CLASSIFICATION OF APP REVIEWS FOR REQUIREMENTS ENGINEERING USING DEEP LEARNING MODELS

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2025

Kezhi Zhang

Supervised by Dr. Liping Zhao

Department of Computer Science

Contents

Word Count: 10250

List of Tables

List of Figures

Abstract

Mobile app reviews have become a valuable source of user feedback for Requirements Engineering (RE), but manually analysing large numbers of reviews is time-consuming and not practical. This project addresses this issue by exploring automated app review classification using both traditional machine learning and modern deep learning models to support RE tasks. The study evaluates five representative models, from a Support Vector Machine (SVM) to transformer-based models (BERT, RoBERTa, BART) and a large language model (LLaMA), across both multi-class and multi-label classification tasks. It also examines how key dataset characteristics —such as class imbalance, training size, and label source (manual vs. GPT-generated)—influence classification performance and stability.

The results show that transformer-based models clearly outperform the SVM baseline, BERT and RoBERTa consistently delivered strong performance across both task types while maintaining moderate computational cost, making them effective and practical choices for real-world applications. These findings support the use of pretrained language models and automated labelling in requirements-related review analysis. Finally, the project outlines several future directions, including enhancing model generalization to handle new types of user feedback, and exploring unsupervised or semi-supervised learning approaches to reduce the dependence on large amounts of labelled data.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my supervisor, Liping Zhao, for her guidance, patience, and continuous support throughout the development of this project. Her feedback and encouragement helped guide the direction of my work and supported me in exploring the core ideas more deeply. I would also like to thank PavelGh, the creator of the dataset used in this study, for making the well-structured data available, which greatly supported my experiments. I appreciate the assistance of ChatGPT in refining the language of this report, which helped me express my thoughts more clearly and effectively in English. Finally, I am deeply grateful to my family for their consistent support and encouragement. In particular, I wish to thank my mother and father for their understanding and help, which have been especially important to me during this challenging year.

Abbreviations and Acronyms

AI	Artificial Intelligence
BART	Bidirectional and Auto-Regressive Transformers
BERT	Bidirectional Encoder Representations from Transformers
CLM	Causal Language Modelling
FN	False Negative
FP	False Positive
FSL	Few-Shot Learning
LLaMA	Large Language Model Meta AI
ML	Machine Learning
MLM	Masked Language Modelling
NLP	Natural Language Processing
NSP	Next Sentence Prediction
RE	Requirements Engineering
RoBERTa	A Robustly Optimized BERT Pretraining Approach
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
TN	True Negative
TP	True Positive
ZSL	Zero-Shot Learning

Chapter 1

Introduction

1.1 Context and Motivation

Nowadays app distribution platforms have become essential communication platforms connecting users and developers in modern mobile application market. In August 2024, leading app stores such as Google Play and Apple APP Store had over 4 million apps available [59]. In addition to downloading apps, users can leave star ratings and textual reviews, which provide developers with a direct source of user feedback [19].

User reviews contain a wealth of information, ranging from bug reports and feature suggestions to subjective opinions and general feedback [44]. Previous research has demonstrated that such feedback can support software maintenance and evolution, making app reviews a valuable resource for continuous improvement in requirement engineering [37, 25].

However, not all reviews are equally informative. Many comments are vague, emotionally driven, or irrelevant to the technical aspects of the application. This noise makes it difficult for developers to extract useful insights efficiently [44]. As the volume of user reviews continues to increase, manual analysis becomes increasingly impractical, especially for popular applications with large and diverse user bases [42].

These challenges have created a clear need for automatic review classification, motivating the use of machine learning techniques to categorize app reviews and support software development [37, 26].

1.2 Aims and Objectives

The primary aim of this project was to conduct a detailed evaluation of automated app review classification methods in the context of Requirements Engineering. In particular, the study aimed to compare a wide range of machine learning and deep learning models and to understand how different task setups and data conditions affect their performance. To achieve this goal, the project followed the objectives below:

Model Comparison: Build and fine-tune five representative text classification models – including a traditional Support Vector Machine (SVM), three transformer-based models (BERT, RoBERTa, BART), and a large language model (LLaMA) – and test their performance on the same app review dataset.

Classification Task Evaluation: Study how the type of classification task (multi-class and multi-label) affects each model's results, and understand the effect of task structure on the models' ability to correctly classify app reviews.

Dataset Characteristic Analysis: Explore how different features of the dataset affect model performance by testing both balanced and imbalanced label distributions, as well as different training sizes (from a few thousand to tens of thousands of reviews). This helps explain how factors like class imbalance and data volume influence accuracy and stability.

Performance and Efficiency Assessment: Review the results to find which model setups give the best performance, and evaluate the trade-offs between prediction accuracy and computing cost. This includes identifying each model's performance (such as training time and data requirements), to give useful advice for applying these methods in real-world requirements engineering work.

1.3 Report Structure

This report consists of the following chapters:

Chapter 1 introduces the research background, motivation, project aims, objectives, and the overall structure of the report.

Chapter 2 provides background knowledge on requirements engineering and the use of app reviews in software analysis, along with an overview of related machine learning approaches.

Chapter 3 describes the research methodology, including data preparation, classification task design and model setup.

Chapter 4 presents and compares the experimental results across different models and configurations.

Chapter 5 reflects on model performance, data limitations, and project challenges.

Chapter 6 concludes the report with a summary of key findings and outlines potential directions for future work.

The Appendix A shows a plan of the work, including tasks and project milestones.

Chapter 2

Background and Related Work

This chapter introduce the background and prior research relevant to the project. Section 2.1.1 discusses app reviews in requirements engineering, the development of language models, and relevant machine learning techniques. Section 2.2 reviews existing literature, highlighting representative approaches and their limitations in app review classification tasks. Finally, Section 2.3 outlines the contributions of this project, which address current research gaps in existing research.

2.1 Background

2.1.1 Requirements Engineering and App Reviews

Requirements Engineering (RE) is the structured process of collecting and analysing the functional and non-functional needs of software stakeholders [58]. Traditionally, requirements are collected through interviews, surveys, and stakeholder meetings [44, 19].

In modern mobile development, user-generated content, particularly app reviews, has become a helpful and increasingly important source of software requirements [44]. These reviews often contain different types of feedback, including bug reports,requests for new features, problems with usability, and general user opinions [36]. Importantly, they reflect real-world usage contexts and containing expectations that may not have been captured during the initial requirements gathering process [6].

To make use of this valuable content, recent research has investigated the automatic classification of app reviews into requirement-relevant categories. This approach enables developers to extract useful information from large volumes of feedback and integrate them into ongoing development [35, 6].

In particular, recent advances in Natural Language Processing (NLP)—especially the development of machine learning models—have made it increasingly feasible to process and classify app reviews with higher accuracy and scalability [14, 26]. These techniques offer a practical way to bridge the gap between unstructured user feedback with structured requirements in modern requirements engineering [19].

2.1.2 Languages Models

To enable automatic classification of app reviews, various types of language models have been explored. These models aim to convert unstructured review texts into structured representations suitable for supervised learning [36]. Over time, language modelling techniques have evolved significantly—from traditional machine learning methods that rely on explicit feature engineering, to deep Pre-trained transformers that learn contextual representations from large corpora, and more recently to large language models (LLMs) capable of generalizing across tasks with minimal supervision [55, 10, 5].

2.1.2.1 Traditional Machine Learning Models

Traditional machine learning models, such as Naive Bayes, Logistic Regression, and Support Vector Machines (SVM), have been widely applied in early text classification tasks [24]. These models rely on supervised learning, where a classifier is trained using labelled examples and fixed-size feature vectors, commonly derived from techniques such as bag-of-words or TF-IDF [39]. As shown in Figure 2.1, TF-IDF represents each review as a sparse vector based on the importance of specific terms. Darker shades indicate higher TF-IDF values, meaning those terms are more representative within that particular review.

Unlike modern language models, traditional models do not learn semantic or contextual representations from raw text. Instead, they operate on manually designed features that focus on word frequency and position. This approach makes them computationally efficient and interpretable, but also limits their ability to capture deeper language patterns or relationships between words.

Among these, SVM have shown strong performance in high-dimensional sparse settings like text classification. They work by learning a hyperplane that separates classes with the largest possible margin, illustrated in Figure 2.2. In this project, SVM

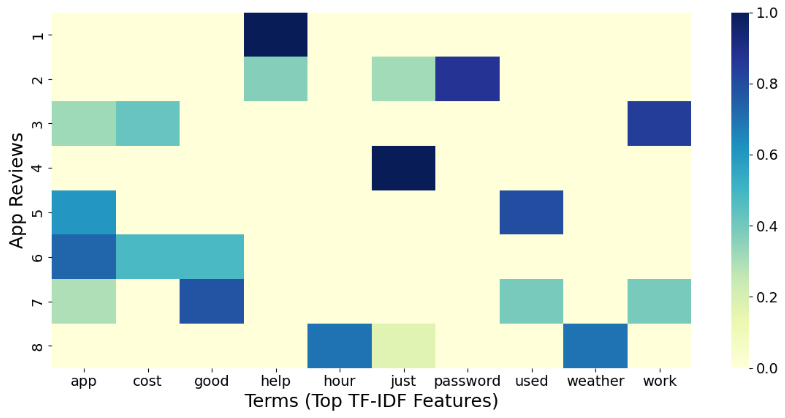


Figure 2.1: TF-IDF representation of app reviews

serves as a representative baseline to compare against Pre-trained and large-scale neural models. For further information on the concept of SVM, see Chapter 3.3.1.

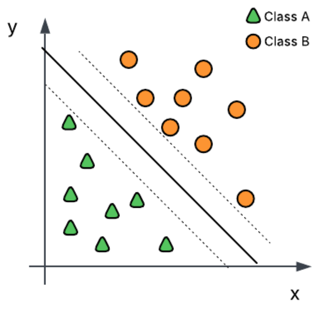


Figure 2.2: SVM Classifier Illustration

2.1.2.2 Pre-trained Language Models

Pre-trained language models have become the foundation of modern NLP [49, 41]. They are built on the Transformer architecture (Figure 2.3)[63], which replaces older methods like RNNs [16] and CNNs [27] with a self-attention mechanism. Unlike those earlier models, Transformers can understand long-range word relationships and capture contextual relationships between all tokens in a sequence at the same time [63, 21]. This mechanism allows the model to assign learned weights to different parts of the input, enabling it to focus on the most relevant tokens when computing each output. The encoder in a Transformer processes input tokens in parallel, computing attention scores that determine how much each token influence others when constructing its contextual representation. The decoder generates output by attending to both its past outputs and the encoder's representations, enabling context-aware text generation. [63].

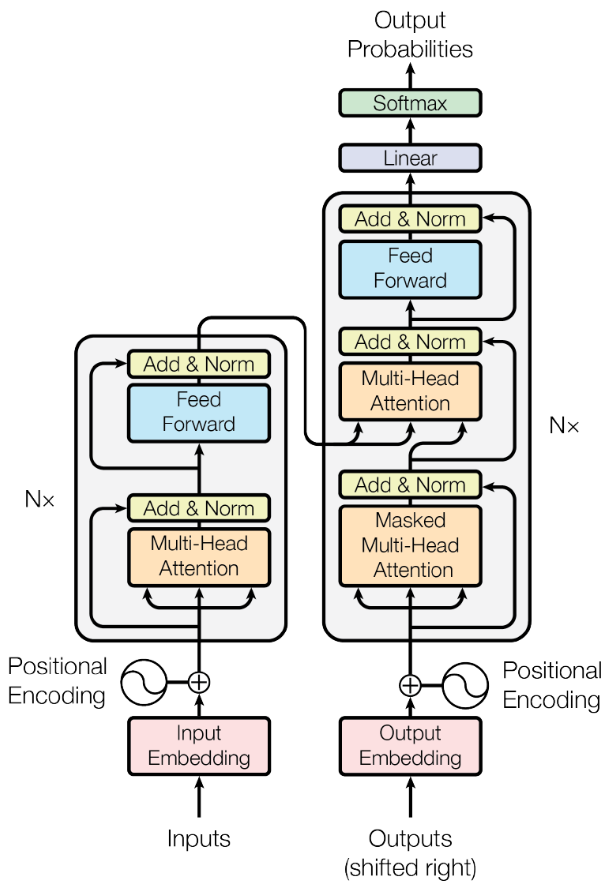


Figure 2.3: Transformer architecture [63]

In Pre-trained models, the Transformer encoder (or encoder-decoder) is trained on large-scale unlabelled text using self-supervised objectives [10, 7]. A common example is Masked Language Modelling (MLM) (Figure 2.4), where some words in the input are hidden, and the model is trained to predict them based on surrounding context. This allows the model to learn semantic patterns, syntax, and word relationships in language [10, 53].

