AUTOMATED USER REVIEW ANALYSIS TO FACILITATE POTENTIAL MOBILE APPLICATION EVOLUTION

A.D.S.R.Gunathilaka

219337X

Master of Science in Computer Science

Department of Computer Science & Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

October 2024

AUTOMATED USER REVIEW ANALYSIS TO FACILITATE POTENTIAL MOBILE APPLICATION EVOLUTION

A.D.S.R.Gunathilaka

219337X

Thesis submitted in partial fulfillment of the requirements for the degree
Master of Science in Computer Science

Department of Computer Science & Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

October 2024

DECLARATION

I declare that this is my own work and this Thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date: 02/27/2025

The supervisor should certify the Thesis with the following declaration.

The above candidate has carried out research for the Master of Science in Computer Science Thesis under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. Nisansa de Silva

Signature of the Supervisor: Date: 27/02/2025

ACKNOWLEDGEMENT

I would like to express my profound gratitude to several individuals and institutions who have been instrumental in the successful completion of this thesis:

- First and foremost, I extend my heartfelt thanks to my supervisor, Dr. Nisansa de Silva, for his invaluable guidance, unwavering support, and insightful feedback throughout the course of this research. His expertise and encouragement have been pivotal in shaping this work.
- I am deeply grateful to the entire faculty and staff of the Department of Computer Science and Engineering for their dedication in delivering high-quality lectures, conducting rigorous examinations, and providing both major and minor support that has contributed significantly to the success of the MSc program.
- My sincere appreciation goes to all the staff members of the University of Moratuwa whose tireless efforts behind the scenes have been crucial in making this MSc program a resounding success.
- I would like to express my heartfelt thanks to my fellow MSc students for their camaraderie, intellectual discussions, and mutual support throughout this academic journey. Your friendship and collaboration have made this experience truly enriching.
- I extend my gratitude to the esteemed panel of examiners for their time, expertise, and constructive feedback in evaluating our research efforts. Your insights have undoubtedly enhanced the quality of this work.
- Last but not least, I wish to acknowledge the broader research and open-source communities worldwide. Your collective efforts in advancing scientific knowledge and providing accessible tools and resources have been instrumental in facilitating this research.

This journey would not have been possible without the support and contributions of all these individuals and groups. I am profoundly thankful for the opportunity to learn, grow, and contribute to the field through this research endeavor.

ABSTRACT

User reviews are crucial for mobile application evolution, but manually analyzing large volumes of feedback is time-consuming and inefficient. This thesis explores advanced techniques for automated user review analysis to facilitate potential mobile application evolution. We introduce a novel Convolutional Neural Network (CNN) based approach for Aspect-Based Sentiment Analysis (ABSA) on app reviews. Our model demonstrates significant improvements over existing baselines, with F1 scores of 0.62, 0.42, and 0.62 for aspect category classification in Productivity, Game, and Social Networking domains respectively. For aspect sentiment classification, we achieve accuracy scores of 0.80, 0.70, and 0.86 in the same domains. We provide empirical evidence on hyperparameter tuning, investigating the effects of batch size, number of epochs, and learning rate on model performance. To enhance our ABSA model, we investigate various word embeddings and data augmentation techniques. We find that Word2Vec embeddings and Round-trip Translation (RTT) augmentation yield the best results, offering insights for future research in this domain. Expanding our exploration of automated review analysis, we evaluate the potential of Large Language Models (LLMs). We provide a comprehensive comparison of state-of-the-art commercial and open-source LLMs in zero-shot and fine-tuned settings. Notably, we demonstrate the feasibility of using commercial LLMs as autonomous annotators, creating a high-quality dataset of 10,000 app reviews while achieving an accuracy of 81.89%. Our research also investigates the impact of various parameters on LLM performance for app review analysis, including training data size, number of epochs, Temperature , and Top_p. We find that fine-tuned open-source models can achieve performance comparable to commercial LLMs, with our best model reaching an F1 score of 0.83416. Our work contributes to the field of mobile application development by advancing automated user review analysis techniques to potentially improve the process of mobile application evolution. This work has the potential to lead to faster and more targeted mobile application improvements, enhancing developers' ability to respond to user needs effectively.

Keywords: Mobile App Review Analysis, Aspect-Based Sentiment Analysis, Convolutional Neural Networks, Large Language Models, Word Embeddings, Data Augmentation, Hyperparameter Tuning, `GPT-3.5`, `LLAMA 2`, `Mistral`, Fine-tuning, Software Evolution

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

With the recent technological advancements in the smartphone industry, the popularity and usage of smartphones are growing at a rapid pace. According to data from Statista[1], the number of smartphone users is expected to have steady growth up to 7516 million by 2026.

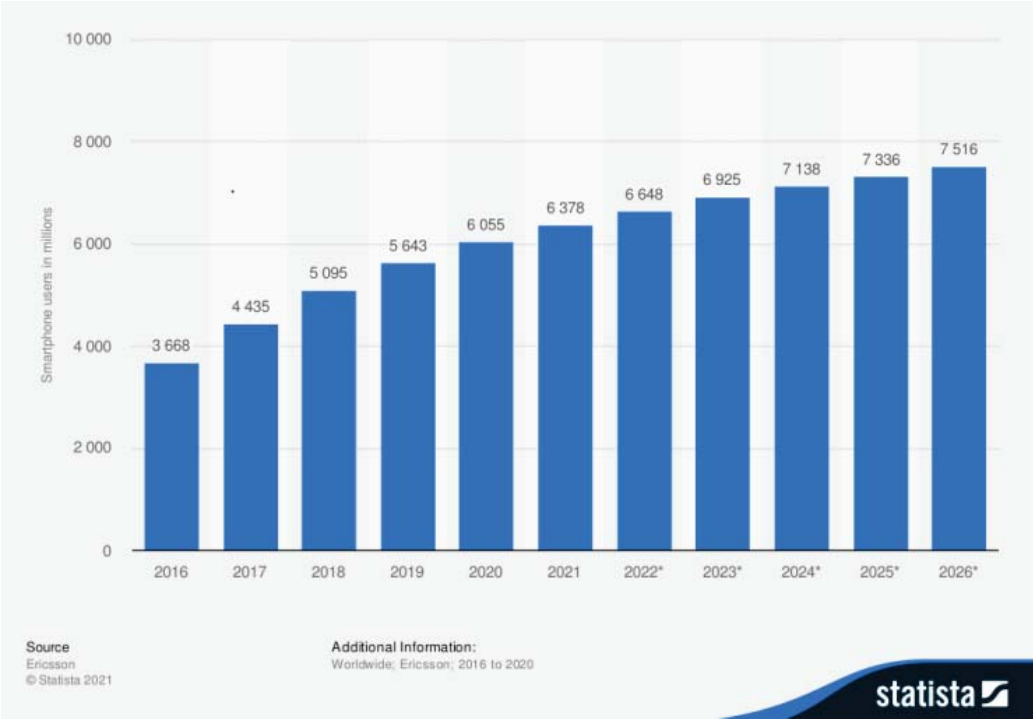


Fig. 1.1: Number of smartphone users from 2016 to 2021 [1]

Due to the high computational power of these devices and the flexibility provided by OS governing bodies such as Google and Apple for publishing 3rd party applications, a huge market for developing mobile applications has emerged. Consequently, the mobile application market has been growing at a rapid pace, becoming highly competitive. Users now have a plethora of mobile applications to choose from to fulfill their needs. This competitive landscape motivates application developers to constantly improve their applications, quickly identify and fix hidden bugs, and add new trending features to stay relevant in the market.

[1]Statista https://www.statista.com/

Mobile applications are typically published on app marketplaces such as Google Play[2] and Apple App Store[3], where users can rate and post feedback on the applications. These reviews often contain valuable information such as hidden bugs and new features that users expect from the application [2, 3]. As a result, analyzing user reviews to extract useful information has become a crucial topic among researchers and developers alike.

1.2 Research Problem

The widespread adoption of mobile applications has fundamentally transformed the way users engage with technology. In today's highly competitive landscape of platforms such as Google Play Store and Apple App Store, the need for rigorous requirements engineering and efficient software evolution processes has become paramount [4, 5]. User reviews on these platforms serve as a valuable source of feedback for developers, containing information about bugs, feature requests, and user experiences [3, 6]. However, the sheer volume and unstructured nature of these reviews pose significant challenges for developers in extracting actionable insights [7].

In the literature, we can see a clear progression of researchers applying traditional machine learning techniques to deep learning techniques to extract useful information that could help developers in the software evolution process [8–10]. This evolution in approaches reflects the growing sophistication of Natural Language Processing (NLP) methods and their application to the domain of app review analysis.

Despite these advancements, there remains significant room for improvement in the accuracy and efficiency of extracting actionable insights from user feedback [7]. The dynamic nature of mobile app development and the continually evolving user expectations create an ongoing need for more advanced and adaptable analysis techniques [10].

Through this study, we aim to contribute to the ongoing effort of improving app review analysis by applying emerging NLP techniques. Our goal is to develop more effective methods for analyzing user feedback, potentially streamlining the mobile application evolution process and enhancing the overall experience of mobile app users.

[2]Google play https://play.google.com/store/apps?hl=en&gl=US.
[3]Apple store https://www.apple.com/app-store/.

1.3 Research Objectives

- Leverage Aspect-Based Sentiment Analysis (ABSA) to identify and analyze user sentiments on various aspects of mobile applications, providing developers with a nuanced understanding of user satisfaction across different app features and functionalities.
- Evaluate the capabilities of Large Language Models (LLMs) in classifying and analyzing app reviews, with the aim of improving the accuracy and efficiency of extracting user-reported issues, feature requests, and experience-related information.

1.4 Contributions

This research contributes to the field of mobile application development, particularly in the area of software evolution, in several ways:

- Proposed and evaluate a CNN-based approach for Aspect-Based Sentiment Analysis (ABSA) on mobile app reviews, demonstrating significant improvements over existing baselines provided by Alturaief et al. [11]
- Provide a comparative analysis of Word2Vec, FastText, and GloVe embeddings for app review analysis, identifying the most effective embedding that work best with our proposed CNN models
- Evaluate Contextual Augmentation by Google Bard and Round-Trip Translation (RTT) data augmentation techniques for improving ABSA model performance on imbalanced datasets by over-sampling minority classes
- Provide empirical evidence on hyperparameter tuning (batch size, number of epochs, learning rate) for Proposed CNN ABSA models in the context of app review analysis
- Conduct a comprehensive comparison of commercial and open-source Large Language Models (LLMs) for app review classification
- Propose and evaluate a method for using commercial LLMs as autonomous annotators to create large-scale, high-quality datasets for app review classification
- Analyze the effects of training data size and number of epochs on the performance of fine-tuned open-source LLMs for app review classification
- Investigate the impact of `Temperature` and `Top_p` parameters on the performance of both commercial and fine-tuned open-source LLMs in app review analysis

CHAPTER 2

LITERATURE SURVEY

In this Chapter we will take a look at the prior work done in the area of mobile app reviews mining. For the convenience of use, this chapter is divided into several sections. In the section 2.1 we discuss about the importance of user reviews with regards to identifying useful information related to mobile applications. After that we will take look at various approaches followed by researchers to extract useful information out of user reviews under two main categories: 1) Traditional Machine Learning (Section 2.2) and 2) Deep learning (Section 2.3). Then in section 2.4 we will go through pre-processing techniques utilized in previous studies to improve accuracy of the results. Then we will summarize all the previous studies in the table 2.4. Finally we go through prior work done on using LLMs (Large Language Models) as a annotator to improve efficiency and reduce costs associated with the manual annotation proess (Section 2.6).

2.1 Importance of user feedback for apps development

In the literature there are number of works highlighted the importance of User Feedback. In an empirical study Pagano and Maalej [12] stated that most of the feedback given by the users after a new release and the frequency of feedback submitted decreases over the time. They also found out that these feedback typically contains multiple topics related to the application such as user experience issues, bug reports, and feature requests. It was pointed out that these feedback content has an impact on download numbers of the application. According to the study majority of low star rating feedback usually contains shortcomings and bug reports of the application where four to five star ratings mainly consist of praise. It was noted that the feature requests are mostly coming from three to five star rating feedback. Another study done by Li et al. [13] showed that analyzing user comments can be used to improve user satisfaction of software products.

Maalej et al. [14] categorize user feedback into two types called *implicit feedback* and *explicit feedback* while describing the concept of *User Feedback Analytics*. There software usage data such as click events,logs, sensor generated data categorized under implicit feedback and user reviews and feedback's are categorized as explicit feedback. It was mentioned that explicit feedback summarizes users' subjective opinions, implicit feedback helps developers understand their objective opinions.

Hence Explicit user feedback is really important since it contains valuable information. However,due to the large volume and noisy-nature of those reviews, manually analyzing them for useful information is a laborious and challenging task [15]. So researches have applied various types of NLP and machine learning techniques to solve

this problem of extracting useful information out of large number of user feedback. For the sake of simplicity we will discuss them under following: 1) Traditional Machine Learning (Section 2.1) and 2) Deep learning (Section 2.3)

2.2 Traditional Machine Learning Approaches

Prior to the emergence of deep learning techniques, researchers employed various traditional machine learning approaches to analyze app reviews. These approaches evolved from basic topic modeling to more sophisticated supervised and hybrid methods, demonstrating a clear progression in addressing the complex challenges of user feedback analysis in mobile applications.

2.2.1 Topic Modeling-Based Approaches

The foundation of automated app review analysis began with topic modeling techniques, starting with Fu et al. [16]'s Wiscom framework that introduced a three-level analysis system combining micro-level review consistency analysis, meso-level temporal evolution tracking, and macro-level marketplace trend identification. While this comprehensive approach broke new ground, it struggled with precise classification of specific user concerns. Addressing these limitations, Chen et al. [6] significantly enhanced the methodology by integrating EMNB classification with topic modeling, improving the identification of informative reviews and reducing noise in the analysis process.

The versatility of topic modeling expanded further when Anchiêta and Moura [17] and Gomez et al. [18] demonstrated its applicability beyond English-language reviews, though challenges remained in handling linguistic nuances and maintaining consistent accuracy across languages. These cross-lingual adaptations highlighted both the flexibility of topic modeling approaches and their inherent limitations in capturing language-specific contextual nuances.

2.2.2 Supervised Learning Approaches

The limitations of pure topic modeling approaches, particularly in classification accuracy and result validation, led to increased adoption of supervised learning techniques. Comprehensive comparative studies by Guzman et al. [8] revealed that ensemble methods consistently outperformed individual classifiers like Naive Bayes, SVMs, and basic Neural Networks, achieving 5-10% better accuracy across various classification tasks. This finding highlighted the importance of combining multiple classification strategies to handle the diverse nature of user feedback.

Building on these insights, Maalej et al. [19] significantly advanced the field by integrating metadata analysis with text classification, achieving unprecedented precision

rates of 88-92% and recall rates of 90-99%. Their multi-faceted approach demonstrated the value of combining multiple data sources and analytical techniques, incorporating:

- Comprehensive review metadata (star ratings, submission timestamps, user history)
- Advanced linguistic feature analysis (tense analysis, bi-gram patterns, semantic relationships)
- Sophisticated sentiment analysis techniques
- Contextual information from app descriptions and update histories

2.2.3 Specialized Analysis Frameworks

As the field matured, researchers developed increasingly sophisticated frameworks addressing specific limitations of earlier approaches. Phong et al. [15]'s MARK framework addressed the rigidity of fully automated systems by introducing analyst-guided processing, enabling:

- Dynamic keyword definition for targeted analysis
- Real-time tracking of temporal trends
- Early detection of emerging issues through change monitoring
- Customizable analysis parameters based on specific app categories

Meanwhile, Gu and Kim [20]'s SUR-Miner overcame the limitations of traditional bag-of-words approaches through pattern-based parsing, offering enhanced semantic understanding and more precise feature extraction. Guzman et al. [10]'s ALERTme further expanded the scope by successfully adapting these techniques to social media data, though each advancement revealed new challenges in scalability and generalization across different platforms and user behaviors.

2.2.4 Active Learning and Hybrid Approaches

The persistent challenge of limited labeled data spurred innovation in hybrid approaches. Dhinakaran et al. [21]'s active learning framework significantly reduced manual labeling requirements while maintaining accuracy comparable to fully supervised methods. This breakthrough addressed one of the major practical limitations in implementing app review analysis systems at scale.

Building on this efficiency improvement, Guo and Singh [22] shifted focus toward extracting actionable insights through action-problem pairs, demonstrating how targeted analysis could provide more practical value than broader classification approaches. Their method achieved:

- More precise identification of specific user problems and their contexts
- Better understanding of user behaviors and interaction patterns
- Improved actionability of extracted insights for development teams
- Enhanced prioritization of issues based on user impact and frequency

2.2.5 Comparative Analysis

The evolution of traditional machine learning approaches reveals several key trends and trade-offs in methodology:

longtable

TABLE 2.1: Comparison of Traditional Machine Learning Approaches

Approach	**Advantages**	**Limitations**
Topic Modeling	• Unsupervised learning • Good for exploratory analysis • Minimal manual labeling • Flexible topic discovery	• Lower precision • Difficult to validate results • High computational cost • Topic coherence issues

Approach	Advantages	Limitations
Supervised Classification	• Higher accuracy • Better handling of specific tasks • More reliable results • Clear performance metrics	• Requires labeled data • Domain-specific training • Limited generalizability • High annotation costs
Specialized Frameworks	• Task-specific optimization • Better semantic understanding • Enhanced user control • Focused analysis capabilities	• Limited application scope • Complex implementation • Higher development overhead • Maintenance challenges
Active Learning	• Reduced labeling effort • Balanced performance • Practical applicability • Adaptive learning capability	• Complex implementation • Iterative training needed • Initial setup overhead • Quality dependency on seed data

These traditional approaches established fundamental principles and methodologies that continue to influence modern app review analysis. The progression from basic topic modeling to specialized frameworks and hybrid approaches demonstrates the field's maturation in addressing increasingly complex challenges. While newer deep learning approaches have since emerged, many of the insights and methodological in-

novations from this era remain relevant, particularly in scenarios where interpretability, control, and efficient resource utilization are prioritized over raw computational power.

The evolution of these approaches also highlights the inherent trade-offs between automation and accuracy, generalizability and specificity, and resource requirements versus performance. These considerations continue to shape current research directions and the development of new methodologies for app review analysis.

2.3 Deep Learning Approaches

Recent research has increasingly turned to deep learning approaches to address the challenges in app review analysis, offering alternatives to the traditional machine learning methods discussed previously. These approaches range from convolutional neural networks to transformer-based architectures, each presenting unique advantages and considerations.

2.3.1 CNN-Based Architectures for App Review Analysis

Convolutional Neural Networks (CNNs) represent one of the first applications of deep learning to app review analysis, demonstrating significant versatility in handling multilingual classification tasks. Stanik et al. [23] compared traditional machine learning with deep learning approaches for classifying user feedback in both English and Italian into problem reports, inquiries, and irrelevant categories. Their CNN architecture (Figure 2.1) featured an embedding layer initialized with word embedding models such as word2vec or FastText.

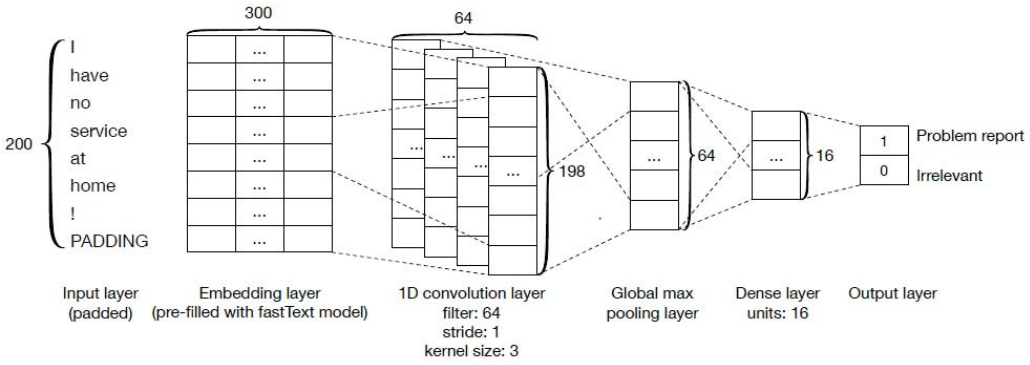


Fig. 2.1: Neural network architecture for the classification [23]

This architecture demonstrated superior performance compared to shallow neural networks, though the researchers noted that traditional machine learning approaches incorporating domain expert knowledge could achieve comparable results due to the powerful features they leveraged. This early implementation of transfer learning and

deep learning techniques established a foundation for subsequent research in app review classification.

Building on CNN approaches, Aslam et al. [24] developed a more comprehensive framework that integrated both textual and non-textual information. Their approach first extracted metadata elements such as total review counts and submission rates, then preprocessed textual information into digital vectors. This multi-faceted approach enabled training a deep learning based multi-class classifier capable of categorizing app reviews into four distinct classes: bug reports, enhancement reports, user experiences, and ratings. This integration of multiple data sources demonstrated the adaptability of CNN architectures for complex app review classification tasks.

2.3.2 Transformer-Based Approaches and Transfer Learning

As the field evolved, researchers began exploring the potential of transformer-based architectures, particularly BERT and its variants, for app review analysis (Figure 2.2). These models offered advanced contextual understanding capabilities beyond what CNN architectures could provide.

