UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Software Engineering Curriculum

Hashika Dhananjanie Agalakepu Watthegedara Marasinghe

Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study

Master's Thesis (30 ECTS)

Supervisor: Faiz Ali Shah, PhD
Institute of Computer Science
University of Tartu

Tartu 2024

Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study

Abstract:

App reviews serve as valuable sources of feedback for application developers, offering insights into the needs and preferences of the user. However, the large volume of user reviews received each day makes manual analysis infeasible, requiring automated solutions to detect information in user reviews relevant for developers to improve software quality. Recent strategies for detecting developer-relevant information in app reviews involve fine-tuning pretrained language models (PLMs) for the review classification task using labelled data. Due to the high cost of labelling data and the continuous emergence of new apps and categories in app marketplaces, it is crucial to evaluate recent techniques like pre-train and prompt-tuning, which has demonstrated success in scenarios with limited data. Pre-train and prompt-tuning strategy allows models to adapt to different tasks independently by leveraging domain knowledge introduced through prompts. The main objective of this study is to assess the effectiveness of few-shot prompt-tuning of language models (LMs) for detecting developer relevant information in app reviews. To achieve this objective, the first research question of this study compares the performance of prompt-tuning with traditional fine-tuning of language model RoBERTa under data constrained situation on three labelled review datasets. The second research question explores the impact of prompt-tuning performance for classifying review information on the selection of LMs (T5 and GPT-2) and its architecture. The third and last research question assess the impact of prompt template design and verbalizer design on the performance of prompt-tuning when classifying review information. The findings of this study reveal that the prompt-tuning approach has the potential to outperform traditional fine-tuning strategy in scenarios with limited labelled data availability. Additionally, this study observed variation in model performance across different review datasets, highlighting the importance of model selection, verbalizer design and prompt template design. These insights provide valuable guidance for leveraging prompt-tuning techniques within the app review domain, particularly in contexts characterized by limited availability of labelled data.

Keywords:

App reviews, User reviews, Prompt-tuning, Prompt-learning, Few-shot, Prompt templates, Verbalizers, OpenPrompt

CERCS: P170 - Computer science, numerical analysis, systems, control

Keelesuurmudelite väheste näidete põhine häälestamine rakenduste arvustuste klassifitseerimiseks: hindamisuuring

Lühikokkuvõte:

Rakenduste arvustused on väärtuslikud tagasisideallikad rakenduste arendajatele, pakkudes ülevaateid kasutajate vajadustest ja eelistustest. Siiski teeb suur päevane kasutajaarvustuste hulk käsitsi analüüsi teostamise ebareaalseks, vajades automatiseeritud lahendusi, et tuvastada kasutajaarvustustest arendajatele olulist teavet tarkvara kvaliteedi parandamiseks. Viimased strateegiad arendajatele olulise teabe tuvastamiseks rakenduste arvustustes hõlmavad eelhäälestatud keelemudelite (PLMide) täppishäälestamist arvustuste klassifitseerimise ülesandeks, kasutades märgendatud andmeid. Andmete märgendamise kõrge kulu ja uute rakenduste ja kategooriate pideva ilmumise tõttu rakendusturgudel on ülioluline hinnata hiljutisi tehnikaid nagu eelhäälestamine ja prompt-häälestamine, mis on näidanud edu piiratud andmete olukordades. Eelhäälestamise ja prompt-häälestamise strateegia võimaldab mudelitel iseseisvalt erinevatele ülesannetele kohaneda, kasutades promptide kaudu tutvustatud domeeniteadmisi. Selle uuringu peamine eesmärk on hinnata keelemudelite (LMide) väheste näidete põhise prompt-häälestamise tõhusust arendajatele olulise teabe tuvastamisel rakenduste arvustustes. Selle eesmärgi saavutamiseks võrdleb uuringu esimene uurimisküsimus prompt-häälestamise ja traditsioonilise keelemudeli RoBERTa täppishäälestamise tulemuslikkust andmekitsikuse olukorras kolmel märgendatud arvustuse andmestikul. Teine uurimisküsimus uurib prompt-häälestamise tulemuslikkuse mõju arvustuste teabe klassifitseerimisel keelemudelite (T5 ja GPT-2) valiku ja nende arhitektuuri põhjal. Kolmas ja viimane uurimisküsimus hindab prompti mallide disaini ja verbalisaatori disaini mõju prompt-häälestamise tulemuslikkusele arvustuste teabe klassifitseerimisel. Selle uuringu tulemused näitavad, et prompt-häälestamise lähenemine võib piiratud märgendatud andmete olemasolu korral ületada traditsioonilise täppishäälestamise strateegia. Lisaks täheldati uuringus mudelite tulemuslikkuse varieerumist erinevate arvustuse andmestike vahel, rõhutades mudelivaliku, verbalisaatori disaini ja prompti mallide disaini tähtsust. Need teadmised pakuvad väärtuslikku juhendit prompt-häälestamise tehnikate kasutamiseks rakenduste arvustuste valdkonnas, eriti kontekstides, kus märgendatud andmeid on vähe.

Võtmesõnad:

Rakenduste arvustused, Kasutajaarvustused, Prompt-häälestamine, Prompt-õppimine, Vähesed näited, Prompt-mallid, Verbalisaatorid, OpenPrompt

CERCS: P170 - Arvutiteadus, arvutusmeetodid, susteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

List of Abbreviations

Abbreviation	Definition
FN	False Negative
FP	False Positive
FSL	Few-shot Learning
LM	Language Model
ML	Machine Learning
NLP	Natural Language Processing
PLM	Pre-trained Language Model
RQ	Research Question
TN	True Negative
TP	True Positive

Thesis specific Abbreviations

Abbreviation	Definition
DS	Data Set
EX	Experiment
MD	Model
PT	Prompt Template
VB	Verbalizer

1 Introduction

In today's app marketplaces like Apple's App Store and Google's Play Store, users have the platform to submit feedback, which contains valuable insights such as feature requests, bug reports and non-functional attributes which are crucial for improving app quality. The importance of classifying and understanding user reviews to enhance and sustain the quality of applications has been widely recognized, prompting researchers to explore various methodologies for extracting developer-relevant information from these reviews. User feedback plays a pivotal role in shaping app visibility, credibility, and overall quality. When it comes to improving the quality, app developers should be able to extract the details they need from user reviews to satisfy the users' expectations. The manual analysis becomes impractical because software vendors often receive thousands of feedback entries daily [1]. Consequently, researchers have investigated methods ranging from traditional machine learning to deep learning paradigms [1, 2] and the pre-train & fine-tune approach [3] to automate the classification of crucial insights from user reviews. However, these methods typically rely on large volumes of labelled data for effective model training. Labelling data needs to be performed by domain experts, and this process is both time-consuming and costly [1, 4], particularly in domains such as app reviews, where app marketplaces offer millions of apps belonging to different categories like games, social networking platforms, and entertainment apps posing significant challenges. Moreover, labelling data for these unseen apps becomes increasingly difficult and expensive with the continuous introduction of new app categories.

A comparative study [1] evaluating traditional machine learning and deep learning methods in classifying user feedback concluded that traditional machine learning could yield satisfactory results within their experimental setting compared to deep learning approaches. Nevertheless, given the dependence of machine learning on extensive training datasets, there is a growing need to explore approaches that either require minimal labelled data (few-shot) or no data (zero-shot).

While prompt-tuning has emerged as a promising technique in natural language processing [3], its application in the app review domain remains largely unexplored, with no published work available for reference. Therefore, this study holds significance for application developers and future researchers seeking to harness the potential of prompt-learning in the context of the app review domain.

Motivated by this need, this study systematically evaluates the effectiveness of few-shot prompt-tuning approach that requires minimal labelled data for fine-tuning pre-trained models for classifying app reviews.

This study aims to answer the following research questions:

RQ1: How does the performance of few-shot prompt-tuning compare with traditional fine-tuning when classifying developer-relevant information from app reviews?

RQ1 aims to evaluate the performance of few-shot prompt-tuning in classifying app reviews into categories such as feature requests, bug reports (functional), excessive energy consumption reports, performance problem reports, security issue reports, usability improvement requests, etc. The experiments pertaining to the first research question assess the performance of prompt-tuning in comparison to traditional fine-tuning under data-constrained scenarios. Pre-trained language model RoBERTa serves as the model for both approaches, with one verbalizer and one prompt template utilized as additional settings for prompt-tuning. These experiments are conducted across four different proportions of few-shot samples, ranging from 70% to 2%, to gauge performance under varying levels of data availability. Verbalizer and prompt template used in RQ1, alongside the outcomes of all these experiments, will serve as the foundational benchmarks for subsequent research inquiries. The insights gleaned from addressing this research question provide perspectives on the performance of prompt-tuning under data-constrained conditions compared to traditional fine-tuning methodologies.

RQ2: How does the performance of few-shot prompt-tuning vary when using different language model architectures and sizes in classifying developer-relevant information in app reviews?

RQ2 explores the comparative performance of few-shot prompt-tuning in classifying app reviews using two additional language models from distinct architectural backgrounds. While the language models differ, all other experimental settings from RQ1 are maintained in RQ2. The findings from this inquiry provide valuable insights into the impact of employing diverse language model architectures in few-shot prompt-tuning on performance. Furthermore, upon completion of this investigation, the identification of the most effective language model for each dataset with 2% training data allows for their targeted utilization in subsequent RQ3 experiments.

RQ3: How do the prompt template design and verbalizer design affect the performance in classifying developer-relevant information in app reviews?

RQ3 is specifically designed to investigate how variations in the prompt template and verbalizer designs influence the performance of prompt-tuning. Leveraging the most successful models identified in RQ1 and RQ2 for each dataset with 2% training data, this inquiry introduces two verbalizers and three distinct prompt templates.

2 Background

Initially, this section introduces the significance of app reviews in facilitating software development activities. Then, it provides knowledge concerning Language Models (LMs) and prompt-based learning, relevant to this study.

2.1 User reviews/ App reviews

App shops such as Google Play and Apple AppStore offer over 3 million apps that cover almost every type of software and service. These apps are downloaded, used, and reviewed by billions of users on a regular basis. Recent studies have demonstrated that user evaluations are a valuable source of information for app vendors and developers, as they contain information about issues, ideas for future features, and documentation of released functionality [5]. The majority of the reviews, however, are pretty non-informative, simply praising the app and repeating the star ratings in words. App stores, such as Google Play and the Apple Store, allow users to submit feedback on apps by leaving review comments and star ratings. These platforms provide an effective electronic means for application developers and consumers to exchange app-related information. It was discovered by previous research that user input includes usage scenarios, problem reports, and feature requests, which can assist app developers with software maintenance and evolution responsibilities [1].

Due to the rapid expansion and intense competition in the mobile application (app) market, app developers must not only provide consumers with appealing new features but also carefully maintain and improve existing features based on user feedback. User reviews reveal a rich source of information for planning such feature maintenance operations, and developers may benefit greatly from evaluating and magnifying the value of certain features to the overall success of their apps.

2.2 Language Models (LMs)

A Language Model is an advanced neural network algorithm capable of performing a wide range of natural language processing (NLP) tasks. These models, built upon transformer architectures, are trained on extensive datasets, allowing them to comprehend, translate, predict, and generate textual or other forms of content[1]. In other words, an LM is a computer program trained on numerous examples to understand and interpret human language or complex data [6].

A transformer model, comprising an encoder and a decoder, stands as the prevailing architecture for language models. This model operates by tokenizing input data and performing concurrent mathematical operations to discover connections between tokens. This helps the computer identify patterns similar to how a person would when given the same query[1]. Transformer models can be categorized based on their architectural design[2].

Encoder-Only Architecture:

In this architecture, only the encoder component is utilized. It generates contextualized representations of input text, which can be used for various downstream tasks such as text classification, sentiment analysis and named entity recognition.

[1] https://www.elastic.co/what-is/large-language-models
[2] https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder

Encoder-Decoder Architecture:

This architecture consists of separate encoder and decoder components. The encoder processes input sequences, while the decoder generates output sequences based on the encoded representations. It is commonly used for tasks like sequence-to-sequence learning and text generation.

Decoder-Only Architecture:

In this architecture, only the decoder component is used. It generates text sequences based on learned patterns and inputs, making it suitable for tasks such as text generation, language translation, and dialogue systems.

Architecture	**Models**
Encoder-Only	BERT, RoBERTa
Encoder-Decoder	BART, T5
Decoder-Only	GPT-2, GPT-3

Table 1: Language models

Below are detailed descriptions of language models[3].

- BERT (Bidirectional Encoder Representations from Transformers): BERT has gained significant attention for its ability to understand the context and nuances of language. It's been employed in NLP tasks such as sentiment analysis, named entity recognition, and question-answering systems.
- RoBERTa (A Robustly Optimized BERT Pretraining Approach): This variant of BERT addresses the limitations of its predecessor and has achieved state-of-the-art performance on various language tasks.
- T5: T5 is an encoder-decoder model that translates all natural language processing difficulties to text. It is trained utilizing instructor coercion. This means that during training, there is always an input sequence and a target sequence. The input sequence is fed into the model via input_ids. Kale and Rastogi's paper suggests a pre-train + fine-tune technique for data-to-text jobs. Their experiments show that text-to-text pre-training, such as T5, allows simple, end-to-end transformer-based models to outperform pipelined neural architectures designed for data-to-text generation, as well as alternative language model-based pre-training techniques like BERT and GPT-2. Importantly, they determined that T5 pre-training leads to stronger generalization, as indicated by significant improvements in out-of-domain test sets [7].
- GPT-2: OpenAI's Generative Pre-trained Transformer 2 (GPT-2) is a big language model and the second in the basic series of GPT models. GPT-2 is a huge language model trained using a massive dataset of text and code.
- GPT-3: OpenAI's Generative Pre-trained Transformer 3 is one of the most remarkable language models to date. At the time of its release, the GPT-3 model comprised an unprecedented 175 billion parameters.

[3] https://www.algolia.com/blog/ai/examples-of-best-large-language-models/

- GPT-4: The largest model in OpenAI's GPT series, Generative Pre-trained Transformer 4 was released in 2023. Like other LMs, it's a transformer-based model. The key differentiator is that its parameter count is more than 170 trillion.

Language models (LMs) are gaining prominence in academia and industry because of their superior performance in a variety of applications. As LMs continue to play an important role in both research and daily use, their evaluation becomes more important, not only at the task level but also at the societal level, in order to have a better understanding of their possible hazards. Over the last few years, major attempts have been undertaken to explore LMs from numerous angles [6]. LMs have recently exhibited amazing capabilities in a variety of natural language processing applications, such as language translation, text production, and question-answering. Furthermore, LMs are a novel and important component of computerized language processing, with the ability to recognize complicated speech patterns and provide coherent and suitable responses in a given environment [8].

2.3 Prompt-Based Learning

Prompt-based learning is an emerging group of ML model training methods. It's a strategy that machine learning engineers can use to train LMs so the same model can be used for different tasks without re-training, which makes it a promising alternative to traditional fine-tuning methods. In prompting, users directly specify the task they want to be completed in natural language for the pre-trained language model to interpret and complete.

For example, when recognizing the emotion of a social media post, "I missed the bus today," we may continue with a prompt "I felt so ____" and ask the LM to fill the blank with an emotion-bearing word. Or if we choose the prompt "English: I missed the bus today. French: ____"), then an LM may be able to fill in the blank with a French translation [3].

There are different prompting strategies such as zero-shot, one-shot and few-shot. Zero-shot learning is a model's ability to be able to complete a task without having received or used any training examples. It is capable of generating answers without modifying the parameters of pre-trained LMs, using only a prompt [9].

According to Mayer et al. their study looks into the possibilities of automated classification employing prompt-based learning methodologies with transformer models (huge language models learned unsupervised) for a domain-specific classification job. Prompt-based learning with zero or few shots has the ability to (1) utilize artificial intelligence without advanced programming abilities and (2) use artificial intelligence without fine-tuning models with vast volumes of labelled training data. They used this unique strategy to conduct an experiment with zero-shot classification as a baseline model and a few-shot classification approach. For comparison, they fine-tuned a language model on the supplied classification job and conducted a second independent human evaluation to compare it to the given human ratings from the first study [10].

The prompt-based learning paradigm bridges the gap between pre-training and fine-tuning, and it performs well in the few-shot context [11]. This study focuses on investigating few-shot prompt-tuning strategy.

2.3.1 Fixed-prompt language model tuning strategy (Few-shot)

Few-shot learning involves tuning the parameters of the pre-trained LM, similar to the traditional pre-train and fine-tune paradigm. However, it distinguishes itself by incorporating fixed prompts to define the model's behaviour [9].

With increased privacy controls on personal devices, data is frequently prohibited from leaving these devices and reaching central servers, where crucial operations are performed to improve research and model training. It is critical to improve such data and optimize the performance of current machine learning models. Machine learning has been extremely successful in data-intensive applications, but it suffers when the data set is limited [12]. Recently, Few-shot Learning has been proposed to address this issue. Using prior knowledge, FSL can quickly generalize to new tasks with only a few samples and supervised information [13].

2.3.2 Prompt templates

The prompt template is a piece of natural language text that uses masked areas to elicit knowledge from a PLM [14]. Recently, prompt-based tuning has emerged as an effective method for few-shot learning, bridging the gap between the pre-training and downstream task stages.

Prompt-based tuning involves wrapping input texts with task-specific templates, transforming the original assignment into a cloze-style problem. For example, in a topic classification task, we can use the template "<text>. This topic is about [MASK]", where <text> represents input sentences. PLMs infer words to fill in [MASK], which are then transferred to matching labels using a verbalizer (e.g., "sports" for "Sports") [15]. Studies have been done to learn how prompt templates can cause accuracy to vary with the choice of the template. According to these studies, it was identified that generating a proper prompt template has a large effect on the accuracy of the model [2, 16].