After pretraining, these models can be adapted to specific tasks through fine-tuning. In fine-tuning, the Pre-trained model is updated using a smaller labelled dataset for a target task, such as text classification. This method is called transfer learning, it uses knowledge learned during pretraining to improve task-specific performance with minimal additional data [49].

Pre-trained models come in various architectural forms. Encoder-only models are well-suited for classification and understanding tasks, while encoder-decoder models

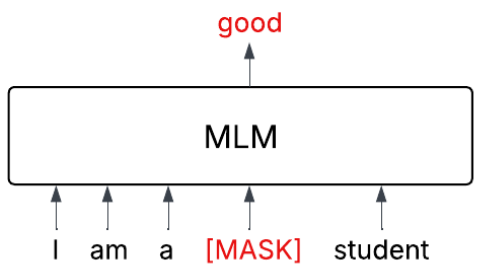


Figure 2.4: Illustration of Masked Language Modelling

are more flexible and can handle both classification and text generation [51, 10]. The key benefit is the ability to learn various, context-aware representations of text, making these models effective in text classification, where input texts are short and diverse [32]. Compared to traditional models, Pre-trained Transformers eliminate the need for manual feature design and significantly improve performance on most NLP tasks [67, 53].

2.1.2.3 Large Language Models

Large Language Models (LLMs) extend Pre-trained language models by significantly increasing parameter scale, training data volume, and generalisation capabilities [68]. While both LLMs and earlier models such as BERT are based on the Transformer architecture, LLMs are typically trained using a causal language modelling (CLM) objective, it is predicting the next token given previous context.

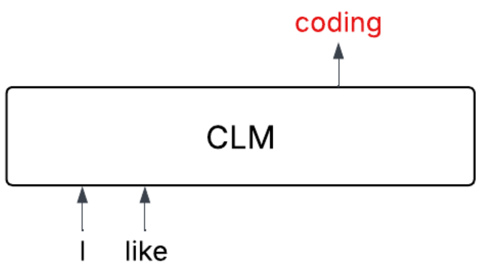


Figure 2.5: Illustration of Causal Language Modelling

Unlike standard Pre-trained models that require fine-tuning for specific tasks, LLMs often support zero-shot and few-shot learning through prompting. This allows them to perform tasks such as text classification, summarisation, and question answering without additional task-specific training. Their ability to generalise across tasks makes them highly flexible in different domains [23, 65].

2.1.3 Relevant Learning Technologies

This study uses a supervised learning approach, which is the most common method for text classification tasks. In supervised learning, models are trained on labelled data to learn a mapping between input text and output categories [24]. Given sufficient examples, the model can learn to classify unseen texts based on patterns learned during training. This method is applicable to both multi-class and multi-label classification settings, depending on whether each input is associated with one or multiple labels [62, 55].

Transfer learning has become a dominant technique in natural language processing, particularly with the rise of large Pre-trained language models [54]. Rather than training models from scratch, transfer learning allows models to reuse general language knowledge learned during pretraining on large corpora. In the fine-tuning stage, task-specific layers are added on top of the Pre-trained model, and the entire network is updated using labelled data for the target task [17]. This approach has been shown to improve classification performance significantly [60].

2.2 Related Work

Early work on app review classification focused on using traditional machine learning models with manually designed features. Studies have shown that user reviews contain rich feedback information, and demonstrated about $1/3$ of the comments involve feature requests, defects, and improvements, which have the potential to extract needs [44]. Traditional algorithms such as Support Vector Machines (SVMs) and Naive Bayes have been identified as among the most commonly used and effective methods for non-functional requirement (NFR) classification tasks after a systematic review of 24 studies [4]. Similarly, a research applied SVMs to classify functional and non-functional requirements, showing that supervised learning methods can effectively support such tasks. The study also demonstrated that performance degradation caused by class imbalance can be solved through sampling strategies and data augmentation, such as increasing user reviews [26].

With the rise of deep learning, more recent studies have adopted neural network models and Pre-trained language representations to improve classification performance on user-generated content. Some research has proposed fine-tuning BERT to better adapt to domain-specific data in Requirements Engineering (RE). For example, BERT4RE, a BERT-based model retrained on requirements texts, showed improved performance over the original BERT on concept extraction tasks. [1]. Fine-tuned transformer models have shown strong performance in extracting software requirements from app reviews. RE-BERT, built on BERT with a local context mechanism, outperforms rule-based traditional models in accuracy and stability when processing user reviews.[9]. The review classification scheme dividing reviews into bug reports, feature requests, user experience, and rating comments has been widely adopted in many studies as a basis for supervised classification tasks [35].

Concurrently, language representation technologies in NLP have evolved to large language models (LLMs). Surveys have outlined this progression, highlighting a shift from traditional models such as SVMs to contextualized models like BERT and RoBERTa, and further to scalable architectures such as GPT and LLaMA [21]. These LLMs have achieved state-of-the-art performance in many NLP benchmarks, but their application to app review classification remains largely untested.

Despite all this progress, there are still important gaps. Few studies compare a wide range of model types, and most evaluations only include one or two models, leaving the impact of architectural choices underexplored. Also, key task settings such as classification type (multi-label and multi-class) are often fixed across experiments, which remains insufficiently studied and requires further exploration.

2.3 Project Contributions

To address the limitations identified in previous research, this project makes three specific contributions to the field of automated app review classification in the context of requirements engineering.

First, a comprehensive model comparison is conducted across five representative classification models, covering a spectrum from traditional machine learning (SVM) to modern transformer-based architectures (BERT, RoBERTa, BART) and a large-scale language model (LLaMA). Although LLaMA has been used in some recent studies, this work is the first to include it in a unified evaluation framework that incorporates traditional and Pre-trained models. The comparison across these architectures provides in-depth evaluation of how model type affects classification performance in this domain.

Second, the project systematically investigates how different classification task is set up (multi-label versus multi-class settings) affect model performance. This contributes to a clearer understanding of how task structure influences performance when extracting useful information from app reviews.

Third, the influence of dataset characteristics is examined, including class distribution (balanced and imbalanced data) and dataset scale (4,000 to 20,000 samples). These experiments provide insights into how model performance varies with real-world constraints.

In summary, these contributions provide a more comprehensive view of model behaviour in the context of app review classification, and support future research and tools for automatically analysing user feedback and requirements extraction.

Chapter 3

Research Methodology

This chapter outlines the experimental methodology. It covers the dataset selection (3.1), including collection and labeling. The classification tasks (3.2) are then discussed, followed by model selection (3.3) and training procedures (3.4). Next the experiment setup (3.5) details the computing environment and tools. Finnaly, the evaluation metrics and performance assessment are presented (3.6).

3.1 Datasets

3.1.1 Data Collection

This project used a publicly available dataset hosted on Hugging Face [1], which was constructed by merging two primary sources of user reviews. These sources contain real-world user feedback from both the Google Play Store and the F-Droid open-source app repository. The dataset was cleaned, standardized, and organized by its creator to ensure consistency and usability across classification tasks.

The final dataset is monolingual, consisting exclusively of user reviews written in English. For data quality purpose, duplicate entries, reviews with insufficient content (e.g., only emojis), and spam-like messages were removed. To examine the impact of training set size on model performance, datasets of varying sizes were used: 4000, 8000, 16000 and 20000 samples. These datasets were used to train and evaluate all classification models in this study.

[1] `https://huggingface.co/datasets/PavelGh/app_reviews`

3.1.2 Data Labelling

Each review in the dataset is labelled to one or more of four predefined categories that are highly relevant to requirements engineering tasks. These include feature request, which captures suggestions for new or enhanced functionality; bug report, which identifies issues such as crashes or errors; rating, which reflects star-based assessments; and user experience, which covers impressions related to usability, design, and overall interaction with the app. Examples of typical user reviews for each category are provided in Table 3.1.

Type	**Examples from User Reviews**
Feature Request	“It would be great if the app had dark mode support.”
Bug Report	“The app crashes every time I try to upload a photo.”
Rating	“Five stars! Works perfectly.”
User Experience	“The layout is confusing and hard to navigate.”

Table 3.1: Descriptions and Examples of Each Type

The dataset is structured to support both multi-label and multi-class classification shown on Table 3.2. The multi-label data is unbalanced and was entirely labelled using ChatGPT [43] through fine-tuned prompt. This format allows each review to belong to multiple categories simultaneously, offering a more detailed representation of user feedback. In contrast, the multi-class data restricts each review to a single dominant label and is divided into balanced and unbalanced datasets. The balanced dataset ensures an equal distribution of samples across categories, while the unbalanced dataset reflects the natural frequency of each class observed in the original data.

The multi-class dataset includes both manually and automatically labelled samples. Manually labelled datasets were reviewed and classified by human, ensuring high fidelity and reliability. The automatically labelled dataset was generated by ChatGPT. To validate the quality of this automated labelling process, approximately three percent of all GPT-labelled data was randomly selected and manually verified.

Task Type	**Dataset Sizes**	**Label Source**
Multi-class (Balanced)	4000 / 8000 / 20000	Manual + GPT
Multi-class (Unbalanced)	4000 / 8000 / 16000	Manual + GPT
Multi-label (Unbalanced)	4000 / 8000 / 16000	GPT only

Table 3.2: Overview of Dataset

3.1.3 Preprocessing

The preprocessing procedures were kept minimal to preserve the nature characteristics of user reviews. A comparative study [57] found that retaining the original textual form can lead to better outcomes for transformer architectures. Accordingly, the original review texts were used for all models without applying spelling correction, stopword removal, case folding, or punctuation stripping. Each model applies its own preprocessing method based on its format and architectural design. These procedures were integrated into respective models' training pipeline, with further details provided in section (3.4).

3.1.4 Dataset Split

To evaluate model performance under consistent and reliable conditions, the dataset was split using the holdout method. Each version of dataset was divided into 80% for training, 10% for validation, and 10% for testing. This splitting strategy allowed for model selection based on validation performance while keeping a separate, unseen test set for final evaluation.

To ensure that the label distribution remained consistent across all splits, stratified sampling [45] was applied. This technique ensures that all classes are adequately represented in each partition. This setup enables fair comparisons across models and experimental conditions while reducing the risk of bias caused by unbalanced label distributions.

3.2 Classification Tasks

The classification task is to determine whether a given review belong to one or more of four predefined categories. Two types of classification tasks were designed: multi-class classification and multi-label classification.

In the multi-class classification task, each review is assigned to exactly one label that best represents its main content. The categories are mutually exclusive, and the model is trained to choose a single most relevant class for each input. The model outputs raw logits, and the softmax function is implicitly applied within the CrossEntropyLoss to model the probability distribution over the classes.

The multi-label classification task allows each review to be associated with multiple labels simultaneously. This reflects real-world scenarios where a review may refer to more than one issue, such as requesting a new feature while reporting a performance problem. In this task, the model produces independent predictions for each label using a sigmoid activation function, and each label is treated as a separate binary classification task.

Although the multi-class and multi-label tasks use different label formats, both task were constructed from the same set of app reviews. Therefore, the two classification tasks are designed to be comparable, and their results are analysed in section (4.6.1) to examine how different task affects model performance.

3.3 Model Selection

All models selected for this study to evaluate the classification performance from the Hugging Face Model Hub[2], which support fine-tuning for specific tasks. These models span a wide spectrum, from traditional machine learning classifiers to state-of-art large language models, allowing for a comparative analysis across varying levels of model complexity [66]. Details of the models are shown in Table 3.3.

