.3: Performance vs temperature for GPT-4o-mini

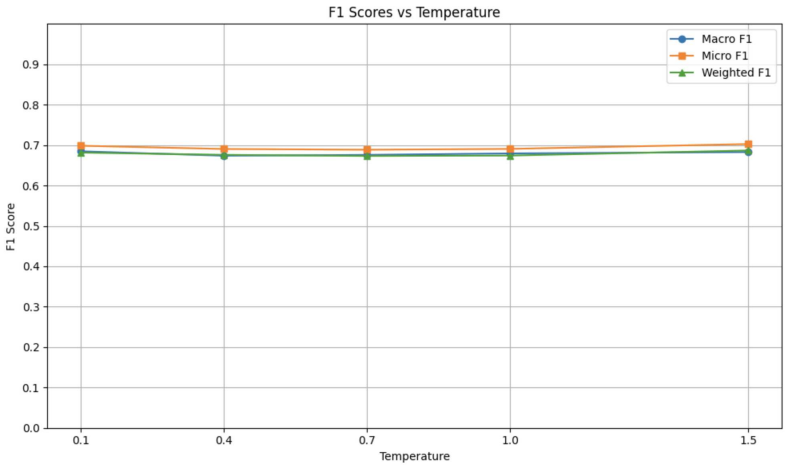


Figure 3.4: Performance vs temperature for GPT-4o

From the graphs above [4], we can observe that varying the temperature does not affect the classification performance in any significant manner. This is in line with the observations made during recent works studying the effect of temperature variations on the classification performance [32]. This could be because the classification labels are returned based on the argmax operation on the highest prediction probabilities and these might remain stable across various temperature ranges.

Nucleus Sampling (Top-p)

Nucleus sampling, also called top-p, is another tunable parameter that can be used to configure GPT's output. This setting tells GPT how many possible tokens to consider while generating the outputs. A higher top-p tells GPT to look at a greater number of words, even the unlikely ones, which results in the resulting output possibly looking more diverse.

For instance, if the top-p value has been specified as 0.4, GPT will consider tokens that will sum up to a total probability of at least 40 percent. Similarly, if the top-p value has been specified as 0.9, GPT will consider tokens that will sum up to at least 90 percent probability. The number of tokens needed to add up to a total 90 percent probability will be higher and thus, this could result in a greater variety of tokens in the output.

Table 3.11: GPT: Comparison of Outputs for Different top_p Values

Prompt	**top_p = 0.4**	**top_p = 0.9**
Describe a serene sunset by the ocean.	The sun gently set over the calm sea, casting a warm glow that mirrored softly on the water's surface.	As the golden orb descended into the endless horizon, the sky ignited with streaks of crimson and amber, reflecting brilliantly on the serene ocean below.

For our experiments, we test the impact of different top-p values on the performance of GPT 4o and 4o-mini. The values we consider are: 0.1, 0.4, 0.7 and 1.0.

[4]The full performance metrics have been presented in the appendix Table B.1

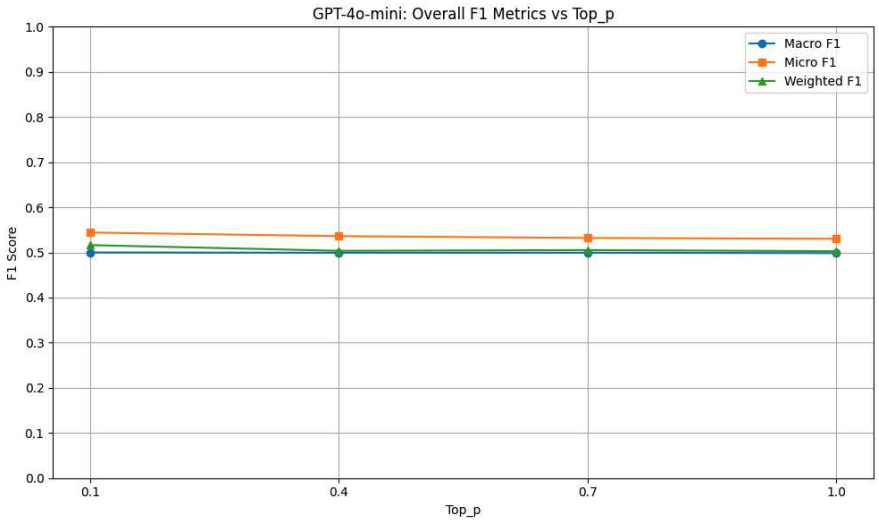


Figure 3.5: Performance vs top-p for GPT-4o-mini

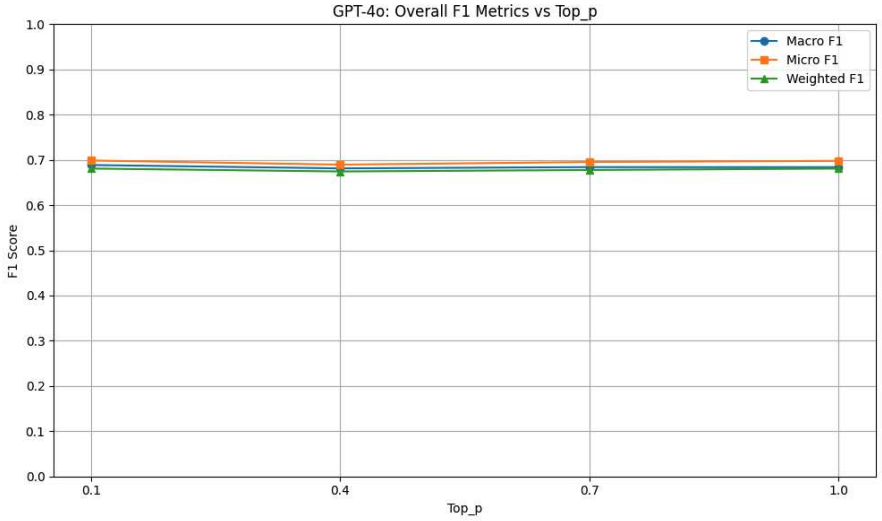


Figure 3.6: Performance vs top-p for gpt-4o

From the graphs above, we can observe that the overall f1 metrics remain unaffected despite the variations in the top-p value. As with the temperature parameter, top-p affects the creativity of the generated output and since our problem is a classification task, it seems reasonable to reason that top-p does not affect the performance in this problem space.

Performance comparison between GPT models

In this section, we present a comparison of the inference performance of two of OpenAI's top models - GPT-4o and GPT-4o-mini. GPT-4o is the flagship model, whereas 4o-mini is a lightweight version of 4o. For this experiment, we use the same 300 sentence dataset we used in our previous experiments and feed the same prompt to both GPT models. The temperature, top-p and other parameters were set to default and we ran the experiments for a total of 3 times and we also report the 95 percent confidence interval values of the metrics.

Table 3.12: Experimental Metrics with 95% Confidence Intervals for GPT-4o mini and gpt-4o (3 runs)

	GPT-4o mini			gpt-4o		
Metric	Macro	Micro	Weighted	Macro	Micro	Weighted
Precision	0.8517 ± 0.0791	0.8320 ± 0.0372	0.8442 ± 0.0582	0.8165 ± 0.0242	0.8069 ± 0.0168	0.8176 ± 0.0040
Recall	0.3876 ± 0.0351	0.3907 ± 0.0177	0.3907 ± 0.0177	0.6116 ± 0.0178	0.6030 ± 0.0082	0.6030 ± 0.0082
F1 Score	0.4952 ± 0.0371	0.5317 ± 0.0236	0.5048 ± 0.0210	0.6763 ± 0.0091	0.6902 ± 0.0063	0.6745 ± 0.0055

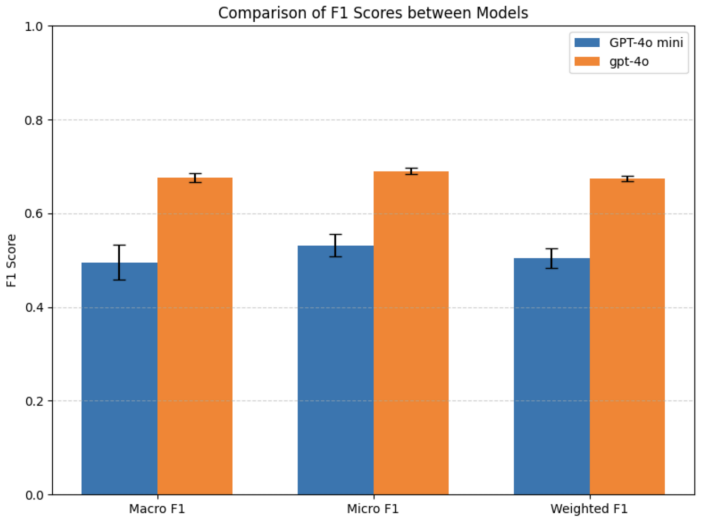


Figure 3.7: Performance of GPT-4o vs GPT-4o-mini on the permission inference task

From the experiments above, it is evident that GPT-4o outperforms GPT-4o-mini for the given task. This doesn't come as huge surprise as GPT-4o has a significantly larger number of parameters

compared to the mini version and thus, is able to capture the linguistic patterns and semantics better than the lightweight model.

3.5 BERT-Based Fine tuned LLMs for permission inference

BERT [33] (Bidirectional Encoder Representations from Transformers) is another transformer-based language model which uses an encoder only architecture. It uses Masked Language Modeling (MLM) as its pre-training objective, which involves masking a token in the given sentence and then pre-training the model to reconstruct this masked token. Ever since its introduction, BERT has proven its utility in several domains and has underwent several modifications and improvements. We use several variants based on the BERT architecture such as RoBERTA as well as distilled variants of these base models in our experiments and evaluate their performance.

Training: BERT-Base-Uncased

This is the base BERT model which is uncased, meaning it does not differentiate between uppercase and lowercase letters during tokenization. For training this model, we use the following hyperparameter configuration (learning rate=2e-5, batch size=16, num train epochs=5). The experiments were performed on Google Colab, making use of an Nvidia A100 GPU (40 GB RAM) to improve training / inference time.

Table 3.13: BERT Base: Comparison of Mean ROC AUC and PR AUC Metrics (Non-augmented vs. AC-Net)

Class	ROC AUC	AC-Net (ROC AUC)	PR AUC	AC-Net (PR AUC)
STORAGE	0.9422	0.94276	0.6524	0.65568
CONTACTS	0.9613	0.97197	0.7273	0.74657
LOCATION	0.9682	0.98376	0.7424	0.77496
CAMERA	0.9570	0.98249	0.7135	0.75821
MICROPHONE	0.9562	0.96293	0.5353	0.49665
SMS	0.9849	0.98991	0.8046	0.83454
CALL_LOG	0.9921	0.99332	0.6323	0.70502
PHONE	0.9770	0.99140	0.5817	0.62455
CALENDAR	0.9831	0.99469	0.7966	0.84444
SETTINGS	0.9459	0.95104	0.4816	0.43251
TASKS	0.9034	0.94972	0.4608	0.48659

Results

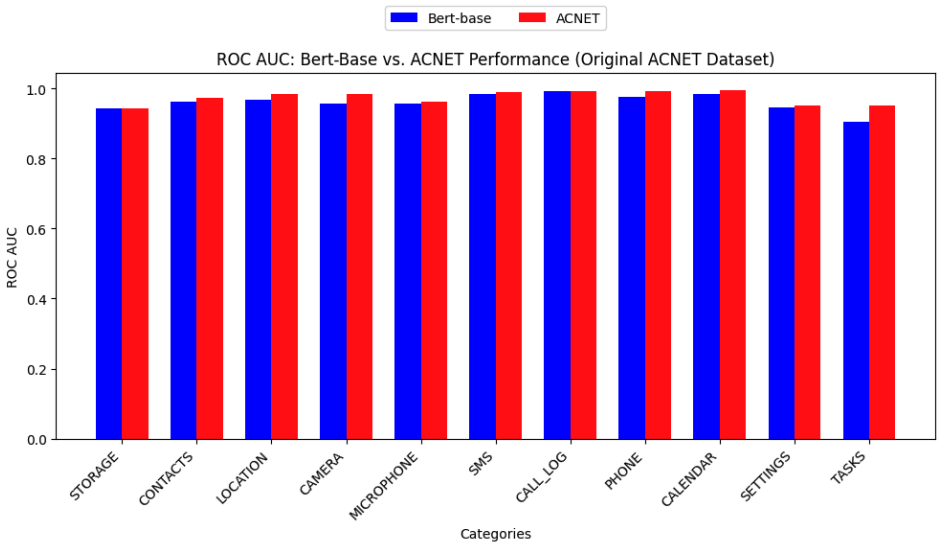


Figure 3.8: ROC-AUC Performance of Bert vs AC-Net

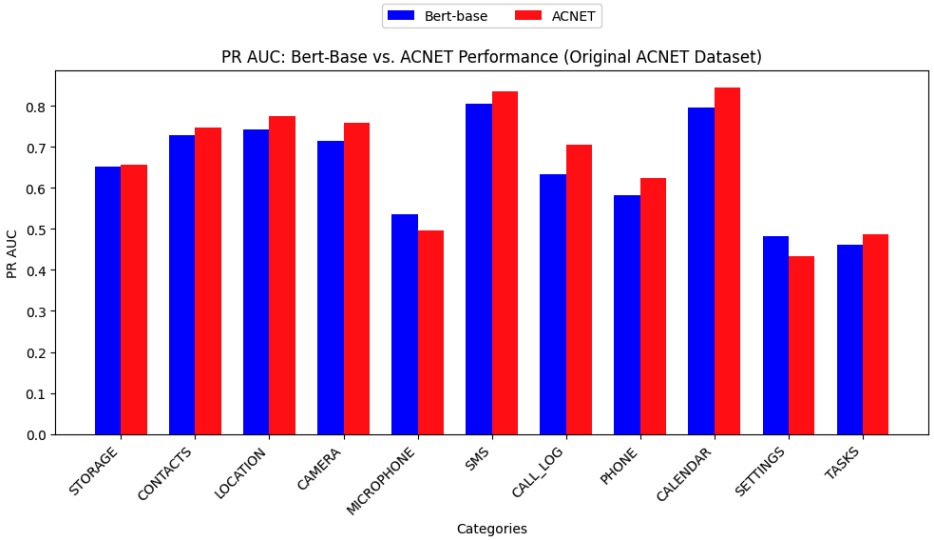


Figure 3.9: PR-AUC Performance of Bert vs AC-Net

Discussion

From 3.8, we can observe that the fine-tuned BERT-large-uncased model has almost identical ROC-AUC scores as AC-Net across all categories. Both models achieve an ROC-AUC score of nearly 1.0, which indicates excellent capability of discriminating between the positive and negative labels. With respect to the PR-AUC scores as outlined in 3.9, AC-Net slightly outperforms the fine-tuned base BERT model. This is true for all categories, except the MICROPHONE and SETTINGS label. It is again interesting to note that the PR-AUC scores for the SETTINGS, TASKS and MICROPHONE label is much lower than the scores for the other labels. This indicates that the models are either missing many true positives (low recall), incorrectly labeling negatives as positives (low precision), or both, hinting to difficulties in distinguishing between the classes.

Training: RoBERTa

RoBERTa [34] (Robustly optimized BERT approach)) is a BERT variant built on top of the base BERT model, which further modifies key hyper-parameters like mini-batch sizes and learning rates and also, removes the next-sentence prediction (NSP) pre-training objective. To fine-tune

the RoBERTa model, we use the same hyperparameter configuration (learning rate=2e-5, batch size=16, num train epochs=5). The experiments were performed on Google Colab, making use of an Nvidia A100 GPU (40 GB RAM) to improve training / inference time.