2.3.3 Verbalizers

The verbalizer, one of the most significant modules in prompt-learning, converts the original labels into a collection of label words. In classification, verbalizers are mappings from labels to words in a language model's vocabulary. The verbalizer, which maps label words to class labels, is a critical component of prompt-tuning. These verbalizers guide the model’s understanding of a specific task.

A verbalizer projects label words into class labels, such as "fix" into "Bug Report" class. To classify the line "I can't install it on my Samsung note iii please fix it" into the class "Bug Report", it must first wrap it in a prompt template such as "I can't install it on my Samsung note iii please fix it. This review is about a: [MASK]" before feeding it into the PLM. If the likelihood of correctly guessing the label word "fix" in the "[MASK]" place above a certain threshold, classify the sentence as "Bug Report" [14]. Recent studies show the effect of verbalizers in prompt-learning and how important it is to design the verbalizers carefully to achieve optimal performance [17].

2.4 Evaluation Metrics

2.4.1 Precision

Precision is a metric that gives you the proportion of true positives to the number of total positives that the model predicts. It answers the question, "Out of all the positive predictions we made, how many were true?" Precision is defined as the number of correct positive predictions (TP) divided by the total number of positive predictions (TP + FP). The best accuracy is 1.0, while the worst is 0.0 [18].

Precision = TP / (TP + FP)

2.4.2 Recall

Recall focuses on how good the model is at finding all the positives. Recall is also called true positive rate and answers the question, "Out of all the data points that should be predicted as true, how many did we correctly predict as true?" Recall is measured by dividing the number of accurate positive predictions (TP) by the total number of positives (P). The best TP rate is 1.0, while the worst is 0.0 [18].

Recall = TP / (TP + FN)

2.4.3 F1 Score

F1 Score is a measure that combines Recall and Precision. This measure represents the average Recall and Precision levels. As we have seen, there is a trade-off between Precision and Recall. F1 can, therefore, be used to measure how effectively our models make that trade-off.

*F1-score = 2 (precision * recall) / (precision + recall)*

2.4.4 Accuracy

The accuracy metric calculates the ratio of correct predictions to the total number of cases assessed. Accuracy is calculated by adding two accurate predictions (TP + TN) and dividing by the entire number of data sets (P + N). The highest accuracy is 1.0, while the lowest is 0.00 [18].

Accuracy = (TP + TN) / (TP + TN + FP + FN)

3 Methodology

This section describes the methodologies employed to fulfil the objectives outlined in the thesis. It comprises two main subsections: Experimental Setup and the Experiments.

The Experimental Setup explains the datasets utilized, the data pre-processing methodologies employed, the data splitting strategies, the language models utilized, and the designs of prompt-tuning verbalizers and prompt templates, alongside the hyperparameter configurations of the experiments.

The subsequent section provides a detailed explanation of the experiments conducted for each research question. The experimental process comprised several sequential steps, commencing with the establishment of baseline performances using the RoBERTa model for traditional fine-tuning and prompt-tuning and subsequently exploring variations in performance when employing models from diverse language model architectures. The third research question investigated how alterations in prompt-tuning verbalizers and prompt templates impact performance.

The OpenPrompt[4] library was employed for all prompt-learning experiments across RQ1, RQ2, and RQ3. OpenPrompt is a unified, easy-to-use toolkit to conduct prompt-learning over pre-trained language models (PLM), renowned for its flexibility and extensibility, facilitates the deployment of prompt-learning pipelines without the need to implement from scratch. Leveraging the OpenPrompt library enables the utilization of various language models and facilitates the design of bespoke prompt-learning workflows [19].

3.1 Experimental Setup

3.1.1 Labelled review datasets

This section describes the datasets utilized in the experiments conducted within this study. Three manually labelled datasets, namely Scalabrino [5], Maalej[5], and Pan[5], were employed to assess the efficacy of app review classification employing prompt-tuning methodologies. The Scalabrino dataset facilitates analysis across seven distinct classes, whereas the Maalej and Pan datasets comprise four classes each. Notably, the feature request and bug report classes were consistently present across all three datasets. Each labelled dataset is described in following subsections.

3.1.1.1 Scalabrino Dataset (DS1)

The Scalabrino dataset [20] utilized in this study comprise a collection of 3000 user reviews that were manually categorized into seven classes by two authors. These reviews were sourced from a diverse array of 705 Android apps spanning categories such as games, books, education, communication, health, sports, travel, weather, and more. Below are the class details of this dataset.

[4] https://github.com/thunlp/OpenPrompt
[5] https://data.mendeley.com/datasets/5fk732vkwr/2

Class name	Class definition	No. of records
Bug	Functional bug report	764
Energy	Report of excessive energy consumption	106
Feature	Suggestion for new feature	333
Other	Non-informative reviews	1505
Performance	Report of performance problems	135
Security	Report of security issues	50
Usability	Request for usability improvements	107

Table 2: Class distribution in Scalabrino dataset

The dataset was obtained through the replication package[6] provided in the corresponding paper [21].

3.1.1.2 Maalej Dataset (DS2)

The second dataset employed in this study was a truth set prepared for the experiments of a study on the automatic classification of app reviews [22]. This dataset contains 3691 reviews from various apps available in Play Store and App Store. Each review underwent manual analysis and labelling into four distinct classes: bug reports, feature requests, user experiences, and ratings. Bug reports describe issues within the application that necessitated rectification, such as crashes, erroneous behaviour, and performance shortcomings. Feature requests encompass user demands for additional functionalities or content, accompanied by suggestions for app enhancements. User experiences narrative detailed users' experiences with the application. Ratings encompass of non-informative reviews expressing praise or criticisms.

Class name	No. of records
Feature Request	252
Problem Discovery/ Bug Report	370
Rating	2462
User Experience	607

Table 3: Class distribution in Maalej dataset

This dataset was sourced from a publicly available repository[1] [23].

[6] https://zenodo.org/records/5733504

3.1.1.3 Pan Dataset (DS3)

The third dataset utilized in this study was a truth set employed in a study on classifying user reviews for software maintenance and evolution [4]. Comprising 1390 reviews from prominent apps like Angry Birds, Dropbox, Evernote, TripAdvisor, PicsArt, Pinterest, and WhatsApp, available in both App Store and Play Store, this dataset underwent manual labelling by two authors into four distinct classes: feature requests, information giving, information seeking, and problem discovery/ bug reports. Feature requests encapsulate reviews containing ideas, suggestions, or needs for enhancing app functionalities. Information giving reviews aim to inform users or developers about various aspects related to the app. Information seeking reviews depict the reviews which attempt to obtain information or assistance from other users or developers. Problem discovery class describe issues and unexpected behaviours encountered within the app.

Class name	No. of records
Feature Request	192
Information Giving	603
Information Seeking	101
Problem Discovery	494

Table 4: Class distribution in Pan dataset

This dataset was also sourced from a publicly available repository[1] [23].

3.1.2 Data Pre-processing

In the pre-processing phase of this study, a series of steps were undertaken to ensure the quality and consistency of the datasets utilised. The pre-processing pipeline included several key steps aimed at standardising the textual data across all three datasets; Scalabrino, Maalej and Pan. The pre-processing phase employed a two-step approach to handle common text-processing tasks:

Expansion of contractions: Contractions, such as "don't" or "can't", were expanded to their full forms (e.g., "do not" or "cannot") to maintain uniformity and improve readability. This ensured the contractions were consistent in all the review texts.

Text cleaning: Various cleaning operations were applied to remove unwanted elements from the text. This involved eliminating special characters, such as punctuation marks and symbols, and replacing them with spaces. Additionally, redundant whitespaces were removed to streamline the text. Finally, any non-ASCII characters, including emojis, were stripped from the text to ensure compatibility with downstream tasks.

By implementing these pre-processing steps consistently across all datasets, the textual data were standardised and prepared for subsequent analyses, including model training and evaluation. This pre-processing approach aimed to mitigate potential noise and inconsistencies in the review texts, ultimately enhancing the quality and reliability of the experimental results obtained in this study.

3.1.3 Training, validation, and test sets

In this study, the data splitting approach served as a pivotal component in evaluating the performance of prompt-tuning under data-constrained scenarios within the app review domain. Four distinct data splitting strategies were employed to comprehensively investigate the efficacy of prompt-tuning compared to traditional fine-tuning, particularly when labelled data availability was limited.

The initial splitting approach designated a standard training dataset size of 70%, validation dataset size of 15%, and test dataset size of 15%, allowing for an assessment of traditional fine-tuning performance under conditions of ample labelled data availability. Subsequent to this baseline evaluation, three additional data splitting strategies were implemented, each featuring significantly reduced training and validation dataset sizes of 5%, 3%, and 2% derived from the initial training dataset and validation dataset. These diminished training dataset sizes were chosen deliberately to mirror scenarios of limited labelled data availability, thereby facilitating an examination of prompt-tuning performance in data-scarce environments.

It is noteworthy that the training dataset sizes of 5%, 3%, and 2% were aligned with the corresponding validation set sizes, ensuring a consistent evaluation framework across the experiments. This alignment enabled a focused analysis of prompt-tuning performance in scenarios characterised by constrained labelled data availability. Conversely, the test set sizes remained consistent across all splitting approaches, ensuring fair and equitable performance comparisons between traditional fine-tuning and prompt-tuning methodologies. Stratification was employed to ensure that the class proportions remained consistent across the training, validation and test sets, a crucial aspect in maintaining the integrity of the experimental design.

Overall, the employed data splitting approaches were meticulously designed to systematically evaluate the performance of prompt-tuning across varying degrees of labelled data scarcity, thereby facilitating nuanced insights into its efficacy for app review classification tasks. Below is a detailed view of data splitting strategies used in this study.

Training set size	Validation set size	Test set size	Reference
70%	15%	15%	EX1
5% of EX1	5% of EX1	15%	EX2
3% of EX1	3% of EX1	15%	EX3
2% of EX1	2% of EX1	15%	EX4

Table 5: Data splitting strategy

Dataset	Data split strategy	Training sample size	Validation sample size	Test sample size
DS1 (3000)	EX1	2100	450	450
	EX2	105	22	450
	EX3	63	13	450
	EX4	42	9	450
DS2 (3691)	EX1	2584	553	554
	EX2	129	27	554
	EX3	77	16	554
	EX4	51	11	554
DS3 (1390)	EX1	973	208	209
	EX2	48	10	209
	EX3	29	6	209
	EX4	19	4	209

Table 6: Sample sizes

3.1.4 Language Models

This section describes the pre-trained language models used in the experiments based on traditional fine-tuning and prompt-tuning. In this study, three models were employed from three distinct language model architectures to investigate the performance of traditional fine-tuning and prompt-tuning in classifying developer-relevant information within app reviews. Each architecture represents a unique approach to natural language processing, providing valuable insights into the effectiveness of prompt-tuning across different model designs.

3.1.4.1 Encoder-only Model: RoBERTa (MD1)

RoBERTa, a variant of the BERT (Bidirectional Encoder Representations from Transformers) architecture, falls under the category of encoder-only models. This architecture processes text inputs by a stack of transformer encoder layers, enabling efficient representation learning from unidirectional context. RoBERTa enhances BERT's pre-training procedure by incorporating additional training data and fine-tuning strategies, leading to improve performance in downstream tasks [24]. This widely used transformer-based model is renowned for its robust performance in various NLP tasks.

Model variant	Details of the model
roberta-base	12-layer, 768-hidden, 12-heads, 125M parameters RoBERTa using the BERT-base architecture

Table 7: RoBERTa model details

3.1.4.2 *Encoder-decoder model: T5 (MD2)*

T5 (Text-To-Text Transfer Transformer) is an encoder-decoder model that operates on the principle of transforming input text into output text. Unlike encoder-only models, T5 employs both encoder and decoder components. The encoder processes input sequences, while the decoder generates target sequences based on the learned representations. T5's unified text-to-text framework enables seamless handling of diverse natural language understanding and generation tasks, facilitating effective transfer learning and task adaptation [25].

Model variant	Details of the model
t5-base	~220M parameters with 12-layers, 768-hidden-state, 3072 feed-forward hidden-state, 12-heads, Trained on English text: The Colossal Clean Crawled Corpus (C4)

Table 8: T5 model details

3.1.4.3 *Decoder-only models: GPT-2 (MD3)*

GPT-2 (Generative Pre-trained Transformer 2), developed by OpenAI, belongs to the category of decoder-only models. In this architecture, text generation is performed autoregressively, where each token in the output sequence is generated based on preceding tokens. GPT-2 utilises a stack of transformer decoder layers to capture dependencies and generate coherent text sequences. While decoder-only models are primarily used for text generation tasks, they can also be adapted for various other NLP tasks through fine-tuning [26].

Model variant	Details of the model
gpt2-medium	24-layer, 1024-hidden, 16-heads, 345M parameters. OpenAI’s Medium-sized GPT-2 English model

Table 9: GPT-2 model details

These models were obtained from the Hugging Face model hub[7], a comprehensive repository of pre-trained language models and associated resources.

3.1.5 Prompt Templates

Prompt templates serve as modifiers of the original input text and play a crucial role in the prompt-learning framework. This study focused on understanding how the performance of classifying app reviews varies when different prompt templates are employed. Four distinct prompt templates were utilized for the experiments conducted in this study.

[7] https://huggingface.co/models

While various techniques exist for deriving effective prompt templates, such as manual template engineering and automated template learning engineering [2] this study opted for handcrafted templates. The templates were crafted by the researcher, drawing inspiration from similar studies in the literature [2, 9].

The objective of this study was not to identify the optimal templates but rather to investigate the impact of different templates on app review classification performance. Therefore, the selected templates were chosen to represent a diverse range of approaches commonly used in prompt-learning frameworks. By examining how the performance varies across different templates, insights can be gained into the relative effectiveness of different prompt template designs in the context of classifying app reviews.

Prompt templates used in this study are shown in the table below.

Template#	**Prompt Template**	**Reference**
Template 1	'{"review_text "} Classify this review: {"mask"}'	PT1
Template 2	'{"review_text"} This review is about a {"mask"}'	PT2
Template 3	'{"review_text"} This belongs to class {"mask"}'	PT3
Template 4	'{"review_text"} This user review belongs to class {"mask"}'	PT4

Table 10: Prompt templates

3.1.6 Verbalizers

The verbalizer serves as an essential part of the prompt-learning pipeline, mapping the outputs of the language model into the necessary labels or categories. This section describes the verbalizers employed in the experiments conducted within this study. Three distinct verbalizers were utilised, each contributing to the classification process in unique ways.

The first verbalizer leveraged logistic regression techniques tailored to the characteristics of each dataset, extracting keywords associated with individual classes to inform the classification process. In contrast, the second verbalizer adopted a straightforward approach by utilising the class names themselves as keywords, simplifying the mapping process. The third verbalizer was constructed utilising insights derived from the OpenAI ChatGPT 3.5 model[8]. This method involved giving the language model a prompt that described the datasets and their classes, prompting it to identify keywords representing each class.

Subsequent sections delve into the details of each verbalizer design, describing the methodologies followed when designing the verbalizers. Through an exploration of these verbalizer strategies, a comprehensive understanding of their impact on classification performance within the context of app review analysis is attained.

[8] https://chat.openai.com/

3.1.6.1 Verbalizer 1 (VB1)

The initial step in the design of the first verbalizer entailed the application of logistic regression models to each of the three datasets individually, aiming to discern keywords pertinent to each class within the datasets. Subsequently, the extracted keywords underwent a meticulous manual analysis and cleaning process to ascertain each class's most salient and representative set. During this iterative process, the researcher and the research supervisor collaborated to individually analyse and identify potential keywords for each class.

Following this individual analysis, a discussion ensued, during which deliberated upon the identified sets of keywords. A final set of keywords was curated through mutual agreement and consensus, capturing the essence of each class across the datasets. Notably, as the bug report/problem discovery and feature request classes were prevalent across all three datasets, and the ratings/other class was prevalent across two datasets, the identified keywords from each dataset were merged to form a comprehensive set.

This methodical approach ensured that the selected keywords were robust and reflective of the distinctive characteristics of each class, thereby enhancing the effectiveness of the verbalizer in accurately mapping language model outputs to the corresponding class labels.

Dataset	**Class**	**Keywords**
DS1	Bug	freeze, fix, bug, error, crash, stuck, issue, problem, fail
	Energy	battery, drain
	Feature	feature, add, wish, improve, lack, miss, need, suggest
	Other	best, useful, love, awesome, fantastic, excellent, rubbish, useless, wow, superb, addict, nice
	Performance	slow, lag, glitch, performance
	Security	virus, hack, permission, secure
	Usability	difficult, ad, annoy, interface, gui, button
DS2	Feature Request	feature, add, wish, improve, lack, miss, need, suggest
	Problem Discovery	freeze, fix, bug, error, crash, stuck, issue, problem, fail
	Rating	best, useful, love, awesome, fantastic, excellent, rubbish, useless, wow, superb, addict, nice
	User Experience	ui, easy, graphic
DS3	Feature Request	feature, add, wish, improve, lack, miss, need, suggest
	Information Giving	recommend, best, great, idea, check
	Information Seeking	why, how, what, where, who
	Problem Discovery	freeze, fix, bug, error, crash, stuck, issue, problem, fail

Table 11: Verbalizer design 1

3.1.6.2 Verbalizer 2 (VB2)

Verbalizer 2 adopts a simplistic yet effective approach by utilizing the class names themselves as keywords. This design strategy draws inspiration from a previous study conducted by Luo [9], wherein one of the verbalizers employed the words from the original intention labels initially established within the dataset. In alignment with this approach, Verbalizer 2 in the current study leverages the class names directly as keywords.

By directly associating the class names with the corresponding labels, this verbalizer design simplifies the mapping process between language model outputs and class categories. This straightforward approach not only streamlines the verbalizer creation process but also ensures clarity and transparency in the interpretation of model predictions.