Model	**Objective**	**Training Corpora**	**#Parameters**
BERT (bert-base-uncased)	MLM + NSP	BooksCorpus, Wikipedia	110M
RoBERTa (roberta-base)	MLM	CC-News, OpenWebText, Books, Stories, Wikipedia	125M
BART (bart-base)	Denoising Autoencoder	Books, Wikipedia, CC-News, OpenWebText, Stories	139M
LLaMA (Llama-3.2-1B)	CLM	Web data, academic papers, code (not fully disclosed)	1.2B

Table 3.3: Details of the selected pre-trained models

3.3.1 SVM

Support Vector Machine (SVM) is a classical linear classifier widely applied in text classification tasks. In this study, the LinearSVC implementation from scikit-learn

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

[47] was used, trained on TF-IDF feature vectors extracted from raw text [20]. It provides a lightweight and interpretable baseline, allowing to evaluate the improvement of modern neural architectures. SVM's strong performance on small datasets and its relatively low computational cost make it a useful comparison point against large-scale transformer models [69, 11]. This model was included as a classical machine learning baseline to establish a performance reference point for comparison with modern deep learning architectures.

3.3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) [10] is a widely used encoder-only transformer model that introduced a new approach to language model pretraining. The model is trained using two self-supervised objectives: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP) [10]. In MLM, a subset of tokens in the input sequence is randomly masked, and the model is trained to predict the original tokens based on the surrounding context. In NSP, the model learns to predict whether two given sentences appear continuously in the original text. This enables BERT to capture bidirectional dependencies in text and modelling sentence relationships.

The specific model used in this study is bert-base-uncased [3], which consists of 12 transformer layers, 768-dimensional hidden state, and approximately 110 million parameters (Table 3.3). The uncased version converts all inputs text to lowercase, improving robustness to surface-level variations. Due to its effective pretraining strategy and general design, BERT has become a standard baseline in many NLP classification tasks and is widely recognised as a representative of encoder-based transformer models [60]. BERT was selected as a widely used Pre-trained transformer model to evaluate the performance.

3.3.3 RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) [33] improves upon BERT by eliminating the Next Sentence Prediction (NSP) objective and adopting a dynamic masking strategy. It retains the Masked Language Modelling (MLM) objective as the pretraining task. RoBERTa is trained on a much larger corpus and for more training

[3] `https://huggingface.co/google-bert/bert-base-uncased`

steps using longer input sequences. These changes lead to more efficient use of training data and stronger contextual representations [33].

The roberta-base model [4] used in this project shares the same architecture as bert-base-uncased, with 12 layers and approximately 125 million parameters (Table 3.3). RoBERTa model was selected to examine the impact of improved pretraining strategies over the original BERT framework on classification performance. RoBERTa was chosen to assess how improvements to BERT's pretraining procedure influence model effectiveness on downstream classification tasks.

3.3.4 BART

BART (Bidirectional and Auto-Regressive Transformers) [29] is a sequence-to-sequence transformer model with both an encoder and a decoder, combining the characteristics of BERT and GPT [50]. It is Pre-trained as a denoising auto-encoder, where input sequences are corrupted using various noise functions, and the model is trained to reconstruct the original text. Corruption methods include token masking, deletion, text infilling, and sentence permutation [29]. This setup enables BART to learn both bidirectional and autoregressive representation.

The bart-base model [5] used in the study consists of 6 encoder layers and 6 decoder layers, totalling approximately 139 million parameters (Table 3.3). Although BART was originally developed for generation tasks such as summarization and translation, it has demonstrated strong performance on discriminative tasks such as classification when fine-tuned on labelled data [29, 31]. In this study, only the encoder component of BART was used during classification. The encoder's final hidden states served as input to a classification head that produced label predictions. The decoder component was not involved during training or testing. BART was included as a representative encoder-decoder architecture to evaluate whether denoising-based pretraining benefits text classification.

3.3.5 LLaMA

LLaMA (Large Language Model Meta AI) [61] is a decoder-only autoregressive language model family optimized for efficiency and general-purpose language modelling.

[4] https://huggingface.co/FacebookAI/roberta-base
[5] https://huggingface.co/facebook/bart-base

Unlike encoder-based models, LLaMA is Pre-trained exclusively using a causal language modelling (CLM) objective, where the model predicts the next token in a sequence given the previous context. This pretraining strategy enables strong performance in text generation and also allows adapting classification through supervised fine-tuning [61]. For classification tasks, a task-specific classification head is attached to the decoder's final hidden state, allowing the model to generate label predictions directly [71].

The selected model, Llama-3.2-1B [6], contains approximately 1.2 billion parameters (Table 3.3) and it represents the decoder-only architecture and allows exploration of how lightweight LLMs perform on text classification tasks. LLaMA was selected as a recent large language model to examine the performance of modern transformer architectures under fine-tuning.

3.3.6 Summary

The selection of these five models provides architectural diversity across traditional machine learning methods 3.3.1, encoder-based transformers 3.3.2,3.3.3, encoder-decoder structures 3.3.4, and decoder-only large language models 3.3.5. This diversity supports an in-depth evaluation on model structure, parameter size, and pretraining objectives influence performance on app review classification tasks.

3.4 Model Training

This study used a supervised learning approach, where models are trained on labelled app reviews to predict categories. Each model was trained using a procedure that reflects its architecture, input format, and classification requirements.

3.4.1 SVM

For the Support Vector Machine (SVM) models, training was conducted using scikit-learn's LinearSVC [47]. For multi-class classification task, the raw app review texts were vectorized using TF-IDF with a vocabulary of 5000 and bi-gram features [64]. For multi-label classification, labels were structured as binary vectors across four dimensions (feature request, bug report, rating, user experience) 3.1, and the model was wrapped with a OneVsRestClassifier [52]. The TF-IDF vectors were directly fed into

[6] https://huggingface.co/meta-llama/Llama-3.2-1B

the model. The training process was conducted using grid search for hyperparameter selection.

3.4.2 Transformers

For all transformer-based models (BERT, RoBERTa, BART, and LLaMA), the training followed a fine-tuning paradigm using Hugging Face Transformers [67]. Input texts were first preprocessed to fill missing values and then tokenized using each model's corresponding tokenizer. Padding and truncation were applied to a fixed sequence length of 256 tokens. The tokenized inputs were converted into PyTorch-compatible dictionaries containing input_ids, attention_mask, and labels, and loaded using the datasets format.

Each model was trained separately on both multi-class and multi-label datasets. Labels were cast as integer class IDs for multi-class tasks and binary label vectors for multi-label tasks. For BERT, RoBERTa, and BART, a linear classification head was attached to the final encoder layer. For LLaMA, the final decoder hidden state was used for classification. Models were trained using a custom training loop, including forward pass, loss computation, backpropagation, and validation per epoch. All transformer models were trained using end-to-end supervised learning.

3.5 Experiment Setup

All experiments were conducted using Jupyter Notebook [22] with Python 3.10 [7] as the programming environment. Two platforms were used to satisfy different resource demands: a local computer equipped with an Intel i7-12700F CPU and an NVIDIA RTX 3080 GPU was used for training the models. The SVM model was trained on the CPU, as LinearSVC of scikit-learn does not support GPU acceleration. Transformer-based models including BERT, RoBERTa, and BART models were trained using the GPU. LLaMA models were trained on Google Colab Pro [8]with access to high-memory NVIDIA A100 GPU, as training on the local GPU resulted in out-of-memory errors due to the model's large memory requirements.

The implementation was based primarily on PyTorch [46] and Hugging Face Transformers [67]. Additional libraries used include scikit-learn [47] for SVM models, dataset [30] for data management, and pandas [40], numpy [15], and matplotlib [18]

[7] https://www.python.org
[8] https://colab.research.google.com

for preprocessing and visualization. All transformer-based models were obtained from the Hugging Face Model Hub and integrated using the AutoModelForSequenceClassification [67] and fine-tuning was implemented using custom PyTorch training loops to each model architecture.

To account for the variability caused by random initialization and stochastic training dynamics, each experimental setting was repeated three times using identical data splits and hyperparameter configurations. The final performance metrics reported are the mean values across these three independent runs, ensuring a more robust and reliable evaluation.

3.5.1 Hyperparameters

Each transformer model was trained with task-specific hyperparameters.

For the SVM-based models, hyperparameters were tuned using grid search with 5-fold cross-validation to ensure generalizable performance. The grid included regularization strength (C values of 0.01, 0.1, 0.5, and 1), tolerance thresholds (tol values of 1e-5, 1e-4, and 1e-3), maximum training iterations (5000, 10000, and 20000), and class weighting strategies (None and balanced). The final configuration was selected based on the best average validation score across folds.

For BERT, RoBERTa, and BART, the batch size was set to 32, and models were trained for 10 epochs as recommended to train models [33, 29]. Learning rates were set to 3e-5 for multi-class tasks and 2e-5 for multi-label tasks. Weight decay was 0.01 for multi-class and 1e-5 for multi-label tasks. To mitigate overfitting and reduce unnecessary computation, early stopping was employed with a patience of 3 epochs, terminating training if no improvement in validation performance was observed over three consecutive epochs [48]. A linear learning rate scheduler with warm-up (10% of total steps) was used to gradually increase the learning rate at the beginning of training and then decay it linearly, which has been shown to improve training stability and convergence behaviour [13]. The AdamW optimizer was chose instead of standard Adam, as it decouples weight decay from the gradient update, resulting in more stable and effective fine-tuning of transformer architectures [34]. All hyperparameter values were selected through initial tuning, based on their performance on the validation set.

Due to the significantly higher memory consumption of LLaMA models [72], LLaMA models were trained with a reduced batch size of 16 and for only 3 epochs. A smaller learning rate of 5e-6 and a weight decay of 0.01 were used to support stable convergence. All other components of the training setup—including optimizer choice, learning rate scheduling, early stopping, and warm-up strategy—were kept consistent with the other transformer models.

3.6 Evaluation Metrics

To evaluate the performance of each model on the classification tasks, four metrics were employed: Precision, Recall, Marco F1 Score and Accuracy. These metrics based on the fundamental classification outcomes: True Positive(TP), False Positive(FP), False Negative(FN), and True Negative(TN). TP refers to correctly predicted positive labels, FP to incorrectly predicted positive labels, FN to missed positive cases, and TN to correctly predicted negative labels. These components form the basis of the evaluation metrics described below.

3.6.1 Precision

Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. A high precision score means the model makes few false positive errors when predicting the positive class. The formula is provided below (3.1).

$$Precision = \frac{TP}{TP+FP} \tag{3.1}$$

3.6.2 Recall

Recall is the proportion of correctly predicted positive instances out of all actual positive instances. It indicates how well the model can correctly identify all actual positive instances, with the corresponding formula given below (3.2).

$$Recall = \frac{TP}{TP+FN} \tag{3.2}$$

3.6.3 Macro F1 Score

Macro F1 Score is calculated by first computing the F1 score independently for each class, and then taking the unweighted mean across all classes. This metric treats each class equally and it is useful in multi-class classification settings where class imbalance exists, since it prevents majority classes from dominating the evaluation [38, 12]. The formula is presented below (3.3).

$$\textit{Macro-F}1 = \frac{1}{N}\sum_{i=1}^{N}\frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{3.3}$$

3.6.4 Micro F1 Score

Micro F1 Score is calculated by summing the contributions of all classes to compute a global precision and recall. And the final F1 score is calculated from these global values. Micro-F1 gives more weight to classes with more samples, making it more sensitive to the performance on frequent labels. This makes it useful in multi-label classification, where class imbalance is common and some labels appear much more often than others. In such cases, micro-F1 offers a more stable and realistic view of the model's overall performance.The formula is provided below (3.4).

$$\textit{Micro-F}1 = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{3.4}$$

3.6.5 Accuracy

Accuracy measures the proportion of correctly predicted labels over the total number of predictions. In multi-class classification, it reflects whether the predicted class exactly matches the true class. However, in multi-label classification, accuracy is interpreted as subset accuracy, which only considers a prediction correct if all labels for a sample are exactly predicted. Due to this strict definition, it was not used for evaluating multi-label performance, as it can underestimate model effectiveness when partial predictions are still useful [70]. The formula is given below (3.5).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

3.6.6 Summary

In summary, the selected metrics allow for a balanced and comprehensive evaluation across both multi-class and multi-label classification tasks. Precision and recall provide insight into error types, while macro and micro F1 scores give a balanced way to compare models. In multi-class classification, accuracy is useful as a general measure of how many predictions are correct.