Results

Table 3.14: RoBERTa: Comparison of Mean ROC AUC and PR AUC Metrics (Non-augmented vs. AC-Net)

Class	Mean ROC AUC	AC-Net (ROC AUC)	Mean PR AUC	AC-Net (PR AUC)
STORAGE	0.9419	0.94276	0.6478	0.65568
CONTACTS	0.9622	0.97197	0.7275	0.74657
LOCATION	0.9735	0.98376	0.7613	0.77496
CAMERA	0.9742	0.98249	0.7235	0.75821
MICROPHONE	0.9698	0.96293	0.5726	0.49665
SMS	0.9885	0.98991	0.7974	0.83454
CALL_LOG	0.9920	0.99332	0.6368	0.70502
PHONE	0.9815	0.99140	0.5691	0.62455
CALENDAR	0.9845	0.99469	0.7809	0.84444
SETTINGS	0.9494	0.95104	0.5003	0.43251
TASKS	0.9330	0.94972	0.5370	0.48659

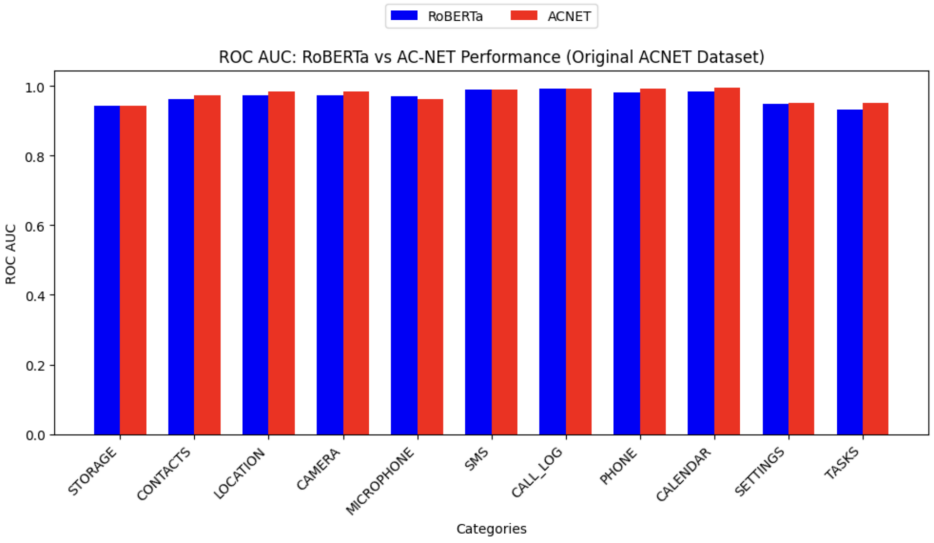


Figure 3.10: ROC-AUC Performance of RoBERTa vs AC-Net

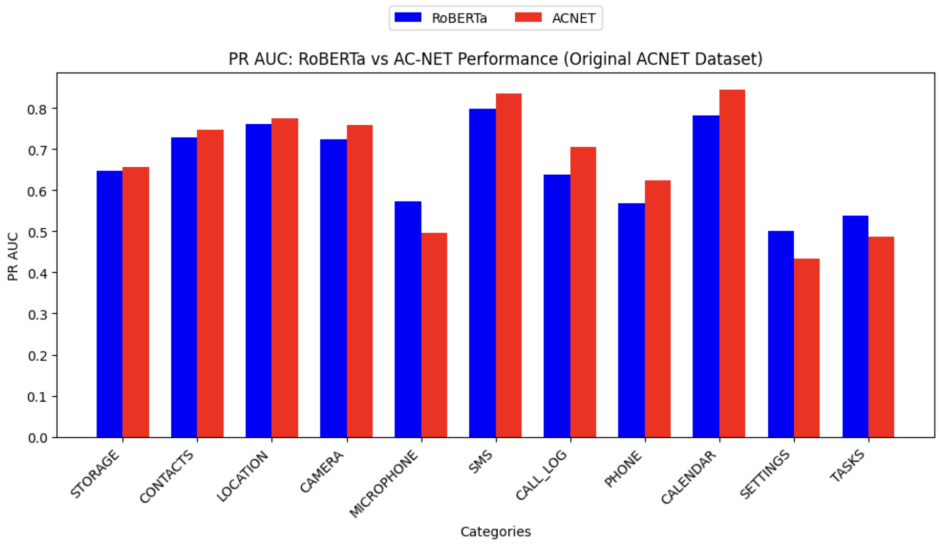


Figure 3.11: PR-AUC Performance of RoBERTa vs AC-Net

Discussion

From 3.10, we can once again observe that the fine-tuned RoBerta model has very comparable ROC-AUC results to AC-Net across all categories. Both AC-Net and RoBERTa have average ROC-AUC scores close to 1.0, which again indicates excellent capability of discriminating between the positive and negative labels. RoBERTa and AC-Net have comparable PR-AUC metrics as well, except for a few instances like MICROPHONE, SETTINGS and TASKS labels where RoBERTa has more noticeable improvements than AC-Net and labels like SMS, CALL LOG, PHONE, CALENDAR where AC-Net performs better than RoBERTa. The overall performance trend for the fine-tuned RoBERTa seem very similar to what was observed for the fine-tuned BERT-base-uncased model, which is natural given that they belong to the same BERT family.

Distilled BERT models

Distillation is a technique that enables knowledge transfer from a larger, powerful model to a smaller, less powerful model. There are several lightweight models trained by distilling the base

BERT model. In this section, we evaluate the fine-tuned performance of these lightweight models on the permission inference task.

Training: DistilBERT and DistilRoBERTa

DistilBERT [35] is a small, fast, cheap, lightweight model trained by distilling the base BERT model. The network has 40 percent less parameters than the original BERT model and runs 60 percent faster than the original model.

Similarly, DistilRoBERTa is a distilled version of RoBERTa. It has 6 layers, 768 dimension and 12 heads, totalizing 82M parameters (compared to 125M parameters for RoBERTa-base) and is in general, twice as fast as the base RoBERTa model.

To fine-tune both distilled models, we use the same hyperparameter configuration (learning rate=2e-5, batch size=16, num train epochs=5). The experiments were performed on Google Colab, making use of an Nvidia A100 GPU (40 GB RAM) to improve training / inference time.

Results

Table 3.15: DistilBERT: Comparison of Mean ROC AUC and PR AUC Metrics (Non-augmented vs. AC-Net)

Class	ROC AUC	AC-Net (ROC AUC)	PR AUC	AC-Net (PR AUC)
STORAGE	0.9405	0.94276	0.6333	0.65568
CONTACTS	0.9622	0.97197	0.7025	0.74657
LOCATION	0.9671	0.98376	0.7358	0.77496
CAMERA	0.9628	0.98249	0.7055	0.75821
MICROPHONE	0.9671	0.96293	0.5182	0.49665
SMS	0.9865	0.98991	0.7828	0.83454
CALL_LOG	0.9931	0.99332	0.6563	0.70502
PHONE	0.9807	0.99140	0.5705	0.62455
CALENDAR	0.9824	0.99469	0.7935	0.84444
SETTINGS	0.9522	0.95104	0.4890	0.43251
TASKS	0.9215	0.94972	0.4807	0.48659

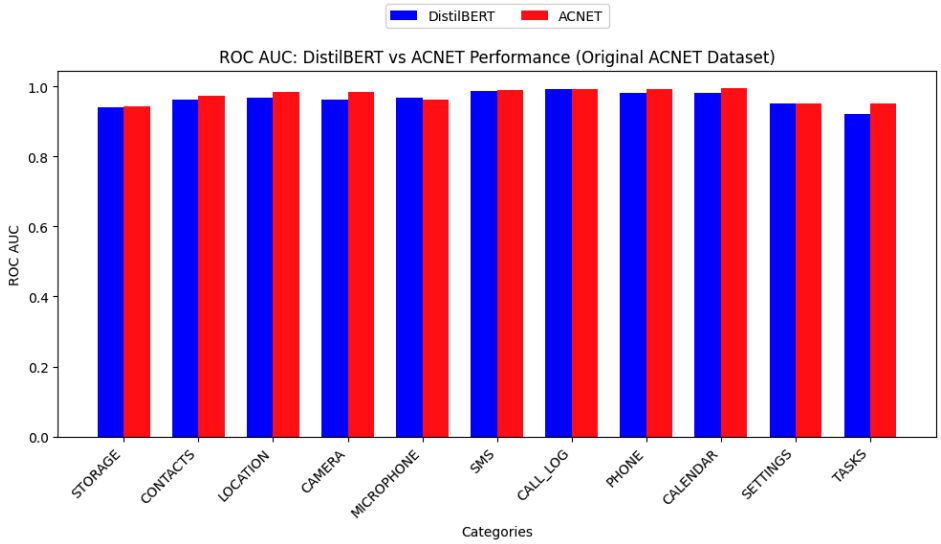


Figure 3.12: ROC-AUC Performance of DistilBERT vs AC-Net

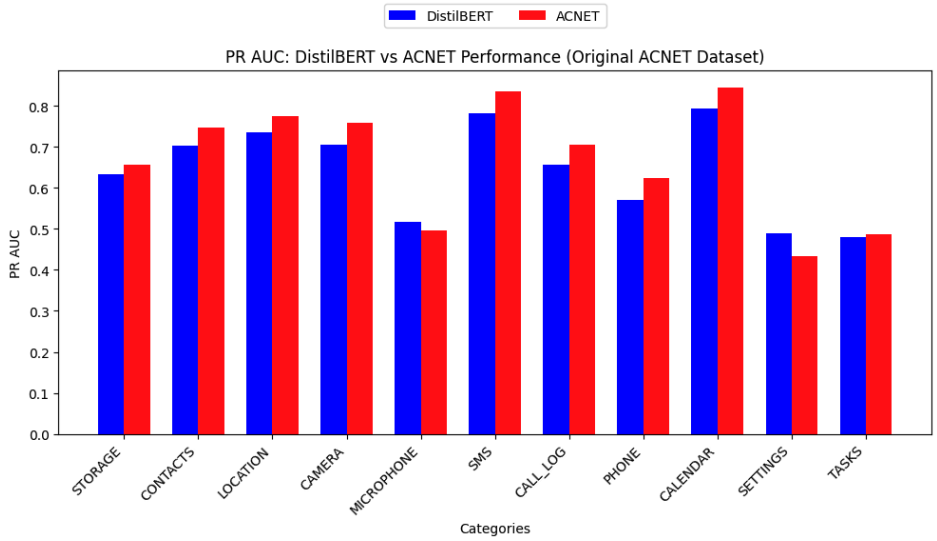


Figure 3.13: PR-AUC Performance of DistilBERT vs AC-Net

Table 3.16: DistilRoBERTa: Comparison of Mean ROC AUC and PR AUC Metrics (Non-augmented vs. AC-Net)

Class	Mean ROC AUC	AC-Net (ROC AUC)	Mean PR AUC	AC-Net (PR AUC)
STORAGE	0.9423	0.94276	0.6436	0.65568
CONTACTS	0.9634	0.97197	0.7302	0.74657
LOCATION	0.9738	0.98376	0.7227	0.77496
CAMERA	0.9716	0.98249	0.6865	0.75821
MICROPHONE	0.9680	0.96293	0.5258	0.49665
SMS	0.9883	0.98991	0.7885	0.83454
CALL_LOG	0.9912	0.99332	0.6347	0.70502
PHONE	0.9787	0.99140	0.5048	0.62455
CALENDAR	0.9802	0.99469	0.7883	0.84444
SETTINGS	0.9528	0.95104	0.4889	0.43251
TASKS	0.9299	0.94972	0.5459	0.48659

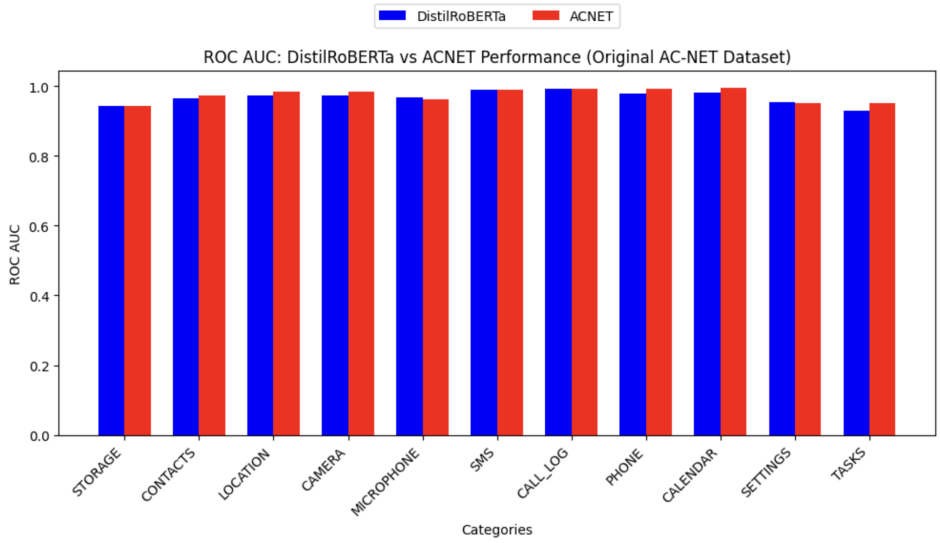


Figure 3.14: ROC-AUC Performance of DistilRoBERTa vs AC-Net

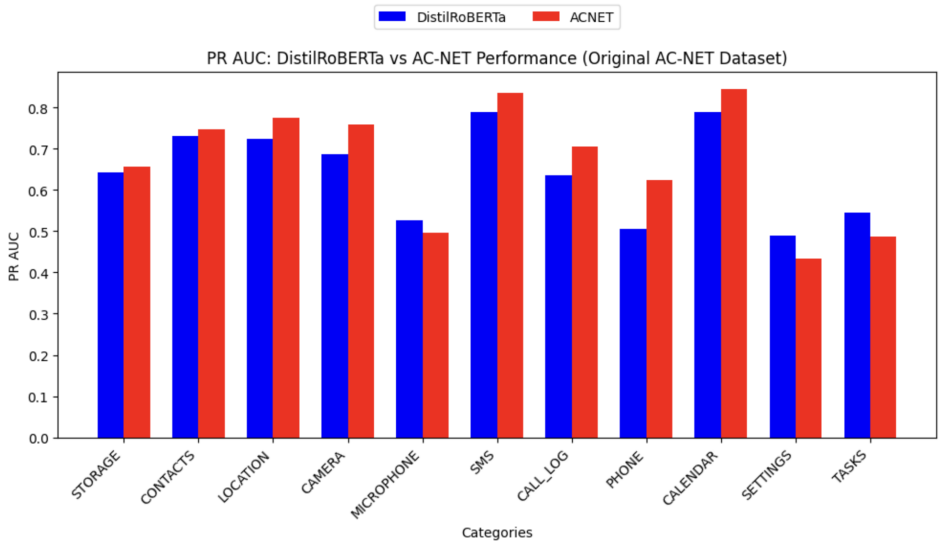


Figure 3.15: PR-AUC Performance of DistilRoBERTa vs AC-Net

Discussion

Like the larger BERT models and AC-Net, the distilled models also exhibit excellent capabilities to discriminate between the positive and negative labels with an average ROC-AUC score of almost 1.0. The distilled models perform similar to the base model with respect to the PR AUC metric. The scores are comparable, but slightly below what AC-Net reports. Nonetheless, the results show that the distilled models, despite being smaller than the base models, perform comparably with respect to the permission inference task. This could be because while the distilled models are smaller than the base models, they still contain millions of parameters, which would make the network sufficiently large enough to learn the semantics of the problem data, enabling it to perform on the same level as the base models.

RQ1: How do fine-tuned LLMs compare against the state-of-the-art when it comes to the app-description permission inference performance?

The fine-tuned BERT and RoBERTa models (both base and distilled) demonstrate near state-of-the-art performance. While they achieve nearly perfect ROC-AUC scores and the PR-AUC scores are comparable to the metrics reported by the state-of-the-art, we can clearly see significant

room for improvement, particularly when we evaluate the raw metrics reported in Table B.3 to Table B.18. The recall scores of all the models seem to be particularly sub-par and thus, the f1 metric is adversely affected. On the other hand, BART-mnli marginally outperforms the fine-tuned BERT models as well as AC-Net after being fine-tuned on the original dataset. This could possibly be attributed to the fact that this model has 4x more parameters than the other fine-tuned models.

RQ2: How well does zero-shot inference using Generative AI perform on the app description - permission inference task? What is the effect of various LLM parameters on their performance?

We observe that the zero-shot performance of GPT-4o across a majority of categories is comparable to the performance of the BERT models after fine-tuning, indicating that GPT-4o's pre-trained capabilities are robust enough to achieve competitive results without additional task-specific training. Bart-NLI, on the other hand, exhibits poor zero shot performance. This is to be expected because GPT-4o has several trillions of parameters compared to the relatively meagre 400 million parameters in the BART model and can thus, be expected to have a greater capability of discerning the semantic and complex linguistic patterns much better, enabling it to perform better at the inference task.

Nonetheless, at the end of the day, given the large amount of compute required to train the LLMs, the fine-tuning performance of the LLMs can be considered underwhelming and not justifying of the training cost. In a later chapter, we will analyze the raw metrics in greater detail and implement targeted solutions to improve specific low-scoring metrics to boost the overall model performance.

Chapter 4

Sparse Data Fine-Tuning

Large language models generally require a substantial amount of data for training. Curating labeled datasets to fine-tune the LLMs involves a lot of resources as well as manual work. The AC-Net dataset had around 27,000 total rows of annotations (20 percent of those were permission sentences). Annotating such a high number of instances was not feasible for us, and thus, we ended up annotating only 250 sentences for our alternative evaluation dataset. In this chapter, we explore how the fine-tuned LLMs perform when trained on a very limited sample such as ours. We also explore the effects of data augmentation on the fine-tuning performance and show how our data augmentation technique yields significant performance improvement, while requiring manual annotations of a very limited number of samples.