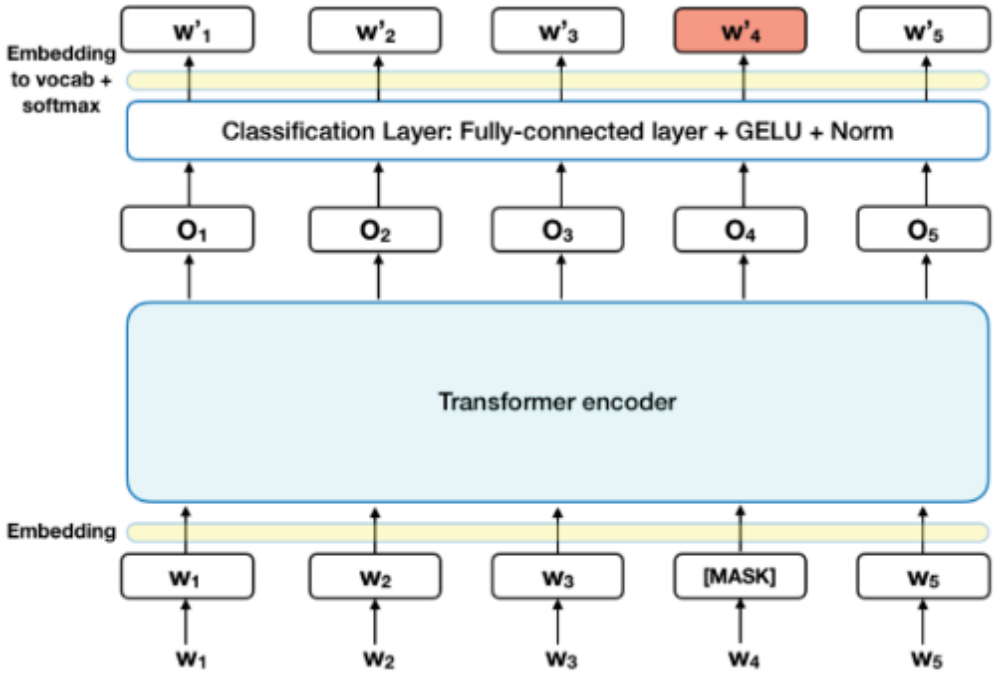


Fig. 2.2: BERT Model Architecture [**?**]

Hadi and Fard [25] conducted a comprehensive empirical study comparing Pre-Trained Models (PTMs) with previous approaches across six datasets from the literature. Their investigation spanned multiple experimental configurations:

- Binary versus multi-class classification scenarios
- Zero-shot classification capabilities
- Multi-task learning settings
- Cross-resource classification effectiveness

Complementing this research, Henao et al. [26] specifically investigated transfer learning potential for user comment classification. Their comparative evaluation of monolingual and multilingual BERT models against state-of-the-art methods revealed that monolingual BERT models consistently outperformed existing baseline approaches in classifying English App Reviews as well as English and Italian Tweets. However, they observed that deploying heavyweight transfer learning models did not automatically guarantee performance improvements, highlighting important considerations about model selection and application.

2.3.3 Comparative Analysis of Deep Learning Approaches

The evolution from CNN-based to transformer-based architectures represents a significant shift in how app review analysis is approached. Each methodology offers distinct advantages and considerations:

TABLE 2.2: Comparison of Deep Learning Approaches for App Review Analysis

Approach	**Advantages**	**Limitations**
CNN-Based Approaches (Stanik et al. [23], Aslam et al. [24])	• Effective integration of word embeddings • Capability to leverage both textual and non-textual data • Efficient handling of multilingual content • Lower computational requirements than transformers	• Limited contextual understanding • Performance comparable to well-optimized traditional ML in some cases • Fixed input size constraints • Less effective for complex semantic tasks

Approach	Advantages	Limitations
Transformer-Based Approaches (Hadi and Fard [25], Henao et al. [26])	• Superior contextual language understanding • Effective pre-training on large datasets • Strong performance in cross-lingual tasks • Better handling of semantic relationships	• Higher computational resource requirements • Not consistently superior to lighter models for all tasks • More complex implementation and fine-tuning • Potential overfitting on smaller datasets

2.3.4 Key Insights and Implications

The research on deep learning approaches for app review analysis reveals several important insights:

1. While CNN architectures provide an effective foundation for classification tasks, particularly when integrating multiple data sources, their performance can sometimes be matched by well-designed traditional machine learning approaches with domain-specific features.

2. Transformer-based models like BERT offer superior contextual understanding and cross-lingual capabilities, though their advantages must be weighed against increased computational requirements and implementation complexity.

3. The monolingual variants of pre-trained models generally outperform multilingual versions for language-specific tasks, suggesting the importance of language-specific tuning even when leveraging transfer learning.

4. Model selection should be guided by specific application requirements rather than assumed superiority of more complex architectures, as demonstrated by cases where heavyweight transfer learning models did not necessarily improve performance.

These findings highlight the need for balanced consideration of model complexity, computational requirements, and task-specific performance when selecting deep learning approaches for app review analysis. The field continues to evolve, with ongoing research exploring optimal architectures and training strategies for various app review analysis tasks.

2.4 Pre-processing user reviews

One of the important step before applying any machine learning and natural language processing to user feedback is pre-processing the user feedback. User feedback often consist of misspelled words, acronyms, abbreviations and other languages. Sometimes user feedback might contain bug reports, feature requests and praise all in the same user feedback. Due to above reasons pre-processing of the user feedback plays a vital role in the process of mining user feedback data for valuable information and it also improve the accuracy [24] of the results. This section discusses about the various technique researchers have used to pre-process the raw user feedback. Figure 5.1 shows an overview of a example pre-processing task.

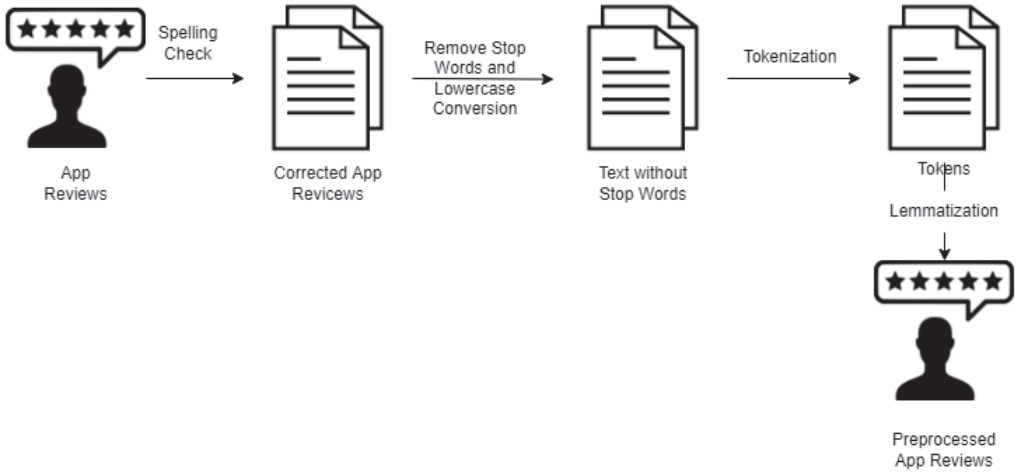


Fig. 2.3: Overview of the Pre-processing

- **Misspelled words, acronyms, and abbreviations:** Most of the time mobile app users use their mobile devices to write reviews. However due to the fact that these devices do not have physical keyboard and being small writing reviews on these devices is difficult and lead to frequent typos (misspelled words). Also mobile users use lots of acronyms and abbreviations while they are typing. To correct these researches have used regular various techniques and common spell expressions Gu and Kim [20] replace common typos they have identified. Phong et al. [15] found out that common spell correcter are ineffective against abbreviation and implemented their own dictionary to deal with spellings and abbreviations.

- **Non-English reviews:** Since mobile applications are globally available when users write their reviews some times they tend to use their language without using English. Since the researches on mining user reviews focusing on reviews

which were written in English these non English reviews add noise [20].So it's important to filter these non-English reviews. Here one prior approach [16] used occurrence of non-ASCII character to filter non-English reviews. But another study done by Gu and Kim [20] pointed out many non-English languages can be written with ASCII, eg. Spanish, French, etc. Hence they developed a custom filter to filter out non-English reviews.

- **Stopwords:** Stopwords are common English words such as *the*, *am* and *their*, which typically carries a very little useful information. Maalej et al. [19] pointed out by removing these words can reduce noise and also allows imformation terms to become more influential. How ever it was also mentioned that some stop words such as *should* and *must* might indicate a feature request and can be important for the review classification. A study [3] suggested that for this task standard list of stopwords provided by Lucene can be used.

- **Lemmatization:** Lemmatization reduces different inflected forms of a word to their basic lemma to be analyzed as a single item .For an example, *fixing*, *fixed*, and *fixes* become *fix*. Lemmatization takes the linguistic context of the term into consideration and uses dictionaries. For an example lemmetization can recognize *good* as the lemma of *better*.

- **Stemming:** Stemming reduces each term to its basic form by removing its post-fix. It is also similar to lemmatization however stemmers just operate on single words therefore it can not distinguish between words which have different meaning depending on part of speech. For example stemmer will reduce *goods* and *good* to the same term. Although lemmatization and stemming reduce terms using different approaches it was noted [3] that both of these techniques can help the classifier to unify keywords which carries same meaning but have different language forms.

Other than these major techniques researchers have applied number of small pre-processing such as lower case conversion [23, 24], POS tagging [3, 15], n-grams [15, 19], removing numeric [27] etc. steps to increase the accuracy of their result. Following table 2.3 form the study by Verma and Patel [27] demonstrate an example user review pre-processing task with it's input text and output text associated with the each step.

2.5 Summery

In above sections we discussed about the previous studies that have been conducted on the particular search area that we are interested in. The table 2.4 summarizes the all pre-processing and processing techniques that are used by the previous studies.

TABLE 2.3: Sample user review pre-processing task

Date Prepossessing Step	Example Input	Output
Remove punctuation especial characters	Hello world, Testing texts for app 5 reviews @ Jhon	Hello world Testing texts for app 5 reviews Jhon
Lowercase Conversion	Hello world Testing texts for app 5 reviews Jhon	hello world testing texts for app 5 reviews jhon
Remove numerals	hello world testing texts for app 5 reviews jhon	hello world testing texts for app reviews jhon
Spelling corrections	hello world testing texts for app reviews jhon	hello world testing texts for None reviews None
Singularization	hello world testing texts for None reviews None	hello world testing text for None review None
Converting all words to base form	hello world testing text for None review None	hello world test text for None review None
Stop-words Removal	hello world test text for None review None	hello world test text review

TABLE 2.4: Summery

Source	Preprocessing	Processing
[2]	Ttokenizing, Lower case Conversion, removing non-words and non-numerical,stopwords	Topic modelling and Sentiment Analysis
Fu et al., 2013	Removing non-english comments, Spliting strings into word using predefined delimiters(. , : ( ) / [ ] ! * ; " ' +), lower case convertion, removal of uncommon words	Analysis of Inconsistant reviews: sentiment analysis and linear regression model, Topic analysis: LDA
Guzman and Maalej, 2014	Noun, verb, and adjective extraction, Stopword removal, Lemmatization.	Sentiment Analysis and Topic modelling with LDA
Chen et al., 2014	converting the raw user reviews into sentencelevel reviews, tokenizing, removal of all non-alphanumeric symbols, lowercase conversion, removal of extra whitespace, stop words and rare words and stemming	Review filtering :EMNB and Topic modelling: LDA and ASUM

Phong et al., 2015	Misspelled words, acronyms, and abbreviations and Non-english reviews removal,Word stemming and PoS tagging	Ranking: sentimant analysis,Clustering:K-means, Search and Trend Analysis: VSM(Vector Space Model)
Gu and Kim, 2015	Seperating sentences,fixing common typos and contractions	Classification: Max Entropy, Text feature Extraction:TrunkWords, Character N-Gram, POS tag and Parsing tree
Guzman et al., 2015	-	Naive Bayes, Support Vector Machines (SVMs), Logistic Regression Neural Networks and Ensembles of them.
Maalej et al., 2016	stop-word removal, stemming, lemmatization, tense detection, and bi-grams,	Classification: Naive Bayes, Decision Tree and Maximum Entropy
Anchiêta and Moura, 2017	Removal of emoji and emoticons, reviews with five stars, and reviews with less than three words,Tokenizing, removing of stop words, and stemming	Clustring with K-means with BoW model and TF-IDF then Topic modelling with LDA and NMF
Guzman et al., 2017	Ttokenizing,Lower case Conversion,extracting n-grams, removing stopwords and stemming	MNB,BTM
Dhinakaran et al., 2018	Removal of stop words and lemmatizing	Naive Bayes, Logistic Regression, and Active Learning.
Stanik et al., 2019	Traditional machine learning :lowercase conversion, masking account names, links,hashtags and lemmatization. Deep Learning: -	Traditional machine learning feature extraction: POS tagging, TF-IDF, sentiment, fastText,Traditional machine classification:Decision Tree, Random Forest, Naive Bayes, and Support Vector Machine, Deep Learning: CNN, Transfer learning , Hyper tuning
Aslam et al., 2020	spell checking, removal of special charactors , stop words,lowercase conversion tokenizing and lemmatization.	Feature extraction:Sentiment analysis, Classification :CNN

Hadi and Fard, 2021	-	PTMs (BERT, XLNet, RoBERTa and ALBERT)
Henao et al., 2021	Tokenizing using BERT tokenizer and adding paddings to tokens	Classification: Transfer Learning and PTMs(BERT and M-BERT)

2.6 LLM-based Annotation

The advent of Large Language Models (LLMs) has introduced new possibilities for automated data annotation, potentially reducing the resource-intensive nature of manual labeling while maintaining annotation quality. This section examines the evolution and comparative effectiveness of LLM-based annotation approaches in addressing these challenges.

2.6.1 Evolution of LLM Annotation Strategies

Initial explorations of LLMs for data annotation focused on basic pseudo-labeling approaches. Wang et al. [28] conducted pioneering research examining GPT-3's capabilities as an annotation tool, demonstrating that augmenting manually labeled data with LLM pseudo-labels could enhance model performance under constrained labeling budgets, though quality still lagged behind human annotations. Building on this foundation, He et al. [29] significantly advanced the field with their two-step "explain-then-annotate" approach, where GPT-3.5 was first prompted to generate explanations for examples, then used these to construct chain-of-thought prompts for unlabeled data annotation. This method demonstrated superior performance compared to both zero-shot and few-shot LLM annotation approaches, establishing the importance of structured reasoning in improving annotation quality.

2.6.2 Advanced Integration Frameworks

As the field matured, researchers developed more sophisticated frameworks integrating LLMs with complementary techniques. Zhang et al. [30] introduced LLMAAA, an innovative framework employing LLMs as active annotators by combining active learning strategies with advanced prompt engineering. Their experiments on named entity recognition and relation extraction demonstrated that models trained on LLM-generated labels could outperform teacher LLMs within hundreds of annotated samples. Similarly, Zhou et al. [31] proposed a specialized framework for e-commerce attribute extraction that combined BERT classification, Conditional Random Fields for value extraction, and LLMs for data annotation, significantly improving attribute recognition from customer queries. These integrated approaches collectively demonstrated the potential of combining LLMs with established machine learning techniques

to enhance annotation effectiveness across various domains.

2.6.3 Human-LLM Collaborative Approaches

Recent research has increasingly focused on optimizing the collaboration between human expertise and LLM capabilities. He et al. [32] examined the integration of LLMs into crowdsourced data annotation pipelines, highlighting both the potential for LLMs to augment human annotation efforts and the importance of maintaining human involvement in the process. Complementing this work, Tang et al. [33] developed PDFChatAnnotator, a collaborative tool combining LLM capabilities with human guidance to improve the quality and efficiency of multi-modal data collection from PDF-format catalogs. These studies demonstrate the emerging consensus that optimal annotation outcomes often result from effectively combining human and LLM strengths rather than relying solely on either approach.

2.6.4 Domain-Specific Applications and Evaluation

LLM-based annotation has been evaluated across diverse specialized domains, revealing both capabilities and limitations. Yu et al. [34] assessed the potential of LLM-assisted annotation for corpus-based pragmatics and discourse analysis, specifically focusing on the annotation of apologies based on a local grammar framework. Their comparison of GPT-3.5 and GPT-4 with human annotators provided insights into model capabilities for complex linguistic tasks. Similarly, Imamovic et al. [35] explored using ChatGPT for annotating complex linguistic phenomena within the Appraisal Theory framework, finding high precision in detecting evaluative meaning but low recall, emphasizing the need for human oversight in specialized annotation tasks.

2.6.5 Limitations and Challenges

Despite promising results, researchers have identified significant limitations in LLM-based annotation approaches. Wang et al. [36] conducted a comprehensive analysis of using LLMs to replace human participants in computational social science research, identifying critical limitations including misportrayal of marginalized groups and flattening of group diversity. These findings highlight the importance of careful consideration when using LLMs for tasks requiring nuanced representation of demographic identities. Pangakis et al. [37] further emphasized the need for validation when using generative AI for automated annotation, proposing workflows that harness LLM potential while ensuring accuracy through human oversight.

2.6.6 Comprehensive Perspectives and Future Directions

Synthesizing these diverse research directions, Tan et al. [38] provided a comprehensive survey on using large language models for data annotation across various domains and tasks. Their work highlights the significant potential of methods like chain-of-thought prompting, explanation generation, active learning, and specialized frameworks for improving annotation quality and efficiency. However, they also emphasize the need for further research to address issues like consistency, representation of diverse perspectives, and identification of optimal approaches for different annotation tasks across various LLM architectures.

2.6.7 Comparative Analysis

Analysis of the different LLM-based annotation approaches reveals distinct trade-offs:

TABLE 2.5: Key Challenges and Solutions in LLM-based Annotation

Approach	**Advantages**	**Limitations**
Basic LLM Annotation (Wang et al. [28])	• Simple implementation • Fast processing • Effective for augmenting human labels	• Limited accuracy compared to humans • Poor handling of edge cases • Lack of reasoning transparency
Explain-then-Annotate (He et al. [29])	• Enhanced reasoning capabilities • Higher accuracy than basic approaches • Transparent decision-making process	• Higher computational cost • Complex prompt engineering required • Increased API usage and costs

Approach	Advantages	Limitations
Active Learning Integration (Zhang et al. [30])	• Efficient sample selection • Rapid improvement with fewer examples • Adaptive annotation strategies	• Complex implementation requirements • Iterative process management • Higher technical overhead
Human-LLM Hybrid (He et al. [32], Tang et al. [33])	• Optimal accuracy • Robust quality assurance • Flexible workflow adaptation	• Higher resource requirements • Complex coordination needed • Training and integration challenges
Domain-Specific Frameworks (Zhou et al. [31], Yu et al. [34], Imamovic et al. [35])	• Tailored to specific task requirements • Enhanced performance in specialized domains • Integration with domain knowledge	• Limited generalizability • Domain expertise requirements • Task-specific customization needed

These studies collectively demonstrate the significant potential of LLMs for data annotation tasks while highlighting important considerations for their effective application. While challenges remain in areas such as demographic representation, consistency, and specialized domain annotation, the rapid evolution of LLM-based annotation approaches suggests a promising future for reducing annotation costs while maintaining data quality.

CHAPTER 3

PUBLICATIONS OF THE RESEARCH

3.1 Aspect-based Sentiment Analysis on Mobile Application Reviews

This paper was published in the 2022 22nd International Conference on Advances in ICT for Emerging Regions (ICTer), where we introduced a novel CNN-based approach for analyzing mobile app reviews using Aspect-Based Sentiment Analysis [39].

3.2 Automatic Analysis of App Reviews Using LLMs

This is our next publication, which we expect to publish in The 31st International Conference on Computational Linguistics (COLING 2025). In this paper, we examine the potential of using both commercial closed-source LLMs and open-source LLMs for the task of mobile app review classification. We investigate the possibility of using a commercial LLM as an annotator to autonomously create a balanced dataset of 10,000 app reviews and use it to instruct and fine-tune open-source models to create cost-effective custom models.

CHAPTER 4

ASPECT BASED SENTIMENT ANALYSIS ON MOBILE APPLICATION REVIEWS

4.1 Introduction

Mobile applications have become an integral part of our daily lives, with millions of apps available across various platforms. As the app market continues to grow, understanding user feedback has become crucial for developers to improve their products and stay competitive. Aspect-based Sentiment Analysis (ABSA) offers a powerful way to extract nuanced insights from user reviews, allowing developers to identify specific aspects of their apps that users appreciate or criticize.

This chapter presents a novel approach to ABSA for mobile application reviews using Convolutional Neural Networks (CNNs). Our method aims to address two key tasks:

1. Aspect category classification: Identifying the specific features or aspects of the app that users are commenting on.

2. Aspect sentiment classification: Determining the sentiment (positive, negative, or neutral) associated with each identified aspect.

By leveraging CNNs, we seek to improve upon existing baseline methods and provide more accurate and detailed analysis of user feedback. This approach can help app developers prioritize improvements, address user concerns more effectively, and ultimately enhance user satisfaction.

In the following sections, we detail our methodology, including data preprocessing, model architecture, and training procedures. We then present our experimental results, comparing our approach to existing baselines and analyzing its performance across different app categories. Finally, we discuss the implications of our findings and potential avenues for future research in this domain.