3.1.6.3 Verbalizer 3 (VB3)

Verbalizer 3 adopts an approach by leveraging the human-like capabilities of ChatGPT [6], to generate relevant keywords for each class in the dataset. This design methodology involved providing specific prompts to ChatGPT, outlining details about the dataset, the classes within it, and the role of verbalizers in prompt-learning. Additionally, examples from the OpenPrompt library illustrating the definition of verbalizers for different classes were provided to offer context. The purpose of utilizing ChatGPT in generating verbalizers was to lessen the dependency on domain experts for verbalizer creation.

Separate prompts tailored to each dataset were utilized, ensuring that the characteristics and nuances of each dataset were accurately conveyed to ChatGPT. By presenting dataset-specific information, the aim was to elicit keywords that were contextually relevant and reflective of the user reviews typically associated with each class.

Unlike Verbalizer 1, which involved manual analysis and curation of keywords, Verbalizer 3 relied on the raw output generated by ChatGPT without further analysis or cleaning. This approach aimed to capture a broad spectrum of keywords that users might use when providing reviews related to each class in the dataset. Through this method, Verbalizer 3 sought to leverage the natural language understanding capabilities of ChatGPT to enhance the relevance and diversity of the generated keywords. The prompts provided to ChatGPT and the corresponding results can be referenced in the appendix.

Dataset	Class	Keywords
DS1	Bug	crash, error, bug, glitch, malfunction, issue, problem
	Energy	battery drain, energy consumption, power usage, energy efficiency, drain, resource-intensive
	Feature	feature, improvement, addition, suggestion, enhancement, request
	Other	praise, criticism, feedback, opinion, irrelevant, off-topic
	Performance	slow, lag, performance, speed, efficiency, bottleneck
	Security	security, vulnerability, breach, hack, exploit, threat

	Usability	usability, interface, user experience, navigation, intuitive, user-friendly
DS2	Feature Request	suggest, add, feature, improvement, enhancement, request, update, new functionality, missing feature, wish
	Problem Discovery	crash, error, issue, bug, freeze, glitch, performance problem, malfunction, software problem, application crash
	Rating	good, great, excellent, fantastic, amazing, poor, terrible, disappointing, satisfactory, exceptional
	User Experience	usability, user-friendly, experience, intuitive, navigation, interface, convenience, ease of use, helpful, smooth
DS3	Feature Request	suggest, idea, improve, enhance, functionality, request, wish, addition, enhancement, new feature
	Information Giving	update, inform, news, announcement, change log, release notes, release, version, details, notification
	Information Seeking	help, question, query, seek, assistance, support, advice, guidance, clarification, answer
	Problem Discovery	issue, problem, bug, glitch, error, malfunction, crash, problem-solving, troubleshooting, unexpected behavior

Table 12: Verbalizer design 3

3.1.7 Hyperparameters

The experimental setup utilized for traditional fine-tuning and prompt-tuning involved a specific configuration of hyperparameters to ensure consistency and reproducibility of the results. These hyperparameters were selected to balance computational efficiency with model performance and stability during training. The below sections elaborate on the experimental setup used in this study.

3.1.7.1 Environmental Setup

Setup	Values
Platform	pegasus2.hpc.ut.ee (Rocket server - OpenStack)
Machine Type	2x AMD EPYC 7713 64-Core Processors (256 cores total), 2 TB RAM, 15 TB of local SSD storage
GPUs	8 x Tesla a100 with 80GB vRAM each

Network	Infiniband with 9 x 100Gb connections
Python Version	3.9.12
OpenPrompt Version	1.0.1

Table 10: Environmental setup

3.1.7.2 Traditional fine-tuning & Prompt-tuning setup

Below is a breakdown of each hyperparameter used in this setup and its purpose in the training process:

Batch Size: Determines the number of samples processed before the model's parameters are updated. A larger batch size can lead to faster training but requires more memory.

Number of Epochs: Specifies the number of times the entire dataset is passed forward and backward through the neural network. Increasing the number of epochs can improve the model's performance, but too many epochs may lead to overfitting.

Random Seed: Sets the initial random state for reproducibility. By fixing the random seed, the same sequence of random numbers will be generated each time the code is run, ensuring consistent results.

Patience: Determines the number of epochs to wait for improvement in the validation loss before terminating the training process. It helps prevent overfitting by stopping training when the model's performance no longer improves on the validation set.

Configuration	**Configured Values**
Batch Size	8
No of Epochs	10
Random Seed	2022
Patience	3

Table 11: Traditional fine-tuning & Prompt-tuning setup

3.2 Experiments

Following sections describe the experiments executed to answer the three research questions in this study.

3.2.1 RQ1 Experiments

The methodology employed for RQ1 involved conducting a series of experiments to compare the performance of traditional fine-tuning and prompt-tuning approaches in classifying app reviews into predefined classes. Specifically, the experiments utilized the RoBERTa model, verbalizer 1, and prompt template 1 across three datasets (DS1, DS2, DS3).

To initiate the experiments, pre-processing logic was applied to each dataset to ensure consistency and readiness for model training. A total of twelve experiments were conducted for RQ1, encompassing various data splitting strategies, including 70%, 5%, 3%, and 2% of the training dataset size. The same samples of training, validation and test datasets were used across both the traditional fine-tuning and prompt-tuning experiments in each scenario to ensure equitable exposure to the data and facilitate fair comparisons between traditional fine-tuning and prompt-tuning methodologies.

The RoBERTa model, obtained from Hugging Face, served as the backbone for both approaches. The experiments were trained over ten epochs with an early stopping criterion set to a patience of three to ensure optimal convergence.

The outcomes of these experiments formed the foundation for subsequent research questions, providing insights into the baseline performance of prompt-tuning compared to traditional fine-tuning under varying data constraints. After execution, a performance report was obtained for each experiment with Precision, Recall, F1 score and Accuracy values.

Datasets	Model	Data split	Prompt template	Verbalizer
Scalabrino	RoBERTa	70%	PT1	VB1
Maalej		5%		
Pan		3%		
		2%		

Table 12: RQ1 experimental setup

3.2.2 RQ2 Experiments

RQ2 aimed to investigate the impact of utilizing different models from distinct language model architectures on the classification performance of app review classes. In contrast to RQ1, which focused on utilizing the RoBERTa model from the encoder-only architecture, RQ2 expanded the scope by incorporating models from diverse architectures.

For this research question, two additional models were introduced: T5 from the encoder-decoder architecture and GPT-2 from the decoder-only architecture. The objective was to compare the performance of these models with the baseline established in RQ1 using RoBERTa, while keeping all other experimental variables constant. This approach allows for a comprehensive understanding of how different language model architectures influence prompt-tuning classification outcomes.

Similar to RQ1, the experiments in RQ2 retained the same experimental setup, including the use of OpenPrompt library, consistent pre-processing logic, and identical data splitting strategies across datasets. Training, validation, and testing datasets remained unchanged to ensure fair comparisons between the different models.

The experiments were conducted over 10 epochs with an early stopping criterion set to a patience of 3, maintaining consistency with the methodology established in RQ1. This approach facilitates a direct comparison of model performances across different architectures under similar training conditions.

The insights gained from RQ2 experiments provided valuable information on the relative strengths and weaknesses of models from various language model architectures in classifying app review classes, contributing to a deeper understanding of the prompt-tuning paradigm in data-constrained scenarios.

Datasets	**Models**	**Data split**	**Prompt template**	**Verbalizer**
Scalabrino	T5	70%	PT1	VB1
Maalej	GPT-2	5%		
Pan		3%		
		2%		

Table 13: RQ2 experimental setup

3.2.3 RQ3 Experiments

RQ3 focused on the examination of how variations in prompt templates and verbalizers impact the performance of classifying app reviews using prompt-tuning. The objective of this research question was to discern the influence of different prompt template designs and verbalizer designs on classification outcomes.

Given the focus on evaluating prompt-tuning performance under data-scarce conditions, the experiments in RQ3 were exclusively conducted using 2% of the training datasets. To establish the baseline for RQ3 experiments, the top-performing experiments from RQ1 and RQ2, conducted with a minimum training dataset percentage of 2%, were selected for each dataset. These experiments, which demonstrated optimal performance in prompt-tuning, served as the foundation for subsequent investigations. For the Scalabrino dataset, the GPT-2 model exhibited superior performance, while the T5 model performed well for the Maalej and Pan datasets. Consequently, these models were utilized in the RQ3 experiments, wherein prompt template and verbalizer modifications were implemented.

In the initial set of experiments for RQ3, the verbalizer (VB1) remained constant while the prompt templates were varied across different iterations. This approach enabled the assessment of how performance fluctuated in response to changes in prompt template designs.

Subsequently, the prompt template (PT1) was held constant, and the verbalizers were altered to observe the variations in performance attributable to different verbalizer designs.

Consistency was maintained with previous experiments, employing OpenPrompt, standardized pre-processing procedures, and identical numbers of epochs and patience values as in the RQ1 and RQ2 experiments.

Through systematic experimentation and analysis, RQ3 aimed to find the impact of prompt template designs and verbalizer designs on the classification performance of app reviews, providing valuable insights into the effectiveness of prompt-tuning methodologies in data-constrained scenarios.

Datasets	**Models**	**Data split**	**Prompt templates**	**Verbalizers**
Scalabrino	T5	2%	PT2	VB2
Maalej	GPT-2		PT3	VB3
Pan			PT4	

Table 14: RQ3 experimental setup

4 Results

4.1 RQ1

RQ1 endeavours to assess the efficacy of prompt-tuning in contrast to traditional fine-tuning methodologies across three distinct datasets. The experiments were conducted employing the RoBERTa model, encompassing varying proportions of training data at 70%, 5%, 3%, and 2%. Within the context of prompt-tuning experiments, a baseline configuration was established utilizing the VB1 and PT1 settings, serving as the foundation for subsequent analyses. To enhance readability, only the overall performance is summarized here; for detailed class-wise performance, refer to the appendix.

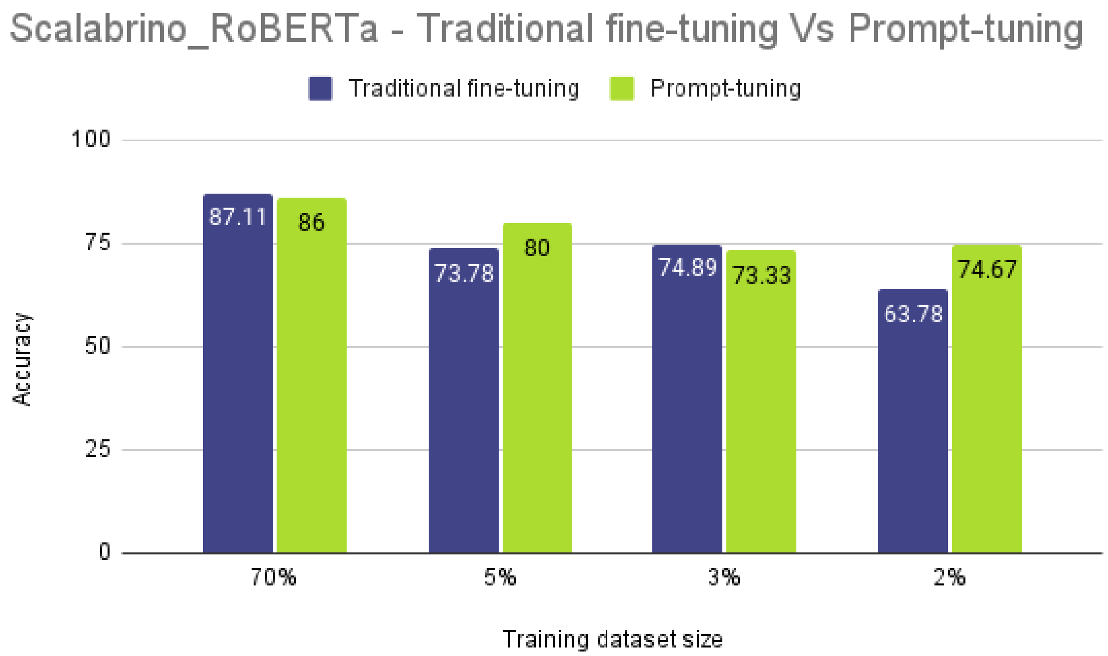

Figure 1: Comparison of RoBERTA model performance between traditional fine-tuning and prompt-tuning on Scalabrino dataset

Figure 1 shows the results produced by the first set of experiments, conducted using the Scalabrino dataset. At a 70% training dataset size, when there's adequate labelled data available, traditional fine-tuning achieved an accuracy of 87.11%. The 5%, 3%, and 2% experiments intended to explore the efficacy of prompt-tuning in scenarios characterized by constrained data availability. At 5% training dataset availability, prompt-tuning achieved an accuracy of 80%, surpassing the accuracy of 73.78% attained by traditional fine-tuning, resulting in a difference of 6.22%. At 3% training data size traditional fine-tuning demonstrated higher accuracy of 74.89%, while prompt-tuning achieved 73.77% accuracy, resulting in a difference of 1.12%. Further reductions to 2% training dataset size showcased the superiority of prompt-tuning, with accuracies of 74.67%, compared to 63.78% for traditional fine-tuning. This performance exhibited a difference of 10.89% compared to traditional fine-tuning methodology, highlighting the effectiveness of prompt-tuning in data-constrained scenarios.

The findings revealed interesting insights into the comparative performance of traditional fine-tuning and prompt-tuning across various training dataset sizes. In the scenario where the training dataset size is limited to 2%, indicating a scarcity of training data, prompt-tuning demonstrated superior performance compared to traditional fine-tuning methodology.

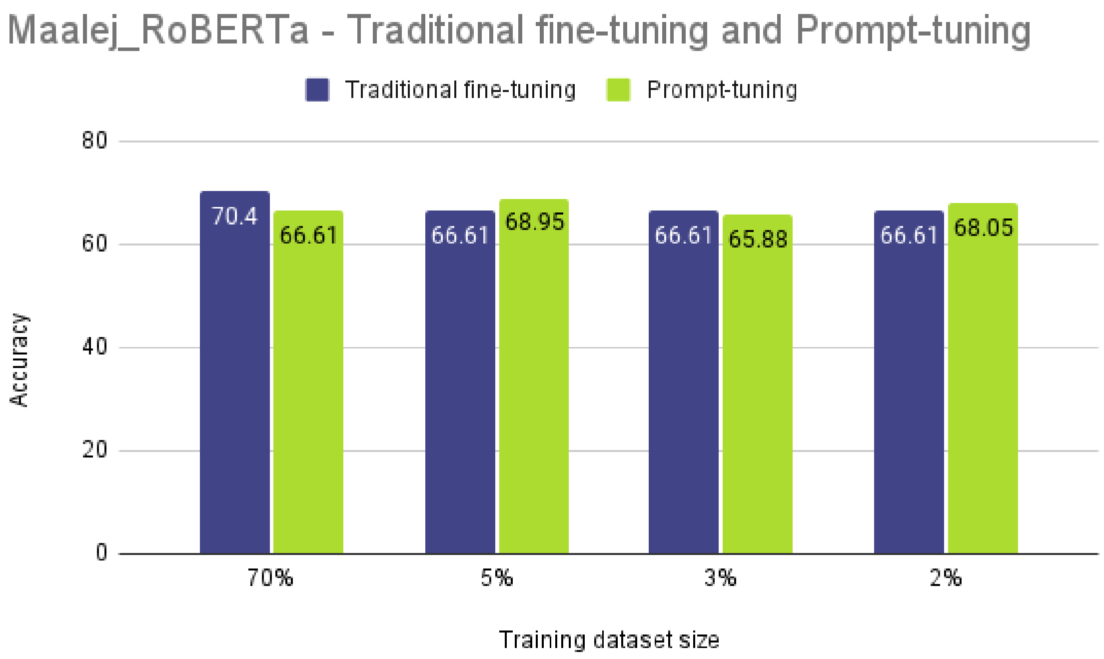

Figure 2: Comparison of RoBERTA model performance between traditional fine-tuning and prompt-tuning on Maalej dataset

Figure 2 shows the results produced by the second set of experiments, conducted using the manually labelled Maalej dataset. At a 70% training dataset size, traditional fine-tuning exhibited an accuracy of 70.4%. At 5% training data availability, prompt-tuning demonstrated a higher accuracy of 68.95% compared to the accuracy of 66.61% achieved by traditional fine-tuning, resulting in a difference of 2.34%. At the 3% training dataset size, traditional fine-tuning achieved an accuracy of 66.61%, slightly surpassing prompt-tuning's accuracy of 65.88%, resulting in a difference of 0.73%. Prompt-tuning exceeded traditional fine-tuning at the 2% dataset size, with an accuracy of 68.05% as compared to 66.61% accuracy of traditional fine-tuning, resulting in a difference of 1.44%.