4.1 Fine tuning using 250 sentences

In our previous experiments, our setup included well over 20,000 sentences allocated for training the LLM. The results showed that the LLMs trained using this setup were able to match the state-of-the-art. In this section, we select the BERT model and train it using only 250 sentences from our Permissions-250 dataset and evaluate the performance. The training hyper-parameters and the training environment from our previous experiments were kept constant, except for the number of epochs (increased to 10 from 5) to impove convergence. For testing the fine-tuned model post-training, we use the AC-Net dataset (3600 randomly shuffled samples)

Results

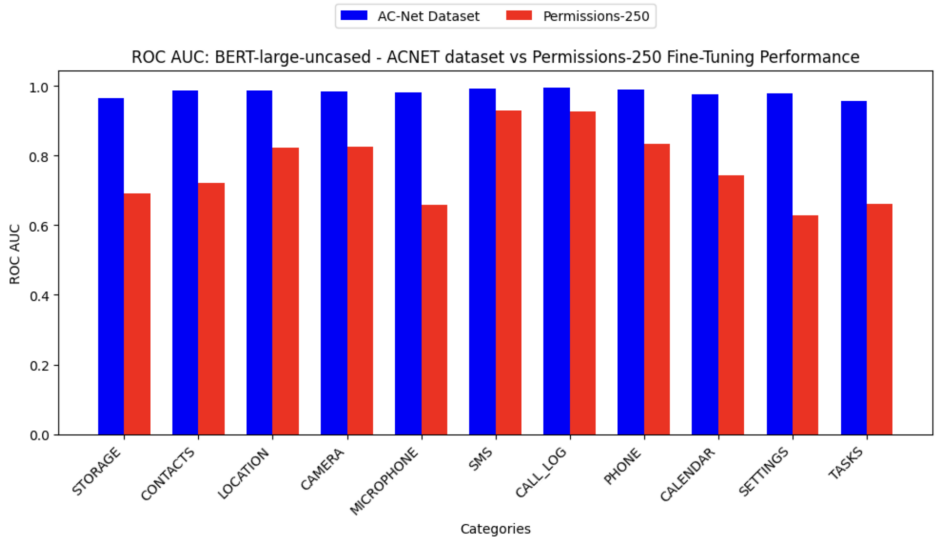


Figure 4.1: ROC-AUC Performance of Bert when fine-tuned using Permissions-250 vs AC-Net

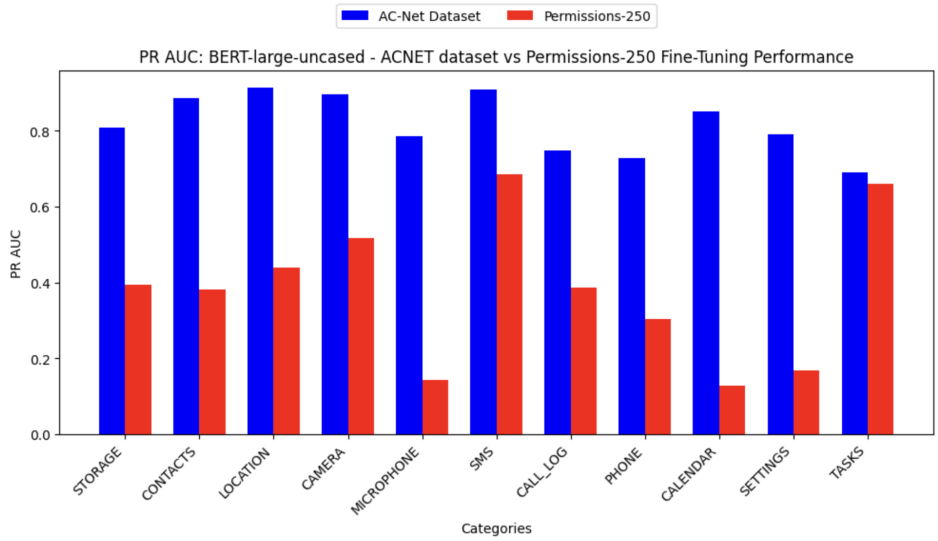


Figure 4.2: PR-AUC Performance of Bert when fine-tuned using Permissions-250 vs AC-Net

The results reported in Table 4.1 and the graphs above show a significant decline in performance when the BERT model was fine-tuned using only 250 samples versus using the 20,000

Table 4.1: BERT: Per-Class ROC AUC and PR AUC Metrics after fine-tuning on 250 samples

Class	ROC AUC	PR AUC
STORAGE	0.6914 ± 0.0033	0.3929 ± 0.0040
CONTACTS	0.7203 ± 0.0027	0.3805 ± 0.0066
LOCATION	0.8229 ± 0.0032	0.4400 ± 0.0079
CAMERA	0.8245 ± 0.0060	0.5171 ± 0.0090
MICROPHONE	0.6578 ± 0.0059	0.1413 ± 0.0050
SMS	0.9307 ± 0.0030	0.6848 ± 0.0071
CALL_LOG	0.9278 ± 0.0033	0.3872 ± 0.0202
PHONE	0.8351 ± 0.0045	0.3034 ± 0.0096
CALENDAR	0.7425 ± 0.0037	0.1283 ± 0.0044
SETTINGS	0.6283 ± 0.0056	0.1686 ± 0.0040
TASKS	0.6612 ± 0.0050	0.0983 ± 0.0023

samples from AC-Net. This clearly indicates that such a limited number of samples may not be sufficient to generate enough data required for the LLM to learn the semantic patterns in the dataset.

The full set of metrics has been reported in Table B.19 and Table B.20. The metrics reported in those tables have numerous 0.00 values because none of the model's predictions exceed the 0.5 threshold, as a result of which all predictions are classified as False (0). This would lead to zero true positives (TP), and hence, metrics like Precision, Recall, and F1 Score would all be 0.0 since they rely on the presence of true positives. The results convey an inability of the models to achieve any sort of usable classification performance, implying a need to revisit our training approach or augment the number of samples in our training data. We focus on the latter approach.

4.2 Data Augmentation

Data augmentation (DA) refers to strategies to artificially increase the number of data samples from existing data, without explicitly collecting more data [36] . This field has seen more work in recent years due to increased NLP work in low-resource domains as well as the rise of large language models which require large amounts of training data. Over the years, several techniques such as synonym replacement [37], back-translation [38], paraphrasing, easy data augmentation

(EDA) [39] techniques such as random insertion, random swap, random deletion or transformer-based approaches such as using GPT, BERT, etc have been leveraged for the DA tasks.

For our purposes, we explore DA with paraphrasing using GPT-4o. This involves passing in the input sentence to GPT along with the prompt and then it outputs approximately the given number of paraphrases for the input sentence.

Table 4.2: An example sentence about app permissions and its five paraphrases.

Original	The app requests location permission to provide personalized recommendations.
Paraphrase 1	The application asks for access to your location so it can tailor its suggestions.
Paraphrase 2	Our software prompts you for location access in order to deliver customized services.
Paraphrase 3	The program requires your location data to offer personalized functionality.
Paraphrase 4	To provide specialized features, the app needs permission to track your location.
Paraphrase 5	In order to customize its content, the app must request the user's location access.

4.2.1 Paraphrasing using GPT-4o

We leverage the GPT-4o model with default temperature and top-p settings to generate the paraphrases. We instruct GPT on the paraphrase generation task using the following prompt:

```
"Please generate 5 paraphrases for the following sentence.
Make sure the paraphrases are different from each other.
Return the output as JSON array with the key \"results\"
```

After GPT returns the array of paraphrases in its response, we parse the individual paraphrases and append them as rows subsequent to the input sentence, while retaining all annotations for the permission labels. After we do this for all our annotated sentences, we have an augmented dataset with a total of 12,193 rows. It is to be noted that we may not end up processing exactly 50 paraphrases for the input sentences. Instead, we observe that around 47-49 paraphrases are

generated by GPT for any given sentence. The next step is to fine-tune BERT using this augmented dataset and observe the results.

4.3 Fine-tuning using Permissions-250-Paraphrased

In this step, we train the base BERT model on our augmented dataset. We designate 10,000 samples for training and the remaining 2,193 samples as the evaluation split. We train the model for 10 training epochs and keep the remaining hyper-parameters and the training environment constant from the previous experiments. For testing the fine-tuned model post-training, we use the AC-Net dataset (3600 randomly shuffled samples)

Results

Table 4.3: BERT: Per-Class ROC AUC and PR AUC Metrics after fine-tuning on the augmented dataset

Class	ROC AUC	PR AUC
STORAGE	0.8258 ± 0.0034	0.6790 ± 0.0049
CONTACTS	0.7802 ± 0.0058	0.6710 ± 0.0070
LOCATION	0.9629 ± 0.0019	0.8499 ± 0.0051
CAMERA	0.9161 ± 0.0057	0.6650 ± 0.0068
MICROPHONE	0.9499 ± 0.0028	0.6831 ± 0.0135
SMS	0.9799 ± 0.0013	0.8215 ± 0.0090
CALL_LOG	0.9834 ± 0.0012	0.6690 ± 0.0157
PHONE	0.9696 ± 0.0025	0.7405 ± 0.0140
CALENDAR	0.9905 ± 0.0016	0.8935 ± 0.0108
SETTINGS	0.8707 ± 0.0037	0.6179 ± 0.0101
TASKS	0.8719 ± 0.0054	0.5964 ± 0.0210

The fine-tuning results obtained when using the paraphrased dataset show significant improvements versus using just the 250 samples for fine-tuning. We report a net average improvement of 15 percent on the ROC-AUC metric, a net improvement of 38 percent on the PR-AUC metric and improvements of 67 percent and 69 percent respectively on the weighted and macro-f1 metrics. The average ROC-AUC score across the labels is 0.91 and the average PR-AUC score across

the labels is 0.71. The PR-AUC score thus, obtained is better than the state-of-the-art average reported by the AC-Net paper (0.66), whereas, the ROC-AUC is slightly less, but comparable to the state-of-the-art (0.97).

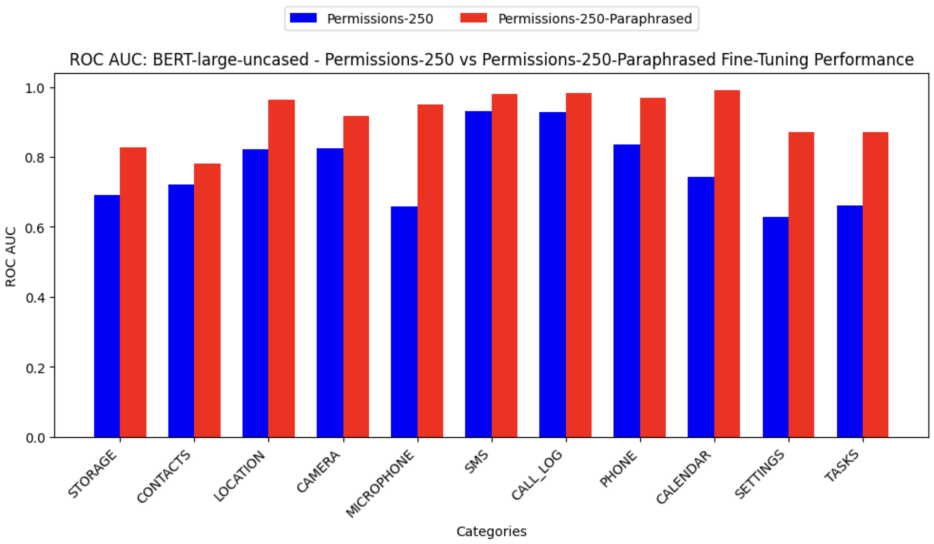


Figure 4.3: ROC-AUC Performance of Bert when fine-tuned using Permissions-250 vs Permissions-250-paraphrased

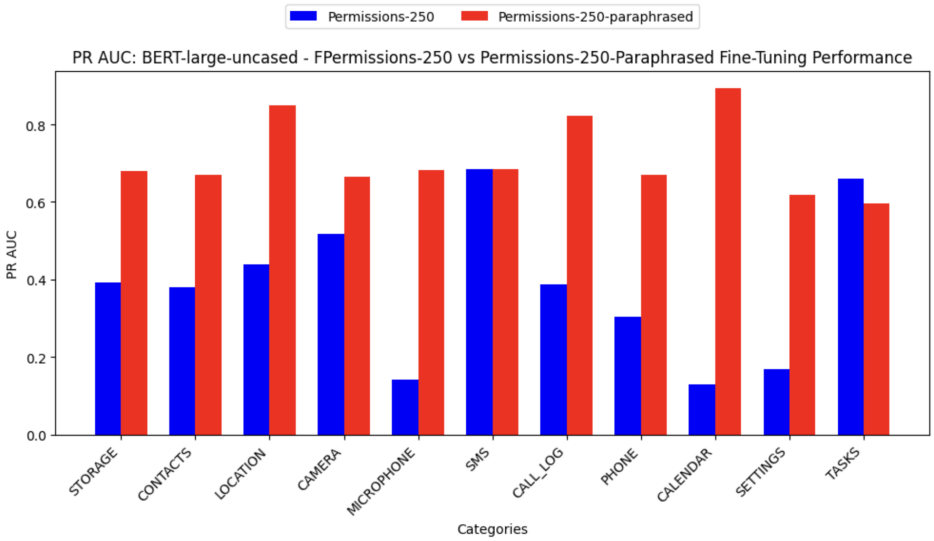


Figure 4.4: ROC-AUC Performance of Bert when fine-tuned using Permissions-250 vs Permissions-250-paraphrased

RQ4: Can we train the LLMs to learn from a limited number of annotated samples? Can data augmentation be leveraged to generate enough high quality samples for the LLM to learn from? Based on our findings, training the LLMs on a very limited number of samples by itself may not be sufficient for the model to learn meaningful patterns with confidence. However, we have shown that by using high quality paraphrases to augment the dataset, we can synthesize high quality samples to boost the fine-tuned performance of the models significantly. We achieved performance comparable to—or even exceeding—that of the state-of-the-art supervised learning model while annotating less than 10 percent of the dataset it used. This shows that paraphrasing can be an effective technique to introduce more diversity to the training dataset enabling the models to learn and improve. All of this comes at no additional cost of having to manually annotate more samples.

It is also important to note that our paraphrased dataset mostly consists of permission sentences (i.e., sentences that include at least one positive permission label). As a result, our model was predominantly trained on examples where at least one positive label was present, which can raise concerns about its ability to effectively handle instances containing only negative labels. This imbalance in the training data may lead the model to under-perform on negative-only cases (non-permission sentences), highlighting the need for future work to incorporate a more diverse range of examples to ensure robust performance across all scenarios. Nonetheless, we can observe the potential of data augmentation to improve the performance of the fine-tuned LLMs.

We will explore the application of the same data augmentation technique to the original AC-Net dataset in the next chapter and analyze if the augmentation yields any performance gains.

Chapter 5

Data Augmentation on AC-Net

Our results after fine-tuning on the original AC-Net dataset show that the LLMs perform similarly to the state-of-the-art. However, given the vast amount of compute required to train these LLMs and the pre-training on vast corpus of data these models undergo, we could expect these models to demonstrate better performance. To investigate further room for improvement, we analyzed the raw metrics during fine-tuning. As depicted in figures 5.1, we observed that our trained models demonstrate poor recall scores and this could be linked back to the severe class imbalance in the original AC-Net dataset. Interestingly, even GPT, for which we did not perform any fine-tuning, demonstrated poor recall scores similar to the other fine-tuned models.

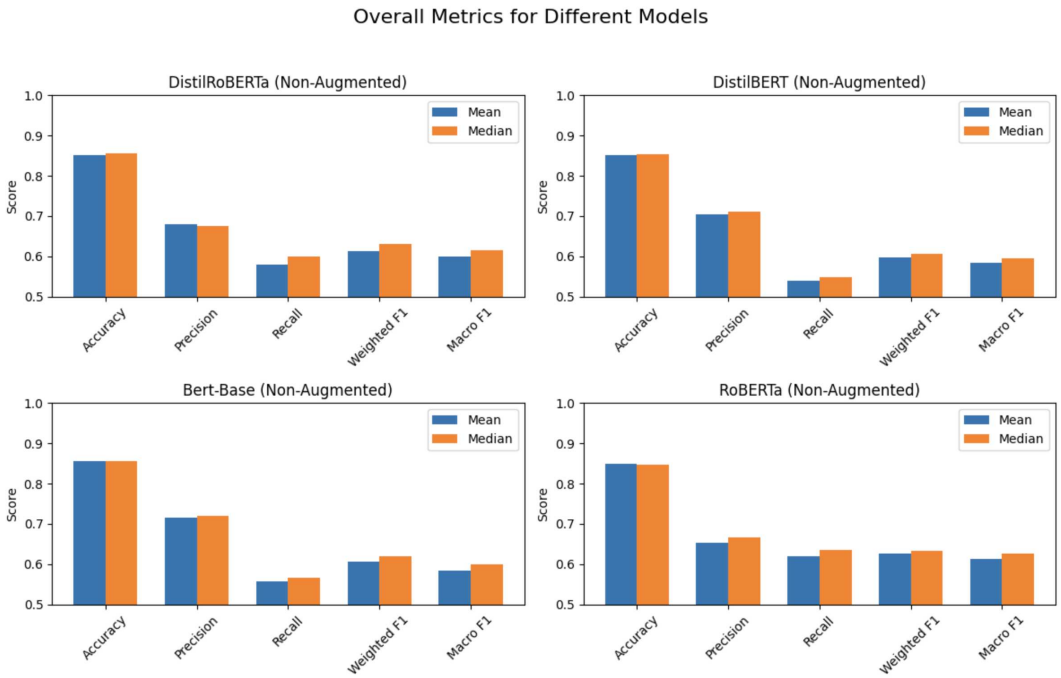


Figure 5.1: BERT, RoBERTa: Global Metrics before augmentation, demonstrating poor recall

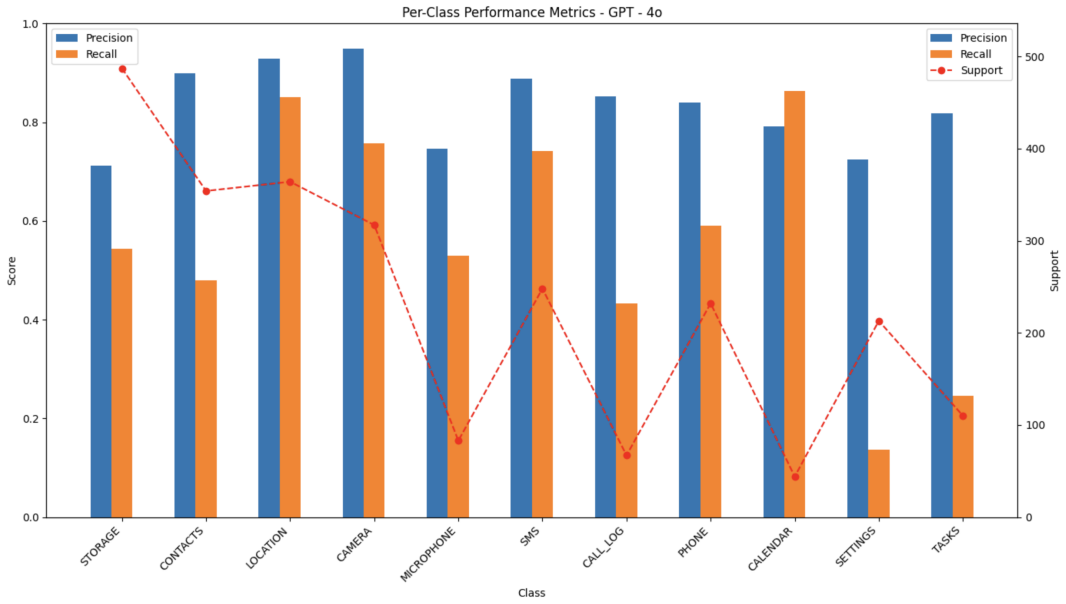


Figure 5.2: GPT-4o: Global Metrics before augmentation, demonstrating poor recall

AC-Net Class Imbalance

From the figures above, it is clear that our models demonstrate poor recall. In other words, our models tend to miss a lot of true positives and falsely label them as negatives. Upon investigating the original AC-Net label distribution, we discovered that this was due to the class imbalance in the dataset. As depicted in 5.3, there is imbalance across the permission labels. However, the imbalance due to the large number of non-permission labels is much more significant. 5.4 illustrates that almost 80 percent of our dataset contains non-permission sentences (that is sentences with no positive permission labels) and as a result, our models' predictions tend to be skewed towards the negative instances, therefore, affecting the recall scores.