4.2 Methodology

The methodology is mainly divided into three major parts; oversampling, pre-processing and embedding, feature extraction, and classification. Different CNN models were implemented during the study and finally selected the CNN model that gave the best performance in both classifying aspect category and aspect sentiment. An overview of the proposed approach is illustrated in Figure 4.1.

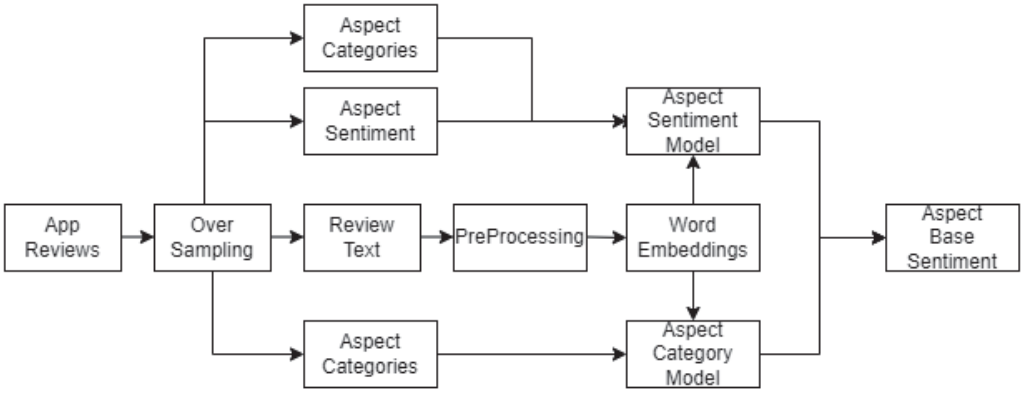


Fig. 4.1: Overview of the Proposed Approach.

4.2.1 OverSampling the Data

Alturaief et al. [11] used Synthetic Minority Oversampling Technique (SMOTE) to handle class imbalance during their study. However, we find that using SMOTE to oversample textual data is somewhat problematic, because SMOTE works in the feature space, therefore it doesn't output synthetic data which has a real-world textual representation. This study employed text augmentation techniques to oversample the training dataset and it was expected that text augmentation would generate multiple representations of user review which in turn improve the accuracy of our CNN models. During the study, the training dataset was oversampled using two text augmenting techniques; Contextual augmentation by Google Bert and Data Augmentation by Round-trip translation (RTT).

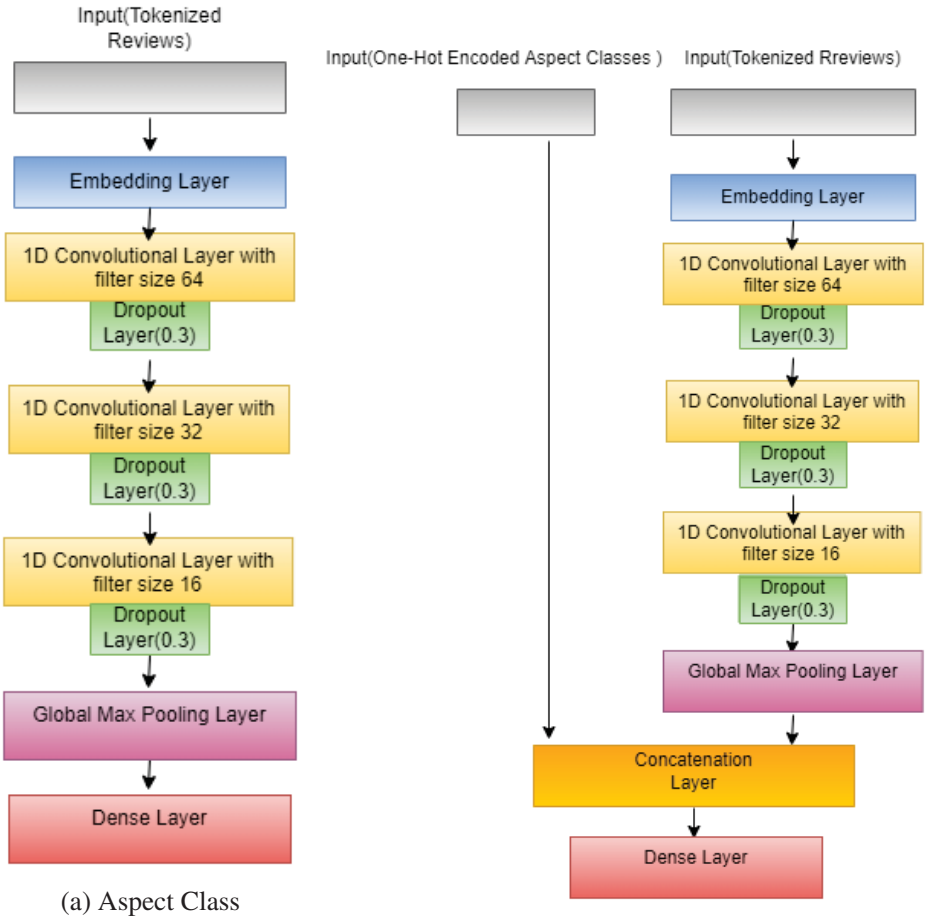
4.2.2 Pre-Processing and Embedding

Several pre-processing steps were applied to review text during the study. First, lowercase conversion was applied to each review. Then removed all the numerals, extra spaces, and punctuation marks. After that, all the stop-words were removed by using a list of English stop-words provided by NLTK Data[1]. Finally, reviews were tokenized into words using NLTK Tokenizer package[2] and lemmatized using NLTK WordNetLemmatizer[3]. After the pre-processing, the reviews were converted to vector format using several popular pre-trained word embedding models.

[1]NLTK Data https://www.nltk.org/nltk_data/
[2]NLTK Tokenizer https://www.nltk.org/api/nltk.tokenize.html
[3]NLTK WordNet Lemmatizer http://bit.ly/3Eoefcn

4.2.3 Feature extraction and classification

As illustrated in Fig 4.2 proposed approach is composed of two distinct CNN models. One for the aspect classification (Fig 4.2a) and another one for the aspect sentiment classification (Fig 4.2b). Convolutional neural networks(CNN) are special kinds of feed-forward artificial neural networks that are used for supervised learning to analyze data. In CNN, nodes are connected in a non-cyclical manner. It uses a multilayer perceptron variation which is designed to require as little pre-processing as possible. Though CNNs are generally used for computer vision recent applications of CNNs to NLP tasks, have shown promising results. This study uses CNN, which is loosely based on the CNN model introduced by Aslam et al. [24].



(a) Aspect Class Classification Model

(b) Aspect Sentiment Classification Model

Fig. 4.2: CNN Model Architectures

4.3 Experiments

4.3.1 Dataset

The study exploits the AWARE dataset introduced by Alturaief et al. [11] as the benchmark dataset. AWARE is an ABSA dataset of 11323 smartphone app reviews extracted from apple store[4] and it is annotated with aspect terms, aspect categories, and sentiments. It provides annotated reviews for three distinct domains: (i) Games, (ii) Productivity, and (iii) Social Networking.

4.3.2 Evaluation Criteria

To measure and compare the performance of the proposed approach against the baselines provided by Alturaief et al. [11], similar evaluation measures were used. Therefore evaluation criteria for the aspect class classification model (multi-class) was micro F1 score (Equation 4.1), which is the harmonic mean of micro precision (Equation 4.2) and micro recall (Equation 4.3).

$$F1_\mu = H(P_\mu, R_\mu) = \frac{2 \times P_\mu \times R_\mu}{P_\mu + R_\mu} \tag{4.1}$$

$$P_\mu = \frac{\sum_{i \in C} TP_i}{\sum_{i \in C} TP_i + \sum_{i \in C} FP_i} \tag{4.2}$$

$$R_\mu = \frac{\sum_{i \in C} TP_i}{\sum_{i \in C} TP_i + \sum_{i \in C} FN_i} \tag{4.3}$$

In all equations, C is the set of classes and TP_i, TN_i, FP_i, FN_i are defined as the *True Positive*, *True Negative*, *False Positive*, *False Negative* counts for class i.

To evaluate the performance of the aspect sentiment model, which is a binary classification, the Accuracy (Equation 4.4) measure was used.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

4.3.3 Data Over Sampling

Mainly two data augmenting techniques were used during the study. The details of the two methods are as follows,

- Contextual augmentation by Google BERT [40] : Contextual word embeddings assign each word a representation based on its context. We used substitute actions for augmenting data. In substitute, the length of the sentence remains the

[4]Apple store https://www.apple.com/app-store/.

same but some words are replaced. We utilized the NLPAug open-source python package[5] for data augmentation.

- Data Augmentation by RTT : Round-trip translation (RTT) is additionally referred to as re-cursive, back-and-forth, and bi-directional translation. It is the method of translating a word, phrase, or text into another language (forward translation), then translating the results back to the first language (back translation). RTT is used as an augmentation technique to extend the training data. To augment the data roundtrip translation python package[6] was used. Four different languages ("German(DE)", "Turkish(TR)", "Japanese(JP)", "Chines(CN)") were used as forward translation languages to find which yields the best results.

4.3.4 Embeddings

The textual format data should be converted to vector format which could be understandable by the machine to be used with the deep learning models. Several context-free embedding models were tested to find out which optimize the results. Following pre-trained models were used during the study: (i) fastText, wiki-news model, with 1 million word vectors and 300 dimensions, trained on Wikipedia 2017, UMBC web-based corpus and statmt.org news dataset, (ii) glove pre-trained model, trained on Wikipedia data with 6 billion tokens, 100 dimensions and a 400,000-word vocabulary, and (iii) pre-trained Google word2vec model, trained on Google news data (about 100 billion words); it contains 3 million words and phrases and was fit using 300-dimensional word vectors.

4.3.5 Feature extraction and classification

The proposed approach consists of two distinct CNN models, the aspect category classification model(Fig 4.2a) and the aspect sentiment classification model(Fig 4.2b). The aspect classification model uses three one-dimensional(1D) convolutional layers with filter sizes 64, 32, and 16 respectively followed by a dropout layer with 0.3 to extract textual features. After that, the output is fed into a global max pooling layer to further extract features. Then a dense layer with softmax as the activation function is used to predict aspect category classes.

The aspect sentiment classification model also follows a similar approach to extract textual features. Since it uses the aspect category as an input to the model, the output of the global max pooling layer is merged with the one-hot encoding of the aspect category. Finally, a dense layer with softmax as the activation function is used to predict the sentiment of the review.

[5]NLPAug python package https://github.com/makcedward/nlpaug
[6]RTT python package https://github.com/samhavens/roundtrip

4.4 Results and Analysis

Dataset	Word Embedding	Preprocessing	BERT	RTT(DE)	RTT(CN)	RTT(TR)	RTT(JP)
Productivity	Fasttext	Disabled	0.60	0.59	0.25	0.61	0.60
		Enabled	0.63	0.61	0.23	0.62	0.59
	Word2Vec	Disabled	0.61	0.62	0.24	0.61	0.61
		Enabled	0.62	0.62	0.26	0.61	0.60
	Glove	Disabled	0.54	0.53	0.24	0.52	0.55
		Enabled	0.56	0.57	0.25	0.58	0.58
Gaming	Fasttext	Disabled	0.42	0.45	0.19	0.35	0.43
		Enabled	0.40	0.39	0.22	0.28	0.45
	Word2Vec	Disabled	0.42	0.41	0.23	0.37	0.44
		Enabled	0.39	0.42	0.21	0.37	0.44
	Glove	Disabled	0.42	0.44	0.20	0.34	0.42
		Enabled	0.30	0.30	0.21	0.24	0.31
Social	Fasttext	Disabled	0.62	0.62	0.58	0.25	0.60
		Enabled	0.60	0.61	0.58	0.27	0.60
	Word2Vec	Disabled	0.60	0.62	0.61	0.29	0.61
		Enabled	0.58	0.62	0.61	0.28	0.61
	Glove	Disabled	0.54	0.56	0.54	0.27	0.55
		Enabled	0.54	0.55	0.55	0.26	0.57
Average	Fasttext	Disabled	0.55	0.56	0.34	0.41	0.55
		Enabled	0.55	0.54	0.35	0.39	0.55
	Word2Vec	Disabled	0.55	0.55	0.36	0.43	0.56
		Enabled	0.53	0.56	0.36	0.42	0.55
	Glove	Disabled	0.50	0.51	0.33	0.38	0.51
		Enabled	0.47	0.48	0.34	0.36	0.49

TABLE 4.1: Aspect Class Classification

4.4.1 Pre-processing

According to results shown in Table 4.1 and Table 4.2, it can be noticed that the pre-processing of reviews doesn't always improve the results and it varies depending on the mobile app domain and the ABSA task.

4.4.2 Optimization

Several pre-trained embedding models and two different text augmenting techniques were experimented with in order to optimize the results of the proposed CNN models. It can be seen that the performance varies with the domain for the same model and the set of pre-processing, word embedding, and the data over-sampling technique used. Hence we considered average performance to select the optimum combination of pre-processing, word embedding, and the data over-sampling technique that would perform well in all three domains. Finally CNN+Word2Vec+RTT(DE)+Preprocessing was selected for the aspect category classification. It could achieve f1 scores of 0.62, 0.42, and 0.62. And for the aspect sentiment classification task, CNN+Word2Vec+RTT(DE) was selected and it could achieve accuracy of 0.80, 0.7, and .086 respectively in the

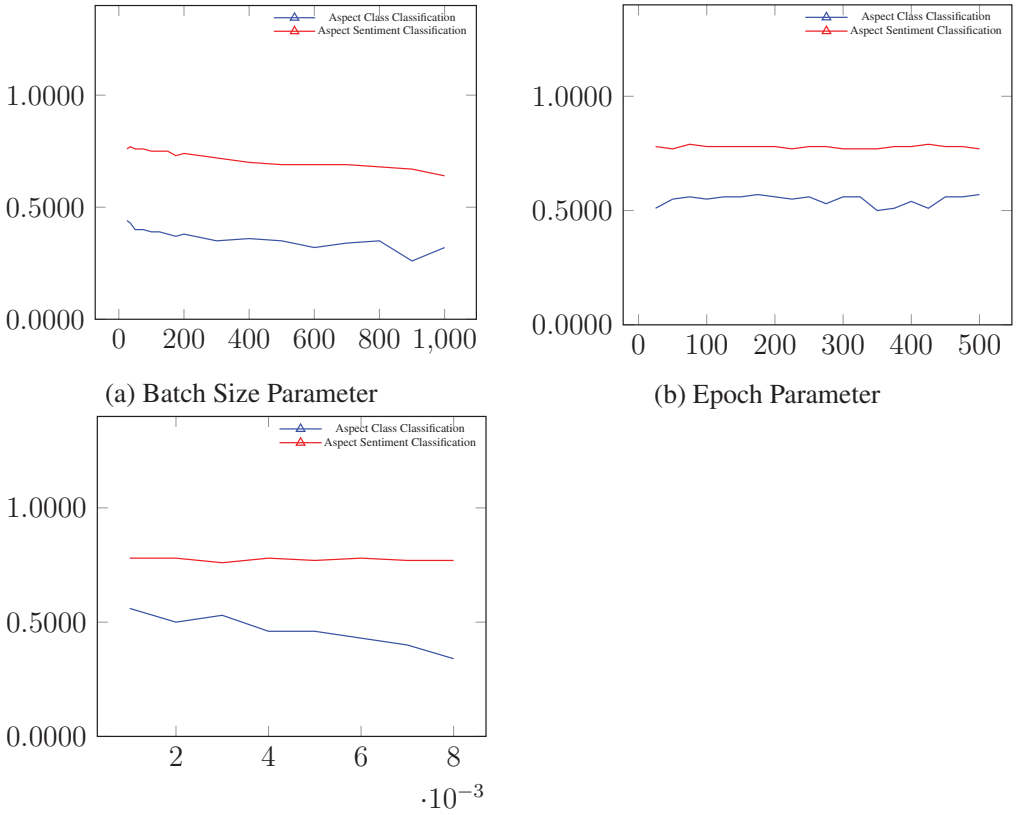
Dataset	Word Embedding	Preprocessing	BERT	RTT(DE)	RTT(CN)	RTT(TR)	RTT(JP)
Productivity	Fasttext	Disabled	0.81	0.81	0.62	0.80	0.78
		Enabled	0.79	0.79	0.63	0.81	0.81
	Word2Vec	Disabled	0.80	0.80	0.62	0.82	0.80
		Enabled	0.79	0.79	0.64	0.82	0.81
	Glove	Disabled	0.80	0.79	0.61	0.79	0.81
		Enabled	0.79	0.80	0.62	0.80	0.77
Gaming	Fasttext	Disabled	0.70	0.71	0.68	0.65	0.70
		Enabled	0.71	0.70	0.68	0.65	0.71
	Word2Vec	Disabled	0.70	0.70	0.67	0.64	0.70
		Enabled	0.72	0.69	0.68	0.66	0.70
	Glove	Disabled	0.71	0.72	0.70	0.65	0.70
		Enabled	0.70	0.69	0.69	0.65	0.70
Social	Fasttext	Disabled	0.83	0.80	0.81	0.63	0.83
		Enabled	0.82	0.83	0.81	0.62	0.82
	Word2Vec	Disabled	0.81	0.86	0.81	0.64	0.80
		Enabled	0.82	0.86	0.82	0.64	0.83
	Glove	Disabled	0.82	0.84	0.82	0.64	0.81
		Enabled	0.79	0.85	0.82	0.63	0.81
Average	Fasttext	Disabled	0.78	0.78	0.71	0.70	0.77
		Enabled	0.78	0.78	0.71	0.70	0.78
	Word2Vec	Disabled	0.77	0.79	0.70	0.70	0.77
		Enabled	0.78	0.78	0.72	0.71	0.78
	Glove	Disabled	0.78	0.79	0.71	0.70	0.78
		Enabled	0.76	0.78	0.71	0.70	0.76

TABLE 4.2: Aspect Sentiment Classification

Productivity, Game, and Social Networking domains. After several experiments, we found the optimum values for the models' parameters as, Learning Rate 0.001, Epoch 75, Batch Size 25, Dense Activation as Sigmoid, and CNN Activation as ReLU for the aspect category classification model and Learning Rate 0.001, Epoch 75, Batch Size 35, Dense Activation as Sigmoid and CNN Activation as ReLU for the aspect sentiment classification model. The relationship between result and batch size, epoch, and learning rate are shown in Fig 4.3a, Fig 4.3b, and Fig 4.3c, respectively.

4.4.3 Final Results and Error Analysis

The final results of both aspect category classification and aspect sentiment classification tasks are shown in Table 4.1 and Table 4.2 respectively. The results show that our approach could achieve 87.88%, 93.75%, and 31.25% improvements in aspect category classification and 16.43%, 23.35%, and 3.72% improvements in aspect sentiment classification over the baselines results provided by Alturaief et al. [11] in productivity, social networking, and game domains respectively. Fig 4.4 shows the generated confusion matrices for the aspect category classification. By analyzing Fig 4.4c we identified that our approach misclassified most of the game domain aspect categories with the enjoyability category, resulting in poorer performance in the game domain compared to other domains.