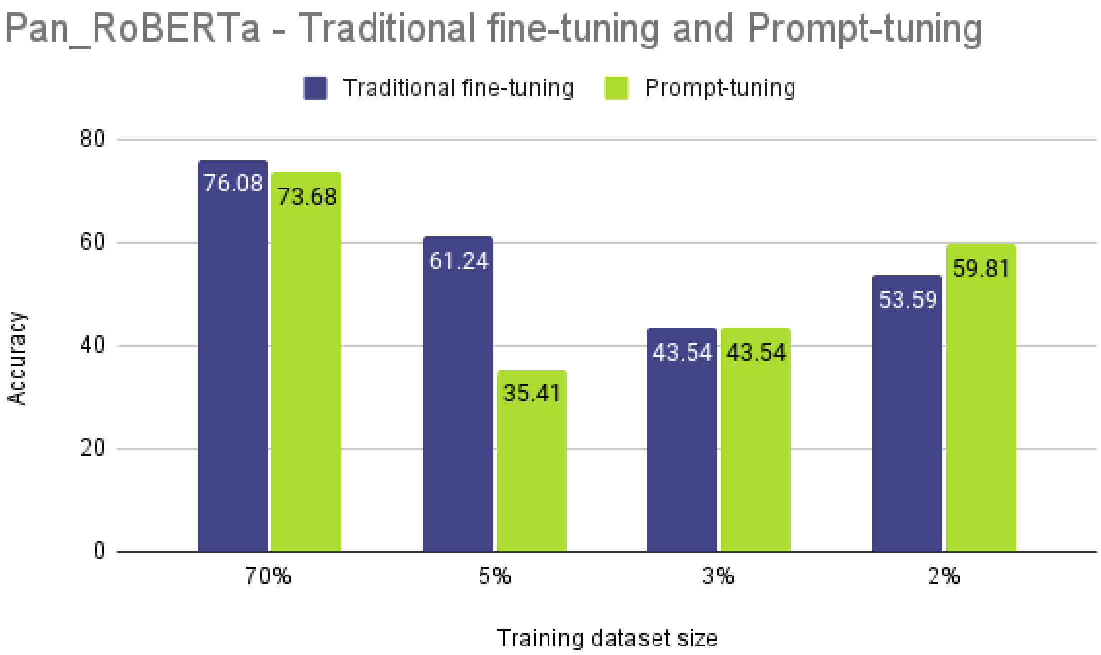

Figure 3: Comparison of RoBERTA model performance between traditional fine-tuning and prompt-tuning on Pan dataset

Figure 3 shows the results of third set of RQ1 experiments, conducted using the manually labelled app review dataset Pan, aimed to evaluate the performance of traditional fine-tuning and prompt-tuning approaches in classifying app reviews into distinct categories. In the initial experiment utilizing 70% of the dataset, traditional fine-tuning yielded an accuracy of 76.08%. At a data availability of 5%, prompt-tuning exhibited an accuracy of 35.41%, contrasting with the 61.24% accuracy achieved by traditional fine-tuning, resulting in a difference of 25.83%. With the 3% dataset size, both traditional fine-tuning and prompt-tuning yielded identical accuracies of 43.54%. The results indicated that at a dataset size of 2%, prompt-tuning notably surpassed traditional fine-tuning, achieving an accuracy of 59.81% compared to 53.59%, having a difference of 6.22%.

In all the scenarios with limited annotated data, specifically at the lowest of 2% training data size of RQ1, prompt-tuning showcased strengths in accuracy compared to traditional fine-tuning. These results underscore the effectiveness of prompt-tuning, particularly in scenarios with limited data, showcasing its potential for superior performance compared to traditional fine-tuning methods.

When using only 2% of the training data, prompt-tuning has improved the classification accuracy over traditional fine-tuning by 10%, 1%, and 6% on the Scalabrino, Maalej, and Pan labelled review datasets, respectively.

4.2 RQ2

RQ2 seeks to evaluate the comparative effectiveness of prompt-tuning against traditional fine-tuning techniques, under the identical experimental parameters as those described in RQ1. However, in this context, the experiments were extended to incorporate T5 and GPT-2 models, with the aim of illuminating the subtle variations in performance attributable to different language model architectures. To enhance readability, only the overall performance is summarized here; for detailed class-wise performance, refer to the appendix.

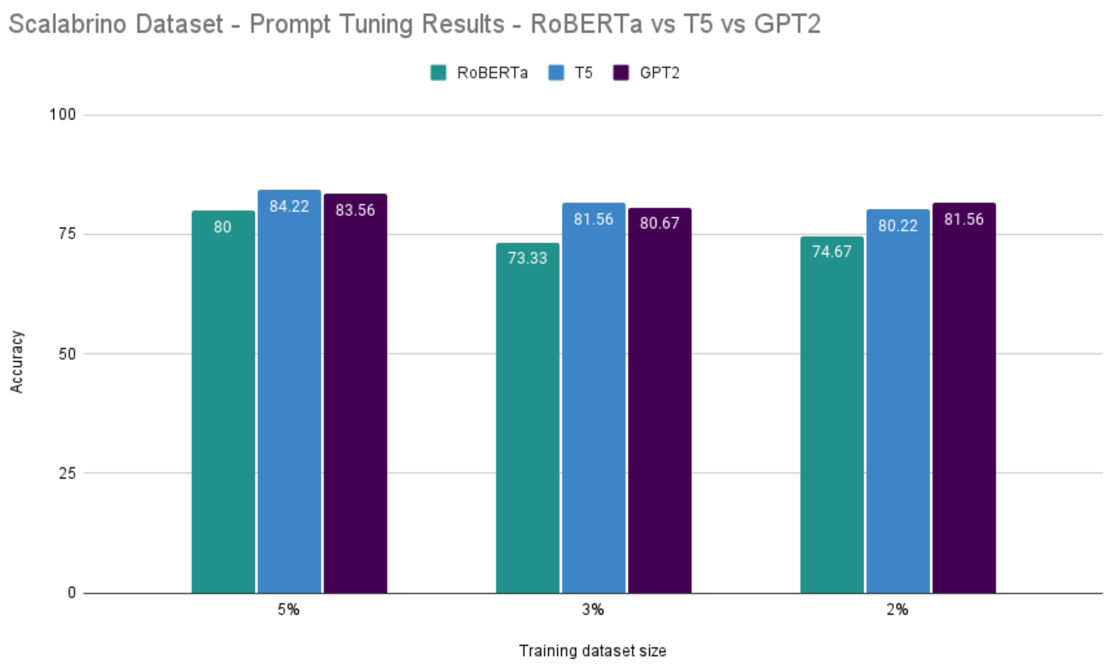

Figure 4: Scalabrino Dataset - Prompt-Tuning Results - RoBERTa vs T5 vs GPT-2

Figure 4 is a comparative analysis of prompt-tuning performance of the Scalabrino dataset across three different language models used in RQ1 and RQ2 - RoBERTa, T5, and GPT-2. This provides valuable insights into the effectiveness of each model in classifying developer-relevant information in app reviews under data-constrained conditions.

When utilizing RoBERTa as the language model, prompt-tuning achieved an accuracy of 74.67% at a 2% training dataset size. Despite exhibiting slightly lower accuracy compared to T5 and GPT-2, RoBERTa's accuracy levels remained above 70% across all dataset sizes.

Prompt-tuning with the T5 model yielded an accuracy of 80.22% at a 2% training dataset size, showcasing its effectiveness in app review classification tasks. T5 demonstrated a notable improvement in accuracy compared to RoBERTa, which suggests that T5's enhanced capability in understanding and generating natural language representations contributes to its superior performance in capturing developer-relevant information.

At a 2% training dataset size, GPT-2 achieved an accuracy of 81.56%, surpassing both RoBERTa and T5 in accuracy. GPT-2's strong performance underscores its efficacy in understanding context and generating coherent text representations, which are crucial for accurate classification of app reviews.

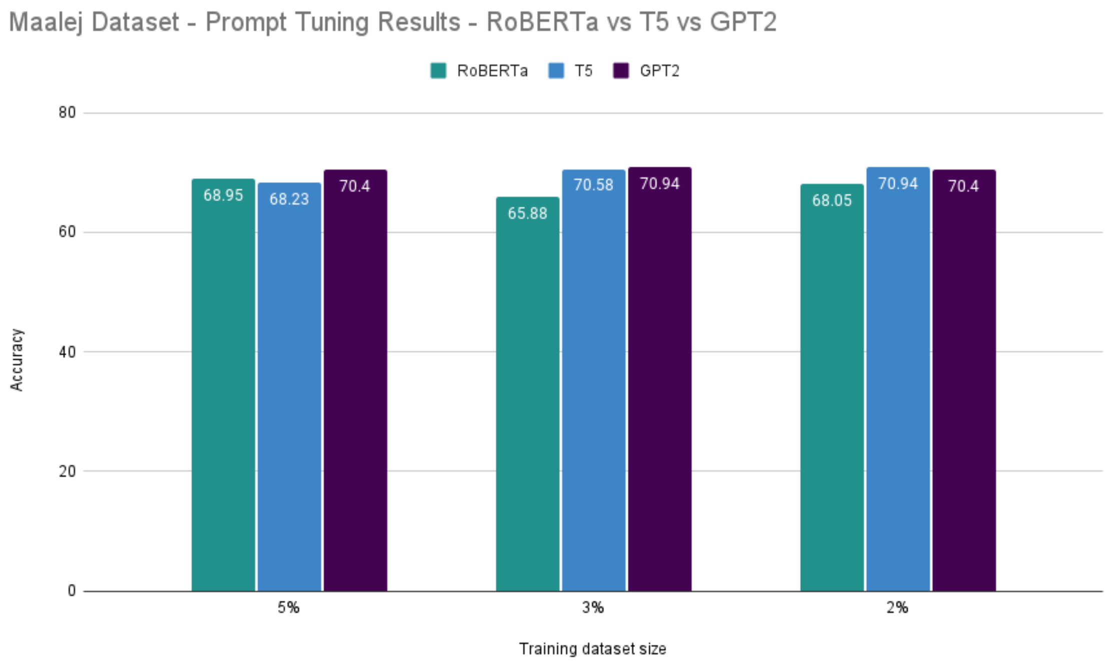

Figure 5: Maalej Dataset - Prompt-Tuning Results - RoBERTa vs T5 vs GPT-2

Figure 5 depicts the comparison of prompt-tuning performance of the Maalej dataset across three different language models RoBERTa, T5, and GPT-2 along with VB1 and PT1 under data-constrained conditions.

RoBERTa achieved an accuracy of 68.05% at a 2% training dataset size. T5 model yielded an accuracy of 70.94% at a 2% training dataset size, slight improvements in accuracy compared to RoBERTa and GPT-2. GPT-2 achieved an accuracy of 70.4% at a 2% training dataset size.

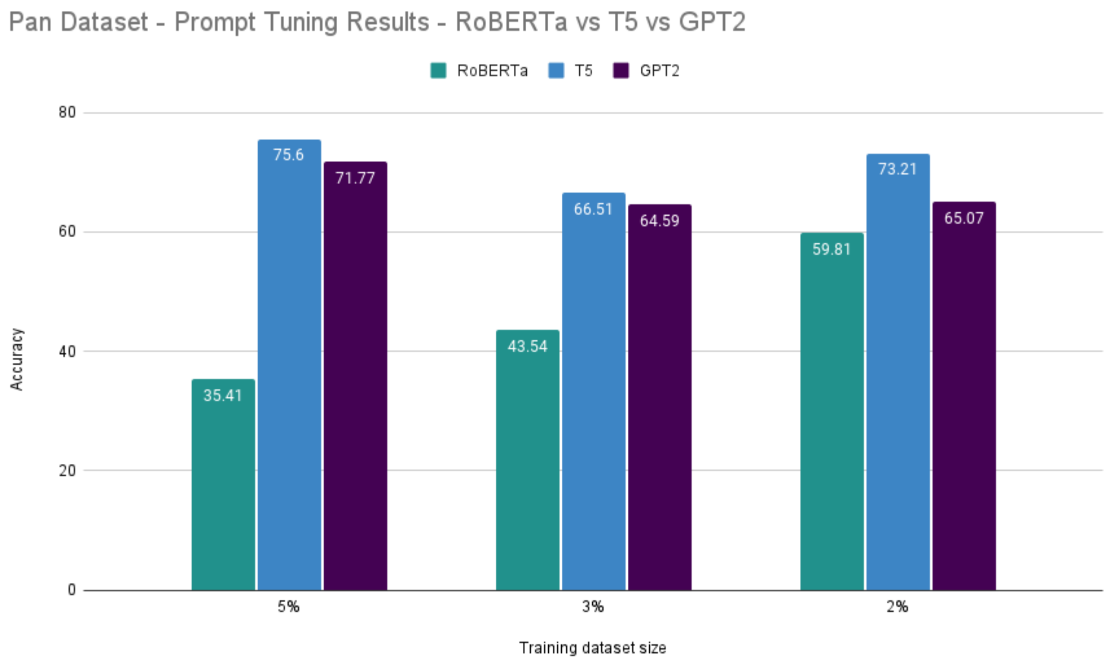

Figure 6: Pan Dataset - Prompt-Tuning Results - RoBERTa vs T5 vs GPT-2

The comparison of prompt-tuning performance using 2% training data from the Pan dataset across three different language models RoBERTa, T5, and GPT-2 along with VB1 and PT1 which depicts in Figure 6 offers valuable insights into the effectiveness of each model in classifying developer-relevant information in app reviews under data-constrained conditions.

RoBERTa achieved an accuracy of 59.81% at a 2% training dataset size, demonstrating moderate performance compared to T5 and GPT-2.

T5 model yielded the highest accuracy of 73.21% at a 2% training dataset size, showcasing its effectiveness in app review classification tasks. T5 consistently outperformed RoBERTa and GPT-2 across all training dataset sizes, indicating its superior ability to understand and generate natural language representations for accurate classification of developer-relevant information in app reviews.

GPT-2 exhibited competitive performance in prompt-tuning experiments, achieving an accuracy of 65.07% at a 2% training dataset size.

At 2% training data availability, GPT-2 performed well for the Scalabrino dataset by 6%, while T5 excelled for the Maalej and Pan datasets by 2%, and 13%, in prompt-tuning, respectively.

4.3 RQ3

RQ3 investigates how the performance of prompt-tuning models is influenced by variations in prompt template design and verbalizer design. By altering these components independently, insights into their respective impacts on the classification of developer-relevant information in app reviews are gained. For the experiments of RQ3, the best settings in RQ1 and RQ2, which produced a higher performance for each dataset at 2% training dataset size, were picked. To enhance readability, only the overall performance is summarized here; for detailed class-wise performance, refer to the appendix.

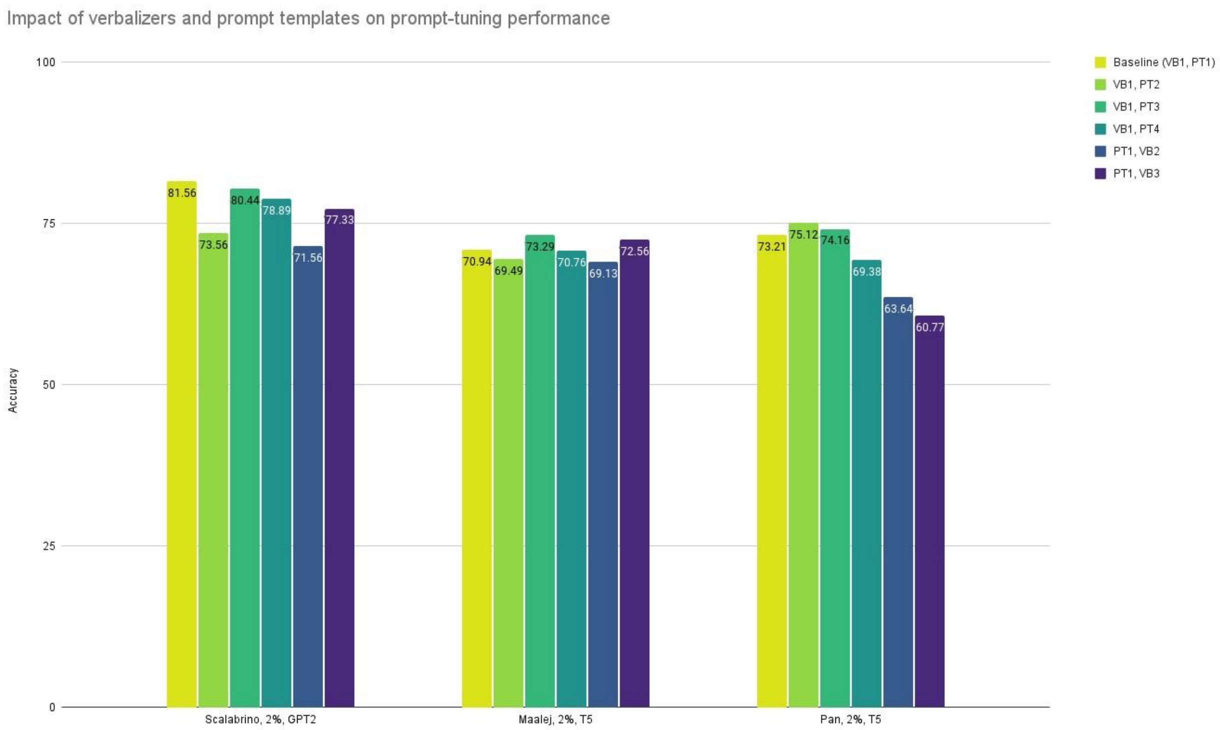

Figure 7: Impact of verbalizers and prompt templates on prompt-tuning performance

Prompt Template Variation:

In the experiments where the prompt template was varied while keeping the verbalizer constant (VB1), several observations could be made:

- For the Scalabrino dataset with a 2% training dataset size and GPT-2 model, prompt template PT3 achieved the highest accuracy of 80.44%, out of PT2 and PT4 but couldn't outperform the baseline of 81.56% of PT1. This suggests that PT1 is the most effective prompt template for Scalabrino in eliciting relevant information from app reviews.

- In the Maalej dataset with a 2% training dataset size and T5 model, prompt template PT3 also exhibited the highest accuracy of 73.29%, surpassing the baseline accuracy of 70.94%. This indicates that PT3 is the best prompt template for Maalej dataset.

- Similarly, in the Pan dataset with a 2% training dataset size and T5 model, prompt template PT2 achieved the highest accuracy of 75.12%, showcasing its superiority over the baseline accuracy of 73.21%.

Verbalizer Variation:

When the verbalizer was varied while keeping the prompt template constant (PT1), the following observations were made:

- In the Scalabrino dataset with a 2% training dataset size and GPT-2 model, verbalizer VB3 resulted in an accuracy of 77.33%, surpassing the performance of VB2, but still lower than the baseline accuracy of 81.56%. This suggests that VB1 is effective in classifying app reviews in Scalabrino dataset.