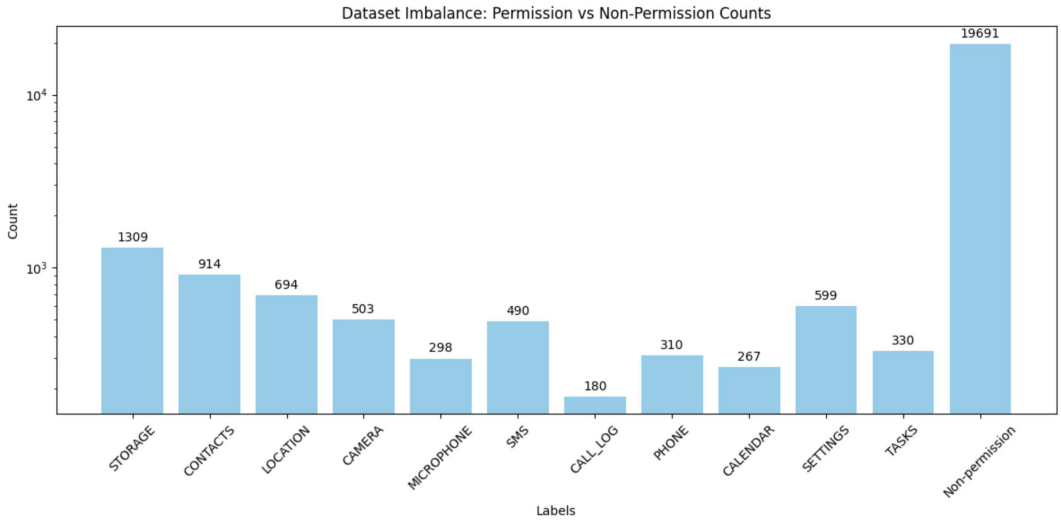


Figure 5.3: AC-Net: Class Imbalance across various permission labels and non-permission labels

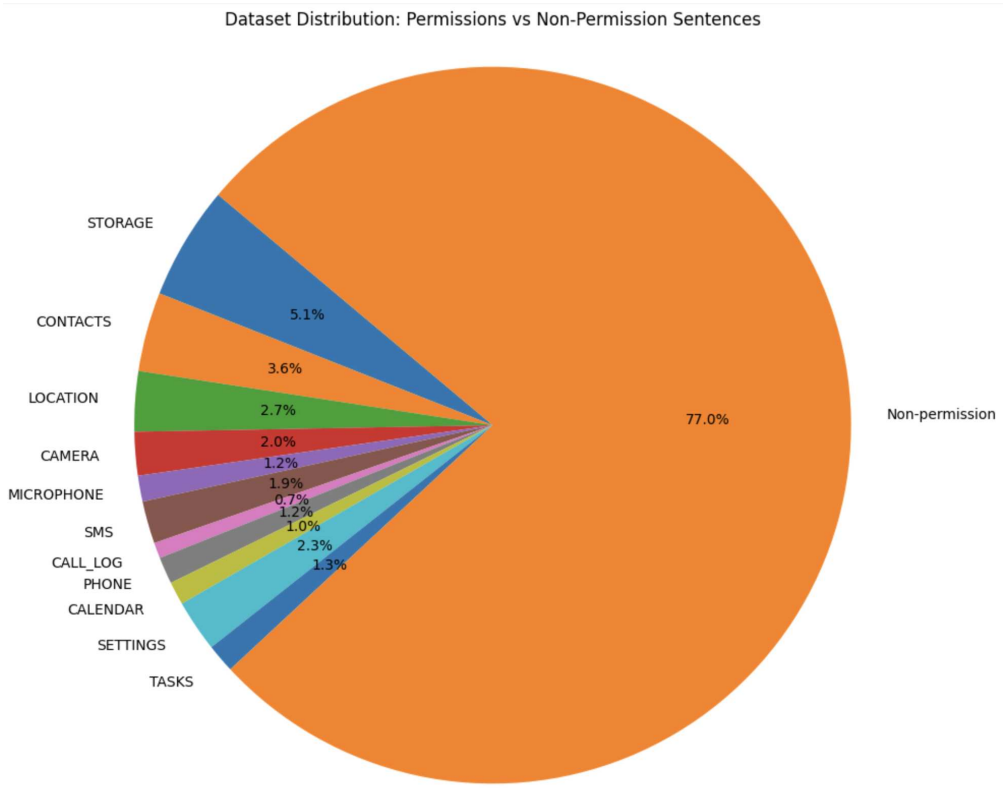


Figure 5.4: AC-Net: Class Imbalance across various permission labels and non-permission labels. Almost 80 percent of the dataset contains sentences with no permission labels, thus making our trained models bias towards negative predictions, thus affecting recall

Solution: Data Augmentation

To combat the data imbalance problem in the training data, we perform data augmentation on AC-Net. We have demonstrated that augmentation via paraphrasing is an effective, automated way to introduce additional samples diverse enough for the LLM to improve upon and learn from. We leverage the same paraphrasing technique here.

We augment the permission sentences by generating a variable number of samples for each permission label with the objective of trying to get the class support for the permission labels to converge towards an approximate lowest common multiple. After augmenting the dataset, we moved slightly past the ideal 50:50 split and attained an approximate 60:40 split, with the positive sentences forming the majority. This was due to numerous sentences being augmented multiple times as these sentences had positive support for multiple labels. Accounting for these multi-label sentences while calculating the appropriate number of paraphrases needed for convergence should help balance the dataset better. Nonetheless, we move forward with our fine-tuning experiments using the newly augmented 60:40 dataset and report our results below for various models.

Training Setup and Experimental Precautions

For training, we use the same hyper-parameter configuration and experimental setup as for the non-augmented dataset. Also, we take special care to not include the paraphrases in the test dataset because:

- When paraphrases that express the same meaning appear in both training and testing, the model might effectively "memorize" these patterns. This leakage can lead to overly optimistic performance estimates since the test set is not fully independent of the training set.
- The model may overfit to the specific phrasing or structure present in the training data, thereby performing exceptionally well on similar paraphrased instances in the test set but failing to generalize to truly novel expressions.

Fine-Tuning Results on the Augmented Training set

Bert-Base-Uncased

Table 5.1: BERT-Base-Uncased - Mean ROC AUC and PR AUC Metrics per Class on the augmented AC-Net dataset

Class	ROC AUC	AC-Net	PR AUC	AC-Net
STORAGE	**0.9669**	0.94276	**0.7369**	0.65568
CONTACTS	**0.9902**	0.97197	**0.8393**	0.74657
LOCATION	**0.9899**	0.98376	**0.8266**	0.77496
CAMERA	**0.9954**	0.98249	**0.8271**	0.75821
MICROPHONE	**0.9904**	0.96293	**0.7402**	0.49665
SMS	**0.9979**	0.98991	**0.9023**	0.83454
CALL_LOG	**0.9995**	0.99332	**0.9433**	0.70502
PHONE	**0.9976**	0.99140	**0.8772**	0.62455
CALENDAR	0.**9984**	0.99469	**0.9089**	0.84444
SETTINGS	**0.9737**	0.95104	**0.6500**	0.43251
TASKS	**0.9866**	0.94972	**0.7179**	0.48659
Mean	**0.9906**	0.9731	**0.8154**	0.6682

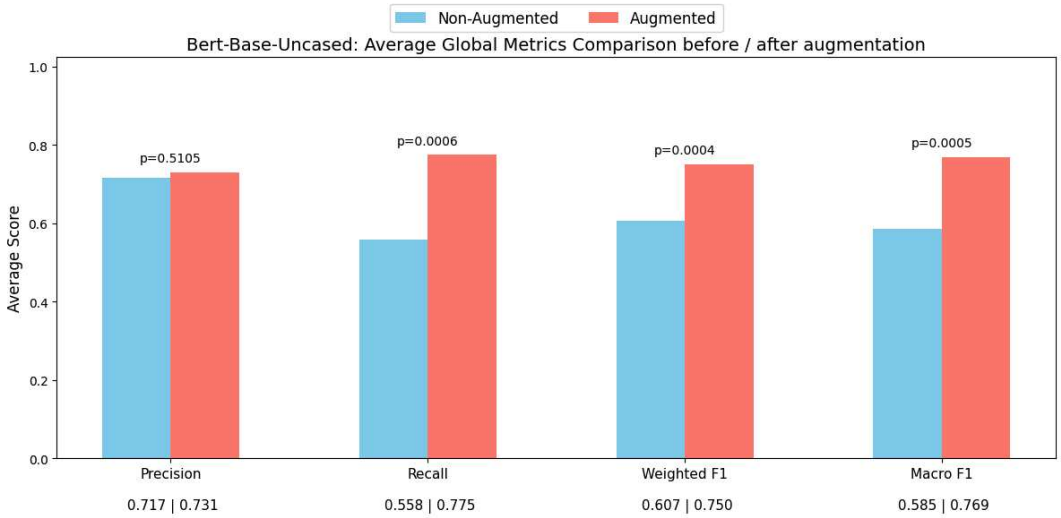


Figure 5.5: BERT-Base-Uncased: Global Metrics before vs after augmentation

DistilBERT

Table 5.2: DistilBERT - Mean ROC AUC and PR AUC Metrics per Class on the augmented AC-Net dataset

Class	ROC AUC	AC-Net	PR AUC	AC-Net
STORAGE	**0.9703**	0.94276	**0.7629**	0.65568
CONTACTS	**0.9895**	0.97197	**0.8196**	0.74657
LOCATION	**0.9914**	0.98376	**0.8198**	0.77496
CAMERA	**0.9952**	0.98249	**0.8229**	0.75821
MICROPHONE	**0.9928**	0.96293	**0.7515**	0.49665
SMS	**0.9979**	0.98991	**0.8870**	0.83454
CALL_LOG	**0.9996**	0.99332	**0.9379**	0.70502
PHONE	**0.9984**	0.99140	**0.8416**	0.62455
CALENDAR	**0.9989**	0.99469	**0.8995**	0.84444
SETTINGS	**0.9791**	0.95104	**0.6349**	0.43251
TASKS	**0.9913**	0.94972	**0.7170**	0.48659
Mean	**0.9913**	0.9731	**0.8086**	0.6682

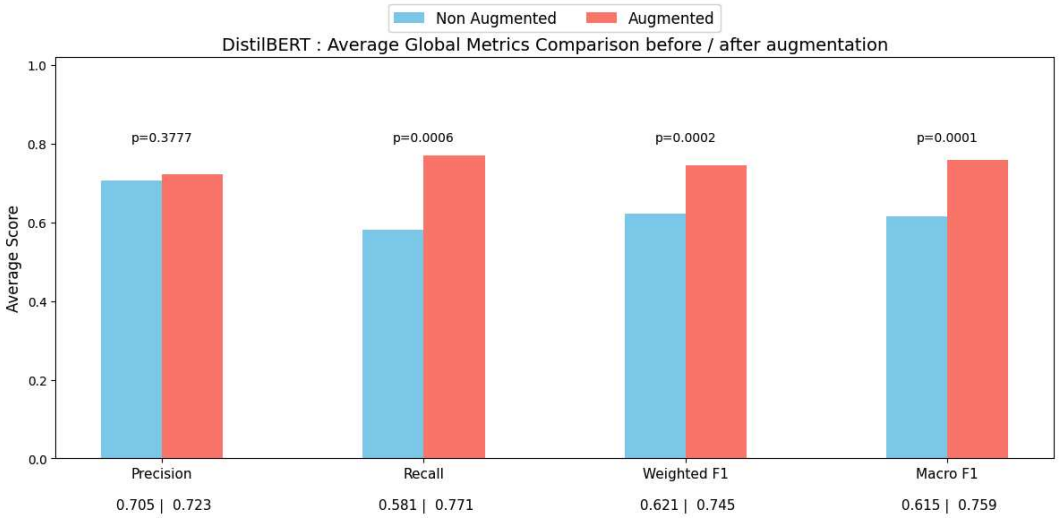


Figure 5.6: DistilBERT: Global Metrics before vs after augmentation

RoBERTa

Table 5.3: RoBERTa - Mean ROC AUC and PR AUC Metrics per Class with AC-Net Values

Class	ROC AUC	AC-Net	PR AUC	AC-Net
STORAGE	**0.9667**	0.94276	**0.7536**	0.65568
CONTACTS	**0.9905**	0.97197	**0.8277**	0.74657
LOCATION	**0.9894**	0.98376	**0.8401**	0.77496
CAMERA	**0.9939**	0.98249	**0.8327**	0.75821
MICROPHONE	**0.9902**	0.96293	**0.7480**	0.49665
SMS	**0.9981**	0.98991	**0.8934**	0.83454
CALL_LOG	**0.9995**	0.99332	**0.9289**	0.70502
PHONE	**0.9983**	0.99140	**0.8572**	0.62455
CALENDAR	**0.9989**	0.99469	**0.8957**	0.84444
SETTINGS	**0.9804**	0.95104	**0.6685**	0.43251
TASKS	**0.9890**	0.94972	**0.7259**	0.48659
Mean	**0.9904**	0.9731	**0.8156**	0.6682

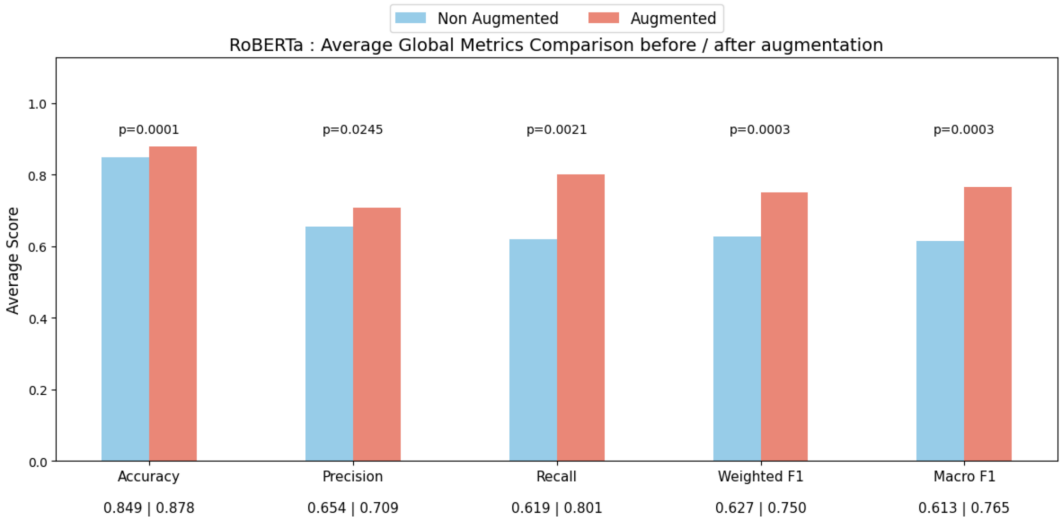


Figure 5.7: RoBERTa: Global Metrics before vs after augmentation

DistilRoBERTa

Table 5.4: DistilRoBERTa - Mean ROC AUC and PR AUC Metrics per Class on the augmented AC-Net dataset

Class	ROC AUC	AC-Net	PR AUC	AC-Net
STORAGE	**0.9660**	0.94276	**0.7285**	0.65568
CONTACTS	**0.9888**	0.97197	**0.8336**	0.74657
LOCATION	**0.9872**	0.98376	**0.8142**	0.77496
CAMERA	**0.9949**	0.98249	**0.7961**	0.75821
MICROPHONE	**0.9923**	0.96293	**0.7512**	0.49665
SMS	**0.9978**	0.98991	**0.8876**	0.83454
CALL_LOG	**0.9994**	0.99332	**0.9138**	0.70502
PHONE	**0.9982**	0.99140	**0.8689**	0.62455
CALENDAR	**0.9991**	0.99469	**0.9231**	0.84444
SETTINGS	**0.9780**	0.95104	**0.6414**	0.43251
TASKS	**0.9886**	0.94972	**0.6984**	0.48659
Mean	**0.9900**	0.9731	**0.8051**	0.6682

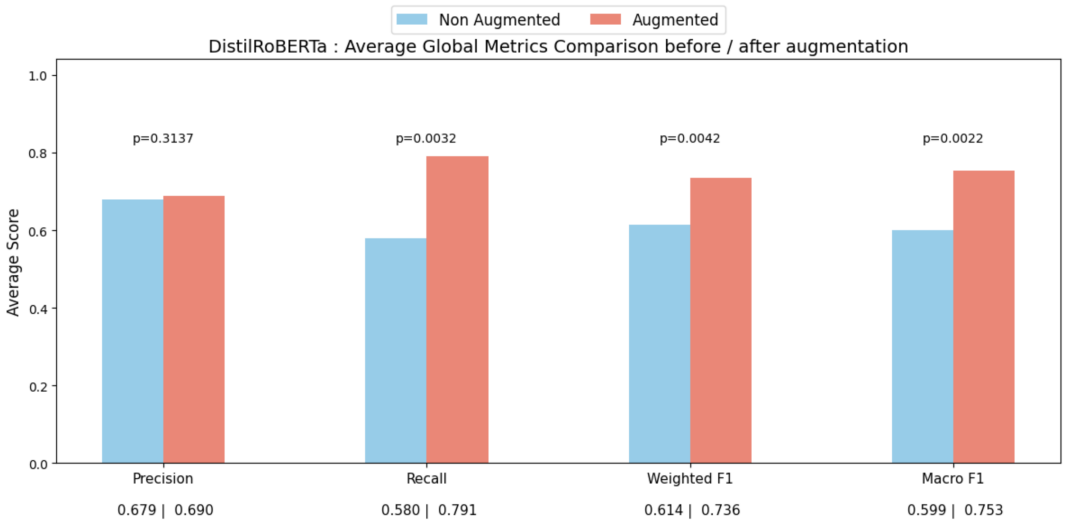


Figure 5.8: DistilRoBERTa: Global Metrics before vs after augmentation

Discussion

Our experimental results on the models fine-tuned using the augmented training data show considerable improvements over the models fine-tuned on the original dataset. The ROC-AUC metrics on all the models improve even further and beat the state-of-the-art across every permission category. The more noticeable improvement is across the PR-AUC metric where we report improvements of over 25 percent over the state-of-the-art as well as across all models trained on the original, unaugmented data. While analyzing the PR-AUC metric, we still notice that the scores for SETTINGS and TASKS label are lower than those for the other categories. We also analyze the global metrics and report significant improvements in recall score across all models, while the precision scores improve slightly or remain constant.

