Automatic Analysis of App Reviews Using LLMs

Sadeep Gunathilaka and Nisansa de Silva
Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka
{*sadeep.21, nisansadds*}*@cse.mrt.ac.lk*

Keywords: App Review Analysis, Large Language Models, Mobile Applications, Software Evolution, Natural Language Processing, Review Classification, GPT-3.5, Zero-Shot Learning, Fine-Tuning, Open-Source Models.

Abstract: Large Language Models (LLMs) have shown promise in various natural language processing tasks, but their effectiveness for app review classification to support software evolution remains unexplored. This study evaluates commercial and open-source LLMs for classifying mobile app reviews into bug reports, feature requests, user experiences, and ratings. We compare the zero-shot performance of `GPT-3.5` and `Gemini Pro 1.0`, finding that `GPT-3.5` achieves superior results with an F1 score of 0.849. We then use `GPT-3.5` to autonomously annotate a dataset for fine-tuning smaller open-source models. Experiments with `Llama 2` and `Mistral` show that instruction fine-tuning significantly improves performance, with results approaching commercial models. We investigate the trade-off between training data size and the number of epochs, demonstrating that comparable results can be achieved with smaller datasets and increased training iterations. Additionally, we explore the impact of different prompting strategies on model performance. Our work demonstrates the potential of LLMs to enhance app review analysis for software engineering while highlighting areas for further improvement in open-source alternatives.

1 INTRODUCTION

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. A number of researches have highlighted the importance of user participation in the software development process (Hosseini et al., 2015; Groen et al., 2017; Maalej et al., 2015). After the deployment of an application, user feedback becomes a vital asset for ongoing enhancement (Pagano and Maalej, 2013). Pagano and Maalej (2013) found that the frequency of feedback reaches its highest point after new releases. This feedback encompasses a wide range of topics, such as user experience concerns and requests for additional features. Li et al. (2010) presented evidence that the analysis of user comments can improve overall user satisfaction with software solutions.

Nevertheless, the overwhelming volume and the unstructured nature of the user reviews pose considerable difficulties for manual analysis (Vu et al., 2015). Accordingly, scholars have investigated many techniques under Natural Language Processing (NLP) and Machine Learning to automate the extraction of information from user reviews. These approaches span supervised methods, such as classification (Di Sorbo et al., 2016; Maalej et al., 2016; Carreno and Winbladh, 2013; Stanik et al., 2019; Hadi and Fard, 2021; Gunathilaka and De Silva, 2022), and unsupervised methods, such as clustering and topic modelling (Chen et al., 2014; Guzman and Maalej, 2014).

Effective supervised approaches frequently require labelled large datasets, which can be resource-intensive to generate (Dhinakaran et al., 2018). The existing difficulty has prompted researchers to investigate alternative approaches, such as the utilisation of Large Language Models (LLMs) like GPT (Radford et al., 2018, 2019; Brown et al., 2020). These models have shown proficient performance in a wide range of software engineering activities (Hou et al., 2023).

Contemporary research has examined the capacity of Large Language Models (LLMs) to optimise data annotation procedures (Yu et al., 2024; He et al., 2023; Ding et al., 2022) and to produce synthetic data (Møller et al., 2023). Building on these advancements, our research intends to utilise Large Language Models (LLMs) to analyse user reviews on mobile application platforms. This will assist developers in in-

Gunathilaka, S. and de Silva, N.
Automatic Analysis of App Reviews Using LLMs.
DOI: 10.5220/0013375600003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 2*, pages 828-839
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

tegrating user feedback into their requirements engineering and software evolution processes.

In pursuit of achieving the above goal, we investigate the following research questions:

1. How effective are commercial and open-source models in classifying user reviews?
2. Can we utilize commercial models to autonomously annotate datasets for fine-tuning smaller open-source models, thereby developing a cost-effective LLM for app review classification?

In order to answer these questions, We evaluated well-known commercial models using carefully designed prompts in a zero-shot setting by benchmarking their performance against a manually annotated dataset. Subsequently, we used the best-performing commercial model to automatically annotate a balanced dataset, which was then used to fine-tune smaller open-source models. Lastly, we evaluated the performance of these fine-tuned models in comparison to their original forms as well as the aforementioned commercially available models.