- In the Maalej dataset with a 2% training dataset size and T5 model, verbalizer VB3 achieved an accuracy of 72.56%, higher than the baseline accuracy of 70.94%. This indicates that VB3 offer improvements in projecting labels to label words compared to VB1.

- However, in the Pan dataset with a 2% training dataset size and T5 model, verbalizer VB2 resulted in an accuracy of 63.64%, substantially lower than the baseline accuracy of 73.21%. This suggests that VB1 is more suitable for this dataset.

These results illustrate the impact of prompt templates and verbalizers on the performance of prompt-tuning.

- Variations in prompt templates caused changes in accuracy of 8% for GPT-2 in the Scalabrino dataset, and 3%, 5% for T5 in the Maalej and Pan datasets, respectively.

- Variations in verbalizers resulted in changes in accuracy of 10% for GPT-2, in the Scalabrino dataset, and 4%, 14% for T5 in the Maalej and Pan datasets, respectively.

5 Discussion

This section delves into the important findings of this study.

Prompt-tuning performance of RoBERTa at 3% training size

The performance of RoBERTa at a 3% training dataset size is interesting, as it exhibits lower performance compared to the 2% dataset. This discrepancy in performance could be attributed to the quality of data available for the 3% training size, which may not have been as robust as that of the 2% dataset. This potentially introduced noise and inconsistencies into the training process, thereby making it more challenging for the model to discern relevant patterns effectively. Additionally, the examples presented for prediction may have differed from those encountered during training, further impacting the model's ability to generalize accurately.

Comparison of Traditional fine-tuning vs Prompt-tuning

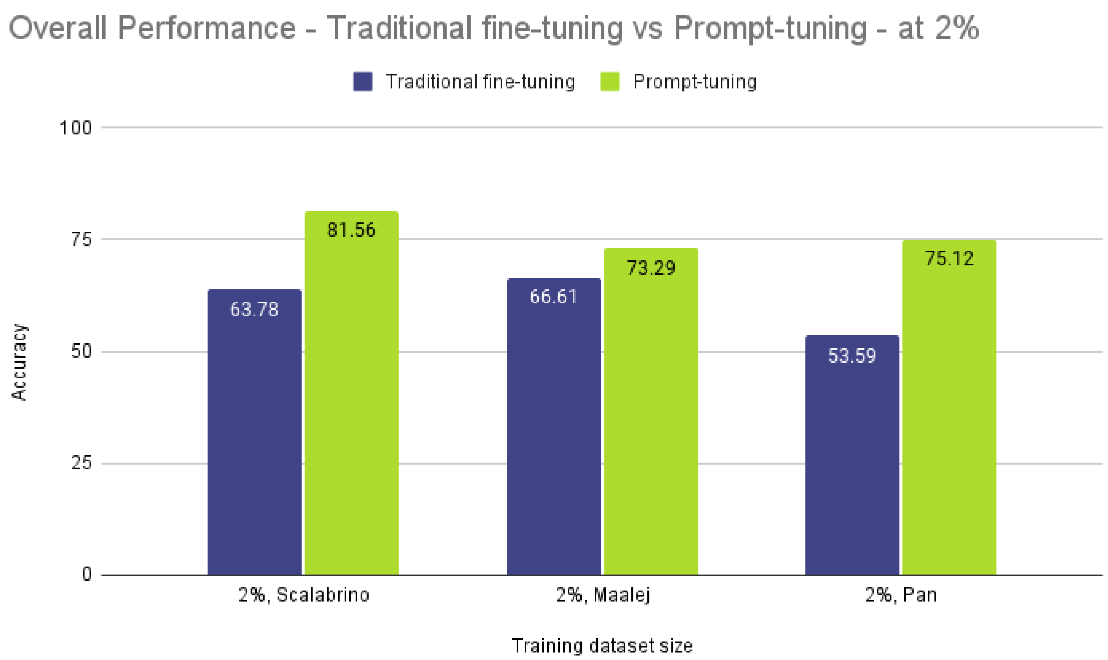

Figure 8: Overall Performance - Traditional fine-tuning vs Prompt-tuning - at 2%

The comparison between traditional fine-tuning and prompt-tuning at a 2% training dataset size illuminates the effectiveness of prompt-tuning in data-constrained scenarios across various datasets. For this comparison, the experiments that yielded the best performance for each dataset in traditional fine-tuning and prompt-tuning were selected, respectively.

For the Scalabrino dataset, traditional fine-tuning with RoBERTa achieved an accuracy of 63.78%, while prompt-tuning with GPT-2, using the VB1 and PT1, attained a significantly higher accuracy of 81.56%. This substantial performance gap highlights the advantage of prompt-tuning in effectively utilizing limited labelled data.

Similarly, in the Maalej dataset, traditional fine-tuning with RoBERTa yielded an accuracy of 66.61%, whereas prompt-tuning with T5 and the VB1 and PT3, achieved a higher accuracy of 73.29%. This also demonstrates the superior performance of prompt-tuning over traditional fine-tuning, even with minimal labelled data.

In the case of the Pan dataset, traditional fine-tuning with RoBERTa achieved an accuracy of 53.59%, whereas prompt-tuning with T5 and the VB1 and PT2 achieved a notably higher accuracy of 75.12%. Once again, prompt-tuning showcased its effectiveness in leveraging limited labelled data to achieve superior classification performance compared to traditional fine-tuning.

Overall, these results underscore the significance of prompt-tuning as a viable approach for classifying developer-relevant information in app reviews, particularly when faced with constraints on labelled data availability.

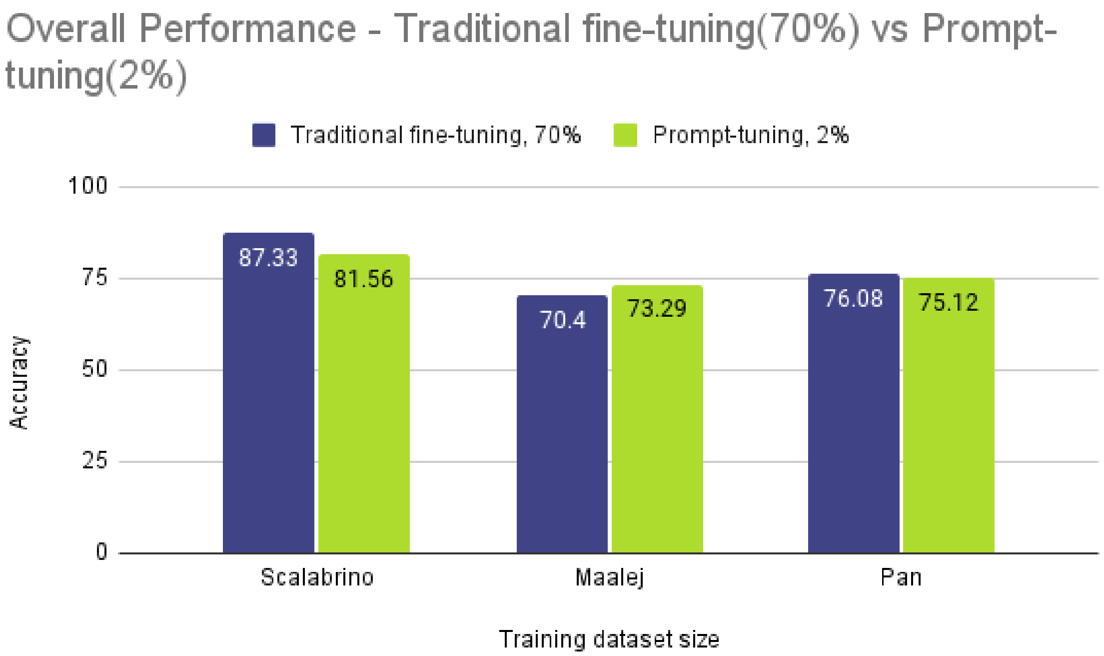

Figure 9: Overall Performance - Traditional fine-tuning (70%) vs Prompt-tuning (2%)

The comparison between the highest performances of traditional fine-tuning with 70% training data and prompt-tuning with 2% training data offers valuable insights into the effectiveness of prompt-tuning in achieving performance levels comparable to traditional fine-tuning with significantly less labelled data.

In the Scalabrino dataset, traditional fine-tuning with T5 at 70% training data achieved an accuracy of 87.33%, while prompt-tuning with GPT-2, using the VB1 prompt template, at 2% training data attained an accuracy of 81.56%. Despite the substantial difference in the amount of labelled data used, prompt-tuning managed to achieve a performance close to traditional fine-tuning, demonstrating its effectiveness in leveraging limited labelled data efficiently.

Similarly, within the Maalej dataset, traditional fine-tuning utilizing RoBERTa at 70% training data yielded an accuracy of 70.4%. Conversely, prompt-tuning employing T5 with the VB1 prompt template at 2% training data achieved a notably higher accuracy of 73.29% compared to traditional fine-tuning. This underscores the capability of prompt-tuning to outperform traditional fine-tuning, even with a mere 2% of the available training data.

In the case of the Pan dataset, traditional fine-tuning with RoBERTa at 70% training data achieved an accuracy of 76.08%, while prompt-tuning with T5 and the VB1 prompt template at 2% training data achieved a slightly lower accuracy of 75.12%. Nevertheless, the performance achieved by prompt-tuning with a fraction of the labelled data approaches that of traditional fine-tuning with a significantly larger dataset, showcasing the efficacy of prompt-tuning in maximizing performance with limited labelled data.

Overall, these results underscore the potential of prompt-tuning as a viable approach for achieving performance levels comparable to traditional fine-tuning with substantially less labelled data, thereby offering practical implications for applications where labelled data availability is limited.

Prompt-tuning class-wise performances

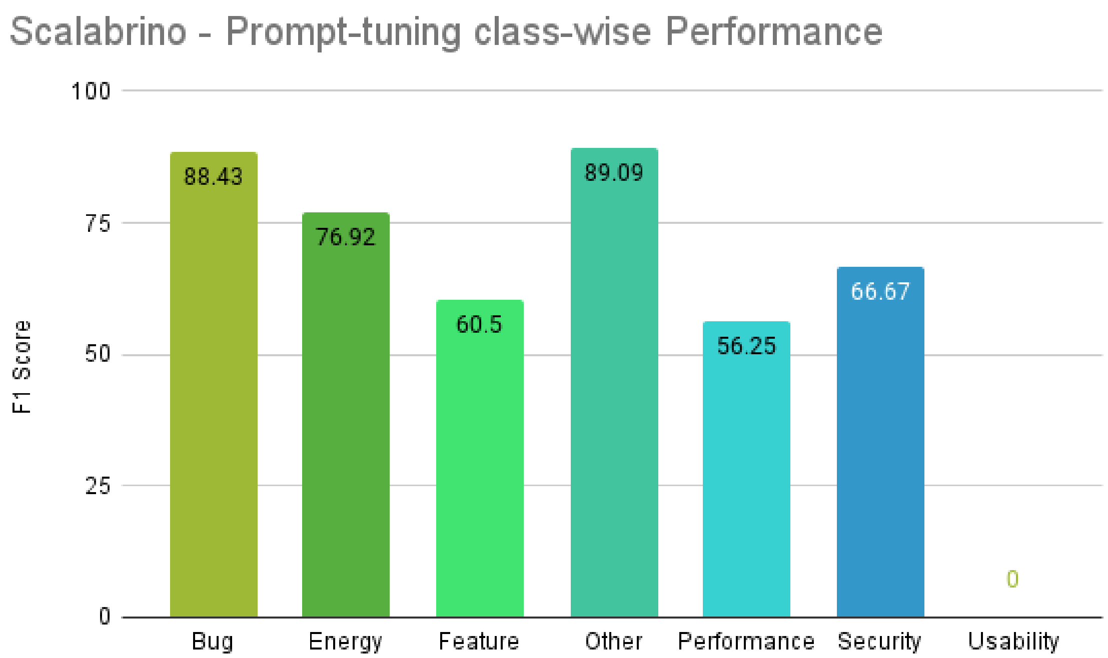

Figure 10: Scalabrino - Prompt-tuning class-wise Performance

This chart provides a snapshot of the class-wise performance of prompt-tuning when only 2% of training data is available. It illustrates the extent of performance achievable even in situations of limited data availability. Among all the few-shot prompt-tuning experiments conducted in RQ1, RQ2, and RQ3 for the Scalabrino dataset, the experiment utilizing the GPT-2 model with VB1 and PT1 settings achieved the highest performance at 2% data. This specific experiment is highlighted here to showcase the class-wise performance.

In the Scalabrino dataset, prompt-tuning achieved impressive performance across various classes. The "Other" class exhibited the highest F1 score of 89.09%, suggesting effective classification of miscellaneous reports. Similarly, the "Other" class achieved a F1 score of 88.43%, indicating the model's ability to accurately identify bug-related reports. "Energy", "Feature" and "Performance" classes achieved 76.92%, 60.5% and 56.25% of F1 scores respectively. However, the "Usability" class exhibited 0% F1 score, indicating potential challenges in accurately identifying usability-related issues with limited labelled data for this dataset.

By looking at some examples of test dataset, it can be assumed that model struggled to accurately classify reviews related to usability issues due to the language used in the reviews is similar to reviews belonging to other classes, such as Bug, Other or Feature classes, and because of the extremely low labelled examples in the training data of Usability class, leading to difficulties in learning distinguishing features and patterns for this class.

Below are few examples of prompt-tuning predictions with the predicted labels and actual labels for the above described scenario.

Review	**Actual label**	**Predicted label**
nice app this app is too good	Other	Other
fix this game needs some fixing it causes my phone to shut on and off plus it freezes my phone	Bug	Bug
cannot show amount of free internal phone memory on desire some monitoring tool	Feature	Feature
my battery seems to be draining faster since update-was one of my favorite apps what happened please fix thanks	Energy	Energy
slow very slow on my symphony w10	Performance	Performance
do not download this is a virus appdo not download	Security	Security
the same pop up ad keeps popping up it is very annoying	Usability	Bug
difficult to use this app	Usability	Other

Table 15: Scalabrino dataset - Prompt-tuning predictions

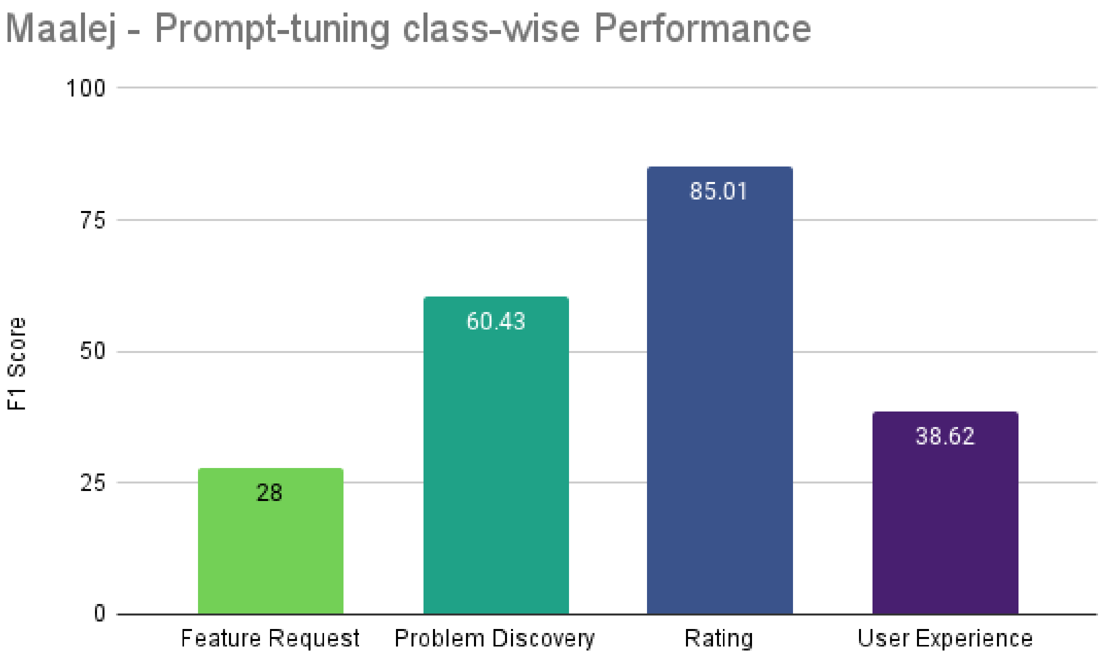

Figure 11: Maalej dataset - Prompt-tuning class-wise Performance

Among the few-shot prompt-tuning experiments carried out across RQ1, RQ2, and RQ3 for the Maalej dataset, the experiment employed the T5 model with VB1 and PT3 settings demonstrated the best performance at a 2% data availability. This particular experiment is emphasized here to illustrate the class-wise performance.

Moving to the Maalej dataset, prompt-tuning demonstrated varying levels of performance across different classes. The "Rating" class achieved the highest F1 score of 85.01%, indicating successful identification of user reviews of non-informative reviews such as praise, dispraise etc. "Problem Discovery" and "User Experience" classes scored 60.43% and 38.62% F1 scores respectively. However, the "Feature Request" class exhibited lower F1 score, suggesting potential difficulties in accurately categorizing feature-related requests with limited labelled data for Maalej dataset. The reason for the lower F1 scores for the "Feature Request" and "User Experience" classes could stem from the ambiguity in language, as reviews may not always explicitly convey the intended class.

Below are few examples of prompt-tuning predictions of Maalej dataset.