The improvement due to augmentation lies in the fact that by creating variations of existing data through paraphrasing, the model is exposed to a broader range of linguistic expressions, which improves its ability to learn more robust representations of language, enabling it to generalize better to unseen inputs.

Statistical Significance of Improvements

In order to evaluate the statistical significance of our improvements due to augmentation, we state the following null-hypothesis:

H_0 : There is no statistically significant difference in large language model performance (recall, F1) between experiments fine-tuned on the non-augmented dataset and those fine-tuned on the augmented dataset.

We perform the paired t-test on our performance data gathered before and after augmentation across the various experimental runs and report the p-values, as depicted in figures 5.5, 5.7, 5.6, 5.8. The results indicate p-values less than 0.005 for all models indicating highly statistically significant improvements across recall and f1-metrics. Thus, we reject the null-hypothesis in favor of the alternative hypothesis which states that there is consistent, statistically significant improvements

in recall and F1 metrics across all models. These results suggest that the augmentation strategy not only enhances model performance in controlled experiments but also has potential for real-world applications.

Thus, in this chapter, we report notable improvements across all metrics after fine-tuning on the augmented dataset. While all models trained on this augmented dataset demonstrate statistically significant performance gains over the state-of-the-art metrics — it remains essential to address a critical question: do these models truly capture the underlying semantics of the data, or are they simply fine-tuned to the specific patterns of the training set? In other words, can these models generalize effectively to new, unseen data, or do they risk being overfitted?

Chapter 6

Generalizability of the Models

The true measure of a model's success lies not only in its performance on familiar data, but its ability to maintain robust performance while it faces new, unseen scenarios. It can very well be true that the model performs great for one dataset and then doesn't do well across novel datasets. This is a classic example of overfitting [40]. While the previous chapter highlighted the impressive discriminative power of our distilled models, this chapter shifts the focus to a critical evaluation: assessing how well these models generalize beyond the training data.

6.1 Alternative Evaluation Dataset (Permissions-250)

To study the generalizability of the models and to further ensure a rigorous evaluation of their real-world applicability, we manually annotated an additional dataset and tested the models on this dataset. This newly annotated dataset serves as a robust benchmark, enabling us to determine whether the observed high performance stems from genuine learning of the underlying semantics or if it is merely an artifact of overfitting to the training set.

The dataset consists of 250 app description sentences where each sentence contains at least one positive instance of some permission. We compiled the dataset from various sources such as AC-Net, play store and also, synthesis using generative AI. The annotation task was performed by two annotators, both graduate students, with several years of combined industry experience in software engineering and artificial intelligence. After the annotation task, we calculated the inter-annotator agreement using the Cohen's Kappa metric [41]. We treat the problem as a multi-label binary classification task and thus, report the cohen's kappa score for each permission label in Table 6.1. We observe an average inter-rate agreement score of 82 percent, with significant agreement for labels such as SMS, microphone, camera, calendar and call log. This strong consensus may be due to the fact that these particular permission types (e.g., SMS, microphone, camera, calendar, and call log) are more easily recognizable, leading raters to align closely on their assessments.

On the other hand, we observe the lowest agreement score for the Settings label. This could be because the settings permission label may not have an obvious keyword indicating its usage - the annotators would have to probe into the sentence in greater detail to assess whether the application somehow would access the WRITE_SETTINGS permission. The conflicts between the annotators were resolved during a meeting where the annotators came up with a common definition for each label and then discussed and amended the labels as per the agreed upon definition.

Table 6.1: Cohen's Kappa for Each Category

Category	Cohen's Kappa
STORAGE	0.64
CONTACTS	0.62
LOCATION	0.81
CAMERA	0.94
MICROPHONE	0.88
SMS	1.00
CALL_LOG	0.91
PHONE	0.64
CALENDAR	0.92
SETTINGS	0.54
TASKS	0.64

After resolving the conflicts, we evaluate the two variants of our fine-tuned models - the first trained on the original AC-Net and the other trained on the augmented version of the original AC-Net. The results have been tabulated below:

Bert-Base

Table 6.2: Performance Metrics for Bert-Base Models evaluated on the Permissions-250-eval dataset

Model	Precision	Recall	Weighted F1	Macro F1
Bert-Base (Original-AC-Net)	**0.8782**	0.6601	0.7368	0.7247
Bert-Base (Original-AC-Net-Augmented)	0.8549	**0.7525**	**0.7896**	**0.7817**

DistilBERT

Table 6.3: Performance Metrics for DistilBERT Models evaluated on the Permissions-250-eval dataset

Model	Precision	Recall	Weighted F1	Macro F1
DistilBERT (Original-AC-Net)	**0.8808**	0.6238	0.7114	0.6971
DistilBERT (Original-AC-Net-Augmented)	0.8637	**0.7525**	**0.7966**	**0.7909**

DistilRoBERTA

Table 6.4: Performance Metrics for DistilRoberta Models evaluated on the Permissions-250-eval dataset

Model	Precision	Recall	Weighted F1	Macro F1
DistilRoberta (Original-AC-Net)	0.8530	0.6106	0.6928	0.6767
DistilRoberta (Original-AC-Net-Augmented)	**0.8592**	**0.7822**	**0.8065**	**0.7994**

RoBERTA

Table 6.5: Performance Metrics for Roberta Model evaluated on the Permissions-250-eval dataset

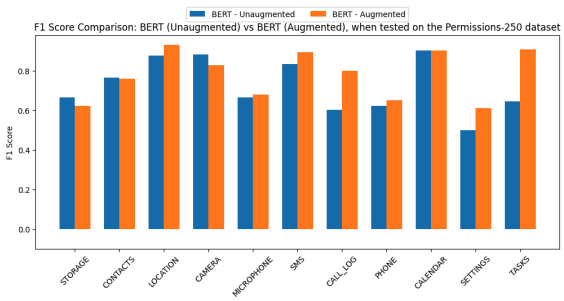
Model	Precision	Recall	Weighted F1	Macro F1
Roberta (Original-AC-Net)	**0.8606**	0.7162	0.7642	0.7533
Roberta (Original-AC-Net-Augmented)	0.8394	**0.7525**	**0.7838**	**0.7766**

Discussion

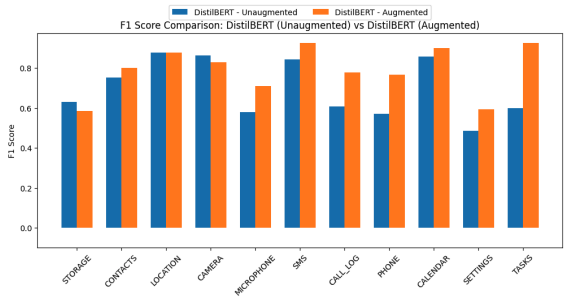
From the tables above, we can observe that all models demonstrate performance improvements after being trained on the augmented dataset. Particular gains are observed for the recall metric for all models, which is in line with our findings reported in the previous chapters. We notice a slight

dip in precision for 3 out of the 4 models indicating the possibility of slight over-fitting due to the training on paraphrases and also, because of the 60:40 skew towards the positive modality after augmentation.

RQ3: How well do LLMs fine-tuned on one permissions dataset generalize across other datasets? Based on the inference performance on the permissions-250 dataset, the reported statistics indicate that the model fine-tuned on the augmented training dataset generalizes better than the model trained on the original imbalanced data. The F1 performance gain due to augmentation is more pronounced for the distilled variants than the base variants, as depicted in the figures 6.1 and 6.2. This could be because the base models, being more powerful than the distilled variants already have a high capacity and might be less sensitive to variations in training data because they can already capture a wide range of patterns from the original data. On the other hand, the distilled models - having fewer parameters and a compressed representation, might be more sensitive to variations in the training data and thus, data augmentation can provide additional variability and robustness, which helps these smaller models generalize better and provides a more distinguished effect when it comes to improvements versus the base model.

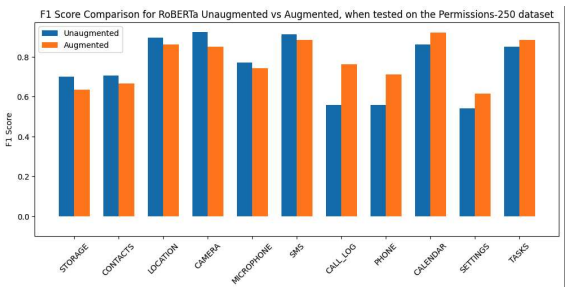


(a) BERT: F1 Score Comparison for Non-augmented vs Augmented variants, tested on the Permissions-250 dataset.

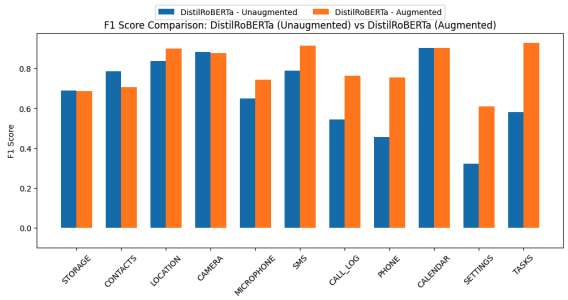


(b) DistilBERT: F1 Score Comparison for Non-augmented vs Augmented variants, tested on the Permissions-250 dataset.

Figure 6.1: Comparison of F1 performance for Non-augmented vs Augmented variants of BERT and DistilBERT, when tested on the Permissions-250 dataset. The gains due to augmentation for the Distilled variant are more noticeable than the gain for the base model



(a) RoBERTa: F1 Score Comparison for Non-augmented vs Augmented variants, when tested on the Permissions-250 dataset.



(b) DistilRoBERTa: F1 Score Comparison for Non-augmented vs Augmented variants, when tested on the Permissions-250 dataset.

Figure 6.2: Comparison of F1 Score performance for Non-augmented vs Augmented variants of RoBERTa and DistilRoBERTa, when tested on the Permissions-250 dataset.

Overall, the final reported F1 metrics are respectable and demonstrate the promise of applying our techniques to real world data. An important takeaway from this chapter and the one before would be the importance of balanced datasets in order to ensure greater LLM performance.

Chapter 7

Limitations and Future Work

Our experiments show the promise of using LLMs to extract permissions from application descriptions more accurately than ever before. However, it is important to point out several possible limitations with our approach and report the possible threats to validity.

7.1 Threats to Validity

7.1.1 External Validity

- Our generalizability study includes a manually annotated dataset of only 250 sentences. While the evaluation on this dataset still yields valuable insights on the general performance of our models, it can be argued that such a small sample size may not be enough to make conclusions about the general applicability of our models. To mitigate this, future work can involve annotating or synthesizing an even larger sample of data and evaluating our models on these datasets.

- The section on sparse data fine-tuning reports significant gains in the performance of LLMs in predicting permission sentences after being trained on the paraphrased dataset versus being trained on only 250 samples. However, it is also important to note that the model needs to be able to separate non-permission sentences in order to be more useful in real world application. Future work can involve augmentation of non-permission sentences and observing how the model performs wholistically across permission and non-permission sentences.

- It is also important to note that our approach involves supervised learning, the quality of which heavily depends on the correctness of the annotations made by humans. However, there is always a chance of some human error during the annotation process. This can result in the models learning incorrect patterns and thus, affecting the ability of the models to perform well in real world scenarios.

7.1.2 Internal Validity

- As mentioned before, the quality of the annotations is crucial for fine-tuning. However, if the manual annotations are subjective or inconsistent (due to unclear guidelines or annotator variability), the measured performance might still report annotation noise rather than the true capability of the model. To mitigate this, while annotating our alternative 250-sentence evaluation dataset, the two annotators agreed upon guidelines as to what properties would be required for any given sentence to fall under a certain permission category. However, the AC-Net dataset reports no such annotator agreement or inter-annotator agreement scores, and thus, can raise concerns about the quality of the annotations.

7.1.3 Construct Validity

- We report micro-f1 scores for many of our experiments. However, it is important to note that in a highly imbalanced dataset, the majority class can dominate these f1 scores, causing the metric to largely reflect performance on that majority class while masking poor performance on the minority classes. This can lead to an overly optimistic assessment of the model's overall performance. To account for this imbalance, we also report macro and weighted f1 scores.

- We use the default classification or decision threshold of 0.5 for our experiments, i.e. if the probabilty for a given input belonging to any given label / class during classification is greater than 0.5, we treat this instance as belonging to the positive class (true or 1), else, we classify it as a negative class (false). However, it can be that a fixed threshold introduces systematic bias in our model's predictions, particularly if the optimal threshold differs from 0.5. To combat with this possibility, we also report the ROC-AUC scores as well as the PR-AUC scores, both of which provide a threshold independent measure of the model's ability to discriminate between classes as well the trade-off between precison and recall.

7.1.4 Statistical Conclusion Validity

- It is important to note that our experiments were conducted for a limited number of times (ranging from 3 to 10, depending on the model and training cost). It can be argued that such a limited number of samples may not be sufficient to form statistically justified conclusions. Future work can involve training and evaluating our models across an even large number of experiments.

Furthermore, an expansion of this study could include the possibility of exploring unsupervised learning for our task. Similar to how GPT was originally created, we could pre-train our LLM on a large corpus of application descriptions and then perform selective fine-tuning on our permission inference task to possibly create an LLM specifically adpated to our problem domain. This could lead to significant performance boost, although at the expense of a significant amount of compute and other resources. Also, this study only focuses on the permission inference task. A natural extension of this would be to use our fine-tuned permission inference models to make its permission predictions based on the app descriptions, while concurrently gathering a list of actual permissions collected by the applications from the app manifest files and then comparing between the two lists to search for possible discrepancies. In other words, this would be the end to end implementation of the permission fidelity analysis.

Chapter 8

Conclusion

In this study, we demonstrate the viability of fine-tuned LLMs for the permission inference task. We fine-tune and primarily assess our model on the AC-Net dataset and show that LLMs can achieve state-of-the-art performance, despite the high imbalance in the training data. Our initial experiments show that the fine-tuned BART-large model performs the best compared to other models such as the base BERT, RoBERTa and their distilled variants. We also conduct experiments where we evaluate the performance of some zero-shot models and show that the trillion parameter GPT reports performance metrics very comparable to the other fine-tuned models, whereas the several hundred million parameter sized Bart zero-shot model performs very poorly. This demonstrates the superior out-of-the-box capabilities of high-end LLMs like GPT which contains such a large number of hyperparameters, enabling them to capture more complex language relationships and handle nuanced prompts. [5]

Furthermore, with training data augmentation, we report significant performance improvement over the state-of-the-art and also, show that the augmented models generalize better than the base models trained on the non-augmented data. This highlights the importance of data augmentation and overall dataset quality in determining the performance of the fine-tuned LLMs. Our experiments on paraphrasing also establishes its effectiveness in the data augmentation task. We report performance gains of over 60 percent across several global metrics when comparing the models trained on the paraphrase augmented data sample versus the models trained on a very limited number of annotated samples. This performance gain was achieved at the cost of no additional annotations, indicating that paraphrasing can introduce enough diverse linguistic patterns for the models to learn and improve upon.

[5] All our fine-tuned models are publicly available on Huggingface: `https://huggingface.co/yuniktmr`

Our study also identifies the SETTINGS and TASKS permission categories as the groups where all models tend to struggle during the prediction task as reported in 8.1. Not only that, we report very low inter-annotator agreement for those two particular labels as well.