2 RELATED WORK

2.1 Large Language Models in Data Annotation

Recent advances in Large Language Models (LLMs) have transformed many aspects of natural language processing, including data annotation. The *explain-then-annotate* method (He et al., 2023) and `GPT-4`'s performance in annotating complex pragma-discursive elements (Yu et al., 2024) show that LLMs have the potential to match or surpass human performance. Researchers have also investigated LLMs' capabilities in generating paraphrases for annotation tasks (Cegin et al., 2023) and optimizing annotation efficiency (Pérez et al., 2023).

LLMs have been applied to various specialised annotation tasks, including rhetorical feature annotation (Hamilton et al., 2024) and multilingual data generation (Choi et al., 2024). Frameworks such as LLMAAA (Zhang et al., 2023a) and Generative Annotation Assistants (Bartolo et al., 2021) incorporate LLMs into active learning loops to improve data annotation. Open-source LLMs have been shown to outperform human-based services in text annotation tasks (Alizadeh et al., 2023).

These advancements have led to significant improvements in annotation efficiency, accuracy, and cost-effectiveness (Wang et al., 2021), highlighting the versatility and potential of LLMs to enhance various aspects of data annotation.

2.2 Mobile App Review Classification: Traditional Approaches

The importance of user feedback in app development and maintenance has been well-established (Pagano and Maalej, 2013; Maalej et al., 2015). The challenges associated with the manual analysis of large volumes of reviews led researchers to explore employing various NLP and machine learning techniques.

Early studies leveraged traditional machine learning methods to extract valuable information from user comments. Approaches included the use of linear regression and LDA (Fu et al., 2013), topic modelling and classification (Chen et al., 2014; Anchiêta and Moura, 2017), and keyword-based analysis (Vu et al., 2015).

Supervised learning approaches gained traction later, with studies comparing individual machine learning algorithms and their ensembles (Guzman et al., 2015). Maalej et al. (2016) achieved high classification precision and recall in categorizing reviews into bug reports, feature requests, user experiences, and text ratings.

More advanced techniques were developed to address specific challenges in app review analysis. Guzman et al. (2017) expanded the analysis beyond app stores to include social media data. Dhinakaran et al. (2018) proposed active learning to reduce the human effort required for labelling training data. Guo and Singh (2020) introduced a method for extracting and synthesizing user-reported mini-stories about app problems from reviews.

These traditional approaches laid the foundation for understanding and categorizing user feedback in mobile app development, paving the way for more sophisticated techniques using Deep learning approaches.

2.3 Mobile App Review Classification: Deep Learning Approaches

Recent studies have increasingly applied deep learning to classify user feedback in mobile app reviews. Stanik et al. (2019) found that Deep Convolutional Neural Networks (CNNs) outperformed traditional methods in multilingual user feedback classification. Aslam et al. (2020) proposed a deep learning approach leveraging both textual and non-textual information for classifying reviews into various categories. Gunathilaka and De Silva (2022) developed a deep learning approach for conducting aspect-based sentiment analysis on app reviews using CNNs. Pre-trained models such as BERT have shown promise in this domain. Hadi and Fard (2021) demonstrated that BERT models performed well across various settings, while Restrepo et al. (2021) found that monolingual BERT models outperformed baseline methods in transfer learning for user review classification.

The literature reveals a progression from traditional machine learning to deep learning and LLM-based methods in analyzing user reviews. Despite this evolution, a notable gap remains in applying recent LLMs, such as `GPT` and `Gemini` models, for mobile app review analysis to support application evolution.

This study bridges this gap through several contributions. We evaluate commercial LLMs (`GPT`-3.5 and `Gemini Pro`) for app review classification, establishing performance benchmarks and optimal parameters. We introduce a method for creating training datasets using commercial LLMs as autonomous annotators, reducing manual effort while maintaining quality.