Review	**Actual label**	**Predicted label**
Great for vacations and tours Love the app	Rating	Rating
Brilliant app but crashes on iPad and iPhone	Problem Discovery	Problem Discovery
I love this app Easy to navigate and easy to post to One of my favorites	User Experience	User Experience
It is almost 2011 Where is the multitasking	Feature Request	Feature Request

This would easily be a 5 star app Best idea ever But it loses 2 stars just because I have to download one photo at a time This is a huge inconvenience	Feature Request	Problem Discovery
Great way to keep up on your crossfit progress	User Experience	Rating

Table 16: Maalej dataset - Prompt-tuning predictions

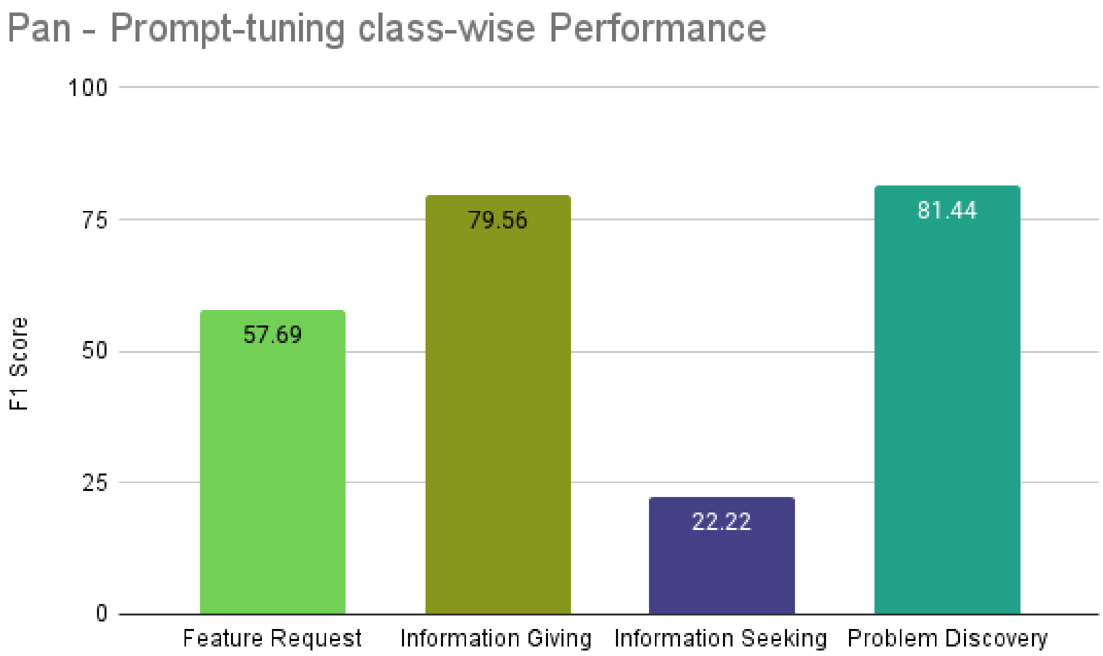

Figure 12: Pan dataset - Prompt-tuning class-wise Performance

Across the few-shot prompt-tuning experiments conducted throughout RQ1, RQ2, and RQ3 for the Pan dataset, the experiment utilizing the T5 model with VB1 and PT2 settings showcased the most superior performance with only a 2% data availability. This particular experiment is used here to illustrate the class-wise performance of Pan dataset.

In the Pan dataset, prompt-tuning showcased mixed performance across different classes. The "Problem Discovery" class exhibited the highest F1 score of 81.44%, indicating effective identification of issues or problems with the app. "Information Giving" and "Feature Request" classes achieved 79.56% and 57.69% of F1 scores respectively. However, the "Information Seeking" class achieved lower F1 score, suggesting challenges in accurately categorizing information-seeking reviews in this dataset. By looking at some examples of "Information Seeking" class, it can be assumed that the lower F1 score might be attributed to the inherent difficulty in distinguishing between requests for information and offers of information based solely on the language used in the reviews.

Below are few examples of prompt-tuning predictions of Pan dataset.

Review	Actual label	Predicted label
I love this site but the app crashes every time I use it	Problem Discovery	Problem Discovery
I planned my daughters entire baby shower using Pinterest	Information Giving	Information Giving
I just wish that you could have more secret boards	Feature Request	Feature Request
How do these things get out of the gate	Information Seeking	Information Seeking
Is there a way I can leave messages to other pinners	Information Seeking	Information Giving
Which one gets you to the list of categories the home magnifying glass plus sign or person	Information Seeking	Information Giving
Can t figure out how to clean up my boards	Information Seeking	Problem Discovery

Table 17: Pan dataset - Prompt-tuning predictions

Overall, the class-wise performance analysis highlights the potential of prompt-tuning in effectively classifying developer-relevant information across various categories, even with a limited amount of labelled data. However, it also underscores the importance of further research and optimization to address challenges in accurately identifying certain classes, particularly those with lower F1 scores.

6 Conclusion

This study explores the effectiveness of prompt-tuning methodologies for classifying developer-relevant information in app reviews, particularly in scenarios with limited labelled data. Through three research questions, prompt-tuning was systematically compared with traditional fine-tuning across various datasets and experimental conditions.

In RQ1, prompt-tuning's performance was assessed against traditional fine-tuning across different datasets and training dataset sizes. Prompt-tuning consistently outperformed traditional methods, particularly when labelled data was scarce, achieving notable improvements in classification accuracy across all datasets examined. Expanding upon the insights gained from RQ1, RQ2 compared prompt-tuning across different language model architectures, reaffirming its superior performance, particularly with T5 and GPT-2 models. RQ3 provided insights into the impact of prompt template design and verbalizer design on prompt-tuning's performance. The findings revealed that both prompt templates and verbalizers play crucial roles in influencing the effectiveness of prompt-tuning, with certain settings yielding superior classification performance across different datasets.

In summary, few-shot prompt-tuning emerges as a practical solution for effectively classifying developer-relevant information in app reviews, particularly in scenarios with limited labelled data. The consistent performance improvements across various datasets and experimental conditions highlight the potential of prompt-tuning for real-world applications where labelled data is scarce. In the ever-evolving app marketplace, where new categories continuously emerge and labelled data for these new apps is limited, few-shot prompt-tuning methods offer a promising approach to classify app reviews effectively. By fine-tuning language models with carefully designed prompt templates and verbalizers, few-shot prompt-tuning facilitates efficient categorization of app reviews into distinct classes, even in scenarios with limited labelled data.

Future research in app review classification using prompt-tuning could focus on optimizing performance by exploring these key areas:

1. **Continuous Prompting:** Investigate the effectiveness of continuous prompting approaches, which involve training the model on a continuous stream of prompts with ongoing updates and adjustments.
2. **Tuning-Free Prompting:** Evaluating tuning-free prompting strategy and compare its performance over few-shot prompt-tuning.
3. **Automatic Prompt Template Design:** Apply techniques for automatic prompt template design and eliminate the dependency of human effort in prompt template design.

By advancing research in these areas, the efficiency and effectiveness of prompt-tuning methodologies can be achieved for app review classification, ultimately enabling better extraction of insights from app review data.

Code related to this study is publicly available at https://github.com/hashikadhananjanie/few-shot-prompting-app-review-classification

Acknowledgement

First and foremost, I extend my heartfelt thanks to my supervisor, Faiz Ali Shah, for his invaluable support, guidance, and encouragement throughout the journey and for being flexible with the arrangements. His expertise and mentorship have been instrumental in shaping the direction of my study.

I am deeply thankful to the University of Tartu for providing me with the necessary resources and facilities to conduct this study. The research environment has greatly facilitated my work and enabled to achieve my objectives.

Furthermore, I extend my appreciation to all the lecturers at University of Tartu for sharing their knowledge through these two years.

Lastly, I express my gratitude to my family, cousins, friends, and colleagues for their unwavering support throughout the duration of this endeavour.

Thank you to everyone who has played a part, no matter how big or small, in the realization of this study.

References

[1] F. Rustam, A. Mehmood, M. Ahmad, S. Ullah, D. M. Khan, and G. S. Choi, "Classification of shopify app user reviews using novel multi text features," *IEEE Access*, vol. 8, pp. 30234–30244, 2020.

[2] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–35, Sep. 2023, doi: 10.1145/3560815.

[3] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "ARdoc: app reviews development oriented classifier," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Seattle WA USA: ACM, Nov. 2016, pp. 1023–1027. doi: 10.1145/2950290.2983938.

[4] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*, IEEE, 2015, pp. 281–290.

[5] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd international requirements engineering conference (RE)*, Ieee, 2014, pp. 153–162.

[6] Y. Chang *et al.*, "A Survey on Evaluation of Large Language Models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, Jun. 2024, doi: 10.1145/3641289.

[7] M. Kale and A. Rastogi, "Text-to-Text Pre-Training for Data-to-Text Tasks." arXiv, Jul. 08, 2021.

[8] C. Garbacea, S. Carton, S. Yan, and Q. Mei, "Judge the Judges: A Large-Scale Evaluation Study of Neural Language Models for Online Review Generation." arXiv, Sep. 05, 2019.

[9] H. Luo, "Prompt-learning and Zero-shot Text Classification with Domain-specific Textual Data." 2023.

[10] C. W. F. Mayer, S. Ludwig, and S. Brandt, "Prompt text classifications with transformer models! An exemplary introduction to prompt-based learning with large language models," *J. Res. Technol. Educ.*, vol. 55, no. 1, pp. 125–141, Jan. 2023, doi: 10.1080/15391523.2022.2142872.

[11] W. Jin, Y. Cheng, Y. Shen, W. Chen, and X. Ren, "A Good Prompt Is Worth Millions of Parameters: Low-resource Prompt-based Learning for Vision-Language Models." arXiv, Mar. 14, 2022.

[12] Y. Ge, Y. Guo, Y.-C. Yang, M. A. Al-Garadi, and A. Sarker, "Few-shot learning for medical text: A systematic review." arXiv, Apr. 21, 2022.

[13] A. Parnami and M. Lee, "Learning from Few Examples: A Summary of Approaches to Few-Shot Learning." arXiv, Mar. 07, 2022.

[14] Y. Ma, S. Luo, Y.-M. Shang, Z. Li, and Y. Liu, “A novel prompt-tuning method: Incorporating scenario-specific concepts into a verbalizer,” *Expert Syst. Appl.*, vol. 247, p. 123204, Aug. 2024, doi: 10.1016/j.eswa.2024.123204.

[15] G. Cui, S. Hu, N. Ding, L. Huang, and Z. Liu, “Prototypical Verbalizer for Prompt-based Few-shot Tuning,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7014–7024. doi: 10.18653/v1/2022.acl-long.483.

[16] Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein and Sameer Singh, “Calibrate Before Use: Improving Few-Shot Performance of Language Models”, https://doi.org/10.48550/arXiv.2102.09690

[17] Flor Miriam Plaza-del-arco, Debora Nozza, and Dirk Hovy, “Respectful or Toxic? Using Zero-Shot Learning with Language Models to Detect Hate Speech. ”
In *The 7th Workshop on Online Abuse and Harms (WOAH)*, pages 60–68, Toronto, Canada. Association for Computational Linguistics, 2023

[18] Ž. Vujović, “Classification model evaluation metrics,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, pp. 599–606, 2021.

[19] Ding, Ning, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun (2021). “OpenPrompt: An Open-source Framework for Prompt-learning”, https://doi.org/10.48550/arXiv.2111.01998

[20] S. Scalabrino, G. Bavota, B. Russo, M. D. Penta and R. Oliveto, "Listening to the Crowd for the Release Planning of Mobile Apps," in *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 68-86, 1 Jan. 2019, doi: 10.1109/TSE.2017.2759112.

[21] P. Devine, Y. S. Koh, and K. Blincoe, “Evaluating software user feedback classifier performance on unseen apps, datasets, and metadata,” *Empir. Softw. Eng.*, vol. 28, no. 2, p. 26, Mar. 2023, doi: 10.1007/s10664-022-10254-y.

[22] Maalej, W., Kurtanović, Z., Nabil, H. *et al.* "On the automatic classification of app reviews. " *Requirements Eng* **21**, 311–331 (2016). https://doi.org/10.1007/s00766-016-0251-9

[23] Al-hawari, assem (2019), “A dataset of Mobile application reviews for classifying reviews into software Engineering's maintenance tasks using data mining techniques”, Mendeley Data, V2, doi: 10.17632/5fk732vkwr.2

[24] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). "Roberta: A robustly optimized bert pretraining approach." *arXiv preprint arXiv:1907.11692.*

[25] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). "Exploring the limits of transfer learning with a unified text-to-text transformer." *Journal of machine learning research*, *21*(140), 1-67.

[26] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). "Language models are unsupervised multitask learners." *OpenAI blog*, *1*(8), 9.

Appendix

I. RQ1 Results

Experiment	Class	Traditional fine-tuning results				Prompt-tuning results			
		Prec	Rec	F1	Acc	Prec	Rec	F1	Acc
RQ1_MD1_DS1_VB1_PT1_EX1	Bug	85.95	90.43	88.14	87.11	90	86.09	88	86
	Energy	93.75	93.75	93.75		62.5	93.75	75	
	Feature	78.72	74	76.29		70.91	78	74.29	
	Other	91.77	93.81	92.78		91.63	92.04	91.83	
	Performance	80	60	68.57		81.82	45	58.06	
	Security	75	85.71	80		87.5	100	93.33	
	Usability	50	37.5	42.86		66.67	62.5	64.52	
RQ1_MD1_DS1_VB1_PT1_EX2	Bug	84.68	81.74	83.19	73.78	77.94	92.17	84.46	80
	Energy	100	18.75	31.58		85.71	75	80	
	Feature	38.46	50	43.48		65.62	42	51.22	
	Other	77.41	92.48	84.27		82.07	91.15	86.37	
	Performance	100	5	9.52		88.89	40	55.17	
	Security	0	0	0		100	57.14	72.73	
	Usability	0	0	0		75	18.75	30	
RQ1_MD1_DS1_VB1_PT1_EX3	Bug	67.74	91.3	77.78	74.89	61.4	91.3	73.43	73.33
	Energy	0	0	0		100	31.25	47.62	
	Feature	61.36	54	57.45		60	36	45	
	Other	81.67	90.71	85.95		82.5	87.61	84.98	
	Performance	0	0	0		100	20	33.33	
	Security	0	0	0		0	0	0	
	Usability	0	0	0		0	0	0	
RQ1_MD1_DS1_VB1_PT1_EX4	Bug	64.35	64.35	64.35	63.78	79.82	75.65	77.68	74.67
	Energy	0	0	0		100	81.25	89.66	
	Feature	0	0	0		48.89	44	46.32	
	Other	63.58	94.25	75.94		79.3	89.82	84.23	
	Performance	0	0	0		31.82	35	33.33	
	Security	0	0	0		75	42.86	54.55	
	Usability	0	0	0		100	6.25	11.76	
RQ1_MD1_DS2_VB1_PT1_EX1	Feature Request	33.33	21.05	25.81	70.4	0	0	0	66.61
	Problem Discovery	51.76	78.57	62.41		0	0	0	
	Rating	81.7	83.47	82.57		66.61	100	79.96	
	User Experience	44.12	32.97	37.74		0	0	0	
RQ1_MD1_DS2_VB1_PT1_EX2	Feature Request	0	0	0	66.61	30.23	34.21	32.1	68.95
	Problem Discovery	63.64	25	35.9		75	5.36	10	
	Rating	79.61	87.8	83.51		75.72	92.14	83.13	
	User Experience	24.8	34.07	28.7		44.83	28.57	34.9	
RQ1_MD1_DS2_VB1_PT1_EX3	Feature Request	0	0	0	66.61	0	0	0	65.88
	Problem Discovery	0	0	0		8.33	1.79	2.94	
	Rating	66.61	100	79.96		67.28	98.64	80	
	User Experience	0	0	0		0	0	0	
RQ1_MD1_DS2_VB1_PT1_EX4	Feature Request	0	0	0	66.61	0	0	0	68.05
	Problem Discovery	0	0	0		46.88	26.79	34.09	
	Rating	66.61	100	79.96		69.62	98.1	81.44	
	User Experience	0	0	0		0	0	0	
RQ1_MD1_DS3_VB1_PT1_EX1	Feature Request	59.09	44.83	50.98	76.08	62.5	17.24	27.03	73.68
	Information Giving	76.47	85.71	80.83		70.27	85.71	77.23	
	Information Seeking	83.33	33.33	47.62		77.78	46.67	58.33	
	Problem Discovery	79.75	85.14	82.35		79.01	86.49	82.58	
RQ1_MD1_DS3_VB1_PT1_EX2	Feature Request	0	0	0	61.24	0	0	0	35.41
	Information Giving	68	74.73	71.2		0	0	0	
	Information Seeking	0	0	0		0	0	0	
	Problem Discovery	55.05	81.08	65.57		35.41	100	52.3	
RQ1_MD1_DS3_VB1_PT1_EX3	Feature Request	0	0	0	43.54	0	0	0	43.54
	Information Giving	43.54	100	60.67		43.54	100	60.67	
	Information Seeking	0	0	0		0	0	0	
	Problem Discovery	0	0	0		0	0	0	
RQ1_MD1_DS3_VB1_PT1_EX4	Feature Request	0	0	0	53.59	18.18	6.9	10	59.81
	Information Giving	54.26	76.92	63.64		56.93	85.71	68.42	
	Information Seeking	0	0	0		0	0	0	
	Problem Discovery	52.5	56.76	54.55		75	60.81	67.16	