(a) Batch Size Parameter

(b) Epoch Parameter

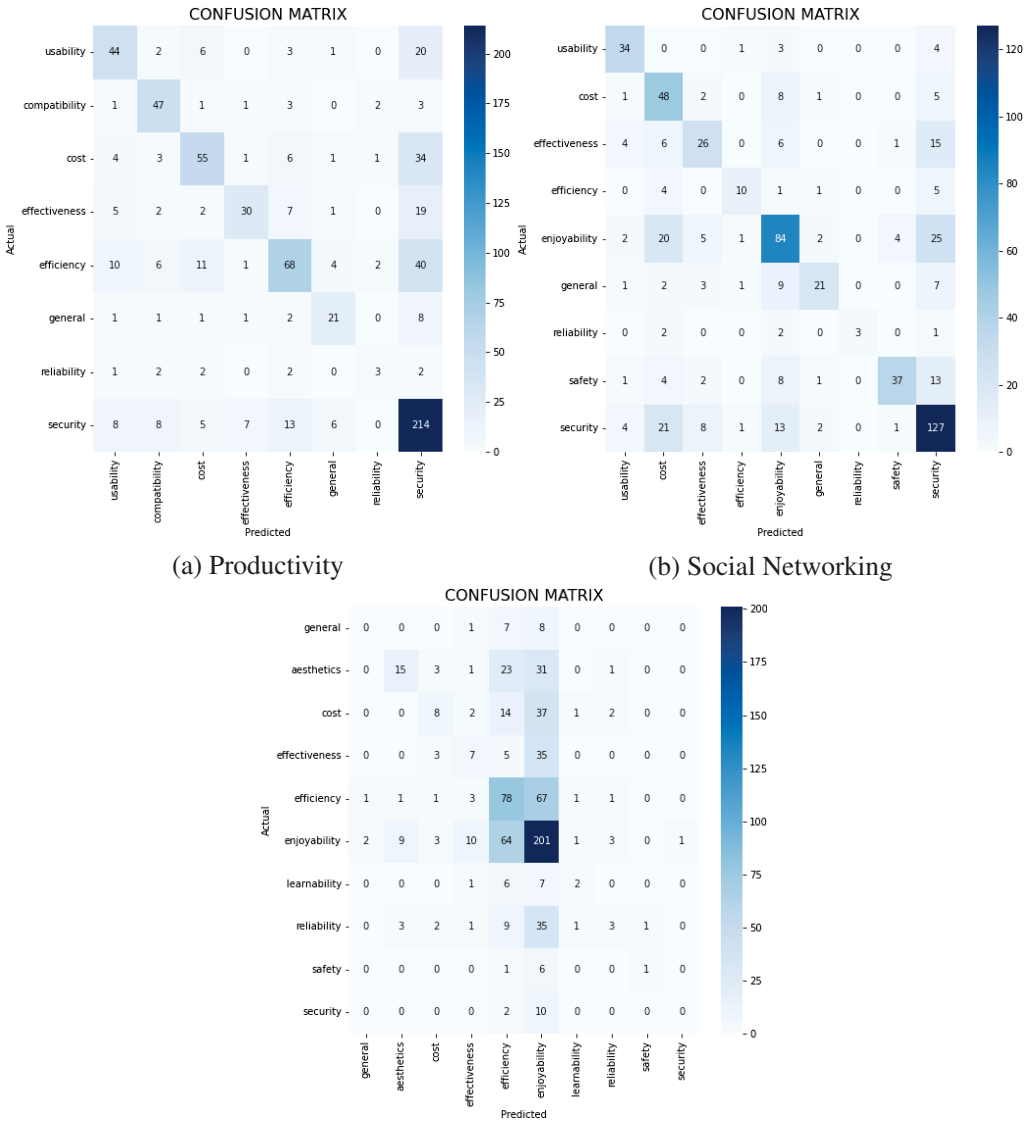
(c) Learning Rate Parameter

Fig. 4.3: Hyper Parameter Tuning

4.5 Conclusion

The proposed CNN models together with pre-trained word embedding models, data pre-processing, and data augmenting techniques were able to comfortably beat the baselines provided by Alturaief et al. [11]. Results show that our approach could archive f1 scores of 0.62, 0.42, and 0.62 in the aspect category classification task, and accuracy of 0.80, 0.7, and .086 for the aspect sentiment classification task in Productivity, Game, and Social Networking domains respectively. As a future work we intend to investigate the possibility of using popular transformer-based models to improve the results further.In the spirit of reproducibility and open science, we have made our datasets and code implementations publicly available[7]

[7] https://github.com/sadeep25/ABSA-Models.git

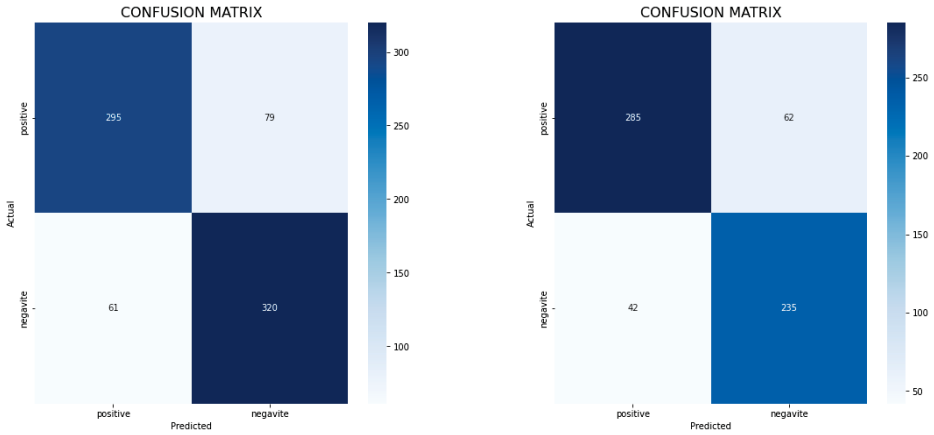


(a) Productivity

(b) Social Networking

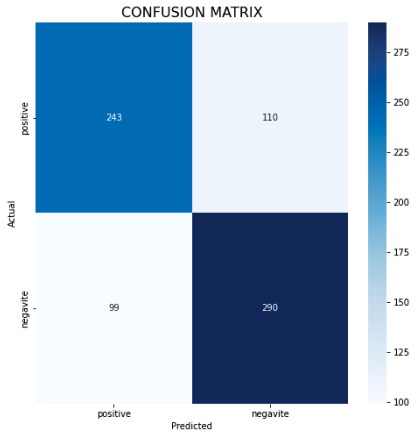
(c) Game

Fig. 4.4: Aspect Category Classification Confusion Matrices



(a) Productivity

(b) Social Networking



(c) Game

Fig. 4.5: Aspect Sentiment Classification Confusion Matrices

CHAPTER 5

AUTOMATIC ANALYSIS OF APP REVIEWS USING LLMS

5.1 Introduction

The mobile application ecosystem continues to evolve rapidly, with millions of apps competing for user attention across various platforms. In this highly competitive landscape, understanding and responding to user feedback has become a critical factor for app success and continuous improvement. While traditional methods of analyzing app reviews have proven valuable, the advent of Large Language Models (LLMs) presents new opportunities for more sophisticated and nuanced analysis of user feedback.

This chapter explores the potential of using state-of-the-art LLMs for automatic analysis of app reviews. Our research aims to leverage the advanced natural language understanding capabilities of these models to improve the accuracy, efficiency, and depth of app review analysis. Specifically, we focus on the following key objectives:

1. Evaluating the performance of commercial closed-source LLMs (such as `GPT-3.5` and `Gemini Pro`) for app review classification in zero-shot settings.
2. Investigating the use of LLMs as autonomous annotators to create large-scale, high-quality datasets for app review analysis.
3. Exploring the potential of fine-tuning open-source LLMs (like `LLAMA 2` and `Mistral`) for app review classification tasks.
4. Analyzing the impact of various parameters (such as `Temperature`, `Top_p`, training data size, and number of epochs) on model performance.

In the following sections, we detail our methodology, including model selection, dataset creation, fine-tuning procedures, and evaluation metrics. We then present our experimental results, offering insights into the relative strengths and limitations of different LLM approaches for app review analysis. Finally, we discuss the implications of our findings for app developers and researchers, and suggest directions for future work in this rapidly evolving field.

5.2 Methodology

Our approach to classifying app store reviews using Large Language Models (LLMs) consists of several key steps, as illustrated in Figure 5.1. We begin by creating two essential datasets: a benchmarking dataset for evaluation and a larger dataset for fine-tuning custom LLMs. The methodology involves selecting and comparing various

LLMs, including both commercial and open-source models. We developed carefully crafted prompts for annotation and fine-tuning, as well as utilized optimization techniques to train custom models on consumer-grade hardware. Throughout the process, we employed a rigorous evaluation strategy to assess the performance of different models and approaches in the task of app review classification.

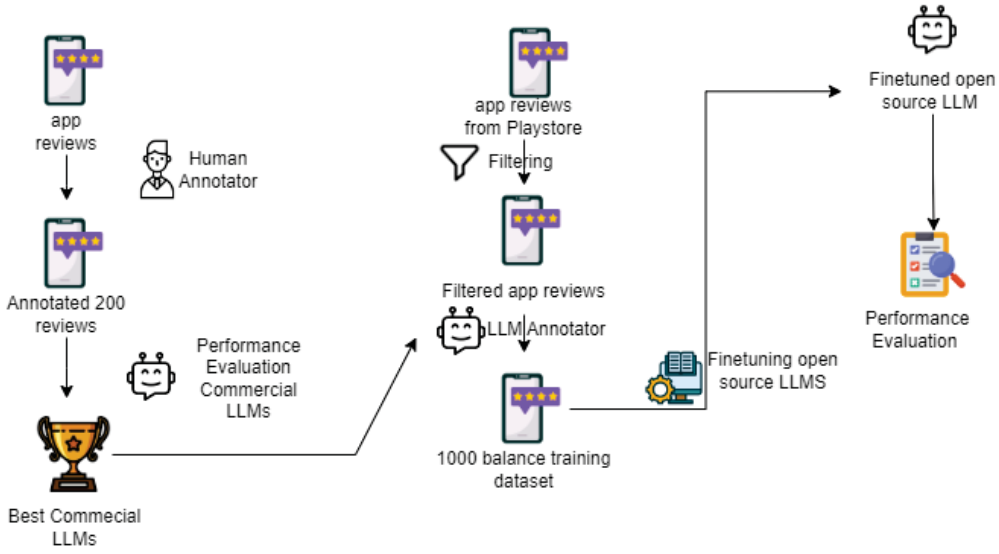


Fig. 5.1: A high-level overview of our approach

5.2.1 LLMs Selection

The selection of appropriate Large Language Models (LLMs) was a crucial step in our research. We carefully evaluated and chose models based on a combination of performance metrics, resource constraints, and task-specific requirements. Our selection process aimed to find the balance between model capabilities and practical implementation considerations, for both our automated annotation approach and custom model development. Section 5.2.1.1 and Section 5.2.1.2 respectively discuss how we selected the appropriate LLMs for our experiments in detail.

5.2.1.1 LLM Selection for Automated Annotation

We selected OpenAI's `GPT-3.5` (gpt-3.5-turbo-0125) [1] and Google's `Gemini Pro 1.0` [2] for automated annotation. These models offer a balance between cost-effectiveness and performance relative to their more advanced counterparts (GPT-4 and Gemini Pro

[1] https://platform.openai.com/docs/models/gpt-3-5-turbo
[2] https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/gemini-pro

1.5). We employed a zero-shot setting to minimize context size and costs, utilizing the annotation prompt explained in the section 5.2.4.1 via respective API endpoints

5.2.1.2 LLM Selection for Custom Models

With the introduction of open-source LLMs, including `LLaMA 2` [41, 42], `Vicuna` [43], `Falcon` [44], `Mistral` [45], and `Zephyr` [46], has enabled fine-tuning LLMs on consumer-grade hardware. Given our hardware constraints (single RTX 4090 GPU with 24GB VRAM) and the requirement for instruction-following capabilities to enforce the required response JSON formatting, we selected `Llama-2-7b-chat-hf` [3], `Mistral-7B-Instruct` [4], and `Falcon 7B Instruct` [5] for our experiments.

We evaluated these models' classification performance against a our manually annotated data, comparing base and fine-tuned versions under various settings. These included the number of annotated reviews, training epochs, and two types of instruction prompts: a zero-shot template with only class definitions and a template incorporating "explain then annotate" technique [29, 47]. We also examined the effects of `Temperature` and `Top_p` parameters on model performance.

5.2.2 Benchmarking Dataset

This study utilizes a subset of a dataset from Maalej and Nabil [48]. Due to discrepancies in the retrieved archived version, we selected a subset for manual review and rectification. We maintained four classes from the original work, slightly modifying definitions for LLM readability:

- **Bug Reports:** Bug reports are user comments that identify issues with the app, such as crashes, incorrect behaviour, or performance problems. These reviews specifically highlight problems which affect the app's functionality and suggest a need for corrective action.
- **Feature Requests:** Feature requests are user suggestions for new features or enhancements in future app updates. These can include requests for features seen in other apps, additions to content, or ideas to modify existing features to enhance user interaction and satisfaction.
- **User Experience:** User experience reviews provide detailed narratives focusing on specific app features and their effectiveness in real scenarios. They offer insights into the app's usability, functionality, and overall satisfaction, often serving as informal documentation of user needs and app performance.

[3] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[4] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1
[5] https://huggingface.co/tiiuae/falcon-7b-instruct

- **Ratings:** Ratings are brief textual comments that reflect the app's numeric star rating, primarily indicating overall user satisfaction or dissatisfaction. These reviews are succinct, focusing on expressing a general sentiment without detailed justification.

We created a balanced dataset of 200 reviews, with 50 per category. For multi-label cases, we applied the precedence: Bug > Feature > User Experience > Rating. ChatGPT-4 assisted in the manual annotation process [49, 50], providing suggestions that the annotator could accept, reject, or modify. This process ensures a reliable, balanced dataset for benchmarking LLM performance in app review classification.

5.2.3 Dataset for Fine-Tuning Custom LLMs

To construct a comprehensive dataset for fine-tuning custom Large Language Models (LLMs), we systematically collected 92,354 reviews from the Google App Store. The selection process targeted popular applications in the United States, as ranked by *appfigures.com*[6] at the time the study was conducted. Our corpus included reviews from over 90 distinct mobile applications. We chose the United States as the target demographic since our study is only focused on English app reviews and the United States is the largest English-speaking market.

For the extraction and filtering process, we utilized the `langdetect` Python library[7] to identify and filter out non-English reviews. Our initial selection criteria prioritized reviews exceeding 10 words in length, which yielded 85,852 reviews. To ensure comprehensive representation, we further extended the corpus with an additional 6,502 reviews, each containing between 2 and 10 words. This was done due to our observation that reviews primarily consisting of simple ratings rarely exceeded 10 words.

To annotate this extensive corpus of 92,354 reviews, we utilized the best-performing commercial LLM model (`GPT 3.5`) according to experiment results from section 5.3.1. The model was configured with a `Temperature` setting of 1 and a `Top_p` value of 0.25 based on our experimental observations and guidelines provided by `Open AI`[8] to achieve a balance between creativity and coherence in the output. We annotated each of the reviews using the annotation prompt described in the section 5.2.4.1 until we arrived at a balanced dataset comprising 10,000 reviews, with an equal distribution of 2,500 reviews across each of the four label categories.

[6] https://appfigures.com/top-apps/ios-app-store/united-states/iphone/top-overall

[7] https://pypi.org/project/langdetect/

[8] https://platform.openai.com/docs/api-reference/chat/create,https://platform.openai.com/docs/guides/text-generation

5.2.4 Selection of Prompts

Prompt quality significantly influences model performance [51].Drawing inspiration from existing work [29, 47, 52–54], we developed three different prompt templates for our experiments. Section 5.2.4.1 describes the prompt we used for benchmark and annotate app reviews and Section 5.2.4.2 describes the prompt templates that we used for instruct fine-tune and benchmark our fine-tuned models.

5.2.4.1 Prompts for Annotation

For benchmark classification performance and annotation of app reviews using commercial LLMs, we developed a comprehensive prompt structure that encourages thorough consideration of all classes. Our approach involves a series of boolean questions and explanations for each class. This prompt was developed to address scenarios where reviews belonging to multiple labels might otherwise be classified into a single category when the model attempts to classify them.

The template, as shown in Figure 5.2, was designed for benchmarking and annotating user reviews using commercial LLMs in a Zero-Shot setting. It classifies reviews based on content and star ratings, with four distinct categories and clear definitions. To address initial confusion between certain categories, we added a "Differentiating Tip" for "User Experience" and "Ratings".

Our prompt structure includes the following key elements:

- Task Description: A clear explanation of the review classification task.
- Definitions for Classification: Detailed descriptions of each category (Bug Reports, Feature Requests, User Experience, and Ratings).
- Structured Questions: Eight questions guiding the classification process, including boolean queries and requests for explanations.
- Instructions: Specific guidelines for the language model on how to process reviews and format the output.
- Output Format: A defined JSON structure for the classification results.

This structure allows for flexible reasoning, particularly for multi-category reviews. The eight structured questions guide the classification process, followed by detailed instructions on the task and expected JSON output format. This approach improved classification outcomes, especially for reviews with characteristics spanning multiple categories.

Post-classification, we apply a precedence order (e.g., Bug > Feature > User Experience > Rating) for final categorization. This priority ordering ensures that we capture

```
Task Description:
Review user reviews for mobile applications based on their
content, sentiment, and ratings. Utilize the definitions
provided to classify each review into the appropriate category.
Definitions for Classification:
Bug Reports:
Definition: Bug reports are user comments that identify issues
with the app, such as crashes, incorrect behavior, or
performance problems. These reviews specifically highlight
problems that affect the app's functionality and suggest a
need for corrective action.
Feature Requests:
Definition: Feature requests are suggestions by users for new
features or enhancements in future app updates.
These can include requests for features seen in other apps,
additions to content, or ideas to modify existing
features to enhance user interaction and satisfaction.
User Experience:
Definition: User experience reviews provide detailed narratives
focusing on specific app features and their effectiveness in
real scenarios. They offer insights into the app's usability,
functionality, and overall satisfaction, often serving as
informal documentation of user needs and app performance.
Differentiating Tip: Prioritize reviews that give detailed
explanations of the app's features and their practical impact
on the user.
Ratings:
Definition: Ratings are brief textual comments that reflect
the app's numeric star rating, primarily indicating overall
user satisfaction or dissatisfaction. These reviews are
succinct, focusing on expressing a general sentiment without
detailed justification.
Differentiating Tip: Focus on reviews that lack detailed
discussion of specific features or user experiences, and
instead provide general expressions of approval or disapproval.
Questions:
Q1.Does it sound like a Bug Report?:
Q2.Explain why Q1 is True/False:
Q3.Does it sound like a missing Feature?:
Q4.Explain why Q3 is True/False:
Q5.Does it sound like a User Experience?:
Q6.Explain why Q5 is True/False:
Q7.Does it sound like a Rating?:
Q8.Explain why Q7 is True/False:
Instructions to the Language Model:
Review Processing: Carefully read the provided app review and
its star rating and answer all questions.
Output Format: Provide the classification results in the
following JSON format:
{
    "Q1.Does it sound like a Bug Report?": "",
    "Q2.Explain why Q1 is True/False": "",
    "Q3.Does it sound like a missing Feature?": "",
    "Q4.Explain why Q3 is True/False": "",
    "Q5.Does it sound like a User Experience?": "",
    "Q6.Explain why Q5 is True/False": "",
    "Q7.Does it sound like a Rating?": "",
    "Q8.Explain why Q7 is True/False": ""
}
Review and Star Rating to Classify:
Review: "Absolutely handy for those pics you don't need
everyone else to see."
Star Rating: 3 out of 5
```

Fig. 5.2: Prompt template: Annotating app review data

the most relevant aspect of each review, reducing ambiguity and aligning with our ex-

perimental goals. For instance, a review classified as both a bug and a user experience issue would be prioritized as a bug, which is our primary focus.

The combination of a structured template, guided questions, and priority ordering significantly enhanced the model's ability to accurately classify user reviews across multiple categories. This approach allows for nuanced classification while maintaining consistency in cases where reviews could potentially fall into multiple categories.

5.2.4.2 Prompts for Fine-Tuning Custom Models

We developed two primary templates for fine-tuning custom models: `Template 1` and `Template 2`. These templates were designed to optimize the performance of open-source models while managing computational constraints. Both templates structures include following key elements:

- Task Description: Outlining the specific requirements for the model.
- Class Definitions: Providing comprehensive descriptions of the review categories.
- Model Instructions: Offering explicit guidelines for input processing and output generation.

To manage computational constraints, we limited the maximum sequence length to 800 tokens, balancing model capacity with resource limitations. We chose JSON format for output due to its ease of programmatic extraction. Class definitions were included in prompts to potentially enhance custom model performance.

5.2.4.2.1 Template 1: Basic Classification Figure 5.3 presents the first template, which focuses on straightforward classification. This template consists of:

- Task Description: Clearly stating the review classification objective.
- Class Definitions: Detailed explanations of Bug Reports, Feature Requests, User Experience, and Ratings categories.
- Model Instructions: Directing the model to classify the review into one of the four categories.
- Output Format: Specifying a JSON structure for the classification result.

The systematic approach in Template 1 ensures consistent, reproducible results across different model architectures. The JSON output format facilitates easy extraction and evaluation of results.

```
Task Description:
Review user reviews for mobile applications based on their
content, sentiment, and ratings. Utilize the definitions
provided to classify each review into the appropriate category.

Definitions for Classification:

Bug Reports:
Definition: Bug reports are user comments that identify issues
with the app, such as crashes, incorrect behavior, or
performance problems. These reviews specifically highlight
problems that affect the app's functionality and suggest a
need for corrective action.

Feature Requests:
Definition: Feature requests are suggestions by users for new
features or enhancements in future app updates. These can
include requests for features seen in other apps, additions
to content, or ideas to modify existing features to enhance
user interaction and satisfaction.

User Experience:
Definition: User experience reviews provide detailed narratives
focusing on specific app features and their effectiveness in
real scenarios. They offer insights into the app's usability,
functionality, and overall satisfaction, often serving as
informal documentation of user needs and app performance.
Differentiating Tip: Prioritize reviews that give detailed
explanations of the app's features and their practical impact
on the user.

Ratings:
Definition: Ratings are brief textual comments that reflect
the app's numeric star rating, primarily indicating overall
user satisfaction or dissatisfaction. These reviews are
succinct, focusing on expressing a general sentiment without
detailed justification.
Differentiating Tip: Focus on reviews that lack detailed
discussion of specific features or user experiences, and
instead provide general expressions of approval or disapproval.

Instructions to the Language Model:
Review Processing: Carefully read the provided app review and
its star rating and Classify the review into one of the
following categories: "Bug", "Feature", "UserExperience",
or "Rating".

Output Format: Provide the classification results in the
following JSON format:
{
   "Class": "<predition>"
}

Review and Star Rating to Classify:
User Review : "Absolutely handy for those pics you don't need
everyone else to see."
User Rating : 3 out of 5
```

Fig. 5.3: Template 1: App review classification prompt for open-source models

5.2.4.2.2 Template 2: *"Explain-then-annotate"* Approach Figure 5.4 illustrates the second template, which adopts an *explain-then-annotate* pattern [29]. This template maintains the structure of Template 1 but introduces a crucial modification:

- Additional Instruction: The model is explicitly directed to provide an explana-

tion for its classification decision before presenting the final category assignment.