Chapter 4

Results

This chapter presents the experimental results for the classification of app reviews using the five selected models: SVM (4.1), BERT (4.2), RoBERTa (4.3), BART (4.4), and LLaMA (4.5). All reported results represent the average performance across three independent runs using identical data splits and hyperparameters, in order to reduce the effects of random initialization and training variability. Section (4.6) offers a comparative discussion of key trends across all models, focusing on differences between multi-label and multi-class classification (4.6.1), the impact of balanced versus unbalanced datasets (4.6.2), and the effect of training dataset size on performance (4.6.3).

4.1 Results for SVM

In the multi-class classification task, SVM demonstrates strong performance improvements as the balanced dataset size increases as shown in Table 4.1. On the manually-labelled balanced dataset, the macro F1 improves significantly from 0.678 at 4,000 samples to 0.920 at 8,000 samples. The corresponding results on the GPT-labelled balanced data exhibit similar trends, achieving a macro F1 of 0.942 at 20,000 samples. This indicates that with sufficient data, SVM is capable of achieving competitive performance even using linear decision boundaries.

On the unbalanced multi-class datasets shown in Table 4.2, SVM's performance is consistently lower at smaller sizes (macro F1 = 0.765 and 0.779), but again improves rapidly with more training data (0.932 at 16,000 samples, GPT-labelled). The performance difference between balanced and unbalanced datasets in the multi-class setting was relatively small, indicating that class imbalance had limited impact on model effectiveness in this context.

Table 4.1: SVM Multi-Class Classification (Balanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.679	0.680	0.678	0.680	Manually-labelled
8000	0.921	0.920	0.920	0.920	Manually-labelled
4000	0.708	0.700	0.701	0.700	GPT-labelled
8000	0.826	0.820	0.822	0.820	GPT-labelled
20000	0.943	0.942	0.942	0.942	GPT-labelled

Table 4.2: SVM Multi-Class Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.776	0.757	0.765	0.794	Manually-labelled
4000	0.807	0.760	0.779	0.789	GPT-labelled
8000	0.887	0.850	0.866	0.872	GPT-labelled
16000	0.943	0.923	0.932	0.937	GPT-labelled

In the multi-label classification task, SVM achieves strong results despite the model's simplicity. Macro F1 increases from 0.854 to 0.969 as the dataset size grows from 4,000 to 16,000. The high precision (up to 0.993) suggests that SVM takes a conservative approach to label assignment, prioritizing precision over recall. Micro F1 scores follow a similar trend, reaching 0.973 at 16,000, which implies accurate predictions across multiple labels. These results may also reflect the lower ambiguity in the GPT-generated label sets.

Table 4.3: SVM Multi-Label Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Micro F1	Label Source
4000	0.884	0.826	0.854	0.863	GPT-labelled
8000	0.968	0.878	0.921	0.929	GPT-labelled
16000	0.993	0.946	0.969	0.973	GPT-labelled

The line chart (4.1) shows that all evaluation metrics improve steadily as the dataset size increases. Across all scales, SVM achieves consistently better performance on multi-label classification than on multi-class classification under unbalanced data

Overall, SVM shows reliable scalability and performance across all configurations. Its ability to achieve macro F1 scores above 0.90 in large datasets makes it a strong baseline.

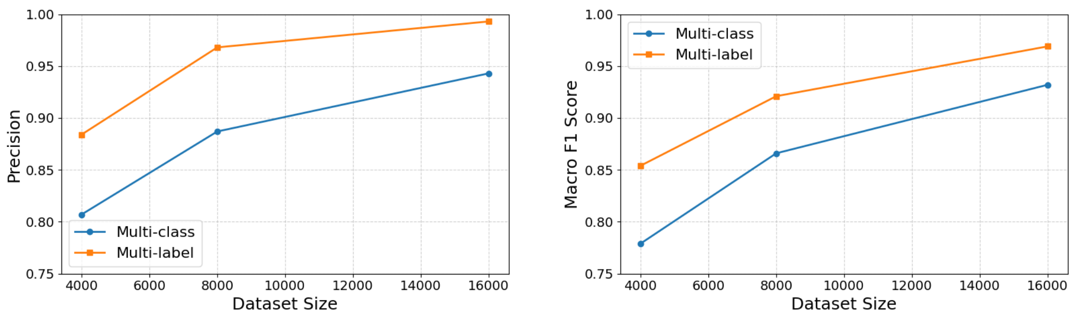


Figure 4.1: SVM Comparison: Multi-class vs Multi-label

4.2 Results for BERT

In the multi-class classification task shown in Table 4.4, BERT demonstrates significant performance improvements with increasing dataset size. On the manually-labelled balanced dataset, the macro F1 improves from 0.706 at 4,000 samples to 0.879 at 8,000 samples. This highlights BERT's ability to learn meaningful representations even from moderately sized datasets. On the GPT-labelled balanced datasets, performance is particularly higher, achieving near-perfect scores (Macro F1 = 0.996) as 8,000 samples.

Figure 4.2 demonstrates that GPT-labelled data consistently outperforms manually-labelled data in the multi-class classification task using BERT. At both 4000 and 8000 samples, the macro F1 score for GPT-labelled data is substantially higher—by approximately 0.27 and 0.12, respectively. This suggests that GPT-generated annotations offer greater consistency and scalability, enabling more effective model learning as the dataset size increases.

Table 4.4: BERT Multi-Class Classification (Balanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.714	0.708	0.706	0.708	Manually-labelled
8000	0.882	0.879	0.879	0.879	Manually-labelled
4000	0.977	0.977	0.977	0.977	GPT-labelled
8000	0.996	0.996	0.996	0.996	GPT-labelled
20000	0.996	0.996	0.996	0.996	GPT-labelled

On the unbalanced multi-class datasets shown in Table 4.5, BERT achieved strong performance on the multi-class classification task using GPT-labelled data, significantly outperforming manually-labelled data at the same scale. This confirms BERT's robustness against class imbalance when enough training data is available. Additionally, the gap between precision and recall remains narrow across all configurations, showing that the model is balanced in predicting both majority and minority classes. These results suggest that BERT is highly robust to class imbalance.

Table 4.5: BERT Multi-Class Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.783	0.768	0.769	0.792	Manually-labelled
4000	0.974	0.975	0.975	0.973	GPT-labelled
8000	0.993	0.995	0.994	0.995	GPT-labelled
16000	0.997	0.996	0.996	0.996	GPT-labelled

For the multi-label tasks shown in Table 4.6, BERT achieves excellent results with the complexity of handling multiple simultaneous labels. The macro F1 increases from 0.926 at 4,000 samples to 0.982 at 16,000 samples. Micro F1 follows a similar upward trend, reaching 0.984. BERT shows consistently high performance in both balanced and unbalanced settings (Figure 4.3). These scores indicate BERT's superior capacity for learning multi-label dependencies and subtle semantic cues in the review text.

Table 4.6: BERT Multi-Label Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Micro F1	Label Source
4000	0.977	0.885	0.926	0.938	GPT-labelled
8000	0.983	0.923	0.951	0.960	GPT-labelled
16000	0.991	0.974	0.982	0.984	GPT-labelled

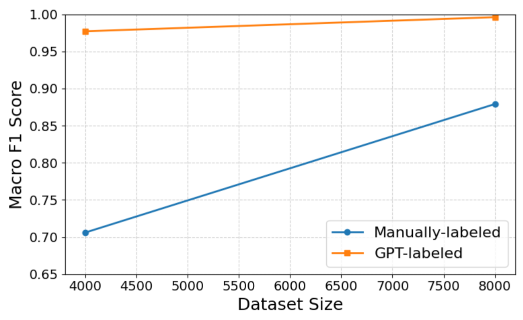


Figure 4.2: BERT Comparison: Manually vs GPT labelled (Multi-class)

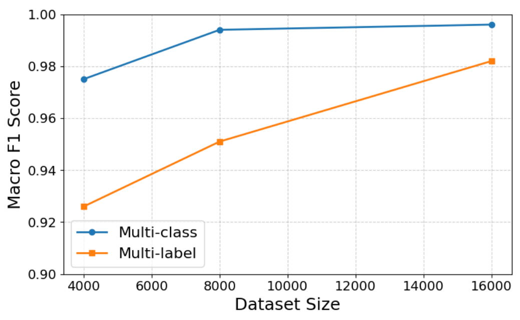


Figure 4.3: BERT Comparison: Multi-class vs Multi-label

4.3 Results for RoBERTa

In the multi-class classification task, RoBERTa shows strong performance across both manually-annotated and GPT-labelled balanced datasets. As shown in Table 4.7, the macro F1 on manually-labelled data increases from 0.755 at 4,000 samples to 0.893 at 8,000 samples, reflecting RoBERTa's capability to effectively learn class representations from moderately sized human-labelled data. On GPT-labelled balanced datasets, RoBERTa achieves very high performance even with only 4,000 training samples (F1 = 0.978), and reaches near-perfect scores (F1 = 0.993) with 20,000 samples. These results confirm RoBERTa's efficiency in adapting to automatically annotated data. Similar to BERT, the GPT-labelled data consistently yields better performance than manually-labelled data across all dataset sizes (Figure 4.4).

Table 4.7: RoBERTa Multi-Class Classification (Balanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.772	0.756	0.755	0.756	Manually-labelled
8000	0.895	0.893	0.893	0.893	Manually-labelled
4000	0.978	0.978	0.978	0.978	GPT-labelled
8000	0.991	0.991	0.991	0.991	GPT-labelled
20000	0.993	0.993	0.993	0.993	GPT-labelled

On the unbalanced multi-class datasets (Table 4.8), RoBERTa performs well, even on the manually-labelled dataset with only 4,000 samples, the macro F1 reaches 0.830. Performance on GPT-labelled unbalanced data is particularly strong, rising from 0.962 at 4,000 samples to 0.994 at 16,000 samples. Precision and recall remain stable across all settings, indicating balanced predictions across both dominant and minority classes. This supports RoBERTa's robustness under label distribution shifts.

Table 4.8: RoBERTa Multi-Class Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.834	0.829	0.830	0.836	Manually-labelled
4000	0.963	0.962	0.962	0.966	GPT-labelled
8000	0.991	0.989	0.990	0.990	GPT-labelled
16000	0.994	0.995	0.994	0.995	GPT-labelled

In the multi-label classification task (Table 4.9), RoBERTa achieves consistently high results. The macro F1 increases from 0.921 to 0.981, and micro F1 from 0.935 to 0.982 as the dataset grows from 4,000 to 16,000. The model maintains high precision and recall, suggesting its ability to simultaneously predict multiple labels with high confidence and coverage. Figure 4.5 shows that RoBERTa achieves high macro F1 scores in both multi-class and multi-label classification tasks, with slightly better performance observed in the multi-class setting across all dataset sizes

Table 4.9: RoBERTa Multi-Label Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Micro F1	Label Source
4000	0.968	0.885	0.921	0.935	GPT-labelled
8000	0.976	0.921	0.947	0.956	GPT-labelled
16000	0.992	0.970	0.981	0.982	GPT-labelled

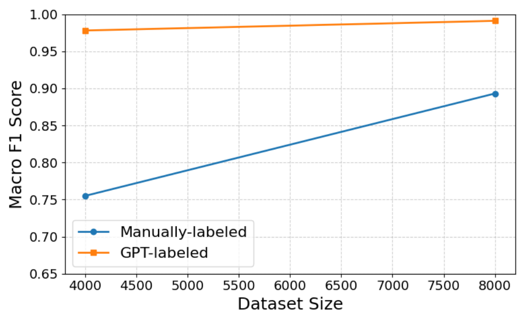


Figure 4.4: RoBERTa Comparison: Manually vs GPT labelled (Multi-class)

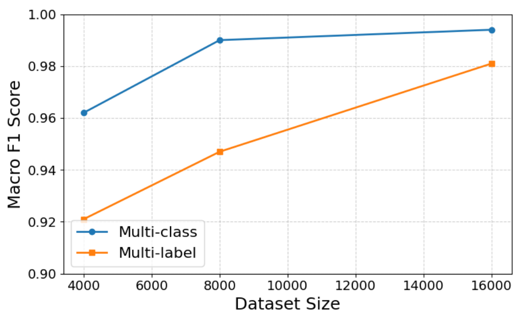


Figure 4.5: RoBERTa Comparison: Multi-class vs Multi-label

Overall, RoBERTa demonstrates superior performance across all experimental conditions. Its capacity to generalize well in both balanced and unbalanced settings, and under both multi-class and multi-label tasks. The results confirm that RoBERTa effectively uses training sets and GPT annotations to deliver highly accurate app review classification.