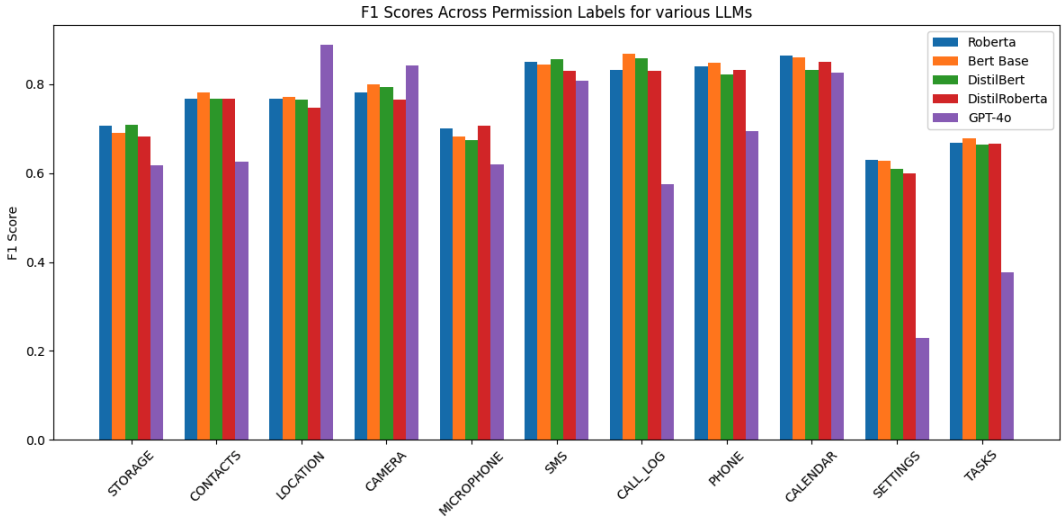


Figure 8.1: F1 scores across various labels for various LLMs (includes models fine-tuned on the augmented data and GPT in zero-shot mode). Performance for the SETTINGS and TASKS label is inferior across all models

These observations lead us to hypothesize that this performance dip maybe due to the vagueness of the sentences which belonging to those two categories. This opens up new avenues towards exploring more focused research questions targeting the permission inference at the fine granularity level of individual permission labels.

Bibliography

[1] Shuang Liu, Baiyang Zhao, Renjie Guo, Guozhu Meng, Fan Zhang, and Meishan Zhang. Have you been properly notified? automatic compliance analysis of privacy policy text with gdpr article 13. In *Proceedings of the Web Conference 2021*, pages 2154–2164, 2021.

[2] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel Reidenberg, N Cameron Russell, and Norman Sadeh. Maps: Scaling privacy compliance analysis to a million apps. *Proceedings on Privacy Enhancing Technologies*, 2019.

[3] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022, 2014.

[4] Hannah Kozlowska. The cambridge analytica scandal affected nearly 40 million more people than we thought. *Quartz*, page 2018, 2018.

[5] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–14, 2012.

[6] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. *arXiv preprint arXiv:1808.05096*, 2018.

[7] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. The privacy policy landscape after the gdpr. *arXiv preprint arXiv:1809.08396*, 2018.

[8] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *Isjlp*, 4:543, 2008.

[9] Ryan Amos, Gunes Acar, Eli Lucherini, Mihir Kshirsagar, Arvind Narayanan, and Jonathan Mayer. Privacy policies over time: Curation and analysis of a million-document dataset. In *Proceedings of the Web Conference 2021*, pages 2165–2176, 2021.

[10] Chenhao Tang, Zhengliang Liu, Chong Ma, Zihao Wu, Yiwei Li, Wei Liu, Dajiang Zhu, Quanzheng Li, Xiang Li, Tianming Liu, et al. Policygpt: Automated analysis of privacy policies with large language models. *arXiv preprint arXiv:2309.10238*, 2023.

[11] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. {WHYPER}: Towards automating risk assessment of mobile applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 527–542, 2013.

[12] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430, 2003.

[13] William Enck, Damien Octeau, Patrick D McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, 2011.

[14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[15] 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, 2014.

[16] 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.

[17] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[19] Adrienne Porter Felt, Serge Egelman, and David Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 33–44, 2012.

[20] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori. Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 241–255, 2015.

[21] Sarang Narkhede. Understanding auc-roc curve. *Towards data science*, 26(1):220–227, 2018.

[22] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[23] Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, and Patrick Zschech. Generative ai. *Business & Information Systems Engineering*, 66(1):111–126, 2024.

[24] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[25] Mike Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[26] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[27] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[28] Alec Radford. Improving language understanding by generative pre-training. 2018.

[29] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[30] Raisa Islam and Owana Marzia Moushi. Gpt-4o: The cutting-edge advancement in multimodal llm. *Authorea Preprints*, 2024.

[31] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *2016 AAAI Fall Symposium Series*, 2016.

[32] Judit M Wulcan, Kevin L Jacques, Mary Ann Lee, Samantha L Kovacs, Nicole Dausend, Lauren E Prince, Jonatan Wulcan, Sina Marsilio, and Stefan M Keller. Classification performance and reproducibility of gpt-4 omni for information extraction from veterinary electronic health records. *arXiv preprint arXiv:2409.13727*, 2024.

[33] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1. Minneapolis, Minnesota, 2019.

[34] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019.

[35] V Sanh. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[36] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

[37] Georgios Rizos, Konstantin Hemker, and Björn Schuller. Augment to prevent: short-text data augmentation in deep learning for hate-speech classification. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 991–1000, 2019.

[38] Tomoki Hayashi, Shinji Watanabe, Yu Zhang, Tomoki Toda, Takaaki Hori, Ramon Astudillo, and Kazuya Takeda. Back-translation-style data augmentation for end-to-end asr. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 426–433. IEEE, 2018.

[39] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[40] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[41] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[42] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[43] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[44] Rebecca Balebako and Lorrie Cranor. Improving app privacy: Nudging app developers to protect user privacy. *IEEE Security & Privacy*, 12(4):55–58, 2014.

[45] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. In *Workshop on Usable Security*, pages 1–10. Citeseer, 2014.

[46] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding challenges for developers to create accurate privacy nutrition labels. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–24, 2022.

[47] Florian Schaub, Rebecca Balebako, Adam L Durity, and Lorrie Faith Cranor. A design space for effective privacy notices. In *Eleventh symposium on usable privacy and security (SOUPS 2015)*, pages 1–17, 2015.

[48] Ruoxi Sun and Minhui Xue. Quality assessment of online automated privacy policy generators: an empirical study. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, pages 270–275, 2020.

[49] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. Privacyflash pro: Automating privacy policy generation for mobile apps. In *NDSS*, volume 2, page 4, 2021.

[50] Shidong Pan, Dawen Zhang, Mark Staples, Zhenchang Xing, Jieshan Chen, Xiwei Xu, and Thong Hoang. Is it a trap? a large-scale empirical study and comprehensive assessment of online automated privacy policy generators for mobile apps. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5681–5698, 2024.

Appendix A

Large Scale Privacy Policy Analysis

We also conduct another stream of research where we develop a tool to automatically scrape over 10000 mHealth apps from the play store and then perform similarity analysis on these privacy policy texts to reveal interesting insights of privacy policy reuse. Our approach and findings have been described in detail in the following sections.

A.1 Data Collection

For our project, we use a previously compiled dataset (csv format) of 10688 mobile health apps as the data source. This dataset contains the package name for the mobile health app, which is what we reference in order to download the privacy policy text. The structure of this dataset is shown in Figure A.1

```
subdirectory_3.zip,homeworkout.homeworkouts.noequipment.base.apk
subdirectory_3.zip,de.biotronik.PatientApp.base.apk
subdirectory_3.zip,com.umpshealth.base.apk
subdirectory_3.zip,com.technogym.mywellness.fitnessmobil.base.apk
subdirectory_3.zip,com.terraillon.wellnesscoach.base.apk
subdirectory_3.zip,com.trainerize.semperstronger.base.apk
subdirectory_3.zip,com.spruce.messenger.base.apk
subdirectory_3.zip,com.technogym.mywellness.yourspace.base.apk
subdirectory_3.zip,com.smsrobot.period.base.apk
subdirectory_3.zip,com.trainerize.maximpact317.base.apk
subdirectory_3.zip,com.takeonestep.android.base.apk
subdirectory_3.zip,fr.interiale.ITE.base.apk
subdirectory_3.zip,eu.findair.base.apk
subdirectory_3.zip,com.wsl.noom.base.apk
subdirectory_3.zip,fitness.online.app.base.apk
subdirectory_3.zip,com.trainerize.freedomfitnessgym.base.apk
subdirectory_3.zip,com.widex.arc.base.apk
subdirectory_3.zip,com.sonlevu.ecgo.base.apk
subdirectory_3.zip,com.stepsappgmbh.stepsapp.base.apk
subdirectory_3.zip,com.zimbiosis.app2.base.apk
subdirectory_3.zip,com.sunrisemedical.intelligentfleet.base.apk
subdirectory_3.zip,com.srp.spf.base.apk
subdirectory_3.zip,com.trainerize.prooffitnessholdings.base.apk
```

Figure A.1: Structure of the csv file containing app package names

A.2 Policy Scraping

The next step in our pipeline is the gathering of privacy policy texts. Manually downloading the policy texts for the 10k+ apps in our dataset is not feasible. Thus, we automate this method using a combination of methods. Our approach can be described as two-prong where we use a combination of methods to enrich our privacy policy repository as much as we can.

A.2.1 Selenium Scraper

This is our preliminary, home-grown approach to extract policy text for mHealth apps based on the package names. We use a combination of Selenium for web automation as well as the requests library for fetching web content.

Selenium Initialization

Selenium is a test automation tool that can be used to perform automated navigation through web pages, simulating user interactions such as clicking links and switching tabs. The Selenium WebDriver is the API that helps us to automate the web interactions. We make use of ChromeDriver so as to leverage Selenium's aforesaid web automation capabilities using Google Chrome.

Web Navigation and Interaction

- For each app's package name, the script navigates to the app's Google Play Store page. We construct the app url using the template:

  `https://play.google.com/store/apps/details?id=[package_name]`

  So, for the application named 'Strava', the formatted url will be

  `https://play.google.com/store/apps/details?id=com.strava`

- The selenium webdriver then automatically opens the formatted link on a web browser. The opened link is a play store page for the given app as show in Figure A.2, provided that it exists.

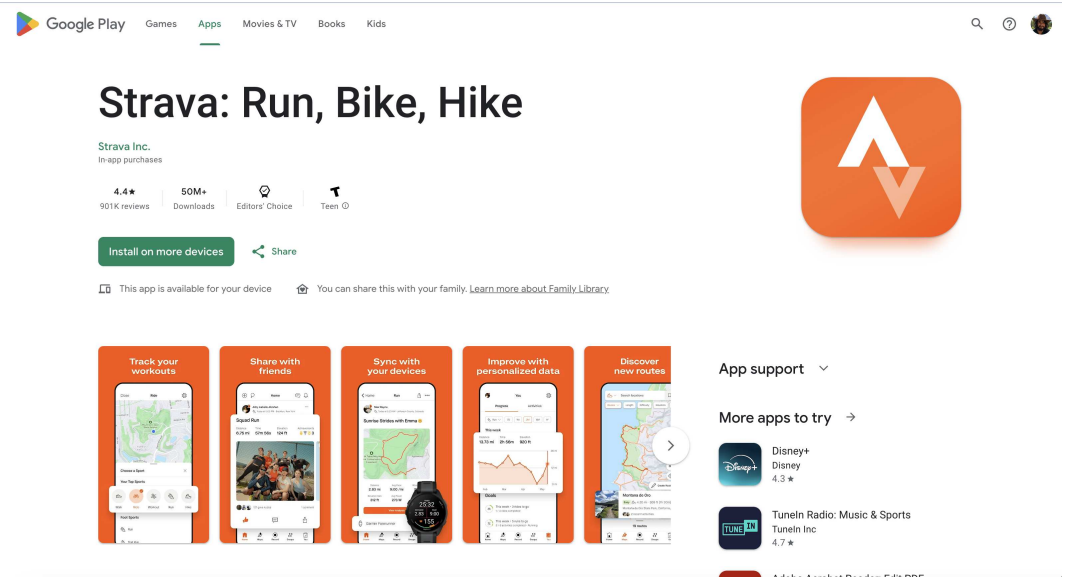


Figure A.2: Play Store Page for Strava, opened by Selenium

- After this, Selenium attempts to click the "See details" within the Data Safety section to reveal the data safety details. To achieve this, Selenium goes through the DOM-tree and scans for either of the following:
  - HTML element with the link text `'See details'`

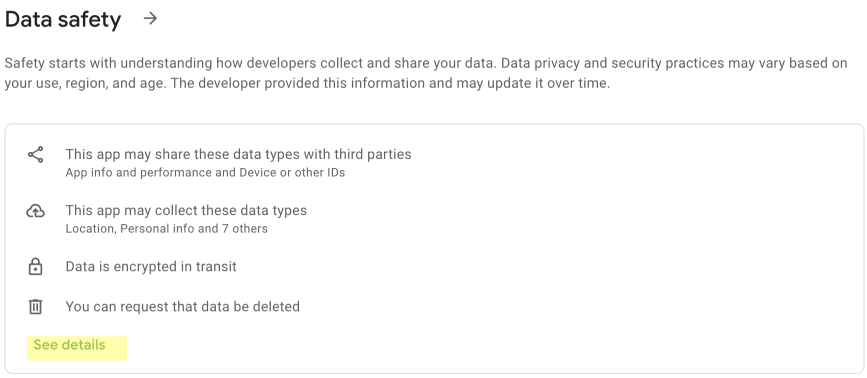
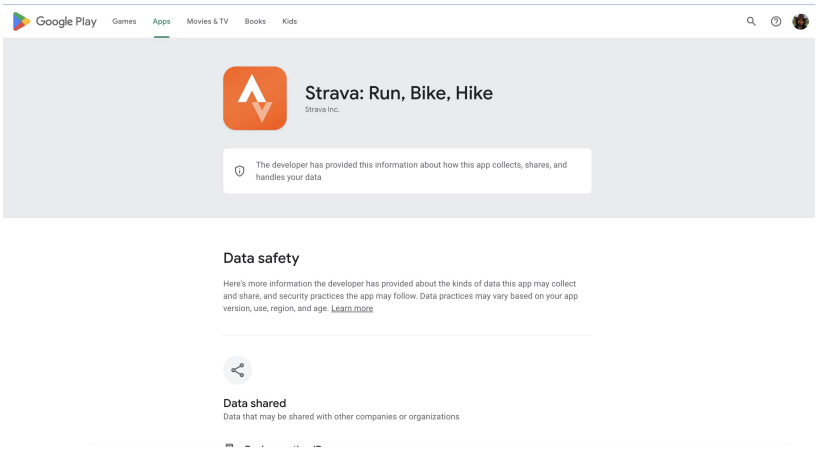


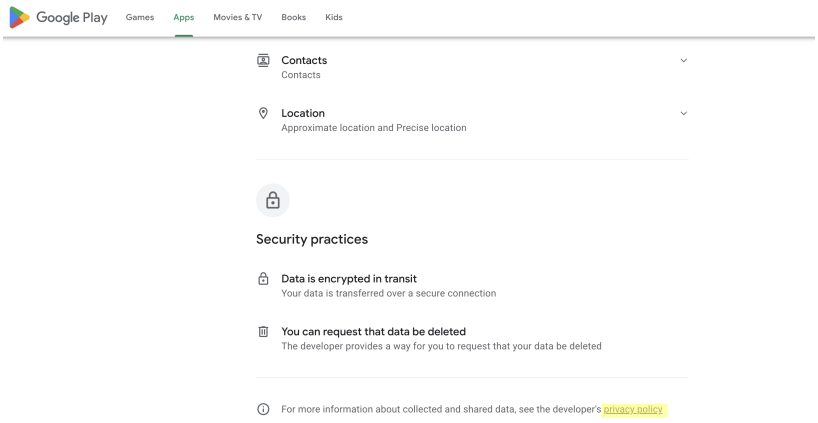
Figure A.3: Selenium searches for this "See Details" element and clicks on it.

  - HTML element with `aria-label`[6]`="See more on data safety"`

- Once the clickable element is found, Selenium clicks on it to navigate to the next page.
- This will lead to the application's Data Safety page where Selenium will search for an element with the link text `"privacy policy"`



(a) Strava: Data Safety Page



(b) Privacy Policy link within the Data Safety Page

Figure A.4: Data Safety Page accessed after clicking on the See Details element (a) The Data Safety page, which is available for most applications. (b) The link to the Privacy policy, which can be accessed after scrolling further down the Data Safety page

[6]The aria-label attribute is part of the Accessible Rich Internet Applications (ARIA) specifications. It provides additional information to assistive technologies (like screen readers), improving web accessibility for users with disabilities.

- Selenium will click on this privacy policy link and then open it in a new Chrome tab.
- The page accessed after clicking the privacy policy link generally contains the actual privacy policy. However, during test runs of the Selenium Webdriver, we observed that this link leads to an intermediary page that requires the user to select the language for viewing the privacy policy. Since we are looking into English texts only, Selenium will automatically click on links named 'Eng' or 'EN', which ultimately leads to the page with the privacy policy text.

Scraping

We then fetch the final URL of the privacy policy page and download the page content using the python `requests` library. This HTML content is parsed with the `BeautifulSoup` library to extract the text, which is then saved as a .txt file in the corresponding directory.