- Enhanced Output Format: The JSON structure now includes both the explanation and the class prediction.

This dual-output approach serves multiple purposes:

1. It provides insight into the model's reasoning process.
2. It serves as a quality control measure, helping to identify instances where the model's explanation might not align with its final classification.
3. It enhances the model's interpretability and decision-making process.

During the fine-tuning phase, we leveraged explanations generated from the prompt template described in section 5.2.4.1 to serve as exemplars for this new requirement. By providing these pre-generated explanations, we aimed to guide the model towards producing coherent and relevant justifications for its decisions.

For both templates, we append the user review content and the star rating given by the user to the prompt. This approach allows for a comprehensive analysis of the review, taking into account both textual content and numerical rating.

The inclusion of explanations opens up new avenues for analysis, potentially allowing for the identification of patterns in the model's reasoning or biases in its decision-making process.

By maintaining a standard format across multiple experiments, we can draw more meaningful comparisons between the performance of different model architectures. This methodical approach to prompt design and model fine-tuning provided a solid foundation for investigating the efficacy of open-source models in app review classification tasks.

5.2.5 Fine-Tuning Open Source Models

We utilized the Hugging Face `SFTTrainer` Library for fine-tuning. We employed several optimization techniques to accommodate our consumer-grade GPU (24 GB VRAM):

- 4-bit quantization via `QLoRA` [55], enabling efficient fine-tuning while maintaining high performance.
- `PEFT` [56] (Parameter-Efficient Fine-Tuning), which updates only a subset of the model's most influential parameters, reducing computational requirements.

We maintained consistent model configurations across all experiments to mitigate potential training biases.

```
Task Description:
Review user reviews for mobile applications based on their content,
sentiment, and ratings. Utilize the definitions provided to classify
each review into the appropriate category.

Definitions for Classification:

Bug Reports:
Definition: Bug reports are user comments that identify issues with
the app, such as crashes, incorrect behavior, or performance problems.
These reviews specifically highlight problems that affect the app's
functionality and suggest a need for corrective action.

Feature Requests:
Definition: Feature requests are suggestions by users for new features
or enhancements in future app updates. These can include requests for
features seen in other apps, additions to content, or ideas to modify
existing features to enhance user interaction and satisfaction.

User Experience:
Definition: User experience reviews provide detailed narratives
focusing on specific app features and their effectiveness in real
scenarios. They offer insights into the app's usability, functionality,
and overall satisfaction, often serving as informal documentation of
user needs and app performance.
Differentiating Tip: Prioritize reviews that give detailed explanations
of the app's features and their practical impact on the user.

Ratings:
Definition: Ratings are brief textual comments that reflect the app's
numeric star rating, primarily indicating overall user satisfaction or
dissatisfaction. These reviews are succinct, focusing on expressing a
general sentiment without detailed justification.
Differentiating Tip: Focus on reviews that lack detailed discussion of
specific features or user experiences, and instead provide general
expressions of approval or disapproval.

Instructions to the Language Model:
Review Processing: Carefully read the provided app review and its star
rating. Give a brief explanation of the classification decision made
for the review and Classify the review into one of the following
categories: "Bug", "Feature", "UserExperience", or "Rating".

Output Format: Provide the classification results in the following JSON
format:
{
   "Explanation": "<explanation>",
   "Class": "<predition>"
}

Review and Star Rating to Classify:
```

Fig. 5.4: Template 2: App review classification prompt for open-source models

5.2.6 Evaluation Strategy

We conducted three experimental runs for each experiment and reported the average results to ensure the reliability of our findings.

Occasionally, even when provided with correct formatting instructions, both commercial and open-source Large Language Models (LLMs) produced responses with inaccurate JSON formatting. These instances were resubmitted to the classification loop until the LLM generated a valid JSON response compatible with our automation script.

To evaluate the performance of our experiments, we selected precision, recall, and F1-score metrics, aligning with methodologies established in prior research [48].

Given our balanced dataset, we employed macro-averaging as an aggregated metric to compute overall performance. Additionally, we manually reviewed and measured the quality of the automatically annotated dataset. To estimate the required sample size, we used the Krejcie and Morgan Table [57].

The manual review process involved three annotators. We calculated inter-annotator agreement using Cohen's kappa to assess consistency among annotators. When determining the class label for a given review, we adopted the majority label. In cases where no majority label emerged, we resolved discrepancies through discussion among the three annotators.

As we utilized an *explain-then-annotate* pattern in one of the prompt templates used to train the open-source LLMs, we also manually reviewed and evaluated the accuracy of the generated explanations. For this evaluation, we considered only the correctly classified instances and assessed the accuracy of generated explanations.

5.3 Experiments

We divided our experiments into two main parts. First, we tested commercially available closed-source models to see how well they could classify app reviews. We tested two well-known Large Language Models (LLMs) in different settings. Then, we used the best model and setting as an annotator to create a dataset fully autonomously. We used this dataset to fine-tune three popular open-source models in various settings.

Section 5.3.1 describes our experiments with commercial LLMs in detail. Section 5.3.4 explains our work with open-source LLMs.

Model Name	Bugs			Feature			Userexperience			Rating			Macro Avg		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Gemini Pro	1.00000	0.69333	0.81642	0.96270	0.66000	0.78264	0.67373	0.89333	0.76808	0.91623	0.50667	0.65246	0.81142	0.76333	0.75490
GPT 3.5 Turbo	0.83261	0.92667	0.87701	0.84969	0.82667	0.83788	0.87150	0.86000	0.86557	0.84871	0.78667	0.81624	0.85063	0.85000	0.84917
Base llama	0.60318	0.88000	0.71542	0.88189	0.28667	0.43142	0.67024	0.48667	0.56284	0.59229	0.74667	0.66046	0.66440	0.60000	0.57753
Base mistral	0.66769	0.73333	0.69882	0.63743	0.22000	0.32649	0.40545	0.80000	0.53798	0.43591	0.25333	0.31912	0.53662	0.50167	0.47060
llama + instruct finetune (10k)	0.84212	0.88667	0.86375	0.78199	0.86000	0.81910	0.85761	0.92000	0.88759	0.87924	0.68000	0.76620	0.84024	0.83667	0.83416
mistral + instruct finetune (10k)	0.85926	0.77333	0.81404	0.74127	0.92667	0.82294	0.77677	0.88000	0.82482	0.87438	0.62000	0.72356	0.81292	0.80000	0.79634
llama + instruct finetune (10k) + explanation	0.83144	0.88667	0.85794	0.74492	0.83333	0.78637	0.85523	0.89333	0.87385	0.85480	0.66667	0.74837	0.82410	0.82000	0.81786
mistral + instruct finetune (10k) + explanation	0.81092	0.85333	0.83119	0.72597	0.84667	0.78152	0.88876	0.90000	0.89410	0.89881	0.68667	0.77778	0.83112	0.82167	0.82115

TABLE 5.1: Model Performance Comparison Including `Gemini Pro` and `GPT 3.5 Turbo`

5.3.1 Evaluating Commercial Model Performance

To address our primary research question, we evaluated the classification performance of two prominent commercial models: Google's `Gemini Pro 1.0` and OpenAI's `GPT-3.5`. Experiments were conducted in a zero-shot setting utilizing prompt template described in section 5.2.4.1. As presented in Table 5.1, we obtained F1 scores of 0.75490 and 0.84917 for `Gemini Pro` and `GPT-3.5`, respectively. `GPT-3.5` demonstrated superior performance with a margin of approximately 0.09427.

5.3.2 Effects of `Temperature` and `Top_p` Parameters on Classification Accuracy of Commercial Models

This section presents our analysis of the impact of `Temperature` and `Top_p` parameters on language model classification accuracy for mobile app reviews. We investigated these performance-tuning parameters for two commercially available closed-source models: `Gemini Pro 1.0` and `GPT-3.5`.

Our study examined how varying `Temperature` and `Top_p` parameters (ranging from 0 to 1) affect the models' performance in classifying mobile app reviews. Tables 5.2 and 5.3 provide detailed performance metrics for `Gemini Pro 1.0` and `GPT-3.5`, respectively. These metrics include precision, recall, and F1 scores for four categories: bugs, features, user experience, and ratings, as well as macro averages to indicate overall model performance across all categories.

Our findings indicate that parameter tuning can significantly enhance model performance. Both models generally maintain high performance across different `Temperature` and `Top_p` settings, with some variations observed. For `Gemini Pro 1.0`, the macro average F1 scores range from approximately 0.75 to 0.78, while `GPT-3.5` shows slightly higher performance with macro average F1 scores ranging from about 0.79 to 0.88.

`GPT-3.5` exhibited greater responsiveness to parameter changes, achieving higher performance in the app review classification task when `Temperature` or `Top_p` was set to lower values. However, when both parameters were set to their maximum values (`Temperature` = 2 and `Top_p` = 1), both models failed to produce results, frequently encountering errors at the REST API level.

The OpenAI documentation recommends modifying only one parameter at a time to achieve predictable outcomes. Lower parameter values result in more focused model responses, while higher values produce more creative responses. Our observations aligned with this recommendation, as changing both parameters simultaneously led to deviations from the expected behavior pattern.

A noteworthy observation was that `Gemini Pro 1.0` produced identical F1 scores at lower `Temperature` and `Top_p` values. To validate this phenomenon, we conducted repeated experiments and found consistent results. Further analysis on

the generated classification responses revealed identical outputs in mutually exclusive classification runs, suggesting that this could be a potential caching effect or model-specific issue.

These findings demonstrate the importance of carefully considering and tuning the `Temperature` and `Top_p` parameters when using commercial language models for classification tasks. The optimal configurations may vary depending on the use case and desired precision-recall trade-off. However, care should be taken when adjusting these parameters, as extreme values or simultaneous changes to both parameters can lead to unexpected results or errors. By understanding and optimizing these parameters, researchers and practitioners can potentially improve the performance and reliability of language models in various classification tasks.

TABLE 5.2: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `Gemini Pro 1.0`

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.25	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.5	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.75	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67350	0.88000	0.76302	0.93021	0.53333	0.67792	0.81984	0.77333	0.76685
	1	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
0.5	0	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.25	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.5	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67350	0.88000	0.76302	0.93021	0.53333	0.67792	0.81984	0.77333	0.76685
	0.75	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67703	0.88000	0.76517	0.90857	0.52667	0.66631	0.81531	0.77167	0.76448
	1	0.70097	1.00000	0.82419	0.98095	0.68000	0.80317	0.64902	0.88000	0.74631	0.91098	0.47333	0.62043	0.81048	0.75833	0.74853
1	0	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.25	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.5	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67514	0.88667	0.76657	0.94130	0.53333	0.68084	0.82302	0.77500	0.76846
	0.75	0.70097	1.00000	0.82419	0.98095	0.68000	0.80317	0.68795	0.88000	0.77209	0.91161	0.54667	0.68299	0.82037	0.77667	0.77061
	1	0.69301	0.99333	0.81642	0.96270	0.66000	0.78264	0.67373	0.89333	0.76808	0.91623	0.50667	0.65246	0.81142	0.76333	0.75490
1.5	0	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.25	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.5	0.70423	1.00000	0.82645	0.97115	0.67333	0.79524	0.68019	0.89333	0.77231	0.94212	0.54000	0.68647	0.82442	0.77667	0.77012
	0.75	0.70097	1.00000	0.82419	0.98095	0.67333	0.79839	0.68331	0.92000	0.78406	0.97575	0.52667	0.68395	0.83524	0.78000	0.77265
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.25	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67692	0.88000	0.76522	0.93103	0.54000	0.68354	0.82090	0.77500	0.76880
	0.5	0.70423	1.00000	0.82645	0.97143	0.68000	0.80000	0.67677	0.89333	0.77011	0.95238	0.53333	0.68376	0.82620	0.77667	0.77008
	0.75	0.69770	1.00000	0.82193	0.99020	0.67333	0.80159	0.70346	0.92667	0.79941	0.97536	0.55333	0.70412	0.84168	0.78833	0.78176
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

TABLE 5.3: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `GPT 3.5`

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.85040	0.94667	0.89589	0.86274	0.86667	0.86415	0.88745	0.89333	0.89036	0.91599	0.80000	0.85360	0.87915	0.87667	0.87600
	0.25	0.85538	0.94667	0.89865	0.85136	0.87333	0.86208	0.85809	0.88667	0.87213	0.89605	0.74667	0.81454	0.86522	0.86333	0.86185
	0.5	0.85567	0.94667	0.89876	0.86768	0.87333	0.87042	0.89436	0.90000	0.89709	0.90165	0.79333	0.84396	0.87984	0.87833	0.87756
	0.75	0.84642	0.95333	0.89660	0.87797	0.86000	0.86865	0.88190	0.89333	0.88744	0.90175	0.79333	0.84386	0.87701	0.87500	0.87414
	1	0.86147	0.95333	0.90506	0.84527	0.87333	0.85906	0.90703	0.90667	0.90642	0.89978	0.77333	0.83148	0.87839	0.87667	0.87550
0.5	0	0.84747	0.96000	0.90012	0.87156	0.86000	0.86572	0.88030	0.88000	0.87998	0.89391	0.78667	0.83679	0.87331	0.87167	0.87065
	0.25	0.83633	0.95333	0.89096	0.85333	0.85333	0.85333	0.89265	0.88667	0.88963	0.91537	0.79333	0.84982	0.87442	0.87167	0.87094
	0.5	0.83438	0.94000	0.88403	0.80917	0.84667	0.82738	0.89276	0.88667	0.88962	0.90399	0.75333	0.82179	0.86007	0.85667	0.85570
	0.75	0.84033	0.94667	0.89032	0.86036	0.86000	0.86012	0.91143	0.86000	0.88397	0.85818	0.79333	0.82285	0.86758	0.86500	0.86431
	1	0.83556	0.94667	0.88754	0.84211	0.84667	0.84403	0.89056	0.86667	0.87837	0.87205	0.77333	0.81971	0.86007	0.85833	0.85741
1	0	0.84983	0.94000	0.89253	0.85554	0.86667	0.86082	0.88931	0.90667	0.89773	0.91473	0.78667	0.84588	0.87735	0.87500	0.87424
	0.25	0.85231	0.96000	0.90289	0.86114	0.86667	0.86382	0.90556	0.89333	0.89932	0.88636	0.78000	0.82979	0.87634	0.87500	0.87395
	0.5	0.84211	0.96000	0.89720	0.86068	0.86000	0.86017	0.87153	0.90000	0.88530	0.91960	0.76000	0.83203	0.87348	0.87000	0.86867
	0.75	0.83653	0.95333	0.89108	0.87148	0.85333	0.86188	0.88974	0.86000	0.87460	0.86754	0.79333	0.82836	0.86632	0.86500	0.86398
	1	0.83261	0.92667	0.87701	0.84969	0.82667	0.83788	0.87150	0.86000	0.86557	0.84871	0.78667	0.81624	0.85063	0.85000	0.84918
1.5	0	0.85213	0.96000	0.90284	0.87191	0.86000	0.86577	0.89419	0.90000	0.89695	0.90152	0.79333	0.84397	0.87994	0.87833	0.87738
	0.25	0.84463	0.94000	0.88966	0.83220	0.86000	0.84586	0.89331	0.89333	0.89326	0.91430	0.78000	0.84165	0.87111	0.86833	0.86760
	0.5	0.84625	0.95333	0.89654	0.86023	0.86000	0.86006	0.87510	0.88667	0.88071	0.89919	0.77333	0.83145	0.87019	0.86833	0.86719
	0.75	0.84288	0.92667	0.88264	0.84244	0.85333	0.84762	0.88889	0.85333	0.87075	0.86309	0.80000	0.83033	0.85933	0.85833	0.85783
	1	0.82270	0.92000	0.86822	0.80495	0.82000	0.81224	0.80928	0.73333	0.76905	0.72008	0.68667	0.70291	0.78925	0.79000	0.78810
2	0	0.84531	0.94667	0.89308	0.84989	0.86667	0.85814	0.90460	0.87333	0.88837	0.89607	0.80000	0.84487	0.87397	0.87167	0.87111
	0.25	0.84520	0.94667	0.89302	0.86585	0.86000	0.86290	0.88958	0.90667	0.89765	0.92407	0.80000	0.85692	0.88118	0.87833	0.87762
	0.5	0.84642	0.95333	0.89660	0.84532	0.86667	0.85535	0.87549	0.88667	0.88091	0.91212	0.76000	0.82911	0.86984	0.86667	0.86549
	0.75	0.84474	0.94000	0.88971	0.85443	0.86000	0.85708	0.88645	0.86000	0.87171	0.89192	0.80667	0.84556	0.86938	0.86667	0.86602
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

5.3.3 Evaluating Dataset Quality through Inter-Annotator Agreement Analysis

To evaluate the quality of the generated dataset, we randomly selected a sample of 370 reviews from the 10,000 app review dataset annotated using `GPT-3.5`. This sample size was chosen to achieve a 95% confidence level with a 5% margin of error, based on the guidelines provided by Krejcie and Morgan [57].

Three experienced developers served as annotators for this sample. We calculated the inter-annotator agreement using Cohen's Kappa (κ) [58]. The κ value was computed between each pair of annotators, and the results are presented in Table 5.4.

Annotator Pair	Kappa Score	Agreement Level
Annotator 1 vs 2	0.9146	Almost perfect
Annotator 1 vs 3	0.9180	Almost perfect
Annotator 2 vs 3	0.9079	Almost perfect
Average	0.9135	Almost perfect

TABLE 5.4: Pairwise Cohen's Kappa Scores and Agreement Levels

For interpreting κ values, we used the scale by Landis and Koch [59]:

- $\kappa \leq 0$: Less than chance agreement
- $0.01 \leq \kappa \leq 0.20$: Slight agreement
- $0.21 \leq \kappa \leq 0.40$: Fair agreement
- $0.41 \leq \kappa \leq 0.60$: Moderate agreement
- $0.61 \leq \kappa \leq 0.80$: Substantial agreement
- $0.81 \leq \kappa \leq 1$: Almost perfect agreement

As shown in Table 5.4, the pairwise Cohen's Kappa scores range from 0.9079 to 0.9180, with an average of 0.9135. These scores all fall within the "almost perfect agreement" category, indicating strong consistency in the annotations across all three annotators and lending credibility to the reliability of our annotation process.

To provide further insight into the composition of our dataset, Table 5.5 displays the distribution of annotations across four categories (Bug, Feature, Rating, and User-Experience) for each annotator.

Annotator	Bug	Feature	Rating	UserExp
Annotator 1	84	55	84	147
Annotator 2	82	69	89	130
Annotator 3	84	68	83	135

TABLE 5.5: Annotation Distribution by Annotator and Category

The similar distributions across annotators in Table 5.5 further support the high inter-annotator agreement observed in the Kappa scores. This distribution provides insight into the composition of our dataset and the relative frequency of each category.

Based on the human annotators' assessment, we calculated the average accuracy of the `GPT-3.5` annotated dataset to be 81.89%. This high accuracy, combined with the strong inter-annotator agreement, suggests that our `GPT-3.5`-generated dataset is of high quality and reliability for app review classification tasks.

In conclusion, our inter-annotator agreement analysis demonstrates the high quality and reliability of our annotated dataset. The strong agreement among annotators, as evidenced by the high Kappa scores and consistent annotation distributions, combined with the high accuracy of the `GPT-3.5`-generated annotations, provides a solid foundation for further research and applications in app review classification.

5.3.4 Evaluating Open Source Models

Motivated by the impressive performance of commercial closed-source models in app review classification, we extended our evaluation to popular open-source models, including `Llama 2`, `Mistral`, and `Falcon`. Initially, we assessed the performance of these models using our benchmark dataset. Table 5.1 presents the base models' performance results.

Our experimental results shows that base `Llama 2` and `Mistral` achieved F1 scores of 0.57753 and 0.47060, respectively. It is noteworthy that we initially intended to include the `Falcon` model in our experiments. However, due to its inability to adhere to our required output response JSON formatting, even after fine-tuning attempts, we excluded it from subsequent experiments.

We proceeded to instruction-fine-tune `Llama 2` and `Mistral` using a dataset with 10,000 samples annotated by `GPT-3.5`. For this process, we used two different instruction prompt templates, which are described in Section 5.2.4.2.

Table 5.1 demonstrates that `Llama 2` exhibits superior performance with prompt `Template 1` from section 5.2.4.2, while `Mistral` achieves optimal performance with the *"explain-then-annotate"* prompt template (`Template 2`).

5.3.5 Effect of Training Dataset Size on Model Performance

To determine the optimal training dataset size, we investigated the relationship between model accuracy and training data sample size, maintaining a single training epoch while varying the dataset size. The results, presented in Table 5.6, indicate that model performance generally increases with the training dataset size.

Our experiments covered a range of dataset sizes from 500 to 10,000 reviews. We evaluated the performance of two models, `LLAMA 2` and `Mistral`, both with Template 1 (fig 5.3) and Template 2 (fig: 5.4). The results show a clear trend of improving

performance as the dataset size increases, with the most substantial gains observed when moving from smaller to mid-sized datasets.

Interestingly, the inclusion of prediction explanations in the prompt consistently improved model performance, especially in small dataset sizes. This suggests that `Template 2`, which follows an *"explain-then-annotate"* pattern, can be particularly beneficial when working with limited training data.

These findings highlight the importance of having a sufficiently large and diverse training dataset for optimal model performance in app review classification tasks. However, they also suggest that there may be diminishing returns beyond a certain dataset size, indicating a potential trade-off between computational resources and marginal performance gains.