We compare open-source LLMs (`Llama 2` and `Mistral`) against commercial alternatives, examining fine-tuning approaches, hyperparameters tunning (`Temperature` and `Top_p`), and training dynamics with varying dataset sizes (500-10000 samples) and epochs. Our experiments show smaller datasets with increased epochs achieve comparable results to larger datasets. We also explore prompting strategies, including an *”explain-then-annotate”* approach, evaluating their impact on classification performance.

By leveraging the advanced capabilities of LLMs, this research seeks to provide a more efficient, accurate, and scalable method for extracting actionable insights from user feedback, potentially enhancing the mobile application evolution process by a significant amount.

3 METHODOLOGY

Our approach to classifying app store reviews using Large Language Models (LLMs) consists of several key steps, as illustrated in Figure 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.

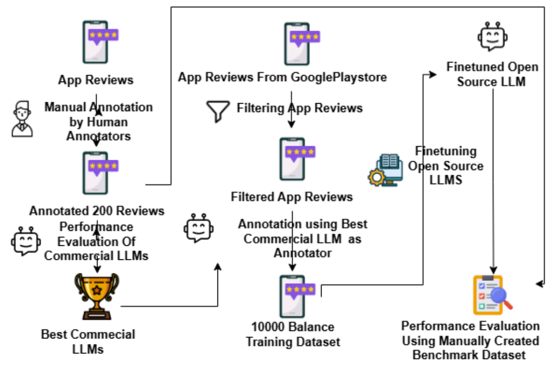

App Reviews
Manual Annotation by Human Annotators
Annotated 200 Reviews
Performance Evaluation Of Commercial LLMs
Best Commecial LLMs
App Reviews From GooglePlaystore
Filtering App Reviews
Filtered App Reviews
Annotation using Best Commercial LLM as Annotator
10000 Balance Training Dataset
Finetuned Open Source LLM
Finetuning Open Source LLMS
Performance Evaluation Using Manually Created Benchmark Dataset

Figure 1: A high-level overview of our approach.

3.1 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 3.1.1 and Section 3.1.2 respectively discuss how we selected the appropriate LLMs for our experiments in detail.

3.1.1 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. Our initial experiments showed that advanced models offered only marginal performance improvements at substantially higher costs. `GPT-3.5` was approximately 45 times more cost-effective than `GPT-4`, while `Gemini Pro 1.0` was three times cheaper than its advanced counterpart `Gemini Pro 1.5`. This cost advantage enabled us to conduct more comprehensive experiments while maintaining acceptable performance. We employed a zero-shot setting to minimize context size and costs, utilizing the annotation prompt explained in Section 3.4.1 via respective API endpoints.

[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

3.1.2 LLM Selection for Custom Models

With the introduction of open-source LLMs, including `LLaMA` (Touvron et al., 2023a,b), `Vicuna` (Chiang et al., 2023), `Falcon` (Almazrouei et al., 2023), `Mistral` (Jiang et al., 2023), and `Zephyr` (Reiter, 2013), 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 (He et al., 2023; Colavito et al., 2024). We also examined the effects of `Temperature` and `Top_p` parameters on model performance.

3.2 Benchmarking Dataset

This study utilizes a subset of a dataset from Maalej and Nabil (2015). 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.
- **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 reviews per category. For reviews potentially belonging to multiple categories, we applied a precedence order (Bug > Feature > User Experience > Rating) based on their impact on software evolution. This hierarchy prioritizes bugs as critical production issues, followed by feature requests for future development cycles, user experience feedback for existing features, and finally ratings which provide minimal actionable insights.