4.4 Results for BART

In the multi-class classification task (Table 4.10), BART achieves steady performance gains across both manually-labelled and GPT-labelled balanced datasets. As shown in Table 4.10, macro F1 improves from 0.769 to 0.811 on human-labelled data when increasing the training size from 4,000 to 8,000. These values are consistent with results from other transformer models but remain lower than GPT-labelled data under the same conditions (Figure 4.6). On the GPT-labelled balanced dataset, BART performs strongly even at smaller scales (F1 = 0.962 at 4,000 samples), reaching 0.995

with 20,000 samples—matching the upper bound observed in models like BERT and RoBERTa.

Table 4.10: BART Multi-Class Classification (Balanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.780	0.772	0.769	0.772	Manually-labelled
8000	0.816	0.812	0.811	0.812	Manually-labelled
4000	0.962	0.962	0.962	0.962	GPT-labelled
8000	0.990	0.990	0.990	0.990	GPT-labelled
20000	0.995	0.995	0.995	0.995	GPT-labelled

On the unbalanced multi-class dataset (Table 4.11), macro F1 is 0.816 with manually-labelled data at 4,000 samples, compared to 0.941 with GPT labels. As dataset size increases, with the model achieving a consistent 0.996 macro F1 at 16,000 samples. These results reinforce the model's capacity to generalize under imbalanced class distributions when trained on large-scale structured annotations.

Table 4.11: BART Multi-Class Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.840	0.803	0.816	0.829	Manually-labelled
4000	0.940	0.943	0.941	0.945	GPT-labelled
8000	0.990	0.991	0.991	0.990	GPT-labelled
16000	0.996	0.996	0.996	0.996	GPT-labelled

For the multi-label classification task (Table 4.12), BART demonstrates strong results across all dataset sizes. Macro F1 improves from 0.921 to 0.981 between 4,000 and 16,000 samples, with micro F1 following a similar trend (from 0.935 to 0.982). The relatively high recall and micro F1 values suggest that BART is capable of accurately identifying multiple labels per instance with both breadth and precision. As shown in Figure 4.7, BART performs slightly higher in multi-class tasks than multi-label tasks. Its performance closely aligns with BERT and RoBERTa, suggesting comparable ability to capture multi-label dependencies in review texts. BART offers a well performance across all tasks and dataset conditions.

4.5 Results for LLaMA

In the multi-class classification task, LLaMA shows highly competitive results. As shown in Table 4.13, the model achieves macro F1 scores of 0.762 and 0.881 on

Table 4.12: BART Multi-Label Classification (Unbalanced, GPT-labelled)

Dataset Size	Precision	Recall	Macro F1	Micro F1	Label Source
4000	0.947	0.898	0.921	0.933	GPT-labelled
8000	0.982	0.920	0.948	0.957	GPT-labelled
16000	0.989	0.974	0.981	0.982	GPT-labelled

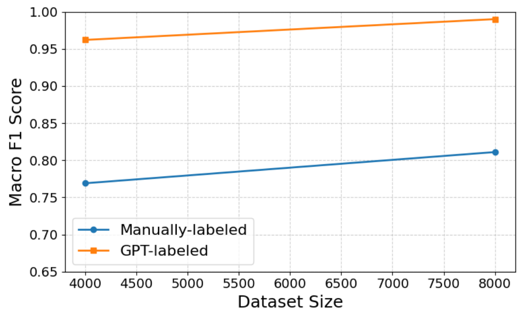


Figure 4.6: BART Comparison: Manually vs GPT labelled (Multi-class)

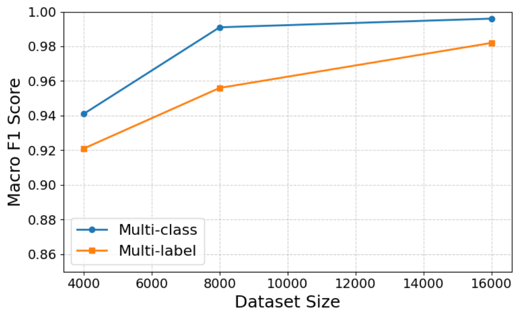


Figure 4.7: BART Comparison: Multi-class vs Multi-label

manually-labelled balanced datasets with 4,000 and 8,000 samples respectively. These values are comparable to those of other transformer models, showing no significant improvement in performance. On GPT-labelled balanced datasets, the performance improves dramatically, reaching near-perfect macro F1 of 0.998 at 20,000 samples. It shows that GPT-labelled data yields higher accuracy compared to manually-labelled data (Figure 4.8).

Table 4.13: LLaMA Multi-Class Classification (Balanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.773	0.764	0.762	0.764	Manually-labelled
8000	0.882	0.881	0.881	0.881	Manually-labelled
4000	0.959	0.958	0.958	0.958	GPT-labelled
8000	0.996	0.996	0.996	0.996	GPT-labelled
20000	0.998	0.998	0.998	0.998	GPT-labelled

On unbalanced multi-class datasets (Table 4.14), LLaMA again achieves exceptional performance. With only 4,000 GPT-labelled samples, it already achieves macro F1 = 0.989 and accuracy = 0.990. The improvement trend continues with larger datasets, where macro F1 reaches 0.996 at 16,000 samples. Compared to the manually-labelled unbalanced dataset (macro F1 = 0.790), the GPT-labelled data yields significantly higher results.

Table 4.14: LLaMA Multi-Class Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Accuracy	Label Source
4000	0.790	0.796	0.790	0.806	Manually-labelled
4000	0.988	0.989	0.989	0.990	GPT-labelled
8000	0.996	0.995	0.996	0.995	GPT-labelled
16000	0.995	0.996	0.996	0.996	GPT-labelled

In the multi-label classification task (Table 4.15), LLaMA maintains outstanding performance. With just 4,000 samples, macro F1 reaches 0.886 and continues to scale up to 0.993 at 16,000 samples. Micro F1 scores also follow this trend, increasing from 0.898 to 0.994. These results indicate that LLaMA handles multi-label dependencies effectively, even in the presence of class imbalance and multi-label tasks.

Table 4.15: LLaMA Multi-Label Classification (Unbalanced)

Dataset Size	Precision	Recall	Macro F1	Micro F1	Label Source
4000	0.928	0.857	0.886	0.898	GPT-labelled
8000	0.985	0.960	0.972	0.976	GPT-labelled
16000	0.997	0.989	0.993	0.994	GPT-labelled

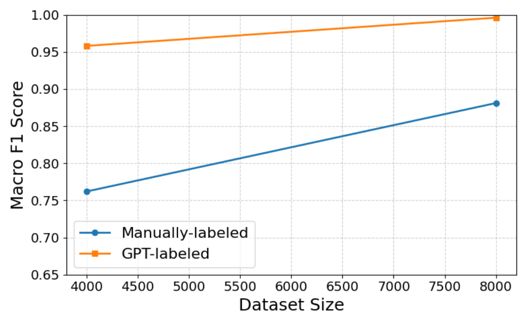


Figure 4.8: LLaMA Comparison: Manually vs GPT labelled (Multi-class)

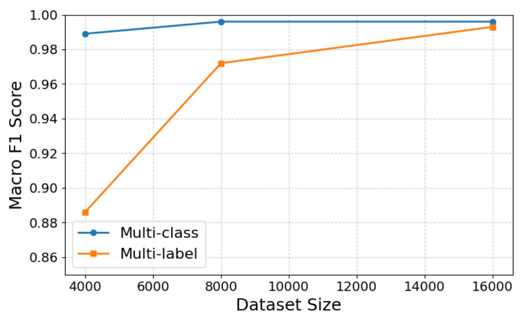


Figure 4.9: LLaMA Comparison: Multi-class vs Multi-label

Overall, LLaMA proves to be an efficient and powerful model for both multi-class and multi-label review classification (Figure 4.9), demonstrating strong performance even with smaller dataset sizes. Its results are largely comparable to those of BERT, RoBERTa, and BART, with no significant differences observed across the evaluated tasks.

4.6 Comparison and Observations

This section presents a model comparison and analysis of the experimental results obtained from five classification models. By systematically analysing model behaviour under varied configurations, this section aims to identify performance trends, model strengths and limitations, and trade-offs between accuracy and computational cost.

4.6.1 Multi-Class (MC) vs Multi-Label (ML) Classification Tasks

This subsection compares model performance and training efficiency between multi-class and multi-label classification tasks across all five models.

At the 4,000-sample scale (Figure 4.10 and Figure 4.11), SVM is the only model that performs better on the multi-label task (macro F1 = 0.854 and accuracy = 0.863) than on multi-class (macro F1 = 0.779 and accuracy = 0.789). In contrast, all transformer-based models (BERT, RoBERTa, BART, and LLaMA) achieve significantly higher macro F1 scores and accuracy on the multi-class task, each exceeding 0.94, with LLaMA achieves the best performance overall, reaching a macro F1 of 0.989 and accuracy of 0.990. BERT demonstrates the most stable performance, maintaining strong results across both tasks (macro F1 of 0.975 on multi-class and 0.926 on multi-label).

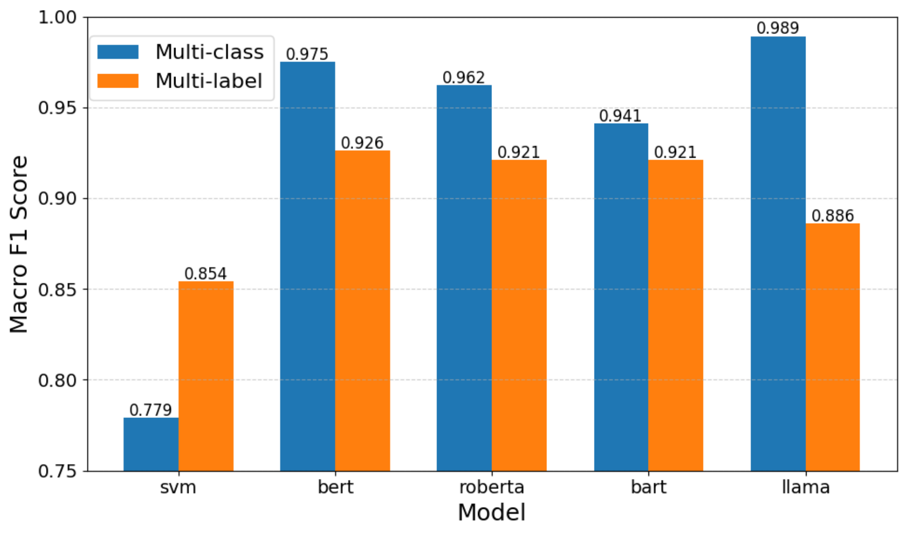


Figure 4.10: Macro F1 Comparison: MC vs ML (4000 samples)

At the 8,000-sample scale (Figure 4.12), the performance gap narrows across all models. SVM is also the only model whose macro F1 score is higher on the multi-label task (0.921) than on multi-class (0.866). This suggests that traditional models