Training Sample Size	Model Name	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
(a) Without label explanations in the prompt																
10000 1 epoch	llama	0.84212	0.88667	0.86375	0.78199	0.86000	0.81910	0.85761	0.92000	0.88759	0.87924	0.68000	0.76620	0.84024	0.83667	0.83416
	mistral	0.85926	0.77333	0.81404	0.74127	0.92667	0.82294	0.77677	0.88000	0.82482	0.87430	0.62000	0.72356	0.81290	0.80000	0.79634
8000 1 epoch	llama	0.83716	0.92667	0.87953	0.78139	0.80000	0.78973	0.91032	0.86667	0.88768	0.79660	0.72667	0.75813	0.83136	0.83000	0.82877
	mistral	0.82840	0.89333	0.85899	0.71691	0.85333	0.77857	0.90198	0.85333	0.87683	0.80189	0.62667	0.70287	0.81230	0.80667	0.80432
6000 1 epoch	llama	0.91069	0.72667	0.80738	0.85391	0.84667	0.84966	0.79138	0.92667	0.85303	0.74222	0.76667	0.75342	0.82455	0.81667	0.81587
	mistral	0.89478	0.79333	0.84099	0.78171	0.88000	0.82789	0.78754	0.86000	0.82176	0.76807	0.68667	0.72446	0.80802	0.80500	0.80377
4000 1 epoch	llama	0.68359	0.96000	0.79824	0.89821	0.52000	0.65758	0.82189	0.79333	0.80723	0.75780	0.79333	0.77510	0.79037	0.76667	0.75953
	mistral	0.87606	0.84000	0.85681	0.85662	0.82000	0.83714	0.75278	0.86000	0.80189	0.76785	0.71333	0.73811	0.81333	0.80833	0.80849
2000 1 epoch	llama	0.77242	0.92667	0.84239	0.90623	0.70667	0.79272	0.73109	0.92000	0.81451	0.87722	0.66667	0.75652	0.82174	0.80500	0.80153
	mistral	0.82115	0.73333	0.77467	0.90922	0.73333	0.81150	0.59876	0.93333	0.72946	0.75237	0.56000	0.64179	0.77038	0.74000	0.73935
1500 1 epoch	llama	0.86375	0.84000	0.85106	0.62651	0.86000	0.72475	0.82639	0.86000	0.84223	0.83390	0.50667	0.62787	0.78764	0.76667	0.76148
	mistral	0.77689	0.88000	0.82518	0.68465	0.78000	0.72879	0.83887	0.68667	0.75468	0.70663	0.64000	0.67134	0.75176	0.74667	0.74500
1000 1 epoch	llama	0.84028	0.82667	0.83196	0.69499	0.88667	0.77829	0.74260	0.77333	0.75690	0.69502	0.48000	0.56660	0.74322	0.74167	0.73344
	mistral	0.73160	0.85333	0.78740	0.70044	0.75333	0.72428	0.77408	0.44667	0.56430	0.57742	0.67333	0.62114	0.69588	0.68167	0.67428
500 1 epoch	llama	0.75900	0.90000	0.82346	0.62743	0.82000	0.71078	0.63354	0.85333	0.72716	0.59259	0.09333	0.15990	0.65314	0.66667	0.60533
	mistral	0.78667	0.82667	0.80590	0.62141	0.79333	0.69668	0.66082	0.74000	0.69806	0.65397	0.36000	0.46352	0.68071	0.68000	0.66604
(b) With label explanations in the prompt																
10000 1 epoch + Explanation	llama	0.83144	0.88667	0.85794	0.74492	0.83333	0.78637	0.86523	0.89333	0.87882	0.85480	0.66667	0.74837	0.82410	0.82000	0.81788
	mistral	0.81092	0.85333	0.83119	0.72597	0.84667	0.78152	0.88876	0.90000	0.89410	0.89881	0.68667	0.77778	0.83112	0.82167	0.82115
8000 1 epoch + Explanation	llama	0.77874	0.95333	0.85667	0.82365	0.75333	0.78381	0.94259	0.84667	0.89139	0.83339	0.79333	0.81113	0.84459	0.83667	0.83575
	mistral	0.83647	0.81333	0.82437	0.74000	0.81333	0.77481	0.81173	0.91333	0.85909	0.82413	0.66000	0.73277	0.80309	0.80000	0.79776
6000 1 epoch + Explanation	llama	0.84879	0.82000	0.83378	0.85209	0.74000	0.79109	0.71503	0.94667	0.81394	0.79033	0.66000	0.71697	0.80156	0.79167	0.78894
	mistral	0.83219	0.66000	0.73546	0.80097	0.78667	0.79232	0.65468	0.90667	0.76009	0.70443	0.58667	0.63985	0.74807	0.73500	0.73193
4000 1 epoch + Explanation	llama	0.68596	0.96000	0.80006	0.82963	0.67333	0.74267	0.89670	0.79333	0.84099	0.80734	0.72667	0.76355	0.80490	0.78833	0.78682
	mistral	0.74446	0.86667	0.79992	0.85739	0.68000	0.75803	0.80125	0.91333	0.85350	0.82662	0.74000	0.77950	0.80743	0.80000	0.79774
2000 1 epoch + Explanation	llama	0.72317	0.90000	0.80162	0.84313	0.61333	0.70870	0.78207	0.94667	0.85611	0.79459	0.64667	0.71189	0.78574	0.77667	0.76958
	mistral	0.70319	0.90000	0.78930	0.91499	0.57333	0.70491	0.73172	0.90667	0.80784	0.78314	0.66000	0.71526	0.78326	0.76000	0.75433
1500 1 epoch + Explanation	llama	0.75800	0.91333	0.82800	0.68278	0.78667	0.73096	0.88903	0.88000	0.88345	0.88668	0.57333	0.69635	0.80412	0.78833	0.78469
	mistral	0.80024	0.84667	0.82147	0.66083	0.79333	0.72095	0.80488	0.93333	0.86423	0.82316	0.48000	0.60566	0.77228	0.76333	0.75308
1000 1 epoch + Explanation	llama	0.82163	0.88667	0.85256	0.77295	0.85333	0.80976	0.75984	0.84000	0.79782	0.80175	0.56667	0.66212	0.78905	0.78667	0.78056
	mistral	0.80321	0.89333	0.84559	0.72971	0.78667	0.75642	0.72553	0.88000	0.79473	0.85381	0.50667	0.63568	0.77806	0.76667	0.75811
0500 1 epoch + Explanation	llama	0.77581	0.87333	0.82126	0.76586	0.70000	0.73042	0.65362	0.90000	0.75704	0.76717	0.44667	0.56236	0.74062	0.73000	0.71777
	mistral	0.85296	0.77333	0.81091	0.81427	0.81333	0.81309	0.63861	0.95333	0.76463	0.79023	0.47333	0.59175	0.77402	0.75333	0.74509

TABLE 5.6: Model Performance Comparison With and Without Label Explanations in the Prompt

5.3.6 Effect of Number of Epochs on Model Performance

We also examined the effect of the number of epochs using 2,000 reviews from the training set. Table 5.7 presents these results. Our analysis reveals that using fewer training samples with multiple epochs can be as effective as using more samples with fewer epochs.

The experiments were conducted with both `LLAMA 2` and `Mistral` models, varying the number of epochs from 1 to 4. We evaluated the models both with and without label explanations in the prompt. The results show that increasing the number of epochs generally leads to improved model performance, with the most significant gains observed in the earlier epochs.

Notably, the performance improvement from increasing epochs was more pronounced for the `Mistral` model compared to `LLAMA 2`. This suggests that different model architectures may have varying sensitivities to the number of training epochs.

An interesting observation is that models trained with 2,000 samples over multiple epochs achieved comparable or sometimes better performance than models trained on larger datasets with a single epoch. For instance, the `Mistral` model trained on 2,000 samples for 4 epochs (with explanations) achieved a macro average F1 score of 0.84782, which is competitive with the performance of models trained on 10,000 samples for a single epoch.

These findings suggest that when working with limited computational resources or smaller datasets, increasing the number of training epochs can be an effective strategy to improve model performance. However, it's important to note that the optimal number of epochs may vary depending on the specific model and dataset characteristics, and care should be taken to avoid overfitting.

Training Sample Size	Model Name	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
(a) Without label explanations in the prompt																
2000 1 epoch	llama	0.77242	0.92667	0.84239	0.90623	0.70667	0.79272	0.73109	0.92000	0.81451	0.87722	0.66667	0.75652	0.82174	0.80500	0.80153
	mistral	0.82115	0.73333	0.77467	0.90922	0.73333	0.81150	0.59876	0.93333	0.72946	0.75237	0.56000	0.64179	0.77038	0.74000	0.73935
2000 2 epoch	llama	0.92097	0.78000	0.84454	0.83281	0.79333	0.81234	0.87657	0.83333	0.85400	0.68463	0.85333	0.75942	0.82875	0.81500	0.81758
	mistral	0.86996	0.84667	0.85802	0.89855	0.82667	0.86111	0.88151	0.74667	0.80787	0.71271	0.89333	0.79230	0.84068	0.82833	0.82982
2000 3 epoch	llama	0.86843	0.82667	0.84663	0.81624	0.82667	0.82113	0.72137	0.90667	0.80306	0.82712	0.64000	0.71965	0.80829	0.80000	0.79762
	mistral	0.91007	0.80667	0.85480	0.86773	0.82667	0.84644	0.74995	0.93333	0.83131	0.79634	0.72667	0.75965	0.83102	0.82333	0.82305
2000 4 epoch	llama	0.87827	0.86000	0.86886	0.82239	0.82667	0.82384	0.88889	0.86667	0.87581	0.81278	0.84000	0.82595	0.85058	0.84833	0.84862
	mistral	0.88675	0.78000	0.82975	0.75054	0.89333	0.81508	0.90089	0.78000	0.83539	0.80544	0.85333	0.82809	0.83591	0.82667	0.82708
(b) With label explanations in the prompt																
2000 1 epoch + Explanation	llama	0.72317	0.90000	0.80162	0.84313	0.61333	0.70870	0.78207	0.94667	0.85611	0.79459	0.64667	0.71189	0.78574	0.77667	0.76958
	mistral	0.70319	0.90000	0.78930	0.91499	0.57333	0.70491	0.73172	0.90667	0.80784	0.78314	0.66000	0.71526	0.78326	0.76000	0.75433
2000 2 epoch + Explanation	llama	0.84654	0.80667	0.82598	0.81867	0.80667	0.81206	0.93663	0.80000	0.86182	0.72556	0.87333	0.79201	0.83185	0.82167	0.82297
	mistral	0.85470	0.82000	0.83693	0.79603	0.82667	0.81070	0.89807	0.82000	0.85689	0.75678	0.82000	0.78572	0.82640	0.82167	0.82256
2000 3 epoch + Explanation	llama	0.88934	0.77333	0.82472	0.84636	0.88000	0.86280	0.68674	0.96667	0.80233	0.89832	0.60667	0.72296	0.83019	0.80667	0.80320
	mistral	0.90242	0.68667	0.77964	0.86148	0.84000	0.84956	0.61443	0.98667	0.75723	0.87756	0.57333	0.69310	0.81397	0.77167	0.76988
2000 4 epoch + Explanation	llama	0.87177	0.86000	0.86571	0.84096	0.80667	0.82340	0.80829	0.92667	0.86338	0.83194	0.75333	0.79054	0.83824	0.83667	0.83576
	mistral	0.88175	0.84000	0.85979	0.82985	0.86667	0.84737	0.87276	0.91333	0.89235	0.81136	0.77333	0.79175	0.84893	0.84833	0.84782

TABLE 5.7: Model Performance Comparison With and Without Label Explanations in the Prompt

5.3.7 Evaluation of Explanation Quality

To assess the effectiveness of our approach in generating meaningful explanations for app review classifications, we conducted a manual review of the explanations generated by the fine-tuned `Mistral` model. This model was chosen for its superior performance when using `Template 2`, which incorporates label explanations in the prompt.

Our evaluation focused solely on correctly classified instances to ensure that we were assessing the quality of explanations for accurate predictions. We employed a four-tier grading system to evaluate the generated explanations:

- **Grade A**: Correct and comprehensive responses containing all relevant and important details from the user review.
- **Grade B**: Correct responses with some relevant details, but not as comprehensive as Grade A.
- **Grade C**: Generic responses lacking specific details from the user review.
- **Grade D**: Incorrect explanations inconsistent with the user review content.

Figure 5.5 presents the distribution of grades for the explanations generated by the fine-tuned `Mistral 7B` model on our 200-review benchmark dataset.

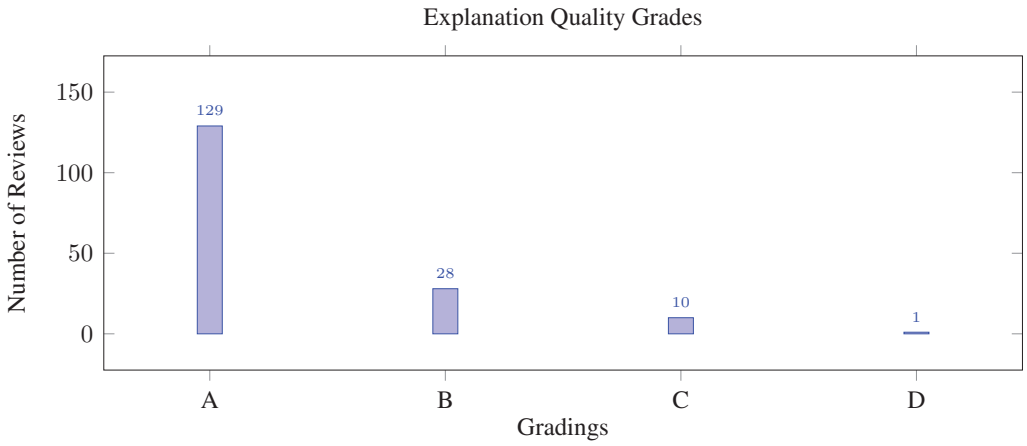


Fig. 5.5: Distribution of grades for explanations generated by the fine-tuned `Mistral 7B` model on our 200-review benchmark dataset, evaluated by human experts. Grades range from A (highest quality) to D (lowest quality).

Out of the 168 correct classifications, 129 explanations (76.79%) were graded as A, 28 (16.67%) as B, 10 (5.95%) as C, and only 1 (0.60%) as D. These results are highly encouraging, with 93.45% of the explanations (grades A and B combined) deemed satisfactory, providing correct and relevant details from the user reviews.

The high proportion of Grade A explanations (76.79%) demonstrates the model's strong capability to generate accurate and detailed explanations for app review classifications. This suggests that the model not only classifies reviews correctly but also understands the nuances and specific details that justify its classifications. The very low rate of Grade D explanations (only one out of 168) further underscores the reliability of the generated explanations.

These findings demonstrate the effectiveness of our approach in combining accurate classification with meaningful explanations. The fine-tuned `Mistral` model's ability to provide high-quality explanations potentially enhances the interpretability and trustworthiness of the model's decisions in real-world applications of app review analysis. This capability is particularly valuable in scenarios where understanding the reasoning behind classifications is as important as the classifications themselves.

5.3.8 Effects of `Temperature` and `Top_p` on Fine-tuned Open-source Models

This subsection examines the impact of `Temperature` and `Top_p` parameters on the performance of fine-tuned open-source language models, specifically `LLAMA 2` and `Mistral`, using two different prompting strategies. Tables 5.8, 5.9, 5.10, and 5.11 present the performance metrics for these models across various parameter settings.

Both models demonstrate strong performance, with macro average F1 scores ranging from approximately 0.70 to 0.87, comparable to commercial models. Performance is generally most stable at lower `Temperature` settings (0 to 0.5) and higher `Top_p` values (0.75 to 1), with some degradation observed at higher `Temperatures` and lower `Top_p` values. The *explain-then-annotate* prompting strategy (Tables 5.9 and 5.11) shows slight improvements in certain scenarios, particularly for the `Mistral` model.

These findings suggest that careful tuning of generation parameters can optimize the performance of fine-tuned open-source models for mobile app review classification tasks.

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.85185	0.92000	0.88462	0.82323	0.90000	0.85989	0.89040	0.92000	0.90494	0.92433	0.73333	0.81771	0.87245	0.86833	0.86679
	0.25	0.84669	0.92000	0.88181	0.82211	0.89333	0.85623	0.87905	0.92000	0.89904	0.92304	0.72000	0.80889	0.86772	0.86333	0.86149
	0.5	0.85092	0.91333	0.88101	0.80861	0.90000	0.85180	0.88462	0.92000	0.90196	0.92240	0.71333	0.80448	0.86664	0.86167	0.85981
	0.75	0.84669	0.92000	0.88181	0.81706	0.89333	0.85348	0.87905	0.92000	0.89904	0.92240	0.71333	0.80448	0.86630	0.86167	0.85970
	1	0.84153	0.92000	0.87900	0.81261	0.89333	0.85088	0.87264	0.91333	0.89251	0.92091	0.70000	0.79504	0.86192	0.85667	0.85436
0.5	0	0.84669	0.92000	0.88181	0.81706	0.89333	0.85348	0.87905	0.92000	0.89904	0.92240	0.71333	0.80448	0.86630	0.86167	0.85970
	0.25	0.84669	0.92000	0.88181	0.82716	0.89333	0.85897	0.87905	0.92000	0.89904	0.92372	0.72667	0.81340	0.86915	0.86500	0.86331
	0.5	0.84052	0.91333	0.87540	0.80606	0.88667	0.84444	0.88462	0.92000	0.90196	0.92240	0.71333	0.80448	0.86340	0.85833	0.85657
	0.75	0.85120	0.91333	0.88105	0.78832	0.89333	0.83754	0.87905	0.92000	0.89904	0.91955	0.68667	0.78601	0.85953	0.85333	0.85091
	1	0.85636	0.91333	0.88386	0.80754	0.89333	0.84821	0.89137	0.92667	0.90862	0.91538	0.72000	0.80599	0.86766	0.86333	0.86167
1	0	0.85092	0.91333	0.88101	0.81331	0.90000	0.85445	0.87349	0.92000	0.89612	0.92173	0.70667	0.79997	0.86486	0.86000	0.85789
	0.25	0.84669	0.92000	0.88181	0.81706	0.89333	0.85348	0.88483	0.92000	0.90202	0.92304	0.72000	0.80889	0.86791	0.86333	0.86155
	0.5	0.84688	0.92000	0.88186	0.80613	0.88667	0.84444	0.86792	0.92000	0.89320	0.92034	0.69333	0.79084	0.86032	0.85500	0.85259
	0.75	0.84285	0.89333	0.86731	0.77307	0.88000	0.82277	0.89062	0.92000	0.90500	0.90526	0.69333	0.78470	0.85295	0.84667	0.84494
	1	0.84212	0.88667	0.86375	0.78199	0.86000	0.81910	0.85761	0.92000	0.88759	0.87924	0.68000	0.76620	0.84024	0.83667	0.83416
1.5	0	0.85092	0.91333	0.88101	0.81331	0.90000	0.85445	0.89062	0.92000	0.90500	0.92368	0.72667	0.81330	0.86963	0.86500	0.86344
	0.25	0.85185	0.92000	0.88462	0.81818	0.90000	0.85714	0.88462	0.92000	0.90196	0.92308	0.72000	0.80899	0.86943	0.86500	0.86318
	0.5	0.84917	0.93333	0.88911	0.82611	0.88667	0.85530	0.86813	0.92000	0.89326	0.90413	0.69333	0.78471	0.86189	0.85833	0.85560
	0.75	0.83809	0.89333	0.86460	0.81077	0.88000	0.84372	0.84616	0.90667	0.87467	0.88761	0.68667	0.77289	0.84566	0.84167	0.83897
	1	0.83015	0.88000	0.85430	0.73052	0.82667	0.77513	0.84925	0.86000	0.85435	0.80136	0.63333	0.70526	0.80282	0.80000	0.79726
2	0	0.85185	0.92000	0.88462	0.82323	0.90000	0.85989	0.89040	0.92000	0.90494	0.92436	0.73333	0.81781	0.87246	0.86833	0.86681
	0.25	0.85185	0.92000	0.88462	0.82323	0.90000	0.85989	0.87927	0.92000	0.89910	0.92304	0.72000	0.80889	0.86935	0.86500	0.86312
	0.5	0.85383	0.88667	0.86955	0.79173	0.88667	0.83647	0.87207	0.90667	0.88864	0.91777	0.73333	0.81492	0.85885	0.85333	0.85239
	0.75	0.80891	0.87333	0.83946	0.75186	0.80667	0.77824	0.83998	0.90667	0.87190	0.84345	0.64667	0.73204	0.81105	0.80833	0.80541
	1	0.83641	0.82000	0.82807	0.68624	0.85333	0.76033	0.77526	0.87333	0.82098	0.77627	0.50000	0.60611	0.76854	0.76167	0.75387

TABLE 5.8: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `LLMA 2` instruct fine-tuned using Prompt Template 1 (Fig: 5.3)