Table 18: RQ1 results

II. RQ2 Results

Experiment	Class	Traditional fine-tuning results				Prompt-tuning results			
		Prec	Rec	F1	Acc	Prec	Rec	F1	Acc
RQ2_MD2_DS1_VB1_PT1_EX1	Bug	88.89	90.43	89.66	87.33	86.4	93.91	90	86.22
	Energy	88.24	93.75	90.91		88.24	93.75	90.91	
	Feature	73.58	78	75.73		78.57	66	71.74	
	Other	90.38	95.58	92.9		88.61	92.92	90.71	
	Performance	91.67	55	68.75		78.57	55	64.71	
	Security	100	57.14	72.73		77.78	100	87.5	
	Usability	50	25	33.33		66.67	25	36.36	
RQ2_MD2_DS1_VB1_PT1_EX2	Bug	30	2.61	4.8	50.44	83.08	93.91	88.16	84.22
	Energy	0	0	0		93.75	93.75	93.75	
	Feature	0	0	0		64	64	64	
	Other	50.91	99.12	67.27		89.32	92.48	90.87	
	Performance	0	0	0		77.78	35	48.28	
	Security	0	0	0		71.43	71.43	71.43	
	Usability	0	0	0		75	18.75	30	
RQ2_MD2_DS1_VB1_PT1_EX3	Bug	29.73	19.13	23.28	50.44	83.74	89.57	86.55	81.56
	Energy	0	0	0		60	93.75	73.17	
	Feature	0	0	0		62	62	62	
	Other	54.52	90.71	68.11		87.39	88.94	88.16	
	Performance	0	0	0		88.89	40	55.17	
	Security	0	0	0		77.78	100	87.5	
	Usability	0	0	0		50	12.5	20	
RQ2_MD2_DS1_VB1_PT1_EX4	Bug	22.86	48.7	31.11	35.56	84.82	82.61	83.7	80.22
	Energy	0	0	0		100	81.25	89.66	
	Feature	0	0	0		50	62	55.36	
	Other	51.74	46.02	48.71		85.54	91.59	88.46	
	Performance	0	0	0		88.89	40	55.17	
	Security	0	0	0		83.33	71.43	76.92	
	Usability	0	0	0		33.33	12.5	18.18	
RQ2_MD2_DS2_VB1_PT1_EX1	Feature Request	0	0	0	70.04	50	7.89	13.64	70.94
	Problem Discovery	50.67	67.86	58.02		62.22	50	55.45	
	Rating	77.44	90.24	83.35		72.75	96.21	82.85	
	User Experience	36.17	18.68	24.64		46.67	7.69	13.21	
RQ2_MD2_DS2_VB1_PT1_EX2	Feature Request	0	0	0	66.61	30.77	21.05	25	68.23
	Problem Discovery	0	0	0		53.52	67.86	59.84	
	Rating	66.61	100	79.96		80	83.47	81.7	
	User Experience	0	0	0		33.33	26.37	29.45	
RQ2_MD2_DS2_VB1_PT1_EX3	Feature Request	0	0	0	66.61	50	5.26	9.52	70.58
	Problem Discovery	0	0	0		55.36	55.36	55.36	
	Rating	66.61	100	79.96		77.04	94.58	84.91	
	User Experience	0	0	0		21.95	9.89	13.64	
RQ2_MD2_DS2_VB1_PT1_EX4	Feature Request	0	0	0	66.61	40	5.26	9.3	70.94
	Problem Discovery	0	0	0		52.94	64.29	58.06	
	Rating	66.61	100	79.96		75.75	95.66	84.55	
	User Experience	0	0	0		13.33	2.2	3.77	
RQ2_MD2_DS3_VB1_PT1_EX1	Feature Request	66.67	55.17	60.38	75.6	58.97	79.31	67.65	79.43
	Information Giving	71.68	89.01	79.41		86.9	80.22	83.43	
	Information Seeking	83.33	33.33	47.62		88.89	53.33	66.67	
	Problem Discovery	84.85	75.68	80		80.52	83.78	82.12	
RQ2_MD2_DS3_VB1_PT1_EX2	Feature Request	0	0	0	44.98	55.88	65.52	60.32	75.6

	Information Giving	44.22	96.7	60.69		81.82	79.12	80.45	
	Information Seeking	0	0	0		66.67	13.33	22.22	
	Problem Discovery	60	8.11	14.29		77.38	87.84	82.28	
RQ2_MD2_DS3_VB1_PT1_EX3	Feature Request	0	0	0	44.02	71.43	17.24	27.78	66.51
	Information Giving	43.84	97.8	60.54		85.33	70.33	77.11	
	Information Seeking	0	0	0		0	0	0	
	Problem Discovery	60	4.05	7.59		55.12	94.59	69.65	
RQ2_MD2_DS3_VB1_PT1_EX4	Feature Request	4.76	3.45	4	38.76	69.23	31.03	42.86	73.21
	Information Giving	42.08	84.62	56.2		77.08	81.32	79.14	
	Information Seeking	0	0	0		50	6.67	11.76	
	Problem Discovery	60	4.05	7.59		70.41	93.24	80.23	
RQ2_MD3_DS1_VB1_PT1_EX1	Bug	87.18	88.7	87.93	85.11	92.38	84.35	88.18	83.78
	Energy	100	87.5	93.33		86.67	81.25	83.87	
	Feature	63.16	72	67.29		62.71	74	67.89	
	Other	88.66	93.36	90.95		85.26	94.69	89.73	
	Performance	84.62	55	66.67		100	30	46.15	
	Security	100	57.14	72.73		85.71	85.71	85.71	
	Usability	71.43	31.25	43.48		57.14	25	34.78	
RQ2_MD3_DS1_VB1_PT1_EX2	Bug	61.06	60	60.53	60.67	85.37	91.3	88.24	83.56
	Energy	0	0	0		100	81.25	89.66	
	Feature	24.62	32	27.83		57.14	72	63.72	
	Other	69.37	83.19	75.65		90.13	88.94	89.53	
	Performance	0	0	0		71.43	50	58.82	
	Security	0	0	0		83.33	71.43	76.92	
	Usability	0	0	0		75	37.5	50	
RQ2_MD3_DS1_VB1_PT1_EX3	Bug	65.31	55.65	60.09	59.56	82.68	91.3	86.78	80.67
	Energy	0	0	0		90.91	62.5	74.07	
	Feature	18.75	12	14.63		67.74	42	51.85	
	Other	62.07	87.61	72.66		84.02	90.71	87.23	
	Performance	0	0	0		60.87	70	65.12	
	Security	0	0	0		83.33	71.43	76.92	
	Usability	0	0	0		37.5	18.75	25	
RQ2_MD3_DS1_VB1_PT1_EX4	Bug	62.16	60	61.06	60.67	84.25	93.04	88.43	81.56
	Energy	0	0	0		65.22	93.75	76.92	
	Feature	10.53	4	5.8		52.17	72	60.5	
	Other	63.32	89.38	74.13		91.59	86.73	89.09	
	Performance	0	0	0		75	45	56.25	
	Security	0	0	0		80	57.14	66.67	
	Usability	0	0	0		0	0	0	
RQ2_MD3_DS2_VB1_PT1_EX1	Feature Request	50	5.26	9.52	67.87	0	0	0	71.48
	Problem Discovery	45.16	75	56.38		56.67	60.71	58.62	
	Rating	78.64	84.82	81.62		78.93	88.35	83.38	
	User Experience	32.2	20.88	25.33		45	39.56	42.11	
RQ2_MD3_DS2_VB1_PT1_EX2	Feature Request	20	15.79	17.65	61.19	25	7.89	12	70.4
	Problem Discovery	20	1.79	3.28		46.32	78.57	58.28	
	Rating	70.13	85.91	77.22		80.9	87.26	83.96	
	User Experience	22.39	16.48	18.99		42.86	23.08	30	
RQ2_MD3_DS2_VB1_PT1_EX3	Feature Request	13.21	18.42	15.38	57.94	0	0	0	70.94
	Problem Discovery	16.67	1.79	3.23		47.06	71.43	56.74	
	Rating	69.28	81.3	74.81		76.82	94.31	84.67	
	User Experience	20.97	14.29	16.99		31.25	5.49	9.35	
RQ2_MD3_DS2_VB1_PT1_EX4	Feature Request	11.7	28.95	16.67	53.97	0	0	0	70.4
	Problem Discovery	0	0	0		71.88	41.07	52.27	
	Rating	67.38	76.69	71.74		72.69	95.93	82.71	

	User Experience	14.29	5.49	7.94		38.24	14.29	20.8	
RQ2_MD3_DS3_VB1_PT1_EX1	Feature Request	50	27.59	35.56	68.9	75	31.03	43.9	72.25
	Information Giving	77.27	74.73	75.98		63.78	89.01	74.31	
	Information Seeking	75	40	52.17		80	26.67	40	
	Problem Discovery	63.92	83.78	72.51		87.69	77.03	82.01	
RQ2_MD3_DS3_VB1_PT1_EX2	Feature Request	25	34.48	28.99	35.89	100	10.34	18.75	71.77
	Information Giving	50.77	36.26	42.31		82.02	80.22	81.11	
	Information Seeking	10.71	20	13.95		66.67	26.67	38.1	
	Problem Discovery	38.16	39.19	38.67		63.06	94.59	75.68	
RQ2_MD3_DS3_VB1_PT1_EX3	Feature Request	19.18	48.28	27.45	20.1	50	10.34	17.14	64.59
	Information Giving	40	2.2	4.17		61.02	79.12	68.9	
	Information Seeking	3.9	20	6.52		50	6.67	11.76	
	Problem Discovery	42.59	31.08	35.94		71.08	79.73	75.16	
RQ2_MD3_DS3_VB1_PT1_EX4	Feature Request	18.42	48.28	26.67	16.27	50	3.45	6.45	65.07
	Information Giving	0	0	0		60.32	83.52	70.05	
	Information Seeking	5.68	33.33	9.71		50	20	28.57	
	Problem Discovery	33.33	20.27	25.21		74.67	75.68	75.17	

Table 19: RQ2 results

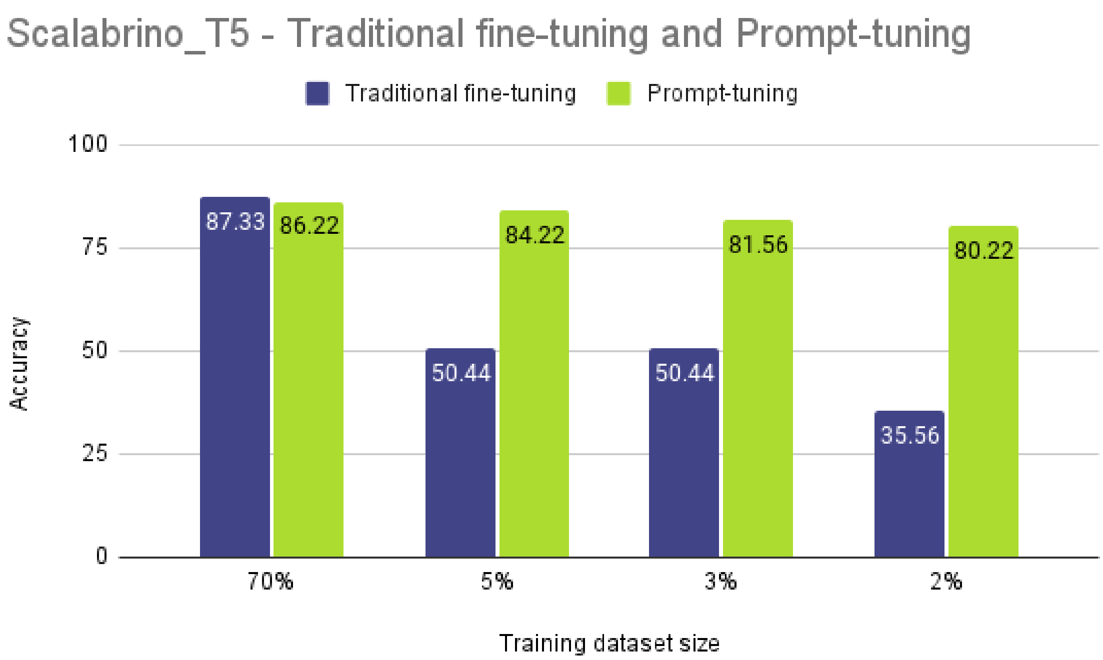

Figure 13: Comparison of T5 model performance between traditional fine-tuning and prompt-tuning on Scalabrino dataset

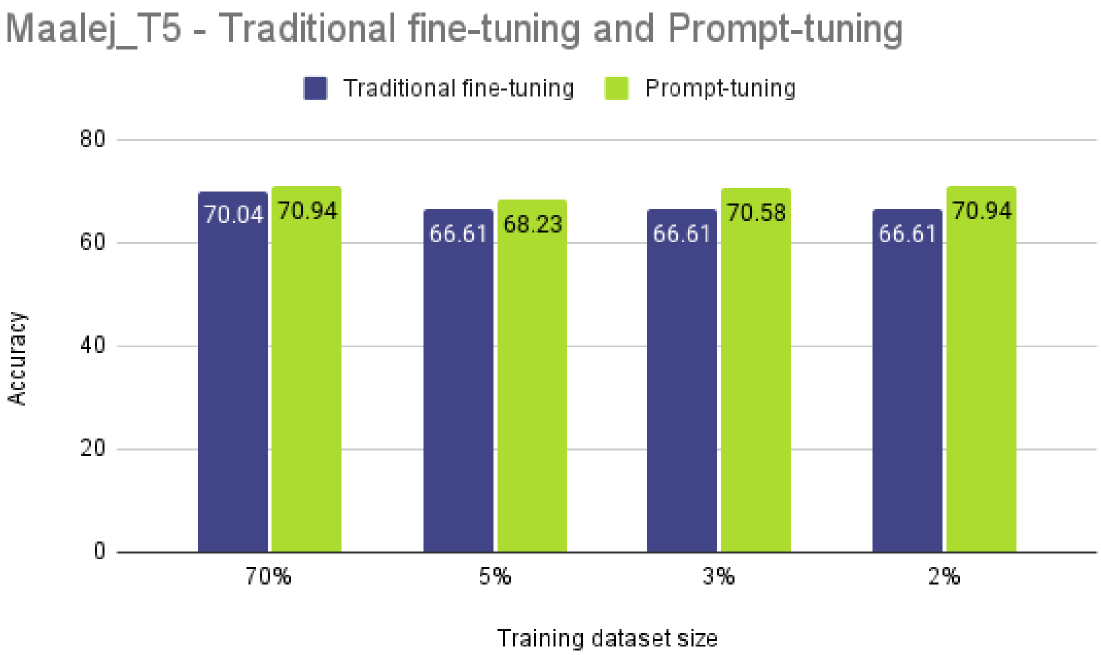

Figure 14: Comparison of T5 model performance between traditional fine-tuning and prompt-tuning on Maalej dataset

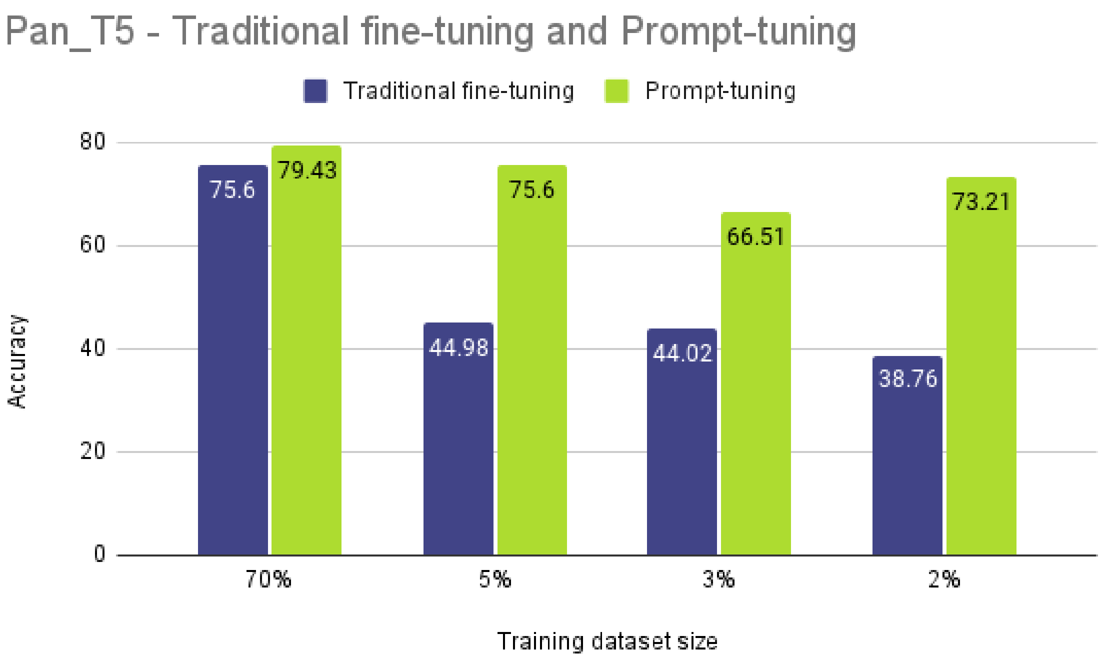

Figure 15: Comparison of T5 model performance between traditional fine-tuning and prompt-tuning on Pan dataset

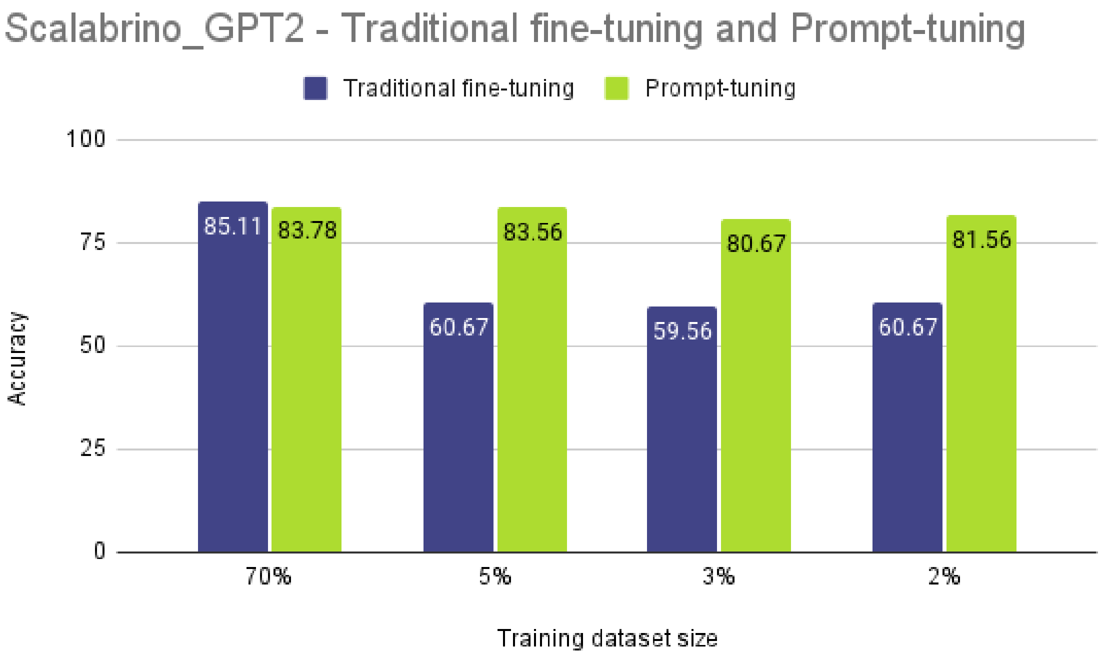

Figure 16: Comparison of GPT-2 model performance between traditional fine-tuning and prompt-tuning on Scalabrino dataset