Three developers with over 5 years of experience annotated the dataset using provided class definitions, examples, and guidelines. Final classifications were determined by majority vote, with disagreements resolved through group discussions.

```
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:
```

[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

```
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?: <True or False>
Q2.Explain why Q1 is True/False: <explanation>
Q3.Does it sound like a missing Feature?: <True or False>
Q4.Explain why Q3 is True/False: <explanation>
Q5.Does it sound like a User Experience?: <True or False>
Q6.Explain why Q5 is True/False: <explanation>
Q7.Does it sound like a Rating?: <True or False>
Q8.Explain why Q7 is True/False: <explanation>
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?": "<True or False>",
   "Q2.Explain why Q1 is True/False": "<explanation>",
   "Q3.Does it sound like a missing Feature?": "<True or
         False>",
   "Q4.Explain why Q3 is True/False": "<explanation>",
   "Q5.Does it sound like a User Experience?": "<True or
         False>",
   "Q6.Explain why Q5 is True/False": "<explanation>",
   "Q7.Does it sound like a Rating?": "<True or False>",
   "Q8.Explain why Q7 is True/False": "<explanation>"
}
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"
```

Prompt Block 1: Prompt template: Annotating app review data.

3.3 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 4.1. 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 3.4.1 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.

```
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
```

[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

```
     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
```

Prompt Block 2: Template 1: App review classification prompt for open-source models.

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

Prompt Block 3: Template 2: App review classification prompt for open-source models.

3.4 Selection of Prompts

Prompt quality significantly influences model performance (Zhang et al., 2023b). Drawing inspiration from existing work (He et al., 2023; Colavito et al., 2024; Wang et al., 2022; Mekala et al., 2024; Schulhoff et al., 2024), we developed three different prompt templates for our experiments. Section 3.4.1 describes the prompt we used for benchmark and annotate app reviews and Section 3.4.2 describes the prompt templates that we used for instruct fine-tune and benchmark our fine-tuned models.

3.4.1 Prompts for Annotation

For benchmark classification performance and annotate app reviews using commercial LLMs we developed a prompt structure that encourages comprehensive class consideration. Our annotation prompt contains a series of boolean questions and explanations for each class as depicted by Prompt Block: 1. We developed this prompt to avoid scenarios where reviews belongs to multiple labels simply get classified to one label when the model trying to classify them. This structure allows flexible reasoning, particularly for multi-category reviews. Post-classification, we apply a precedence order (e.g., Bug > Feature > User Experience > Rating) for final categorization. For example, if the LLM returns True for both Q3 and Q5 in the preprocessing stage, we classify the review as a Feature request, following our defined precedence order.

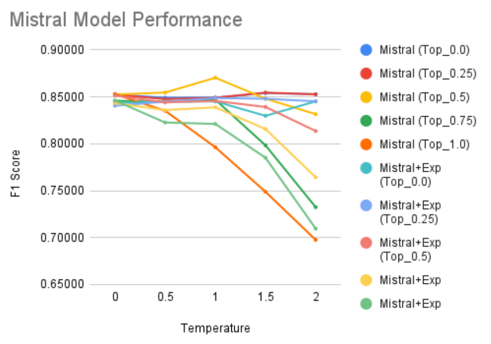
3.4.2 Prompts for Fine-Tuning Custom Models

We developed two primary templates for fine-tuning: `Template 1` (Prompt Block: 2) and `Template 2`, with their main difference in the ”Instructions to the Language Model” section (shown in Prompt Block: 3). Both templates share identical task descriptions, classification definitions, and example formats, but `Template 2` follows the *”explain-then-annotate”* pattern (He et al., 2023). This is implemented through modified instructions requiring the model to provide an explanation before classification, as illustrated in Prompt Block: 3. The difference is reflected in their JSON output formats: `Template 1` outputs only the classification class, while `Template 2` requires both an explanation and the class prediction. 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.

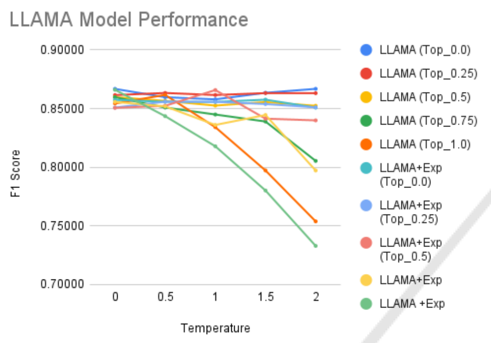
3.5 Fine-Tuning Open Source Models

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

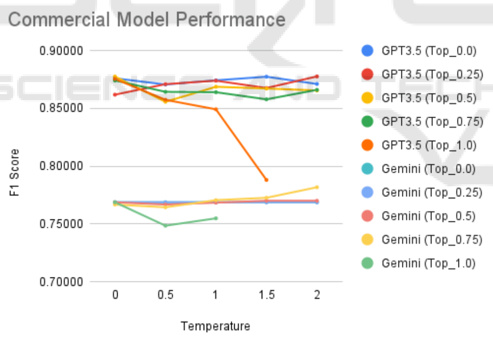
- 4-bit quantization via `QLoRA` (Dettmers et al., 2023), enabling efficient fine-tuning while maintaining high performance.