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.83019	0.88000	0.85437	0.81151	0.86000	0.83500	0.88132	0.94000	0.90970	0.92616	0.75333	0.83066	0.86229	0.85833	0.85743
	0.25	0.81994	0.88000	0.84889	0.79881	0.84667	0.82200	0.88607	0.93333	0.90907	0.91809	0.74667	0.82352	0.85573	0.85167	0.85087
	0.5	0.82506	0.88000	0.85163	0.80510	0.85333	0.82847	0.87512	0.93333	0.90322	0.91761	0.74000	0.81910	0.85572	0.85167	0.85060
	0.75	0.82506	0.88000	0.85163	0.80510	0.85333	0.82847	0.88210	0.94667	0.91318	0.93368	0.74667	0.82950	0.86149	0.85667	0.85569
	1	0.83039	0.88000	0.85442	0.81938	0.84667	0.83278	0.89431	0.96000	0.92587	0.93612	0.78000	0.85093	0.87005	0.86667	0.86600
0.5	0	0.82506	0.88000	0.85163	0.81011	0.85333	0.83114	0.88132	0.94000	0.90970	0.92622	0.75333	0.83085	0.86068	0.85667	0.85583
	0.25	0.82506	0.88000	0.85163	0.80510	0.85333	0.82847	0.88679	0.94000	0.91262	0.92622	0.75333	0.83085	0.86079	0.85667	0.85589
	0.5	0.83019	0.88000	0.85437	0.81132	0.86000	0.83495	0.87421	0.92667	0.89968	0.91057	0.74667	0.82051	0.85657	0.85333	0.85238
	0.75	0.82611	0.88667	0.85530	0.81406	0.84667	0.82999	0.86627	0.94667	0.90457	0.92484	0.73333	0.81767	0.85782	0.85333	0.85188
	1	0.81366	0.90000	0.85455	0.78323	0.84000	0.81012	0.88157	0.93333	0.90650	0.93065	0.70667	0.80297	0.85228	0.84500	0.84354
1	0	0.83551	0.88000	0.85716	0.80130	0.86000	0.82960	0.88677	0.94000	0.91255	0.91809	0.74667	0.82352	0.86042	0.85667	0.85571
	0.25	0.83019	0.88000	0.85437	0.80130	0.86000	0.82960	0.88679	0.94000	0.91262	0.92561	0.74667	0.82654	0.86097	0.85667	0.85578
	0.5	0.83039	0.88000	0.85442	0.83044	0.84667	0.83832	0.87346	0.96667	0.91763	0.95155	0.77333	0.85277	0.87146	0.86667	0.86579
	0.75	0.82218	0.89333	0.85622	0.78179	0.85333	0.81558	0.85834	0.92667	0.89114	0.91880	0.68000	0.78128	0.84528	0.83833	0.83606
	1	0.83144	0.88667	0.85794	0.74492	0.83333	0.78637	0.86523	0.89333	0.87882	0.85480	0.66667	0.74837	0.82410	0.82000	0.81788
1.5	0	0.83039	0.88000	0.85442	0.81155	0.86000	0.83492	0.88607	0.93333	0.90907	0.91960	0.76000	0.83203	0.86190	0.85833	0.85761
	0.25	0.83019	0.88000	0.85437	0.80631	0.86000	0.83228	0.87584	0.94000	0.90677	0.92497	0.74000	0.82213	0.85933	0.85500	0.85389
	0.5	0.82735	0.89333	0.85903	0.79226	0.84000	0.81524	0.84426	0.94000	0.88955	0.93878	0.70000	0.80178	0.85066	0.84333	0.84140
	0.75	0.84483	0.90667	0.87459	0.78302	0.88667	0.83152	0.86659	0.90667	0.88594	0.92022	0.68667	0.78606	0.85367	0.84667	0.84453
	1	0.84492	0.80000	0.82170	0.68729	0.89333	0.77676	0.81761	0.78667	0.80034	0.82385	0.64667	0.72198	0.79342	0.78167	0.78020
2	0	0.81994	0.88000	0.84889	0.79874	0.84667	0.82201	0.88607	0.93333	0.90907	0.91809	0.74667	0.82352	0.85571	0.85167	0.85087
	0.25	0.81994	0.88000	0.84889	0.80890	0.84667	0.82734	0.88459	0.92000	0.90189	0.90510	0.76000	0.82615	0.85463	0.85167	0.85107
	0.5	0.82043	0.90667	0.86110	0.78093	0.84667	0.81141	0.87369	0.92000	0.89618	0.92123	0.69333	0.79108	0.84907	0.84167	0.83994
	0.75	0.83499	0.90000	0.86549	0.72057	0.84000	0.77446	0.84427	0.86667	0.85498	0.82668	0.60000	0.69406	0.80663	0.80167	0.79725
	1	0.80849	0.84000	0.82358	0.63904	0.84667	0.72818	0.85617	0.74667	0.79671	0.67698	0.51333	0.58343	0.74517	0.73667	0.73298

TABLE 5.9: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `LLMA 2` instruct fine-tuned using *explain-then-annotate* Prompt Template 2 (Fig: 5.4)

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.83974	0.87333	0.85621	0.84382	0.90000	0.87099	0.82190	0.95333	0.88271	0.94545	0.69333	0.79996	0.86273	0.85500	0.85247
	0.25	0.83974	0.87333	0.85621	0.84382	0.90000	0.87099	0.82190	0.95333	0.88271	0.94545	0.69333	0.79996	0.86273	0.85500	0.85247
	0.5	0.83974	0.87333	0.85621	0.84906	0.90000	0.87379	0.81742	0.95333	0.88011	0.94542	0.69333	0.79986	0.86291	0.85500	0.85249
	0.75	0.83556	0.84667	0.84106	0.82828	0.90000	0.86264	0.81723	0.95333	0.87999	0.94539	0.69333	0.79975	0.85661	0.84833	0.84586
	1	0.83571	0.88000	0.85721	0.84277	0.89333	0.86731	0.82982	0.94000	0.88137	0.93812	0.70667	0.80608	0.86160	0.85500	0.85299
0.5	0	0.84013	0.87333	0.85625	0.83333	0.90000	0.86538	0.82128	0.94667	0.87934	0.94494	0.68667	0.79533	0.85992	0.85167	0.84908
	0.25	0.83450	0.87333	0.85341	0.84906	0.90000	0.87379	0.80577	0.94000	0.86771	0.94494	0.68667	0.79533	0.85857	0.85000	0.84756
	0.5	0.83442	0.87333	0.85342	0.84926	0.90000	0.87384	0.82574	0.94667	0.88202	0.94642	0.70667	0.80913	0.86396	0.85667	0.85460
	0.75	0.83002	0.84667	0.83821	0.84986	0.90000	0.87401	0.80119	0.94000	0.86505	0.93782	0.70000	0.80143	0.85472	0.84667	0.84467
	1	0.83465	0.80667	0.82029	0.79048	0.89333	0.83797	0.81572	0.94000	0.87336	0.94722	0.70667	0.80896	0.84702	0.83667	0.83514
1	0	0.83882	0.86667	0.85245	0.83352	0.90000	0.86544	0.82083	0.94667	0.87925	0.94542	0.69333	0.79986	0.85965	0.85167	0.84925
	0.25	0.83442	0.87333	0.85342	0.83857	0.90000	0.86819	0.82083	0.94667	0.87925	0.94494	0.68667	0.79533	0.85969	0.85167	0.84905
	0.5	0.86867	0.87333	0.87064	0.84475	0.90667	0.87460	0.84232	0.96000	0.89718	0.95747	0.74667	0.83894	0.87830	0.87167	0.87034
	0.75	0.87004	0.84667	0.85817	0.80400	0.87333	0.83717	0.82052	0.94000	0.87593	0.92457	0.73333	0.81652	0.85478	0.84833	0.84695
	1	0.85926	0.77333	0.81404	0.74127	0.92667	0.82294	0.77677	0.88000	0.82482	0.87430	0.62000	0.72356	0.81290	0.80000	0.79634
1.5	0	0.84515	0.87333	0.85899	0.84382	0.90000	0.87099	0.82291	0.96000	0.88617	0.94545	0.69333	0.79996	0.86433	0.85667	0.85403
	0.25	0.84414	0.86667	0.85524	0.84382	0.90000	0.87099	0.82184	0.95333	0.88272	0.94642	0.70667	0.80913	0.86405	0.85667	0.85452
	0.5	0.87170	0.86000	0.86555	0.82751	0.89333	0.85901	0.80593	0.94000	0.86776	0.92173	0.70667	0.79997	0.85671	0.85000	0.84807
	0.75	0.83668	0.78667	0.81084	0.75000	0.90000	0.81818	0.77073	0.89333	0.82735	0.89562	0.62667	0.73709	0.81326	0.80167	0.79837
	1	0.83244	0.78667	0.80863	0.69248	0.85333	0.76282	0.72974	0.82667	0.77492	0.80781	0.54667	0.64910	0.76562	0.75333	0.74887
2	0	0.83761	0.86000	0.84864	0.83857	0.90000	0.86819	0.82668	0.95333	0.88549	0.94642	0.70667	0.80913	0.86232	0.85500	0.85286
	0.25	0.83874	0.86667	0.85246	0.83857	0.90000	0.86819	0.82658	0.95333	0.88543	0.94595	0.70000	0.80460	0.86246	0.85500	0.85267
	0.5	0.83683	0.82000	0.82827	0.76624	0.87333	0.81625	0.85445	0.94000	0.89504	0.89804	0.70000	0.78667	0.83889	0.83333	0.83155
	0.75	0.79670	0.80000	0.79784	0.68130	0.82667	0.74683	0.72907	0.84000	0.78036	0.78544	0.49333	0.60505	0.74813	0.74000	0.73252
	1	0.75170	0.66667	0.70576	0.66257	0.80000	0.72366	0.69671	0.80667	0.74260	0.73931	0.53333	0.61870	0.71257	0.70167	0.69768

TABLE 5.10: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `Mistral` instruct fine-tuned using Prompt Template 1 (Fig: 5.3)

Temperature	Top_p	Bugs			Feature			Userexperience			Rating			Macro Avg		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0	0	0.81836	0.90000	0.85719	0.75909	0.84000	0.79748	0.89370	0.94667	0.91920	0.94539	0.69333	0.79975	0.85413	0.84500	0.84341
	0.25	0.81212	0.89333	0.85079	0.75331	0.83333	0.79124	0.89248	0.94000	0.91560	0.94592	0.70000	0.80449	0.85096	0.84167	0.84053
	0.5	0.81543	0.91333	0.86157	0.78285	0.84000	0.81036	0.90007	0.96000	0.92905	0.94595	0.70000	0.80460	0.86107	0.85333	0.85139
	0.75	0.80267	0.89333	0.84529	0.77803	0.84000	0.80764	0.89318	0.94667	0.91909	0.93741	0.70000	0.80146	0.85282	0.84500	0.84337
	1	0.83256	0.89333	0.86176	0.77706	0.86000	0.81641	0.87733	0.95333	0.91374	0.93644	0.68667	0.79230	0.85584	0.84833	0.84605
0.5	0	0.81825	0.90000	0.85713	0.76364	0.84000	0.80000	0.89378	0.95333	0.92258	0.94545	0.69333	0.79996	0.85528	0.84667	0.84492
	0.25	0.81261	0.89333	0.85073	0.76964	0.84667	0.80628	0.89408	0.95333	0.92263	0.94545	0.69333	0.79996	0.85544	0.84667	0.84490
	0.5	0.80915	0.87333	0.83983	0.76836	0.84000	0.80248	0.88770	0.94667	0.91616	0.94734	0.72000	0.81808	0.85314	0.84500	0.84414
	0.75	0.81002	0.88000	0.84345	0.75611	0.84667	0.79879	0.89553	0.91333	0.90428	0.91502	0.70667	0.79709	0.84417	0.83667	0.83590
	1	0.80681	0.86000	0.83244	0.74541	0.84000	0.78978	0.85984	0.89333	0.87611	0.91314	0.70000	0.79237	0.83130	0.82333	0.82268
1	0	0.81846	0.90000	0.85711	0.76413	0.84000	0.80014	0.89308	0.94667	0.91909	0.94592	0.70000	0.80449	0.85540	0.84667	0.84521
	0.25	0.82360	0.90000	0.86000	0.76690	0.83333	0.79863	0.89378	0.95333	0.92258	0.94689	0.71333	0.81365	0.85779	0.85000	0.84872
	0.5	0.80511	0.88000	0.84083	0.77285	0.86000	0.81399	0.89868	0.94000	0.91863	0.94688	0.70667	0.80920	0.85588	0.84667	0.84566
	0.75	0.82906	0.87333	0.85055	0.74228	0.88000	0.80482	0.88401	0.90667	0.89495	0.95508	0.70000	0.80500	0.85261	0.84000	0.83883
	1	0.81092	0.85333	0.83119	0.72597	0.84667	0.78152	0.88876	0.90000	0.89410	0.89881	0.68667	0.77778	0.83112	0.82167	0.82115
1.5	0	0.80463	0.88000	0.84052	0.74886	0.83333	0.78879	0.87037	0.94000	0.90385	0.94392	0.67333	0.78596	0.84194	0.83167	0.82978
	0.25	0.81356	0.87333	0.84228	0.75909	0.84000	0.79748	0.91026	0.94667	0.92810	0.94036	0.73333	0.82395	0.85582	0.84833	0.84795
	0.5	0.82069	0.88000	0.84910	0.76504	0.86000	0.80911	0.87723	0.90000	0.88799	0.93053	0.72000	0.81074	0.84837	0.84000	0.83924
	0.75	0.83837	0.89333	0.86472	0.71015	0.86667	0.78061	0.87487	0.88667	0.88065	0.89568	0.62667	0.73671	0.82977	0.81833	0.81567
	1	0.80187	0.82000	0.80915	0.67014	0.86667	0.75571	0.89350	0.84667	0.86878	0.83939	0.61333	0.70698	0.80123	0.78667	0.78515
2	0	0.81731	0.89333	0.85352	0.76380	0.84000	0.80005	0.89387	0.95333	0.92257	0.94595	0.70000	0.80460	0.85523	0.84667	0.84519
	0.25	0.79411	0.90000	0.84356	0.79331	0.84000	0.81564	0.88700	0.94000	0.91268	0.94666	0.70667	0.80921	0.85527	0.84667	0.84527
	0.5	0.80239	0.81333	0.80770	0.72981	0.86000	0.78930	0.84530	0.91333	0.87795	0.92649	0.67333	0.77947	0.82600	0.81500	0.81361
	0.75	0.83902	0.76667	0.80107	0.64770	0.86667	0.74092	0.82554	0.86667	0.84498	0.82208	0.56667	0.67053	0.78358	0.76667	0.76438
	1	0.84335	0.75333	0.79497	0.58675	0.84000	0.69070	0.76639	0.74000	0.75170	0.72704	0.51333	0.60137	0.73088	0.71167	0.70969

TABLE 5.11: Effects of `Temperature` and `Top_p` on Model Performance Metrics of `Mistral` instruct fine-tuned using *explain-then-annotate* Prompt Template 2 (Fig: 5.4)

5.3.9 Model Performance Visualization

To provide a comprehensive view of our models' performance in classifying mobile app reviews, we present a series of confusion matrices. These visualizations offer insights into the strengths and weaknesses of each model across four categories: Bug, Feature, Rating, and UserExperience.

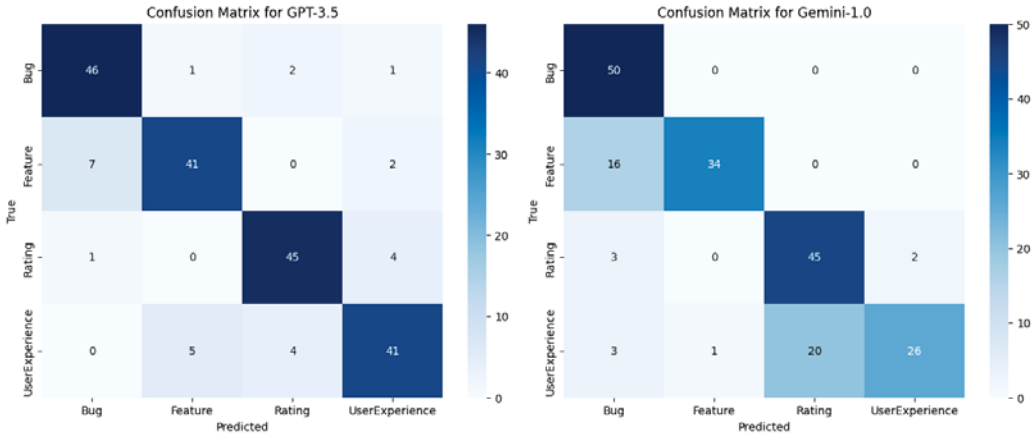


Fig. 5.6: Confusion matrices for `GPT-3.5` (left) and `Gemini-1.0` (right) models in classifying software engineering tasks.

Figure 5.6 presents the confusion matrices for `GPT-3.5` and `Gemini Pro 1.0` models. Both models demonstrate strong overall performance in task categorization. The diagonal elements represent correct classifications, while off-diagonal elements indicate misclassifications. `GPT-3.5` shows slightly better accuracy in the Feature and UserExperience categories, correctly identifying 41 instances in each, compared to `Gemini Pro 1.0`'s 34 and 26 respectively. Conversely, `Gemini Pro 1.0` performs marginally better in Bug identification, correctly classifying 50 instances versus `GPT-3.5`'s 46.

Figure 5.7 displays the confusion matrices for `LLAMA 2` and `Mistral` models. The `LLAMA 2` model demonstrates strong performance, particularly in the Bug and Rating categories, correctly identifying 46 and 45 instances respectively. `Mistral` shows comparable performance, with notably high accuracy in the Feature category, correctly classifying 44 instances. Both models exhibit some misclassifications, particularly between the Feature and UserExperience categories, suggesting potential areas for improvement in distinguishing these review types.

Figure 5.8 presents confusion matrices for the `LLAMA 2` and Mistral models when explanation is enabled during the app review classification task. This allows for comparison of model performance with and without explanations. The `LLAMA 2` model with explanation shows improved performance in the Bug category (46 correct classifications) compared to without explanation, while maintaining strong performance in

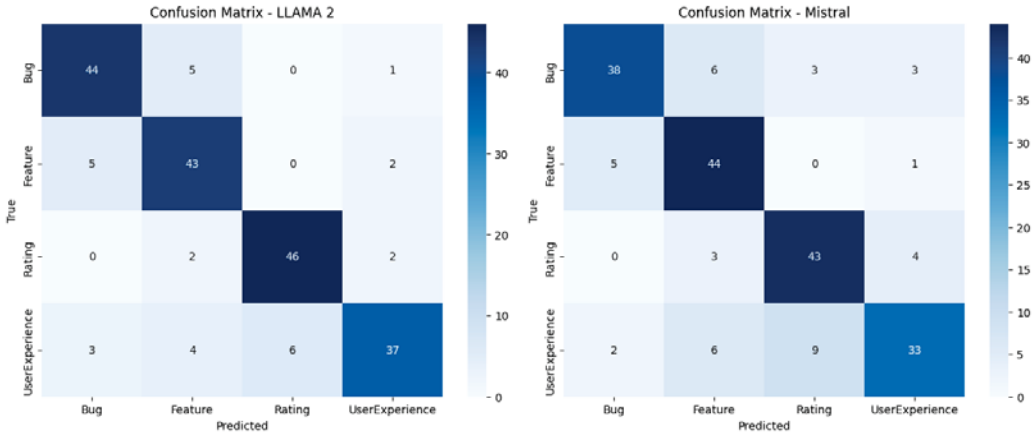


Fig. 5.7: Confusion matrices for `LLAMA 2` (left) and `Mistral` (right) models in classifying app reviews.

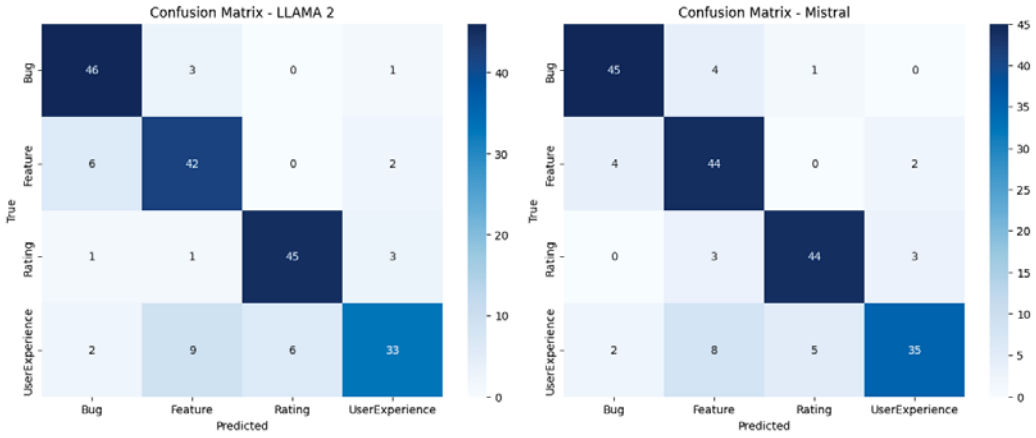


Fig. 5.8: Confusion matrices for `LLAMA 2` (left) and `Mistral` (right) models in classifying app reviews when explanation enabled.

other categories. The `Mistral` model with explanation demonstrates enhanced accuracy in the Bug and Rating categories (45 and 44 correct classifications respectively) compared to its performance without explanation.

These visualizations provide valuable insights into each model's strengths and potential areas for improvement in mobile app review classification. The comparison between models with and without explanations suggests that enabling explanations may lead to improved classification accuracy for certain categories, potentially due to the models' ability to articulate their reasoning process. This observation underscores the potential value of incorporating explanations in enhancing model performance and interpretability in app review classification tasks.