Using this approach, we were able to scrape around 6000 privacy policy texts. The remaining 4000 policy texts ran into various errors such as:

- The play store link to the application was no longer valid, suggesting that the app was no longer available on play store or the app underwent renaming
- The applications contained nested links that required the user to click through more pages (beyond the language screen that the selenium scraper already handled)
- The links exceeded the 30 second timeout limit set for Selenium so as to prevent stalling while trying to scrape a page.

During further exploration of ways to improve the scraping results, we explored other tools such as the Google Play scraper.

A.2.2 Google Play Scraper

The Google Play Scraper is a Python package that provides APIs to easily crawl the Google Play Store without any external dependencies. Using Google Play Scraper, we were able to scrape

potential policy documents for around 8642 apps. However, more than 1,063 policy links also encountered various errors with this approach. We were able to scrape around 113 of these erroneous links using our Selenium scraper, to form an initial corpus of around 8755 policy texts.

A.3 Language Filtering

To filter out non-english policy texts, we use the langdetect tool. We removed 1314 invalid texts from the original sample with a precision of around 91 percent. The detected false positives were still non-policy texts which we remove in the next step.

A.4 Non-Policy Filtering

In order to filter out non-policy texts such as random homepages, we train a distilBERT model on 1000 GPT generated policy texts and another 1000 GPT generated non-policy texts. The synthetic dataset was created using ChatGPT and the prompts included examples of policy texts and non-policy texts as instructions to the model. Using this approach, we removed 789 non-policy docs with a precision of around 98.74 percent. The resulting dataset contained around 6554 privacy policy documents.

A.5 Data Analysis

During the inspection of our refined dataset from the steps described previously, we noticed several policy texts that shared common package names. Upon inspecting the contents of various policies under these umbrellas, we noticed that the privacy policies were almost identical. This led us to conduct a similarity analysis across our corpus to gain insights on the similarity trends across the policy texts.

A.5.1 Similarity Analysis

In order to compute the similarity between various texts, we first need to convert the text into machine-readable forms. This involves converting the natural language text into machine-readable vector representations.

Vectorization

We generate the vectors using the Term Frequency - Inverse Document Frequency (TF-IDF) algorithm [7], which is a very popular and standard algorithm for generating embeddings. It can be defined as the relative frequency of words in a specific document compared to the inverse proportion of that word throughout the corpus of the document. Terms with a higher TF-IDF score are treated as more relevant or important [42]. The following are some term definitions:

- **Term frequency (TF)**: The normalized count of given words in a document. Thus, the weight or Normalized Term Frequency of a term 't' in a given document 'd' is given by:

$$\text{tf}(t, d) = \frac{\text{count of } t \text{ in } d}{\text{number of words in } d}$$

- **Document frequency**: The number of documents in the corpus that contain the term at least once.

$$\text{df(t)} = \text{occurrence of t in documents}$$

- **Inverse Document Frequency (IDF)**: The IDF of a term is calculated to determine how important or rare a term is across the entire document corpus. It is inversely proportional to the frequency of the term document, which means that the more documents contain the term,

[7] Implemented in python using the TfidfVectorizer() method in the sklearn module.

the lower the IDF value as the term is less unique. The IDF is given by:

$$\text{idf}(t) = \log\left(\frac{N}{\text{df}(t)}\right) = \log\left(\frac{N}{N(t)}\right)$$

where:

- N is the total number of documents in the corpus.
- $\text{df}(t)$ or $N(t)$ is the number of documents containing the term t.

- **TF-IDF**: Finally, the TF-IDF metric is computed using the formula:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

where:

- $\text{tf}(t, d) = \frac{\text{count of } t \text{ in } d}{\text{number of words in } d}$ is the term frequency of t in document d.
- $\text{idf}(t) = \log\left(\frac{N}{\text{df}(t)}\right)$ is the inverse document frequency, where N is the total number of documents, and $\text{df}(t)$ is the document frequency of term t.

Terms that are common in a single document or a small number of documents in a corpus tend to have a higher TF-IDF score than very common words such as articles or prepositions and thus, are considered more important or relevant. This weighted approach helps filter out stopwords and also, helps tailor the weight depending on the domain that the input corpus belongs to (Example: the word 'plaintiff' may have a higher TF-IDF score in legal literature, and low score in some geography literature.) It is important to note that the vectors generated with this approach are based on word-document occurrences and don't capture semantic relationships. It only considers the frequency of words in documents relative to a corpus, so "cat" and "kitten" would not be close in the TF-IDF space, even though they are semantically related.

Cosine Similarity

Cosine similarity is a widely used metric to compute the similarity (lexical or semantic) between two documents. As per this approach, the cosine value between any two documents' vector representations is calculated [43], which is then treated as the similarity score. Considering, A and B as the vector representation of the documents, the cosine similarity is given by the formula:

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

This formula gives a value between 0 to 1, where:

- 1 indicates identical orientation (perfect similarity),
- 0 indicates orthogonality (no similarity),

Note that, negative values between -1 and 0 are not feasible when using TF-IDF vectors because TF-IDF values are calculated based on term frequency (TF) and inverse document frequency (IDF), both of which are non-negative by nature.

A.5.2 Cosine Similarity Results using TF-IDF vectors

We converted the policy texts in our dataset to the TF-IDF vectors and then computed the pairwise similarity for each document pair. Our analysis revealed the presence of a substantial number of duplicate policy texts in our corpus. We performed agglomerative clustering on the pairwise similarity scores to group documents with high similarity scores together in one cluster. Our analysis revealed several key clusters. The top 10 clusters ranked on the number of member elements is given in Table A.1

Table A.1: Top 10 clusters based on agglomerative clustering of TF-IDF based cosine similarity scores.

Cluster Rank	Developer ID	Cluster Membership	Average Similarity Score
1	com.trainerize	2151	1.0
2	N/A	110	0.86
3	N/A	97	0.8766
4	com.phorest	92	1.0
5	com.technogym	83	0.9998
6	com.fitnessmobileapps	79	1.0
7	N/A	55	0.8594
8	com.cofox	45	0.9975
9	N/A	45	0.8707
10	N/A	41	0.9770

From the analysis presented in table Table A.1, we can observe that:

- Developers with distinct IDs like 'com.trainerize', 'com.phorest', 'com.technogym', etc have clusters with a very high number of similar policy texts. These texts are almost identical to each other with cosine similarity scores equal 1.0 or very close to 1.0. This shows a pattern where the same developer, responsible for building multiple applications, share identical or nearly identical privacy policies across their suite of applications.

- We also see several clusters with no distinct developer IDs i.e the N/A values. These N/A Developer ID values refer to clusters where the member privacy policies belong to a wide set of Developer IDs and thus, cannot be linked to a particular developer ID. We can observe quite a few of these heterogeneous clusters in the table. This shows another pattern where even different developers tend to share privacy policy texts. We hypothesize that this could

be because of the usage of similar open-source or publicly available policy generation templates, Online Automated Privacy Policy Generators (APPGs) or outsourced legal services. The similarity scores for these heterogeneous clusters are lesser than that of the homogeneous clusters. This observation might be attributed to the fact that different developers, while sharing the majority of the policy template bodies, tend to tweak different developer or business-specific details within the policy texts, as outlined in Figure A.5

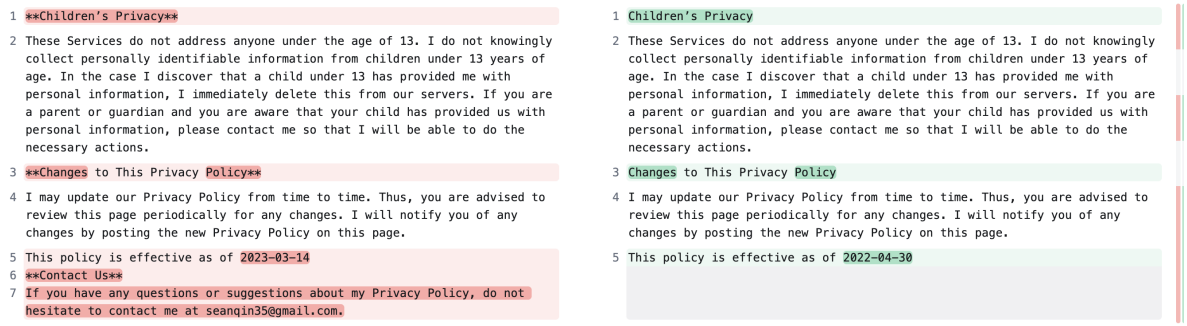


Figure A.5: Minor differences between the policy texts of two applications by different developers. (Cosine similarity = 0.9727)

Upon manual inspection of some of these identical privacy policies (belonging to different developer IDs), we verified the existence of embedded links to the page/tool that was used to generate these policies as shown in Figure A.6

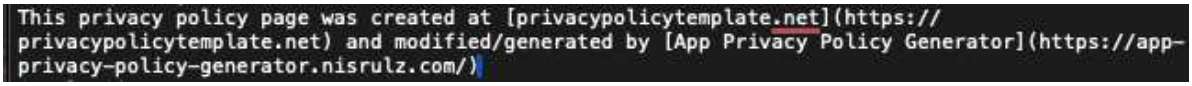


Figure A.6: Link to a policy generator

This finding corroborates our hypothesis of the involvement of common / open-source 3rd party policy template generators which results in multiple developers responsible for different applications possibly sharing the same privacy policy. This may be attributed to the fact that privacy policy development is a complex process requiring knowledge about both the application's engineering as well as legal aspect. While large companies have the resources and legal expertise

to craft highly detailed and well-reviewed policy texts, individual developers do not have access to the resources required to craft such high quality privacy policies [44] [45] [46] [47]. Therefore, the developers tend to use these APPGs to automate and simplify the process of creating policy texts. The developers are increasingly using APPGs to generate the policy texts, but they are not aware about several hidden issues prevalent in the APPGs [48] [49]. As a result, this can lead to various compliance issues, which raises concerns about the effectiveness of the privacy policy texts in informing users about their rights, the types of personal information collected, and how their data is processed or shared. These concerns highlight the potential risks associated with relying on automated policy generation tools, as such tools may not fully account for the legal nuances and specific regulatory requirements necessary to ensure user transparency and data protection compliance [50]. Consequently, policies generated by APPGs may fall short in meeting both legal standards and user expectations for clear, comprehensive privacy disclosures.

Appendix B

Miscellaneous Tables

GPT-4o-Metrics

Table B.1: GPT-4o-mini Performance Metrics at Various Temperature Values

Temperature	Macro Stats			Micro Stats			Weighted Stats		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
0.1	0.7909	0.3888	0.4952	0.8424	0.3994	0.5419	0.8084	0.3994	0.5130
0.4	0.8693	0.3902	0.4965	0.8263	0.3932	0.5328	0.8561	0.3932	0.5055
0.7	0.8463	0.3886	0.4963	0.8333	0.4000	0.5405	0.8397	0.4000	0.5146
1.0	0.8396	0.3855	0.4972	0.8353	0.4046	0.5451	0.8319	0.4046	0.5188
1.5	0.8267	0.4128	0.5119	0.8229	0.4103	0.5475	0.8211	0.4103	0.5166

Table B.2: GPT 4o Performance Metrics at Various Temperature Values

Temperature	Macro Stats			Micro Stats			Weighted Stats		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
0.1	0.8246	0.6119	0.6847	0.8224	0.6068	0.6984	0.8180	0.6068	0.6815
0.4	0.7921	0.6147	0.6738	0.7978	0.6086	0.6904	0.7996	0.6086	0.6759
0.7	0.8141	0.6030	0.6758	0.8164	0.5954	0.6886	0.8146	0.5954	0.6732
1.0	0.8019	0.6163	0.6795	0.8030	0.6057	0.6906	0.8001	0.6057	0.6742
1.5	0.8128	0.6091	0.6827	0.8175	0.6160	0.7026	0.8060	0.6160	0.6866

RoBERTa-Metrics

Table B.3: RoBERTa Overall Metrics (Non-Augmented)

Metric	Mean	Median
Accuracy	0.8493	0.8481
Precision	0.6543	0.6675
Recall	0.6188	0.6350
Weighted F1	0.6274	0.6342
Macro F1	0.6134	0.6261

Table B.4: RoBERTa Performance Metrics for Different Categories (Non-augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6336	0.6158	0.5877	0.5850	0.6080	0.6000	0.9419	0.9455	0.6478	0.6541
CONTACTS	0.7021	0.7018	0.6586	0.6620	0.6790	0.6939	0.9622	0.9632	0.7275	0.7353
LOCATION	0.6505	0.6476	0.7559	0.7573	0.6987	0.6834	0.9735	0.9760	0.7613	0.7716
CAMERA	0.7052	0.7206	0.7112	0.7042	0.7060	0.7123	0.9742	0.9748	0.7235	0.7063
MICROPHONE	0.5797	0.5625	0.4997	0.5000	0.5350	0.5373	0.9698	0.9672	0.5726	0.5745
SMS	0.7428	0.7160	0.7282	0.7342	0.7332	0.7451	0.9885	0.9912	0.7974	0.8388
CALL_LOG	0.7359	0.8235	0.3815	0.4000	0.4978	0.5238	0.9920	0.9913	0.6368	0.6409
PHONE	0.5619	0.5283	0.5776	0.6341	0.5485	0.5825	0.9815	0.9809	0.5691	0.5610
CALENDAR	0.6860	0.6667	0.8418	0.8298	0.7541	0.7368	0.9845	0.9960	0.7809	0.7889
SETTINGS	0.5323	0.5479	0.4831	0.4706	0.5013	0.5000	0.9494	0.9500	0.5003	0.4977
TASKS	0.6686	0.6500	0.3932	0.4091	0.4861	0.5155	0.9330	0.9319	0.5370	0.5387

Table B.5: RoBERTa Overall Metrics (Augmented)

Metric	Mean	Median
Accuracy	0.8775	0.8767
Precision	0.7087	0.7057
Recall	0.8011	0.7971
Weighted F1	0.7505	0.7479
Macro F1	0.7646	0.7640

Table B.6: RoBERTa Performance Metrics for Different Categories (Augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6792	0.6679	0.7396	0.7268	0.7068	0.7100	0.9667	0.9672	0.7536	0.7645
CONTACTS	0.7274	0.7188	0.8149	0.8273	0.7679	0.7649	0.9905	0.9908	0.8277	0.8329
LOCATION	0.7168	0.7213	0.8284	0.8224	0.7684	0.7686	0.9894	0.9888	0.8401	0.8473
CAMERA	0.7157	0.7083	0.8637	0.8710	0.7818	0.7674	0.9939	0.9945	0.8327	0.8525
MICROPHONE	0.6475	0.6271	0.7655	0.7838	0.7007	0.6988	0.9902	0.9891	0.7480	0.7681
SMS	0.7869	0.7941	0.9282	0.9219	0.8509	0.8553	0.9981	0.9978	0.8934	0.8697
CALL_LOG	0.7385	0.7407	0.9558	0.9630	0.8314	0.8163	0.9995	0.9995	0.9289	0.9430
PHONE	0.7998	0.8043	0.8867	0.8864	0.8404	0.8571	0.9983	0.9987	0.8572	0.8818
CALENDAR	0.7980	0.7500	0.9471	0.9455	0.8640	0.8352	0.9989	0.9988	0.8957	0.8952
SETTINGS	0.6130	0.6100	0.6529	0.6437	0.6298	0.6250	0.9804	0.9796	0.6685	0.6733
TASKS	0.6481	0.6667	0.6969	0.6818	0.6681	0.6517	0.9890	0.9881	0.7259	0.7211

BERT-Base Metrics

Table B.7: Bert-Base Overall Metrics (Non-Augmented)

Metric	Mean	Median
Accuracy	0.8558	0.8556
Precision	0.7165	0.7198
Recall	0.5580	0.5653
Weighted F1	0.6069	0.6199
Macro F1	0.5851	0.5997

Table B.8: Bert-Base Uncased Performance Metrics for Different Categories (Non-augmented)

Class	**Precision**		**Recall**		**F1 Score**		**ROC AUC**		**PR AUC**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6935	0.7154	0.5343	0.5410	0.6015	0.6041	0.9422	0.9430	0.6524	0.6550
CONTACTS	0.7509	0.7207	0.6276	0.6324	0.6816	0.6944	0.9613	0.9619	0.7273	0.7470
LOCATION	0.6942	0.7079	0.7164	0.7228	0.7023	0.6986	0.9682	0.9689	0.7424	0.7485
CAMERA	0.7568	0.7692	0.6689	0.7164	0.7057	0.7101	0.9570	0.9585	0.7135	0.6954
MICROPHONE	0.6653	0.6667	0.3958	0.4000	0.4865	0.4706	0.9562	0.9617	0.5353	0.5390
SMS	0.7665	0.8028	0.7011	0.6761	0.7317	0.7407	0.9849	0.9844	0.8046	0.8478
CALL_LOG	0.8714	0.8571	0.3113	0.2800	0.4554	0.4103	0.9921	0.9930	0.6323	0.6763
PHONE	0.5850	0.6154	0.5050	0.6098	0.5038	0.5495	0.9770	0.9700	0.5817	0.5758
CALENDAR	0.7449	0.7500	0.7960	0.8205	0.7673	0.7647	0.9831	0.9831	0.7966	0.8006
SETTINGS	0.6212	0.5616	0.3901	0.4353	0.4463	0.5103	0.9459	0.9489	0.4816	0.4744
TASKS	0.7985	0.8333	0.2337	0.2273	0.3540	0.3571	0.9034	0.9092	0.4608	0.4916

Table B.9: Bert-Base Overall Metrics (Augmented)