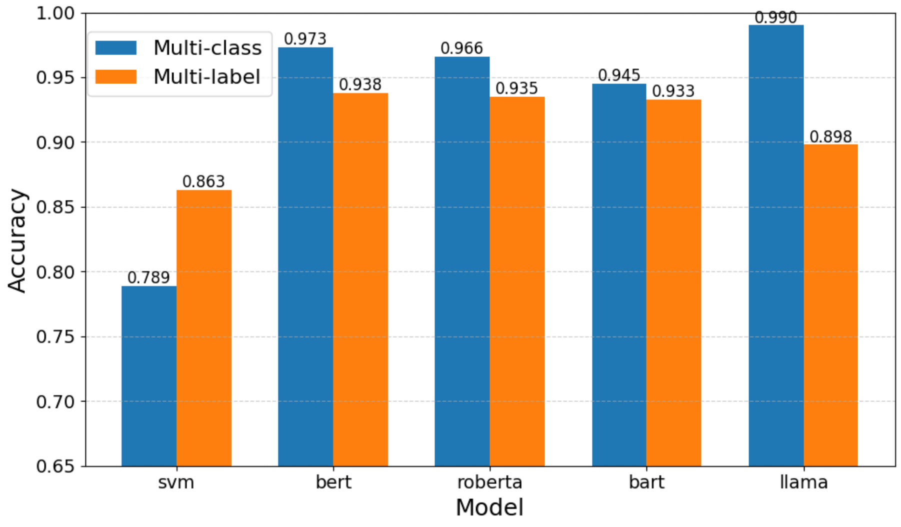


Figure 4.11: Accuracy Comparison: MC vs ML (4000 samples)

may do better from assigning multiple relevant labels when class boundaries are not clearly defined. In contrast, all transformer-based models achieve outstanding results on both tasks. On multi-class classification, every transformer model achieves a macro F1 score above 0.99, indicating near-perfect classification accuracy. Among them, LLaMA continues to lead in performance, reaching a macro F1 of 0.996. On the multi-label task, performance also remains high, which all transformer models exceed a macro F1 score of 0.94.

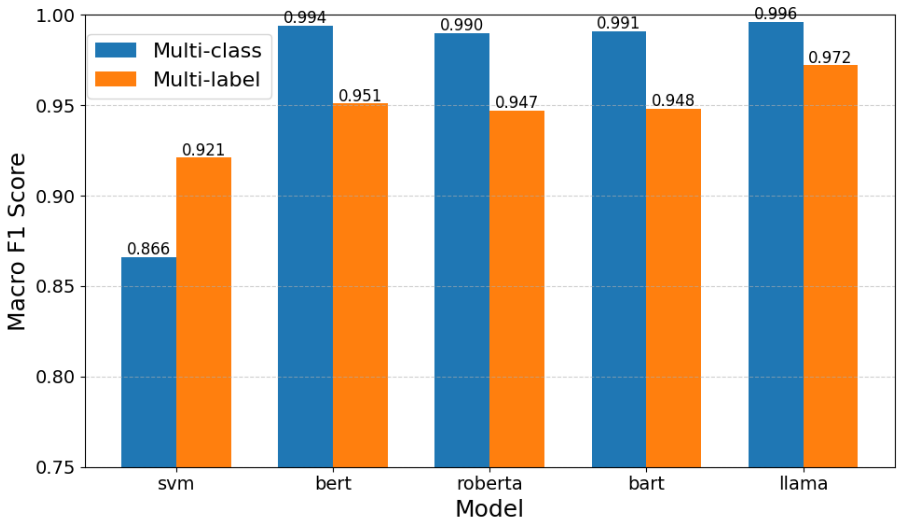


Figure 4.12: Macro F1 Comparison: MC vs ML (8000 samples)

In terms of training time (Table 4.16), multi-label classification generally requires more computational time than multi-class task, with the exception of SVM. SVM completes training in under one second for multi-class task and 25 seconds for multi-label task. For all transformer-based models, multi-class task training is notably slower. BERT and RoBERTa both show nearly double the training time on multi-class task compared to multi-label task (e.g., BERT: 734s and 352s). BART takes the longest time, requiring approximately three times more time for multi-class task (2001s and 587s). LLaMA demonstrates relevant training time and minimal variation between the two tasks (514s and 539s). Although LLaMA shows relatively moderate training time, it was trained on a more powerful GPU, meaning that the actual computational resources consumed were much higher than the number reported.

Table 4.16: Training Time on MC and ML Tasks (8000 samples)

Model	**Multi-Class**	**Multi-Label**
SVM	0.17	25
BERT	734	352
RoBERTa	733	374
BART	2001	587
LLaMA	514	539

In summary, while LLaMA demonstrate the highest performance, it come at the cost of significantly higher computational resources. In contrast, BERT and RoBERTa achieve strong and consistent performance across both classification tasks while maintaining relatively lower training time, making them favourable choices when balancing effectiveness and efficiency.

4.6.2 Balanced and Unbalanced Datasets

This subsection investigates the impact of class distribution—balanced versus unbalanced datasets—on model performance in multi-class classification tasks.

On the manually-labelled dataset of 4000 samples (Figure 4.13), every model achieves better performance on the unbalanced dataset compared to the balanced one. This difference is especially clear for SVM (0.678 to 0.765) and RoBERTa (0.755 to 0.830). For models like BERT, BART, and LLaMA, the unbalanced setup consistently provides higher macro F1 scores. This may indicate that the models benefited from the dominance of majority classes, or that balancing the dataset introduced additional noise and increased the learning difficulty of minority classes.

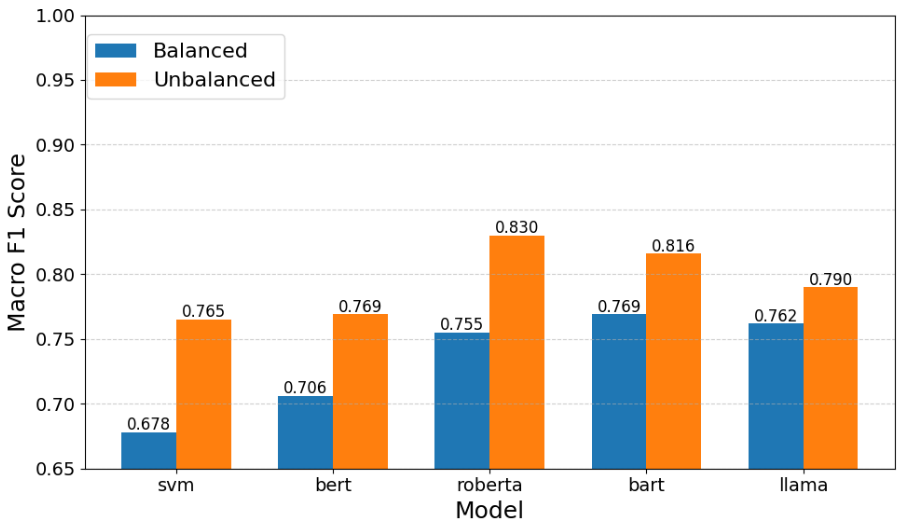


Figure 4.13: Macro F1 Comparison: Balanced vs Unbalanced (Manual)

On the GPT-labelled dataset of 4000 samples (Figure 4.14), the performance differences between balanced and unbalanced configurations are much smaller. For transformer models, the gap is minimal and often within 0.01–0.02 macro F1. BERT (0.977 and 0.975), RoBERTa (0.978 and 0.962), and BART (0.962 and 0.941) all perform slightly better on the balanced dataset, whereas LLaMA performs slightly better on the unbalanced version (0.958 and 0.989). SVM once again shows a difference under the unbalanced setup (0.701 to 0.779). The results suggest that transformer-based models are robust to class imbalance, especially when trained on GPT-labelled data.

Class imbalance has a greater effect on manually-labelled datasets, where the unbalanced configuration consistently yields better performance. In contrast, for GPT-labelled data, the performance difference between balanced and unbalanced settings is marginal. These results suggest that the impact of balancing strategies depends on label quality. In datasets with less consistent annotations, enforcing class balance may disrupt the natural distribution and degrade performance. In contrast, when label consistency is high, transformer-based models can learn robust representations even from imbalanced data, reducing the necessity of balancing. Therefore, balancing should not be applied by default, but rather evaluated based on the characteristics of the dataset and the model through empirical validation.

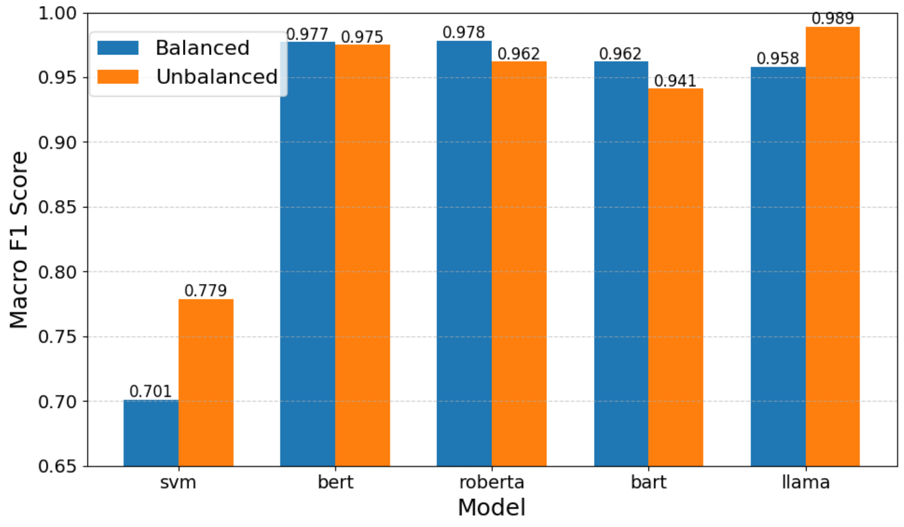


Figure 4.14: Macro F1 Comparison: Balanced vs Unbalanced (GPT)

4.6.3 Dataset Size

This subsection explores how increasing dataset size influences model performance across different classification tasks. Figures 4.15, 4.16, 4.17 illustrate macro F1 scores for all models under three settings: multi-class balanced, multi-class unbalanced, and multi-label unbalanced. For visualization purposes, two setting includes a zoomed-in view (excluding SVM) on the right to better illustrate trends among the transformer-based models, since values of SVM are substantially lower.

Across all configurations, macro F1 scores consistently improve with larger datasets, showing that more data helps models generalize better. However, the amount of improvement varies between models.

In the multi-class balanced setting (Figure 4.15), all models show a clear upward trend. SVM shows the largest gain, increasing from 0.701 at 4,000 samples to 0.942 at 20,000, reflecting its reliance on larger training data to learn reliable decision boundaries. Transformer-based models already perform well at 4,000 samples and improve only slightly after 8,000. By 8,000 samples, all transformer models exceed macro F1 = 0.99, showing that transformer models are able to extract sufficient semantic representations from even moderate-sized datasets under balanced conditions.

In the multi-class unbalanced setting (Figure 4.16), the pattern is similar. Again, SVM shows steady improvement with scale, reaching 0.932 at 16,000 samples. Transformer models quickly converge to high performance: all exceed 0.99 at 8,000 samples. The unbalanced nature of the dataset does not influence performance when there

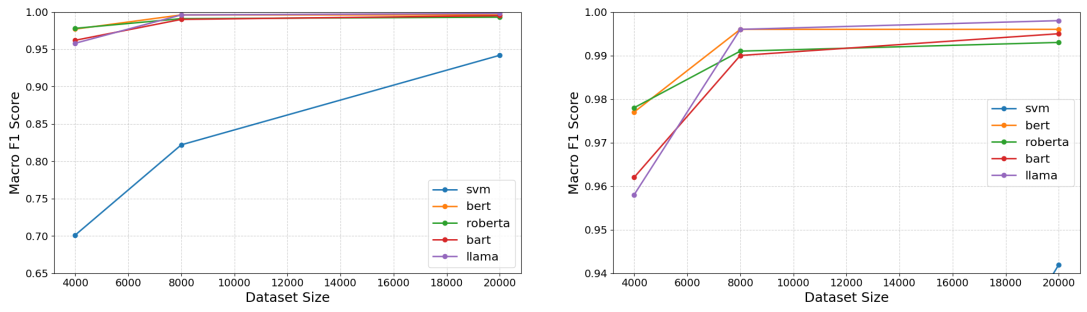


Figure 4.15: Model Comparison by Size (multi-class, balanced)

is sufficient data, further highlighting the robustness of deep architectures.