5.4 Conclusion

In this study, we explore the performance of commercial and open-source LLMs for app review classification. We investigated the possibility of using commercially available closed-source models to classify mobile application reviews in zero-shot settings. We further explored the effects of `Temperature` and `Top_p` parameters on classification performance.

Inspired by commercial LLM results, we explored small open-source models for app review classification, comparing them to closed-source alternatives. We used the top-performing commercial model to autonomously generate a large dataset and fine-tune open-source models. Our findings reveal that fine-tuned open-source models exhibited substantial performance gains, offering a cost-effective alternative for app review classification task.

We carried out various experiments to determine the effects of training data size, number of epochs, `Temperature`, and `Top_p` parameters and also measured the quality of generated explanations by fine-tuned LLMs. To encourage other researchers, we have publicly published the related code and dataset[9].

As future work, we plan to investigate the generalizability of our results by experimenting on datasets from multiple domains.

5.5 Limitations

While our study provides valuable insights into the use of Large Language Models (LLMs) for app review classification, it is important to recognize its limitations:

Our investigation concentrated on a small number of commercial (`GPT-3.5`, `Gemini Pro 1.0`) and open-source (`Llama 2`, `Mistral`) LLMs, which may not accurately represent the wide range of available models. The manually annotated benchmark dataset of 200 reviews, while carefully curated, may be considered insufficient for comprehensive evaluation.

Our study focused solely on mobile app reviews published in English on Google Play Store, extracted from top-rated apps in the United States according to appfigures.com[10]. This focus may limit the generalisability of our findings to app reviews from other geographical demographics, multilingual contexts, or app categories not covered by our datasets.

Due to resource and cost constraints, we only considered the smaller 7 billion parameter models. While we can assume that larger model sizes improve performance, more research is needed to confirm this hypothesis.

[9] https://github.com/sadeep25/LLM-Mobile-App-Review-Analyzer.git

[10] https://appfigures.com/top-apps/ios-app-store/united-states/iphone/top-overall

Although we conducted experiments to find prompts better suited to this task in the early stages and chose the best templates for our experiments, a more thorough investigation of prompt engineering techniques could potentially yield even greater performance improvements.

According to Chen et al. [60], the performance of commercial models, such as `GPT-3.5`, can vary significantly over time. As a result, it is critical to understand that the results of this experiment represent a snapshot of each model's performance at the time the experiment was conducted.

Finally, our study did not delve deeply into the computational resources required to fine-tune and run these models. In practical applications, cost, processing time, and energy consumption may be important factors in model selection and deployment.

Despite these limitations, we believe that our research provides useful comparative insights into the performance of commercial and open-source language models for app review classification. Future research can build on these findings while addressing the limitations and advancing the field.

5.6 Ethical considerations

We took several ethical considerations into account when conducting our research on app review classification using language models. First and foremost, we ensured that all app reviews in our dataset were anonymized to protect user privacy. During our experiments, no personally identifiable information was included or processed, ensuring the privacy of app users.

The use of large language models, especially for data annotation, raises serious concerns about bias and fairness. We acknowledge that the `GPT-3.5` model used to generate our training dataset may inherit biases from its training data. These biases have the potential to influence classification results, particularly for under-represented groups or app categories that are less common. Future research should look into ways to detect and mitigate such biases, ensuring equitable results across user groups and app types.

Our study also considered the environmental impact of LLM usage. While we focused on efficiency by determining the minimum effective training dataset size and number of epochs, we acknowledge the need for broader discussions about the energy consumption and carbon footprint of NLP research. The AI community must continue to develop more sustainable practices and technologies to reduce the environmental impact of large-scale language model training and deployment, particularly when fine-tuning models on consumer-grade hardware, as in our study.

We also acknowledge that this technology has the potential for dual use. While our research aims to improve app development and user experience, similar techniques could be used to manipulate app store rankings or spread misinformation. We strongly

encourage future researchers and practitioners to think about these ethical implications and create appropriate safeguards to prevent misuse.

In the spirit of reproducibility and open science, we have made our datasets and code implementations publicly available[11]. This transparency enables peer review and advancement of the field while upholding ethical standards.

Moving forward, we recommend that future research in this field prioritise ethical considerations. This includes creating robust methods for detecting and mitigating bias in language models used for app review classification, investigating more energy-efficient approaches to training and deploying these models, developing guidelines and best practices for responsible use of app review classification technologies, and researching the long-term effects of automated review classification on app ecosystems and user trust.

By addressing these ethical concerns, we hope to contribute to the responsible development and use of language models in app review classification and related fields. As technology evolves, researchers and practitioners must remain vigilant in identifying and addressing any new ethical challenges that may arise.

[11] https://github.com/sadeep25/LLM-Mobile-App-Review-Analyzer.git

CHAPTER 6

FINDINGS

6.1 Aspect-Based Sentiment Analysis on App Reviews

Our first significant contribution in this research was the development of a CNN-based model for Aspect-Based Sentiment Analysis (ABSA) on app reviews. We focused specifically on aspect class classification and aspect sentiment classification, with our work published in the ICTER conference [39].

6.1.1 CNN-based Approach for Aspect-Based Sentiment Analysis

We developed two distinct CNN models: one for aspect category classification and another for aspect sentiment classification. Both models demonstrated substantial improvements over the baseline results provided by the AWARE dataset authors:

- Aspect Category Classification: Our model achieved F1 scores of 0.62, 0.42, and 0.62 for the Productivity, Game, and Social Networking domains respectively, representing improvements of 87.88%, 31.25%, and 93.75% over the baselines.
- Aspect Sentiment Classification: Our model reached accuracy scores of 0.80, 0.70, and 0.86 for the same domains, showing improvements of 16.43%, 3.72%, and 23.35% respectively.

These results underscore the effectiveness of our CNN-based approach in capturing the nuances of app review sentiment and categorization.

6.1.2 Impact of Word Embeddings

We examined three popular word embedding techniques: FastText, GloVe, and Word2Vec. Our findings revealed:

- Word2Vec consistently outperformed the other embeddings across all domains for both tasks.
- For aspect category classification, Word2Vec achieved the highest average F1 score of 0.56.
- In aspect sentiment classification, Word2Vec again led with an average accuracy of 0.79.

These results suggest that Word2Vec's ability to capture semantic relationships between words is particularly beneficial for app review analysis.

6.1.3 Effectiveness of Data Oversampling Techniques

To address the class imbalance in the AWARE dataset, we experimented two oversampling techniques:

- Contextual augmentation using Google BERT
- Round-trip translation (RTT) with four languages: German (DE), Turkish (TR), Japanese (JP), and Chinese (CN)

Our analysis showed:

- RTT with German (DE) as the intermediate language yielded the best results for both tasks.
- For aspect category classification, RTT (DE) improved the average F1 score by 2% compared to no oversampling.
- In aspect sentiment classification, RTT (DE) increased the average accuracy by 1.5%.

These findings highlight the potential of linguistic-based data augmentation techniques in improving model performance, particularly for imbalanced datasets.

6.1.4 Hyperparameter Tuning

We fine-tuned three key hyperparameters: batch size, number of epochs, and learning rate. Our experiments revealed:

- Optimal batch sizes: 25 for aspect category classification and 35 for aspect sentiment classification.
- Ideal number of epochs: 75 for both tasks.
- Most effective learning rate: 0.001 for both models.

These optimized parameters contributed significantly to the overall performance improvements of our models.

6.2 Automatic Analysis of App Reviews Using LLMs

As our next step we explored the potential of Large Language Models (LLMs) for app review analysis. This ongoing work is currently under review for COLING 2025.

6.2.1 Evaluation of Popular Commercial Models

We conducted a comparative study of two prominent commercial LLMs: Google's `Gemini Pro 1.0` and OpenAI's `GPT-3.5`, in a zero-shot setting. Our findings showed:

- `GPT-3.5` outperformed `Gemini Pro` with an F1 score of 0.84917 compared to `Gemini Pro`'s 0.75490.
- Both models demonstrated sensitivity to `Temperature` and `Top_p` parameters, with optimal performance generally achieved at lower values (`Temperature` 0.5, `Top_p` 0.25).

These results underscore the potential of commercial LLMs in app review classification tasks, even without task-specific fine-tuning.

6.2.2 LLMs as Autonomous Annotators

Leveraging the superior performance of GPT-3.5, we used it as an autonomous annotator to create a balanced dataset of 10,000 app reviews. Our quality assessment revealed:

- A random sample of 370 reviews (95% confidence level, 5% margin of error) was manually reviewed.
- Inter-annotator agreement reached an "almost perfect" level with an average Cohen's Kappa score of 0.9135.
- The GPT-3.5-generated annotations achieved an accuracy of 81.89% according to the results of the manual review by human annotators.

These findings suggest that LLMs can be effectively used to create large, high-quality datasets for app review analysis, potentially reducing the need for extensive manual annotation.

6.2.3 Fine-tuning and Evaluating Open-source LLMs

We evaluated the performance of open-source LLMs (`LLAMA 2`, `Mistral`, and `Falcon`) on the app review classification task. Key findings include:

- Base models showed lower performance compared to commercial LLMs, with F1 scores of 0.57753 for `LLAMA 2` and 0.47060 for `Mistral`.
- Fine-tuned models achieved performance comparable to commercial LLMs, with F1 scores of 0.83416 for `LLAMA 2` and 0.79634 for `Mistral`.

- The *"explain-then-annotate"* prompting strategy improved performance for Mistral, achieving an F1 score of 0.82115.
- `Temperature` and `Top_p` parameters showed similar effects as observed in commercial models, with best performance at lower values.
- Manual evaluation of explanations generated by the *"explain-then-annotate"* approach showed high quality, with 76.79% rated as comprehensive and correct.

These results demonstrate the potential of fine-tuned open-source LLMs as a cost-effective alternative to commercial models for app review analysis tasks.

Our findings collectively highlight the evolving landscape of app review analysis techniques, from traditional machine learning approaches to the application of state-of-the-art large language models. They underscore the potential for significant improvements in accuracy and efficiency in extracting valuable insights from user feedback.

CHAPTER 7

DISCUSSION AND CONCLUSION

First, we developed a CNN-based approach for Aspect-Based Sentiment Analysis (ABSA) on app reviews (see Chapter 4), which demonstrated substantial improvements over existing baselines. For aspect category classification, we achieved F1 scores of 0.62, 0.42, and 0.62 for Productivity, Game, and Social Networking domains respectively, representing improvements of up to 93.75% over baselines. In aspect sentiment classification, our model reached accuracy scores of 0.80, 0.70, and 0.86 for the same domains, showing improvements of up to 23.35% (see Section 4.4.3). These results underscore the effectiveness of our CNN-based approach in capturing the nuances of app review sentiment and categorization.

Then, we investigated the impact of different word embedding techniques and data augmentation methods on our ABSA model (see Section 4.3). We found that Word2Vec consistently outperformed FastText and GloVe across all domains and tasks, suggesting its superior ability to capture semantic relationships in app review contexts. We also demonstrated the effectiveness of data augmentation techniques, particularly Round-trip Translation (RTT) with German as an intermediate language, in improving model performance on imbalanced datasets.

Further, we explored the potential of Large Language Models (LLMs) for app review analysis (see Chapter 5). We evaluated both commercial and open-source LLMs in zero-shot and fine-tuned settings. Commercial LLMs like GPT-3.5 demonstrated high performance in zero-shot settings, achieving an F1 score of 0.84917 (see Section 5.3.1). We successfully used GPT-3.5 as an autonomous annotator to create a large, high-quality dataset of 10,000 app reviews, with an accuracy of 81.89% according to the manual review done on the dataset by human annotators (see Section 5.3.3). This approach offers a promising method for creating extensive, reliable datasets for training specialized models, potentially reducing the need for manual annotation.

We then fine-tuned open-source LLMs and found they could achieve performance comparable to commercial models, with `LLAMA 2` reaching an F1 score of 0.83416 (see Section 5.3.4). We utilized *"explain-then-annotate"* prompting strategy, which improved performance for some models, with Mistral achieving an F1 score of 0.82115 using this approach (see Section 5.2.4.2). These results demonstrate the potential of both commercial and open-source LLMs in automating and improving the app review analysis process.

Our research has several important implications for the field of mobile app development and user feedback analysis. The improved accuracy in aspect-based sentiment analysis can help developers prioritize issues and feature requests more effectively, leading to better-informed development decisions. The use of LLMs as autonomous

annotators offers a cost-effective approach to creating large, high-quality datasets. The comparable performance of fine-tuned open-source LLMs to commercial models suggests a viable alternative for app review analysis, especially beneficial for smaller development teams or academic researchers.

Without being confined to a single app category or language, we verified our experimental results across multiple domains and languages. Our ABSA model was tested on Productivity, Game, and Social Networking app categories (see Section 4.3), while our LLM experiments covered a wide range of app types. This broad applicability enhances the generalizability of our findings and their potential impact on the mobile app development ecosystem.

In conclusion, this research has demonstrated the potential of advanced machine learning and natural language processing techniques in revolutionizing app review analysis. By improving the accuracy and efficiency of extracting insights from user feedback, our work contributes to the ongoing advancements of mobile app review analysis and potentially helps mobile app developers in the process of mobile application evolution. As the field continues to advance, the integration of these technologies promises to enhance the responsiveness of app developers to user needs, ultimately leading to improved mobile experiences for users worldwide. Future research could explore the integration of our ABSA and LLM-based approaches for more comprehensive app review analysis, develop methods to detect and mitigate biases in LLM-based systems, and investigate the long-term impact of automated review analysis on app quality and user satisfaction.

REFERENCES

[1] "Number of smartphone users worldwide," https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[2] L. V. G. Carreno and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *2013 35th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 582–591.

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

[4] D. Pagano and B. Bruegge, "User involvement in software evolution practice: a case study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 953–962.

[5] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *2012 9th IEEE working conference on mining software repositories (MSR)*. IEEE, 2012, pp. 108–111.

[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*, 2014, pp. 767–778.

[7] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Software*, vol. 137, pp. 143–162, 2018.

[8] E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (n)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 771–776.

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

[10] E. Guzman, M. Ibrahim, and M. Glinz, "A little bird told me: Mining tweets for requirements and software evolution," 09 2017.

[11] N. Alturaief, H. Aljamaan, and M. Baslyman, "Aware: Aspect-based sentiment analysis dataset of apps reviews for requirements elicitation," in *2021 36th*

IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). IEEE, 2021, pp. 211–218.

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

[13] H. Li, L. Zhang, L. Zhang, and J. Shen, "A user satisfaction analysis approach for software evolution," vol. 2, pp. 1093–1097, 2010.

[14] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE software*, vol. 33, no. 1, pp. 48–54, 2015.

[15] M. V. Phong, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," pp. 749–759, 2015.

[16] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," pp. 1276–1284, 2013.

[17] R. T. Anchiêta and R. S. Moura, "Exploring unsupervised learning towards extractive summarization of user reviews," in *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, 2017, pp. 217–220.

[18] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier, "A recommender system of buggy app checkers for app store moderators," pp. 1–11, 2015.

[19] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[20] X. Gu and S. Kim, ""' what parts of your apps are loved by users?"(t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 760–770.

[21] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah, "App review analysis via active learning: reducing supervision effort without compromising classification accuracy," in *2018 IEEE 26th international requirements engineering conference (RE)*. IEEE, 2018, pp. 170–181.

[22] H. Guo and M. P. Singh, "Caspar: Extracting and synthesizing user stories of problems from app reviews," 2020, p. 628–640.

[23] C. Stanik, M. Haering, and W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in *2019 IEEE 27th international requirements engineering conference workshops (REW)*. IEEE, 2019, pp. 220–226.

[24] N. Aslam, W. Y. Ramay, K. Xia, and N. Sarwar, "Convolutional neural network based classification of app reviews," *IEEE Access*, vol. 8, pp. 185 619–185 628, 2020.

[25] M. A. Hadi and F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews," 2021.

[26] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2021, pp. 80–86.

[27] J. Verma and A. Patel, "Evaluation of unsupervised learning based extractive text summarization technique for large scale review and feedback data," *Indian Journal of Science and Technology*, vol. 10, pp. 1–6, 05 2017.

[28] S. Wang, Y. Liu, Y. Xu, C. Zhu, and M. Zeng, "Want to reduce labeling cost? gpt-3 can help," *arXiv preprint arXiv:2108.13487*, 2021.

[29] X. He, Z. Lin, Y. Gong, A. Jin, H. Zhang, C. Lin, J. Jiao, S. M. Yiu, N. Duan, W. Chen *et al.*, "Annollm: Making large language models to be better crowdsourced annotators," *arXiv preprint arXiv:2303.16854*, 2023.

[30] R. Zhang, Y. Li, Y. Ma, M. Zhou, and L. Zou, "Llmaaa: Making large language models as active annotators," *arXiv preprint arXiv:2310.19596*, 2023.

[31] J. Zhou, W. Du, M. O. F. Rokon, Z. Wang, J. Xu, I. Shah, K.-c. Lee, and M. Wen, "Enhanced e-commerce attribute extraction: Innovating with decorative relation correction and llama 2.0-based annotation," *arXiv preprint arXiv:2312.06684*, 2023.

[32] Z. He, C.-Y. Huang, C.-K. C. Ding, S. Rohatgi, and T.-H. K. Huang, "If in a crowdsourced data annotation pipeline, a gpt-4," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–25.

[33] Y. Tang, C.-M. Chang, and X. Yang, "Pdfchatannotator: A human-llm collaborative multi-modal data annotation tool for pdf-format catalogs," in *Proceedings of the 29th International Conference on Intelligent User Interfaces*, 2024, pp. 419–430.

[34] D. Yu, L. Li, H. Su, and M. Fuoli, "Assessing the potential of llm-assisted annotation for corpus-based pragmatics and discourse analysis: The case of apology," *International Journal of Corpus Linguistics*, 2024.

[35] M. Imamovic, S. Deilen, D. Glynn, and E. Lapshinova-Koltunski, "Using chatgpt for annotation of attitude within the appraisal theory: Lessons learned," in *Proceedings of The 18th Linguistic Annotation Workshop (LAW-XVIII)*, 2024, pp. 112–123.

[36] A. Wang, J. Morgenstern, and J. P. Dickerson, "Large language models cannot replace human participants because they cannot portray identity groups," *arXiv preprint arXiv:2402.01908*, 2024.

[37] N. Pangakis, S. Wolken, and N. Fasching, "Automated annotation with generative ai requires validation," *arXiv preprint arXiv:2306.00176*, 2023.

[38] J. Tan, A. Zhang, X. Zhang, C. Xiao, Z. Ding, Y. Peng, C. Wu, X. Zhu, J. Zhou, and X. Huang, "Large language models for data annotation: A survey," *arXiv preprint arXiv:2402.13446*, 2024.

[39] S. Gunathilaka and N. De Silva, "Aspect-based sentiment analysis on mobile application reviews," in *2022 22nd International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, 2022, pp. 183–188.

[40] S. Kobayashi, "Contextual augmentation: Data augmentation by words with paradigmatic relations," 2018.

[41] 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.

[42] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[43] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90

[44] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, E. Goffinet, D. Heslow, J. Launay, Q. Malartic *et al.*, "Falcon-40b: an open large language model with state-of-the-art performance," 2023.

[45] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[46] L. Reiter, "Zephyr," *Journal of Business Finance Librarianship*, vol. 18, no. 3, pp. 259–263, 2013.

[47] G. Colavito, F. Lanubile, N. Novielli, and L. Quaranta, "Leveraging gpt-like llms to automate issue labeling," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 2024, pp. 469–480.

[48] 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)*. IEEE, 2015, pp. 116–125.

[49] D. Yu, L. Li, H. Su, and M. Fuoli, "Assessing the potential of llm-assisted annotation for corpus-based pragmatics and discourse analysis."

[50] K. Hamilton, L. Longo, and B. Bozic, "Gpt assisted annotation of rhetorical and linguistic features for interpretable propaganda technique detection in news text." in *Companion Proceedings of the ACM on Web Conference 2024*, 2024, pp. 1431–1440.

[51] T. Zhang, I. C. Irsan, F. Thung, and D. Lo, "Revisiting sentiment analysis for software engineering in the era of large language models," *arXiv preprint arXiv:2310.11113*, 2023.

[52] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language models with self-generated instructions," *arXiv preprint arXiv:2212.10560*, 2022.

[53] R. R. Mekala, Y. Razeghi, and S. Singh, "Echoprompt: Instructing the model to rephrase queries for improved in-context learning," 2024. [Online]. Available: https://arxiv.org/abs/2309.10687

[54] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. Kroiz, F. Li, H. Tao, A. Srivastava, H. D. Costa, S. Gupta, M. L. Rogers, I. Goncearenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle, and P. Resnik, "The prompt report: A systematic survey of prompting techniques," 2024.

[55] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

[56] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "Peft: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022.

[57] R. V. Krejcie and D. W. Morgan, "Determining sample size for research activities," *Educational and psychological measurement*, vol. 30, no. 3, pp. 607–610, 1970.

[58] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." *Psychological bulletin*, vol. 70, no. 4, p. 213, 1968.

[59] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[60] L. Chen, M. Zaharia, and J. Zou, "How is chatgpt's behavior changing over time?" *arXiv preprint arXiv:2307.09009*, 2023.