Metric	Mean	Median
Accuracy	0.8808	0.8822
Precision	0.7305	0.7355
Recall	0.7750	0.7655
Weighted F1	0.7502	0.7526
Macro F1	0.7686	0.7715

Table B.10: Bert-Base Uncased Performance Metrics for Different Categories (Augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6863	0.6979	0.6977	0.6900	0.6907	0.6827	0.9669	0.9692	0.7369	0.7468
CONTACTS	0.7656	0.7667	0.8001	0.7931	0.7821	0.7986	0.9902	0.9912	0.8393	0.8446
LOCATION	0.7365	0.7381	0.8100	0.8158	0.7711	0.7745	0.9899	0.9902	0.8266	0.8370
CAMERA	0.7384	0.7568	0.8744	0.8701	0.8001	0.8072	0.9954	0.9952	0.8271	0.8431
MICROPHONE	0.6592	0.6667	0.7123	0.7045	0.6822	0.6667	0.9904	0.9922	0.7402	0.7513
SMS	0.8074	0.8228	0.8895	0.9062	0.8452	0.8227	0.9979	0.9979	0.9023	0.8972
CALL_LOG	0.7979	0.7917	0.9584	0.9655	0.8686	0.8525	0.9995	0.9997	0.9433	0.9297
PHONE	0.8085	0.8667	0.8957	0.8864	0.8481	0.8764	0.9976	0.9978	0.8772	0.9169
CALENDAR	0.8149	0.8222	0.9188	0.9189	0.8608	0.8810	0.9984	0.9989	0.9089	0.9281
SETTINGS	0.6370	0.6222	0.6212	0.6395	0.6269	0.6180	0.9737	0.9754	0.6500	0.6478
TASKS	0.6948	0.6600	0.6651	0.6750	0.6792	0.6667	0.9866	0.9907	0.7179	0.7381

DistilBERT Metrics

Table B.11: DistilBERT Overall Metrics (Non-augmented)

Metric	Mean	Median
Accuracy	0.8514	0.8536
Precision	0.7034	0.7102
Recall	0.5388	0.5492
Weighted F1	0.5968	0.6073
Macro F1	0.5831	0.5946

Table B.12: DistilBERT Performance Metrics for Different Categories (Non-augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6705	0.6780	0.4941	0.5000	0.5655	0.5707	0.9405	0.9399	0.6333	0.6243
CONTACTS	0.7378	0.7345	0.5856	0.6087	0.6527	0.6667	0.9622	0.9623	0.7025	0.7181
LOCATION	0.7228	0.7209	0.6820	0.6832	0.6979	0.6983	0.9671	0.9710	0.7358	0.7299
CAMERA	0.7764	0.8000	0.6208	0.6456	0.6873	0.7154	0.9628	0.9628	0.7055	0.6804
MICROPHONE	0.7088	0.6800	0.3562	0.3654	0.4623	0.4750	0.9671	0.9707	0.5182	0.5319
SMS	0.7640	0.8000	0.6815	0.6757	0.7176	0.6906	0.9865	0.9841	0.7828	0.7946
CALL_LOG	0.8673	0.8462	0.3393	0.3438	0.4811	0.5116	0.9931	0.9942	0.6563	0.6764
PHONE	0.5417	0.5116	0.5557	0.5435	0.5350	0.5263	0.9807	0.9784	0.5705	0.5995
CALENDAR	0.7272	0.7273	0.7773	0.7551	0.7495	0.7536	0.9824	0.9876	0.7935	0.7608
SETTINGS	0.5817	0.5682	0.3947	0.4235	0.4690	0.4902	0.9522	0.9531	0.4890	0.4880
TASKS	0.7066	0.6452	0.2991	0.3514	0.3965	0.4598	0.9215	0.9207	0.4807	0.5164

Table B.13: DistilBERT Overall Metrics (Augmented)

Metric	Mean	Median
Accuracy	0.8772	0.8739
Precision	0.7234	0.7067
Recall	0.7705	0.7731
Weighted F1	0.7451	0.7363
Macro F1	0.7593	0.7565

Table B.14: DistilBERT Performance Metrics for Different Categories (Augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.7041	0.7000	0.7127	0.7251	0.7081	0.7195	0.9703	0.9729	0.7629	0.7605
CONTACTS	0.7543	0.7465	0.7802	0.7681	0.7665	0.7708	0.9895	0.9894	0.8196	0.8364
LOCATION	0.7339	0.7103	0.8008	0.8000	0.7658	0.7525	0.9914	0.9918	0.8198	0.8264
CAMERA	0.7602	0.7848	0.8320	0.8226	0.7933	0.7949	0.9952	0.9951	0.8229	0.8355
MICROPHONE	0.6616	0.6842	0.6904	0.7115	0.6745	0.6835	0.9928	0.9959	0.7515	0.7919
SMS	0.8001	0.7778	0.9237	0.9296	0.8566	0.8589	0.9979	0.9976	0.8870	0.8704
CALL_LOG	0.7842	0.7812	0.9574	0.9545	0.8592	0.8772	0.9996	0.9996	0.9379	0.9383
PHONE	0.7901	0.7556	0.8585	0.8605	0.8225	0.8113	0.9984	0.9985	0.8416	0.8509
CALENDAR	0.7869	0.7872	0.8868	0.8966	0.8318	0.8333	0.9989	0.9990	0.8995	0.9039
SETTINGS	0.5957	0.5938	0.6234	0.6211	0.6089	0.6178	0.9791	0.9777	0.6349	0.6501
TASKS	0.6517	0.6486	0.6829	0.6818	0.6648	0.6742	0.9913	0.9914	0.7170	0.7158

DistilRoBERTa metrics

Table B.15: DistilRoBERTa - Overall Metrics (Non-Augmented)

Metric	Mean	Median
Accuracy	0.8521	0.8567
Precision	0.6793	0.6763
Recall	0.5803	0.5990
Weighted F1	0.6142	0.6303
Macro F1	0.5991	0.6151

Table B.16: DistilRoBERTa - Per-Class Metrics (Non-Augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6361	0.6196	0.5695	0.5700	0.5976	0.6042	0.9423	0.9435	0.6436	0.6690
CONTACTS	0.7569	0.7658	0.6358	0.6471	0.6895	0.6692	0.9634	0.9663	0.7302	0.7282
LOCATION	0.6717	0.6786	0.6972	0.7033	0.6825	0.6701	0.9738	0.9741	0.7227	0.7252
CAMERA	0.7224	0.7241	0.6352	0.6479	0.6747	0.6767	0.9716	0.9729	0.6865	0.6865
MICROPHONE	0.5852	0.6000	0.4251	0.4400	0.4809	0.5060	0.9680	0.9654	0.5258	0.5275
SMS	0.7562	0.7386	0.6912	0.6771	0.7208	0.7361	0.9883	0.9873	0.7885	0.8106
CALL_LOG	0.7687	0.7333	0.3943	0.4000	0.5116	0.5128	0.9912	0.9929	0.6347	0.6183
PHONE	0.5162	0.4828	0.5476	0.6087	0.5105	0.5200	0.9787	0.9820	0.5048	0.5380
CALENDAR	0.7153	0.7292	0.8360	0.8974	0.7670	0.7750	0.9802	0.9945	0.7883	0.7964
SETTINGS	0.5696	0.5571	0.4287	0.4138	0.4867	0.4741	0.9528	0.9503	0.4889	0.4872
TASKS	0.7919	0.7931	0.3673	0.4318	0.4941	0.5614	0.9299	0.9379	0.5459	0.5386

Table B.17: DistilRoBERTa - Overall Metrics (Augmented)

Metric	Mean	Median
Accuracy	0.8683	0.8700
Precision	0.6895	0.6848
Recall	0.7914	0.7777
Weighted F1	0.7356	0.7341
Macro F1	0.7527	0.7520

Table B.18: DistilRoBERTa - Per-Class Metrics (Augmented)

Class	Precision		Recall		F1 Score		ROC AUC		PR AUC	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
STORAGE	0.6328	0.6385	0.7445	0.7450	0.6833	0.6748	0.9660	0.9639	0.7285	0.7277
CONTACTS	0.7399	0.7219	0.8016	0.8000	0.7684	0.7609	0.9888	0.9891	0.8336	0.8325
LOCATION	0.6885	0.6789	0.8165	0.8224	0.7468	0.7419	0.9872	0.9856	0.8142	0.8150
CAMERA	0.6893	0.7113	0.8590	0.8621	0.7648	0.7853	0.9949	0.9955	0.7961	0.8220
MICROPHONE	0.6781	0.6579	0.7402	0.7115	0.7060	0.6981	0.9923	0.9950	0.7512	0.7324
SMS	0.7722	0.7917	0.8970	0.8889	0.8297	0.8352	0.9978	0.9977	0.8876	0.8792
CALL_LOG	0.7291	0.7297	0.9749	1.0000	0.8308	0.8438	0.9994	0.9995	0.9138	0.9158
PHONE	0.7947	0.7885	0.8774	0.8864	0.8329	0.8298	0.9982	0.9979	0.8689	0.8996
CALENDAR	0.7741	0.7800	0.9471	0.9388	0.8506	0.8762	0.9991	0.9990	0.9231	0.9423
SETTINGS	0.5828	0.5851	0.6178	0.6180	0.5997	0.6011	0.9780	0.9797	0.6414	0.6334
TASKS	0.6449	0.6765	0.6922	0.6981	0.6665	0.6916	0.9886	0.9865	0.6984	0.6893

Limited Resource Fine-Tuning

Table B.19: Bert Performance after being fine-tuned on 250 sentences

Metric	Value (95% CI)
Accuracy	0.0000 ± 0.0
Precision	0.0000 ± 0.0
Recall	0.0000 ± 0.0
Weighted F1	0.0000 ± 0.0
Macro F1	0.0000 ± 0.0

Table B.20: Bert Performance per permission label after being fine-tuned on 250 sentences

Class	Precision	Recall	F1 Score	ROC AUC	PR AUC
STORAGE	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.6914 ± 0.0033	0.3929 ± 0.0040
CONTACTS	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.7203 ± 0.0027	0.3805 ± 0.0066
LOCATION	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.8229 ± 0.0032	0.4400 ± 0.0079
CAMERA	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.8245 ± 0.0060	0.5171 ± 0.0090
MICROPHONE	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.6578 ± 0.0059	0.1413 ± 0.0050
SMS	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.9307 ± 0.0030	0.6848 ± 0.0071
CALL_LOG	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.9278 ± 0.0033	0.3872 ± 0.0202
PHONE	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.8351 ± 0.0045	0.3034 ± 0.0096
CALENDAR	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.7425 ± 0.0037	0.1283 ± 0.0044
SETTINGS	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.6283 ± 0.0056	0.1686 ± 0.0040
TASKS	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.6612 ± 0.0050	0.0983 ± 0.0023

Table B.21: Bert Performance after being fine-tuned on the augmented dataset

Metric	Value ± 95% CI
Accuracy	0.5318 ± 0.0022
Precision	0.7475 ± 0.0018
Recall	0.6584 ± 0.0013
Weighted F1	0.6789 ± 0.0015
Macro F1	0.6943 ± 0.0012

Table B.22: Bert Performance per permission label after being fine-tuned on the augmented dataset

Class	Precision	Recall	F1 Score	ROC AUC	PR AUC
STORAGE	0.7570 ± 0.0046	0.5378 ± 0.0067	0.6289 ± 0.0055	0.8258 ± 0.0034	0.6790 ± 0.0049
CONTACTS	0.8944 ± 0.0048	0.4050 ± 0.0069	0.5575 ± 0.0062	0.7802 ± 0.0058	0.6710 ± 0.0070
LOCATION	0.8555 ± 0.0050	0.8686 ± 0.0064	0.8619 ± 0.0025	0.9629 ± 0.0019	0.8499 ± 0.0051
CAMERA	0.6850 ± 0.0082	0.7608 ± 0.0085	0.7208 ± 0.0062	0.9161 ± 0.0057	0.6650 ± 0.0068
MICROPHONE	0.6050 ± 0.0101	0.7627 ± 0.0096	0.6746 ± 0.0079	0.9499 ± 0.0028	0.6831 ± 0.0135
SMS	0.7378 ± 0.0072	0.9398 ± 0.0049	0.8266 ± 0.0044	0.9799 ± 0.0013	0.8215 ± 0.0090
CALL_LOG	0.5721 ± 0.0098	0.8160 ± 0.0136	0.6725 ± 0.0092	0.9834 ± 0.0012	0.6690 ± 0.0157
PHONE	0.6824 ± 0.0103	0.7387 ± 0.0115	0.7093 ± 0.0073	0.9696 ± 0.0025	0.7405 ± 0.0140
CALENDAR	0.7012 ± 0.0091	0.9764 ± 0.0033	0.8162 ± 0.0066	0.9905 ± 0.0016	0.8935 ± 0.0108
SETTINGS	0.6915 ± 0.0063	0.4595 ± 0.0058	0.5520 ± 0.0046	0.8707 ± 0.0037	0.6179 ± 0.0101
TASKS	0.6246 ± 0.0142	0.6094 ± 0.0063	0.6168 ± 0.0087	0.8719 ± 0.0054	0.5964 ± 0.0210

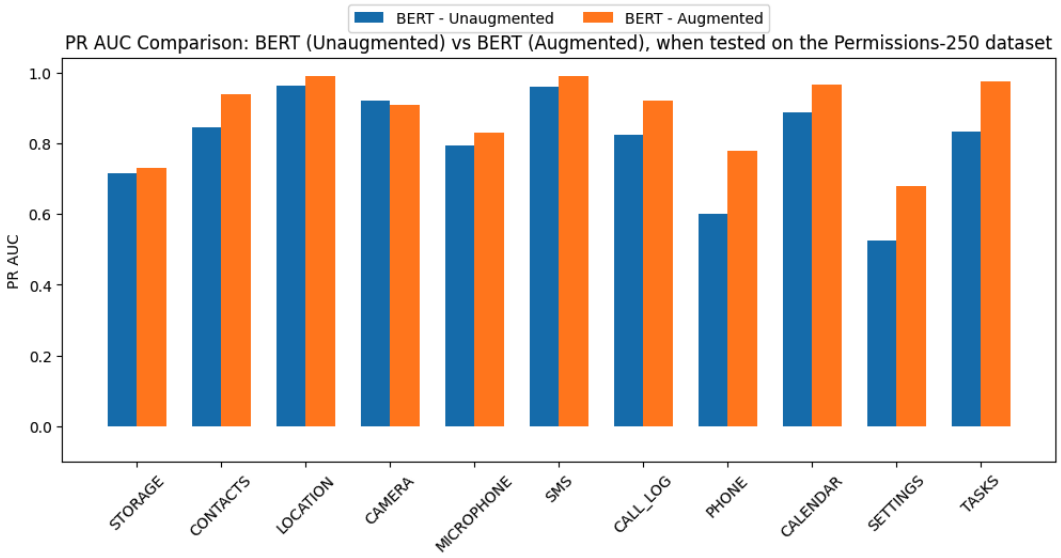


Figure B.1: BERT: PR AUC Comparison for Non-augmented vs Augmented variants, when tested on the Permissions-250 dataset

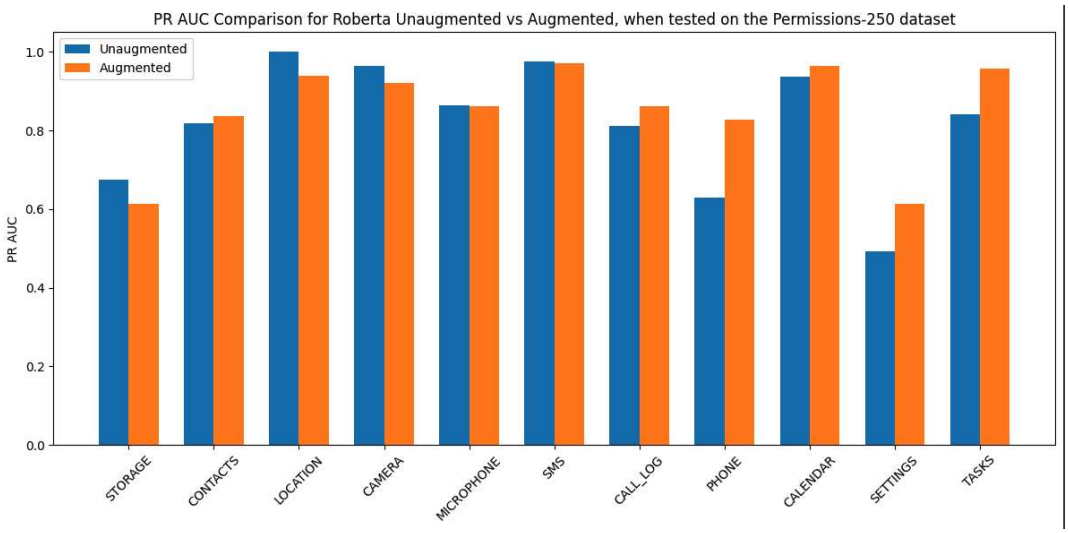


Figure B.2: RoBERTa: PR AUC Comparison for Non-augmented vs Augmented variants, when tested on the Permissions-250 dataset

Huggingface Deployments

Our fine-tuned models have been deployed on Huggingface for public access. Our best performing models are:

- yuniktmr/distilbert-augmented
- yuniktmr/bert-base-augmented
- yuniktmr/distilroberta-augmented
- yuniktmr/roberta-augmented

Random Number Generator Initialization

We use the following formula to generate the seed used for the Random Number Generator initialization during the random test, train, eval split selection during model fine-tuning

$$\text{seed} = (\text{experiment}_i \times 10) + 42$$