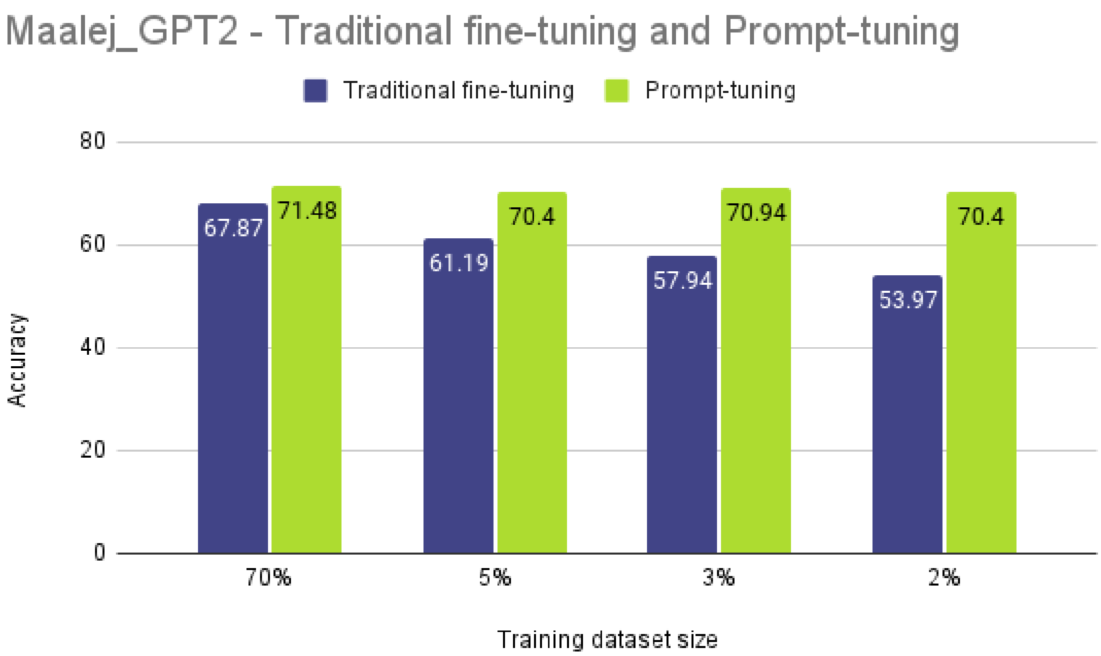

Figure 17: Comparison of GPT-2 model performance between traditional fine-tuning and prompt-tuning on Maalej dataset

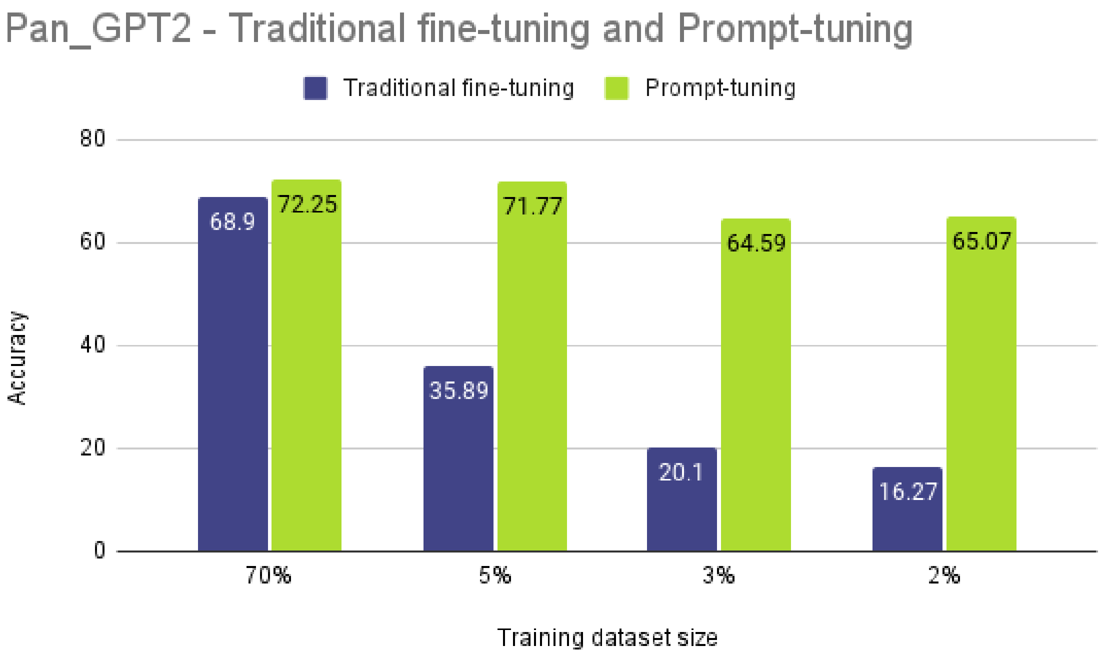

Figure 18: Comparison of GPT-2 model performance between traditional fine-tuning and prompt-tuning on Pan dataset

III. RQ3 Results

Experiment	Class	Prompt-tuning results			
		Prec	Rec	F1	Acc
RQ3_MD2_DS2_VB1_PT2_EX4	Feature Request	28.21	28.95	28.57	69.49
	Problem Discovery	58.33	50	53.85	
	Rating	78.85	88.89	83.57	
	User Experience	35.29	19.78	25.35	
RQ3_MD2_DS2_VB1_PT3_EX4	Feature Request	58.33	18.42	28	73.29
	Problem Discovery	50.6	75	60.43	
	Rating	81.23	89.16	85.01	
	User Experience	51.85	30.77	38.62	
RQ3_MD2_DS2_VB1_PT4_EX4	Feature Request	20	2.63	4.65	70.76
	Problem Discovery	57.63	60.71	59.13	
	Rating	75.97	95.12	84.48	
	User Experience	21.43	6.59	10.08	
RQ3_MD2_DS2_VB2_PT1_EX4	Feature Request	57.14	10.53	17.78	69.13
	Problem Discovery	65.52	33.93	44.71	
	Rating	75.5	91.87	82.89	
	User Experience	30.43	23.08	26.25	
RQ3_MD2_DS2_VB3_PT1_EX4	Feature Request	33.33	5.26	9.09	72.56
	Problem Discovery	62.5	53.57	57.69	
	Rating	76.2	94.58	84.4	
	User Experience	50	23.08	31.58	
RQ3_MD2_DS3_VB1_PT2_EX4	Feature Request	65.22	51.72	57.69	75.12
	Information Giving	80	79.12	79.56	
	Information Seeking	66.67	13.33	22.22	
	Problem Discovery	73.12	91.89	81.44	
RQ3_MD2_DS3_VB1_PT3_EX4	Feature Request	90	31.03	46.15	74.16
	Information Giving	75	85.71	80	
	Information Seeking	0	0	0	
	Problem Discovery	71.58	91.89	80.47	
RQ3_MD2_DS3_VB1_PT4_EX4	Feature Request	100	3.45	6.67	69.38
	Information Giving	70.09	82.42	75.76	
	Information Seeking	100	6.67	12.5	
	Problem Discovery	68	91.89	78.16	
RQ3_MD2_DS3_VB2_PT1_EX4	Feature Request	50	10.34	17.14	63.64
	Information Giving	65.38	74.73	69.74	
	Information Seeking	0	0	0	
	Problem Discovery	62.63	83.78	71.68	
RQ3_MD2_DS3_VB3_PT1_EX4	Feature Request	0	0	0	60.77
	Information Giving	65.59	67.03	66.3	
	Information Seeking	33.33	6.67	11.11	
	Problem Discovery	64.36	87.84	74.29	
RQ3_MD3_DS1_VB1_PT2_EX4	Bug	86.96	69.57	77.29	73.56
	Energy	78.57	68.75	73.33	
	Feature	41.03	64	50	
	Other	78.46	85.4	81.78	
	Performance	69.23	45	54.55	
	Security	80	57.14	66.67	
	Usability	100	12.5	22.22	
RQ3_MD3_DS1_VB1_PT3_EX4	Bug	80.95	88.7	84.65	80.44
	Energy	81.25	81.25	81.25	
	Feature	47.83	66	55.46	
	Other	89.24	88.05	88.64	
	Performance	100	35	51.85	
	Security	85.71	85.71	85.71	
	Usability	100	12.5	22.22	
RQ3_MD3_DS1_VB1_PT4_EX4	Bug	93.68	77.39	84.76	78.89
	Energy	100	62.5	76.92	
	Feature	50	70	58.33	
	Other	81.42	91.15	86.01	
	Performance	60	45	51.43	
	Security	66.67	28.57	40	
	Usability	100	25	40	
RQ3_MD3_DS1_VB2_PT1_EX4	Bug	96.97	55.65	70.72	71.56
	Energy	83.33	31.25	45.45	
	Feature	53.33	64	58.18	

	Other	70.37	92.48	79.92	
	Performance	71.43	50	58.82	
	Security	66.67	28.57	40	
	Usability	0	0	0	
RQ3_MD3_DS1_VB3_PT1_EX4	Bug	80.77	91.3	85.71	77.33
	Energy	100	43.75	60.87	
	Feature	75	18	29.03	
	Other	76.33	95.58	84.87	
	Performance	75	45	56.25	
	Security	50	14.29	22.22	
	Usability	25	6.25	10	

Table 20: RQ3 results

IV. Scalabrino VB3 design using ChatGPT

Prompt:

I'm doing a research on 'Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study'. I'm using OpenPrompt library in this research. When using this library there's a step to define a Verbalizer. Below is the example they've provided in the documentation.

Step: Define a Verbalizer

A Verbalizer is another important (but not necessary) in prompt-learning, which projects the original labels (we have defined them as classes) to a set of label words. Here is an example that we project the negative class to the word bad, and project the positive class to the words good, wonderful, great.

eg:

```
from openprompt.prompts import ManualVerbalizer

promptVerbalizer = ManualVerbalizer (
                        classes = classes,
                        label_words = {"negative": ["bad"],
                                       "positive": ["good", "wonderful", "great"]},
                        tokenizer = tokenizer)
```

I need to define label_words according to the classes that I have in the dataset I use in my research. This app review dataset has 3000 user reviews, randomly selected from 705 different Android apps. This was manually labelled by two people by independently analyzing the 3,000 reviews and assigning each of them to a category (class) among the seven. Below are the details of the seven categories.

In total, of the considered 3,000 reviews we labeled 764 as functional bug report, 333 as suggestion for new feature, 50 as report of security issues, 135 as report of performance problems, 107 as request for usability improvements, 106 as report of excessive energy consumption and 1505 as other (other - i.e., non-informative reviews like praise or dispraise, a distractive critique, or a dissuasion).

Could you please suggest the most relevant keywords that a user might use when providing reviews related to above seven classes.

Result:

Certainly! Here are some suggestions for relevant keywords that users might use when providing reviews related to the seven classes/categories in your dataset:

1. Functional Bug Report:

crash, error, bug, glitch, malfunction, issue, problem

2. Suggestion for New Feature:

feature, improvement, addition, suggestion, enhancement, request

3. Report of Security Issues:

security, vulnerability, breach, hack, exploit, threat

4. Report of Performance Problems:

slow, lag, performance, speed, efficiency, bottleneck

5. Request for Usability Improvements:

usability, interface, user experience, navigation, intuitive, user-friendly

6.Report of Excessive Energy Consumption:

battery drain, energy consumption, power usage, energy efficiency, drain, resource-intensive

7. Other (Non-informative Reviews):

praise, criticism, feedback, opinion, irrelevant, off-topic

These keywords should help in defining the label words for your Verbalizer in a way that accurately captures the essence of each class based on user-provided reviews. Adjust them as necessary based on the specific characteristics and terminology of your dataset.

V. Maalej VB3 design using ChatGPT

Prompt:

I'm doing a research on 'Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study'. I'm using OpenPrompt library in this research. When using this library there's a step to define a Verbalizer. Below is the example they've provided in the documentation.

Step: Define a Verbalizer

A Verbalizer is another important (but not necessary) in prompt-learning, which projects the original labels (we have defined them as classes) to a set of label words. Here is an example that we project the negative class to the word bad, and project the positive class to the words good, wonderful, great.

eg:

```
from openprompt.prompts import ManualVerbalizer

promptVerbalizer = ManualVerbalizer (
                    classes = classes,
                    label_words = {"negative": ["bad"],
                                   "positive": ["good", "wonderful", "great"]},
                    tokenizer = tokenizer)
```

I need to define label_words according to the classes that I have in the dataset I use in my research. This app review dataset has 3691 user reviews, manually labelled by assigning each of them to a category (class) among four. Below are the details of the four categories.

Bug reports describe problems with the app which should be corrected, such as a crash, an erroneous behavior, or a performance issue. In feature requests, users ask for missing functionality (e.g., provided by other apps) or missing content (e.g., in catalogs and games) and share ideas on how to improve the app in future releases by adding or changing features. User experiences document the experience of users with the app and how it helped in certain situations. They can be seen as documentation of the app, its requirements, and features. Finally, ratings are simple text reflections of the numeric star rating. Ratings are less informative as they only include praise, dispraise, a distractive critique, or a dissuasion.

Could you please suggest the most relevant keywords that a user might use when providing reviews related to above four classes (Bug reports, Feature requests, User experience, Ratings)

Result:

Certainly! Here are some relevant keywords that users might use when providing reviews related to the four classes:

1. Bug Reports:

crash, error, issue, bug, freeze, glitch, performance problem, malfunction, software problem, application crash

2. Feature Requests:

suggest, add, feature, improvement, enhancement, request, update, new functionality, missing feature, wish

3. User Experience:

usability, user-friendly, experience, intuitive, navigation, interface, convenience, ease of use, helpful, smooth

4. Ratings:

good, great, excellent, fantastic, amazing, poor, terrible, disappointing, satisfactory, exceptional

These keywords capture the essence of each class and are commonly used by users when providing reviews related to app experiences. Feel free to adjust or expand the list based on the specific characteristics of your dataset and the classes you're interested in.

VI. Pan VB3 design using ChatGPT

Prompt:

I'm doing a research on 'Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study'. I'm using OpenPrompt library in this research. When using this library there's a step to define a Verbalizer. Below is the example they've provided in the documentation.

Step: Define a Verbalizer

A Verbalizer is another important (but not necessary) in prompt-learning, which projects the original labels (we have defined them as classes) to a set of label words. Here is an example that we project the negative class to the word bad, and project the positive class to the words good, wonderful, great.

eg:
from openprompt.prompts import ManualVerbalizer

```
promptVerbalizer = ManualVerbalizer (
                    classes = classes,
                    label_words = {"negative": ["bad"],
                                   "positive": ["good", "wonderful", "great"]},
                    tokenizer = tokenizer)
```

I need to define label_words according to the classes that I have in the dataset I use in my research. This app review dataset has 1390 user reviews, which contains reviews of the AngryBirds, Dropbox and Evernote apps available in Apple's App Store and reviews from the apps TripAdvisor, PicsArt, Pinterest and Whatsapp available in Android's Google Play store. These were manually labelled by assigning each of them to a category (class) among four. Below are the details of the four categories.

- Information Giving: sentences that inform or update users or developers about an aspect related to the app.

- Information Seeking: sentences related to attempts to obtain information or help from other users or developers.

- Feature Request: sentences expressing ideas, suggestions or needs for improving or enhancing the app or its functionalities.

- Problem Discovery: sentences describing issues with the app or unexpected behaviors.

Could you please suggest the most relevant keywords that a user might use when providing reviews related to above four classes (Bug reports, Feature requests, User experience, Ratings)

Results:

Certainly! Here are some relevant keywords that users might use when providing reviews related to the four classes in your dataset:

1. Information Giving:

update, inform, news, announcement, change log, release notes, release, version, details, notification

2. Information Seeking:

help, question, query, seek, assistance, support, advice, guidance, clarification, answer

3. Feature Request:

suggest, idea, improve, enhance, functionality, request, wish, addition, enhancement, new feature

4. Problem Discovery:

issue, problem, bug, glitch, error, malfunction, crash, problem-solving, troubleshooting, unexpected behavior

These keywords are tailored to capture the essence of each class based on the descriptions provided. However, you may need to adjust or expand the list based on the specific characteristics of your dataset and the context of the reviews from the selected apps.

VII. Writing Assistance

Grammarly and ChatGPT were employed to enhance the writing process of this thesis, addressing grammar errors and refining the writing style.

VIII. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Hashika Dhananjanie Agalakepu Watthegedara Marasinghe**,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

Few-Shot Prompt-Tuning of Language Models for App Review Classification: An Evaluation Study

supervised by Faiz Ali Shah.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Hashika Dhananjanie Agalakepu Watthegedara Marasinghe

15/05/2024