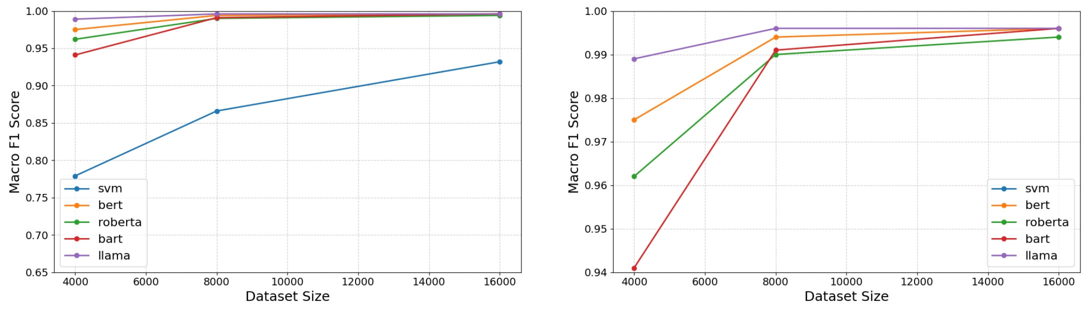


Figure 4.16: Model Comparison by Size (multi-class, unbalanced)

In the multi-label unbalanced setting (Figure 4.17), the trend remains consistent. All models are improved as dataset size increases. SVM improves from 0.854 at 4,000 to 0.969 at 16,000. LLaMA achieves the highest macro F1 at 16,000 samples (reaching to 0.993), followed closely by BERT, RoBERTa and BART.

In summary, increasing dataset size clearly helps all models, especially SVM. Transformer-based models benefit substantially from the initial increase in data size (from 4,000 to 8,000), after which performance tends to stabilize. These models achieve nearly optimal macro F1 scores even before the largest data scale is reached. These results highlight the scalability and high data efficiency of deep learning models.

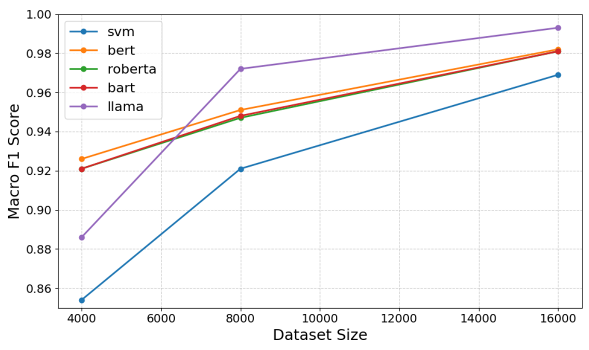


Figure 4.17: Model Comparison by Size (multi-label, unbalanced)

Chapter 5

Project Evaluation

This chapter consists of two parts. Section 5.1 provides a comprehensive summary of the performance of all five models. Section 5.2 provides a critical reflection on the project and identifies several limitations related to the dataset, labelling approach, and learning methodology.

5.1 Model Performance

Overall, transformer-based models consistently outperformed the traditional machine learning model. SVM showed weak performance on smaller datasets and in multi-class classification across all metrics. While its performance only improved when large training data was available, especially in multi-label tasks, but it still lagged behind the other models. SVM performed better on unbalanced datasets than on balanced ones at smaller data sizes like 4,000. This suggests that oversampling in the balanced setting may have introduced noise or weakened majority-class patterns, which traditional models like SVM rely on to form effective decision boundaries.

In contrast, BERT and RoBERTa demonstrated strong, stable performance under all experimental conditions. Both models achieved high macro F1 scores (often large or equal to 0.96) in multi-class tasks and remained robust in multi-label classification, even on small datasets. With just 4,000 samples, they already delivered competitive results, indicating excellent data efficiency and strong generalization ability.

BART showed decent classification performance, particularly in the multi-class task with large datasets. However, its results were generally slightly lower than those of BERT and RoBERTa, and it required at least twice the training time under similar settings. These factors make BART a less efficient choice when considering both effectiveness and resource usage.

LLaMA showed strong performance on small datasets in the multi-class task, performing competitively even at the 4,000 samples level. However, in the multi-label task, its performance was significantly lower than BERT, RoBERTa, and BART at the same scale, suggesting limited ability to handle complex label dependencies with limited data. As the dataset size increased, LLaMA's performance improved across both task types, eventually achieving the highest macro F1 scores in both multi-class and multi-label classification at 20,000 samples. This demonstrates that while LLaMA is highly effective with sufficient data, it is less reliable in low-resource, multi-label scenarios. Additionally, its high computational cost during training and inference may limit its practical use, despite its great performance.

Across the two task types, multi-class classification generally resulted in higher performance than multi-label classification, particularly for transformer-based models. This is likely because multi-class classification is structurally simpler, requiring the model to assign only one label per input, whereas multi-label classification involves learning more complex relationships between labels. Additionally, the evaluation in multi-label classification are typically stricter. A prediction is considered correct only if the full set of predicted labels exactly matches the true set. This makes multi-label tasks more sensitive to small errors such as missing or extra labels, leading to lower macro F1 scores overall.

In contrast, SVM often performed better in multi-label tasks. This outcome can be explained by the One-vs-Rest strategy used for multi-label classification simplifies the learning task into a series of binary problems, which fits well with SVM's model structure. This reduces the need to learn complex class boundaries in low-resource settings. Moreover, multi-label classification avoids the mutual exclusivity constraint, enabling the model to assign multiple relevant categories to a single review.

Comparing results from GPT-labelled and manually labelled datasets, it suggests that these models may benefit from keeping natural label distributions when labels are more variable, as balancing might introduce noise or weaken useful class patterns. In contrast, when labels are consistent and systematically generated, data balance has minimal impact on performance.

These results indicate that encoder-based transformers, especially BERT and RoBERTa, are the most robust and adaptable models for app review classification under various task conditions and dataset constraints.

5.2 Project Reflection

This section provides a reflection on the dataset, labelling approach, and learning methodology used in the project. While the results were encouraging, several limitations and design choices may affect how well the models work in real-world situations. The following subsections highlight key considerations and limitations of the work.

5.2.1 Data Limitations

The dataset used in this study supported well for testing and analysis. It provided a consistent structure, clear labelling, and sufficient volume to evaluate model performance across various experimental settings. However, it was limited in terms of language and variety.

First, all app reviews were written in English. While this ensured compatibility with widely used pre-trained language models and simplified preprocessing, it restricted the models to English-only settings. In real-world applications, user feedback often appears in multiple languages and reflects local expressions or cultural phrases. Since the models developed in this study were not trained on such language diversity, their effectiveness in multilingual or non-English environments remains untested and may not work well.

Second, the dataset was limited to reviews from only two Android app platforms: Google Play and F-Droid. As a result, the dataset does not capture user feedback from other major platforms, such as the Apple App Store, which is used by a large number of people around the world. User needs, habits, and review styles may be different on other platforms due to variations in users, interface design, and platform-specific rules [3]. Therefore, the current dataset may not fully reflect the diversity of user reviews across the broader app platforms.

5.2.2 Reliability of Automatically Generated Labels

One important factor that made large-scale training possible in this project was the use of GPT-generated labels to expand the dataset. This approach had two main advantages: labelling consistency and scalability. It made it possible to create a large training set in a short amount of time without the cost of manual labelling. The consistency of the generated labels also contributed to model stability during training, particularly for transformer-based models that benefit from structured input-output mappings.

However, the use of automatically generated labels has some limitations. Although GPT is capable of generating high-quality output, it is still weak to semantic ambiguity and systematic misclassification, especially when label categories are hard to separate or not clearly defined. In this study, categories such as user experience and rating text were especially difficult to distinguish due to their subjective nature and common co-occurrence in real-world reviews. For example, a review like "Love the design, still need improvement. Now 3 stars." could be interpreted as both user experience and a rating comment, depending on the labelling perspective.

To partially validate the reliability of the GPT-labelled dataset, a manual check of 3% of the data was conducted. The small sample size limits the confidence in the reliability of the dataset. The true quality of the training signal remains uncertain, particularly for ambiguous or multi-faceted reviews. Further manual labelling is needed to better understand potential patterns of mislabelling and analysis the true consistency of the GPT-generated labels.

5.2.3 Dependence on Supervised Learning

All five models in this study were developed using a supervised learning approach, which mainly depends on having a large amount of labelled data. While this method is still widely used for text classification, it comes with practical challenges—especially in areas where collecting high-quality labels is costly, slow, or hard to manage.

Although Pre-trained transformer models like BERT and RoBERTa showed fairly strong results even with smaller datasets (such as 4,000 labelled samples), they still needed thousands of task-specific examples to produce stable and reliable performance. This was especially true for multi-label classification, where consistent performance was only achieved when the training data increased to 8,000 samples or more. Likewise, LLaMA, despite being Pre-trained on a large amount of data, was more affected by limited training size. Its performance on smaller datasets was clearly weaker than BERT and RoBERTa, and only became competitive after training on 8,000 samples.

This strong reliance on large labelled datasets limits how easily and widely these models can be used. In real-world cases—such as specialized domains, or less commonly used languages—it can be difficult or expensive to collect enough labelled data. Creating labelled data in such cases often requires expert knowledge, making the process more time-consuming and complex.

In addition, this study did not explore other learning methods like few-shot, zero-shot, or semi-supervised learning. These techniques have gained interest because they reduce the need for large labelled datasets. For example, large language models can be guided to do classification tasks using only a few examples (few-shot) or even without specific training data (zero-shot), by using their general language understanding. Adding such approaches could help models adapt more easily to new topics or categories and perform better in situations with limited labelled data.

To summarize, although the supervised learning setup used in this study allowed for clear and consistent evaluation, its heavy dependence on labelled data is a key limitation. Future work should explore label-efficient learning methods to make app review classification more flexible and usable in real-world scenarios.

Chapter 6

Conclusion

This chapter presents a summary of the project's key outcomes and proposes directions for future research. Section 6.1 outlines the main achievements, while Section 6.2 discusses how the work can be extended to improve the scalability and applicability in requirements engineering.

6.1 Achievements

This project successfully addressed the challenge of classifying app reviews for requirements engineering through the application of supervised machine learning and deep learning techniques. By implementing and comparing five representative models, ranging from a traditional SVM to modern transformer-based architectures such as BERT, RoBERTa, BART, and LLaMA, the study provided a comprehensive evaluation of different modelling approaches across multiple classification tasks.

Three experimental settings were investigated: multi-class and multi-label classification, balanced and unbalanced data, and varying dataset sizes. Through these comparisons, the study demonstrated that transformer-based models consistently outperform traditional machine learning methods in both accuracy and robustness. In particular, BERT and RoBERTa achieved excellent results while requiring fewer computational resources and shorter training time. On smaller datasets, BERT and RoBERTa even outperformed LLaMA, highlighting their efficiency and suitability for data-constrained situations.

In addition to testing model performance, the project also reflected on several key challenges encountered during implementation. These included the lack of enough high-quality manually-labelled data and the limitations of depending only on supervised learning methods. By recognizing and discussing these challenges, the study provided a realistic perspective on the practical constraints of applying deep learning to requirements engineering, and set the foundation for identifying directions for future research.

6.2 Future Work

Building on the reflections discussed in Section 5.2, several directions are proposed to improve the work of app review classification for future work.

The first direction for future work is to improve the language and platform variety in the dataset. Since this study focused only on English reviews from two Android platforms, it does not capture the full range of language use and context found in global app markets. To support multilingual use, future research should collect and include user reviews in other languages, such as Chinese, Hindi, Spanish, Arabic, and Russian. This can be done by directly gathering data from app stores. To process multilingual input, Pre-trained models like mBERT [10] or XLM-RoBERTa [8] could be trained with tasks across multiple languages. Future work could also examine whether multilingual models perform differently across languages, especially in terms of classification accuracy and category consistency.

Along with language coverage, future work should also improve platform coverage by adding data from iOS platforms like the Apple App Store, which may show different review patterns in terms of tone, length, and topics [3]. Reviews from enterprise or less common app stores (e.g., Huawei AppGallery, Amazon AppStore) could also add variety by showing different user habits. Also, the performance of models in different platforms could be examined by using techniques such as domain-adaptive pretraining or domain adaptation methods. For example, models could first be trained on large amounts of platform-specific text before being fine-tuned on the classification task. This would help models better handle different writing styles and user expectations when used in real-world environments.

The second important direction is to reduce the need for large amounts of labelled data by using more efficient training methods. Although GPT-generated labels made large-scale experiments possible in this study, they also introduced some label errors. To improve label use and quality, future work could use semi-supervised learning

[73]. For example, pseudo-labelling allows an initial model trained on a small human-labelled set to generate labels for the rest of the data, which can then be refined by selecting the most confident predictions for further training [28].

In addition, active learning could help reduce the cost of manual labelling. With this method, the model selects the most uncertain or varied examples for annotation, making each labelled sample more useful [56]. This is especially helpful when expert input is needed or labelling is costly. Future work could test different sampling methods, such as choosing samples based on uncertainty and diversity, to build smaller but more useful training sets.

To further reduce data needs, few-shot and zero-shot learning should be explored [5, 2]. Large language models have shown strong performance in classification tasks by using in-context learning, where a few examples are provided at prediction time without further training [2]. Prompt-based zero-shot classification is also possible by describing each label in natural language and asking the model to choose the best match. These methods are especially useful for quickly applying models in new domains or in areas where labelled data is not available.

In summary, future research should focus on two main goals: increasing the variety and realism of training data and improving how models learn from fewer high-quality labels. These steps will help build review classification systems that are more flexible, reliable, and practically deployable in different real-world software situations. Such progress is essential for improving the performance of automated user review analysis across diverse application ecosystems and user communities.

Bibliography

[1] M. Ajagbe and L. Zhao. Retraining a bert model for transfer learning in requirements engineering: A preliminary study. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*, pages 309–315. IEEE, 2022.

[2] W. Alhoshan, A. Ferrari, and L. Zhao. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology*, 159:107202, 2023.

[3] M. Ali, M. E. Joorabchi, and A. Mesbah. Same app, different app stores: A comparative study. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 79–90. IEEE, 2017.

[4] M. Binkhonain and L. Zhao. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, 1:100001, 2019.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, pages 767–778, 2014.

[7] R. Colin. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21, 2020.

[8] A. Conneau, A. Baevski, R. Collobert, A. Mohamed, and M. Auli. Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*, 2020.

[9] A. F. de Araújo and R. M. Marcacini. Re-bert: automatic extraction of software requirements from app reviews using bert language model. In *Proceedings of the 36th annual ACM symposium on applied computing*, pages 1321–1327, 2021.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[11] B. Dou, Z. Zhu, E. Merkurjev, L. Ke, L. Chen, J. Jiang, Y. Zhu, J. Liu, B. Zhang, and G.-W. Wei. Machine learning methods for small data challenges in molecular science. *Chemical Reviews*, 123(13):8736–8780, 2023.

[12] M. Grandini, E. Bagli, and G. Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[13] K. Gupta, B. Thérien, A. Ibrahim, M. L. Richter, Q. Anthony, E. Belilovsky, I. Rish, and T. Lesort. Continual pre-training of large language models: How to (re) warm your model? *arXiv preprint arXiv:2308.04014*, 2023.

[14] 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)*, pages 153–162. Ieee, 2014.

[15] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[18] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

[19] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *2013 10th working conference on mining software repositories (MSR)*, pages 41–44. IEEE, 2013.

[20] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[21] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*, 2021.

[22] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[23] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[24] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[25] S. Kujala. User involvement: a review of the benefits and challenges. *Behaviour & information technology*, 22(1):1–16, 2003.

[26] Z. Kurtanović and W. Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 490–495. Ieee, 2017.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] D.-H. Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896. Atlanta, 2013.

[29] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training

for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[30] Q. Lhoest, A. V. Del Moral, Y. Jernite, A. Thakur, P. Von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, et al. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*, 2021.

[31] S. Li, H. Yan, and X. Qiu. Contrast and generation make bart a good dialogue emotion recognizer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 11002–11010, 2022.

[32] Y. Liu. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*, 2019.

[33] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[34] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[35] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21:311–331, 2016.

[36] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE, 2015.

[37] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. Toward data-driven requirements engineering. *IEEE software*, 33(1):48–54, 2015.

[38] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern recognition*, 45(9):3084–3104, 2012.

[39] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[40] W. McKinney et al. Data structures for statistical computing in python. *SciPy*, 445(1):51–56, 2010.

[41] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[42] E. Noei, F. Zhang, and Y. Zou. Too many user-reviews! what should app developers look at first? *IEEE Transactions on Software Engineering*, 47(2):367–378, 2019.

[43] OpenAI. Chatgpt. `https://chat.openai.com`, 2023.

[44] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)*, pages 125–134. IEEE, 2013.

[45] V. L. Parsons. *Stratified Sampling*, pages 1–11. John Wiley & Sons, Ltd, 2017.

[46] A. Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[48] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[49] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *Science China technological sciences*, 63(10):1872–1897, 2020.

[50] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[51] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[52] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine learning*, 85:333–359, 2011.

[53] A. Rogers, O. Kovaleva, and A. Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the association for computational linguistics*, 8:842–866, 2021.

[54] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18, 2019.

[55] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[56] B. Settles. Active learning literature survey. 2009.

[57] M. Siino, I. Tinnirello, and M. La Cascia. Is text preprocessing still worth the time? a comparative survey on the influence of popular preprocessing methods on transformers and traditional classifiers. *Information Systems*, 121:102342, 2024.

[58] I. Sommerville. Software engineering 9th edition. *ISBN-10*, 137035152:18, 2011.

[59] Number of apps available in leading app stores 2024. Available at: `http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`, Accessed: 2025-03-02.

[60] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? In *China national conference on Chinese computational linguistics*, pages 194–206. Springer, 2019.

[61] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[62] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications*, pages 64–74, 2008.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[64] S. I. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94, 2012.

[65] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[66] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[67] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[68] L. Wu, Z. Zheng, Z. Qiu, H. Wang, H. Gu, T. Shen, C. Qin, C. Zhu, H. Zhu, Q. Liu, et al. A survey on large language models for recommendation. *World Wide Web*, 27(5):60, 2024.

[69] P. Xu, X. Ji, M. Li, and W. Lu. Small data machine learning in materials science. *npj Computational Materials*, 9(1):42, 2023.

[70] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2013.

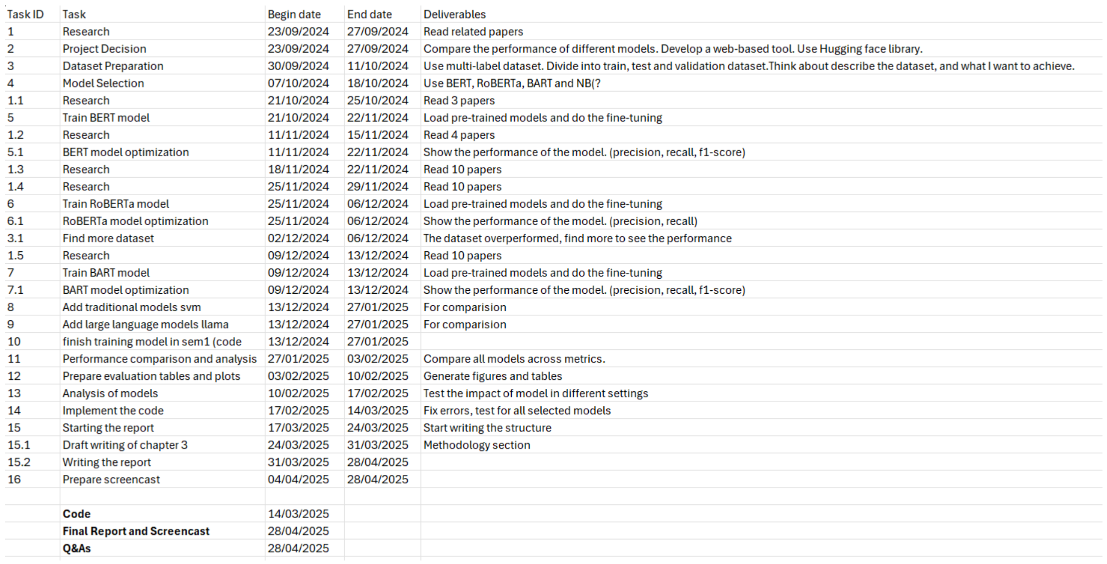
[71] R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[72] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

[73] X. Zhu and A. Goldberg. *Introduction to semi-supervised learning*. Morgan & Claypool Publishers, 2009.

Appendix A

Plan



Task ID	Task	Begin date	End date	Deliverables
1	Research	23/09/2024	27/09/2024	Read related papers
2	Project Decision	23/09/2024	27/09/2024	Compare the performance of different models. Develop a web-based tool. Use Hugging face library.
3	Dataset Preparation	30/09/2024	11/10/2024	Use multi-label dataset. Divide into train, test and validation dataset.Think about describe the dataset, and what I want to achieve.
4	Model Selection	07/10/2024	18/10/2024	Use BERT, RoBERTa, BART and NB(?
1.1	Research	21/10/2024	25/10/2024	Read 3 papers
5	Train BERT model	21/10/2024	22/11/2024	Load pre-trained models and do the fine-tuning
1.2	Research	11/11/2024	15/11/2024	Read 4 papers
5.1	BERT model optimization	11/11/2024	22/11/2024	Show the performance of the model. (precision, recall, f1-score)
1.3	Research	18/11/2024	22/11/2024	Read 10 papers
1.4	Research	25/11/2024	29/11/2024	Read 10 papers
6	Train RoBERTa model	25/11/2024	06/12/2024	Load pre-trained models and do the fine-tuning
6.1	RoBERTa model optimization	25/11/2024	06/12/2024	Show the performance of the model. (precision, recall)
3.1	Find more dataset	02/12/2024	06/12/2024	The dataset overperformed, find more to see the performance
1.5	Research	09/12/2024	13/12/2024	Read 10 papers
7	Train BART model	09/12/2024	13/12/2024	Load pre-trained models and do the fine-tuning
7.1	BART model optimization	09/12/2024	13/12/2024	Show the performance of the model. (precision, recall, f1-score)
8	Add traditional models svm	13/12/2024	27/01/2025	For comparision
9	Add large language models llama	13/12/2024	27/01/2025	For comparision
10	finish training model in sem1 (code	13/12/2024	27/01/2025	
11	Performance comparison and analysis	27/01/2025	03/02/2025	Compare all models across metrics.
12	Prepare evaluation tables and plots	03/02/2025	10/02/2025	Generate figures and tables
13	Analysis of models	10/02/2025	17/02/2025	Test the impact of model in different settings
14	Implement the code	17/02/2025	14/03/2025	Fix errors, test for all selected models
15	Starting the report	17/03/2025	24/03/2025	Start writing the structure
15.1	Draft writing of chapter 3	24/03/2025	31/03/2025	Methodology section
15.2	Writing the report	31/03/2025	28/04/2025	
16	Prepare screencast	04/04/2025	28/04/2025	
	Code	14/03/2025		
	Final Report and Screencast	28/04/2025		
	Q&As	28/04/2025		

Figure A.1: Project plan containing key tasks and milestones

Figure A.1 presents the project plan, outlining the main tasks and milestones throughout the work process. It structured with clear timelines to ensure steady progress and deliverable tracking.