(a) `Mistral` model performance across different `Temperature` settings and `Top_P` settings.



(b) `LLAMA` model performance across different `Temperature` settings and `Top_P` settings.



(c) Commercial models (`GPT-3.5` and `Gemini Pro`) performance comparison across different `Temperature` settings and `Top_P` settings.

Figure 2: Performance comparison of different language models across varying `Temperature` settings (0-2.0) and sampling parameters. All models demonstrate performance variations with `Temperature` changes, with notable differences in stability across different `Top-p` values (0.0-1.0). (a) `Mistral` model exhibits varying performance patterns with different exponential variants. (b) `LLAMA` model shows similar parameter sensitivity with distinct stability characteristics. (c) Commercial models display unique performance patterns compared to open-source alternatives, particularly in high-`Temperature` regions. The y-axis represents model performance metrics, while the x-axis shows `Temperature` values from 0 to 2.0. Key observations include optimal performance at lower `Temperatures` (0-0.5) and general performance degradation at higher `Temperatures` (>1.5).

- `PEFT` (Mangrulkar et al., 2022) (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.

3.6 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 (Maalej and Nabil, 2015).

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 (Krejcie and Morgan, 1970).

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.

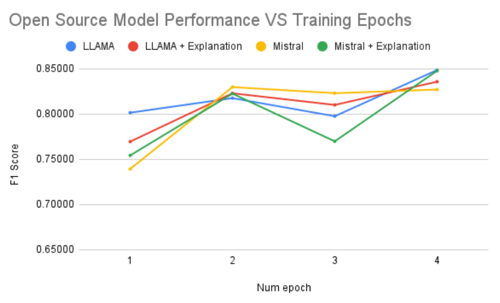
As we utilized an *"explain-then-annotate"* pattern in one of the prompt templates used to train the open-source LLMs, we also manually reviewed and evaluated the accuracy of the generated explanations. For this evaluation, we considered only the correctly classified instances and assessed the accuracy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

4 EXPERIMENTS

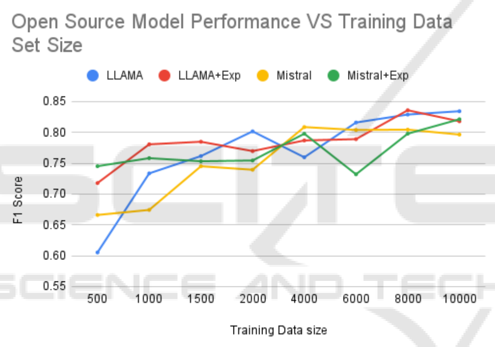
We divided our experiments into two main parts. First, we tested commercially available closed-source models to see how well they could classify app reviews. We tested two well-known Large Language Models (LLMs) in different settings. Then, we used

Table 1: Model Performance Comparison Including `Gemini Pro` and `GPT 3.5`.

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



(a) Model performance across training epochs for `LLAMA` and `Mistral`, both with and without explanation capabilities.



(b) Model performance scaling with training data size, comparing base models and their explanation-enhanced variants.

Figure 3: Training dynamics analysis of open-source language models. (a) Performance evolution across training epochs shows consistent improvement patterns, with both `LLAMA` and `Mistral` achieving F1 scores above 0.80 by epoch 4. Models with explanation capabilities demonstrate comparable learning trajectories to their base variants. (b) Impact of training data size on model performance, ranging from 500 to 10000 samples. All models show significant improvements with increased data, with initial performance variations converging at larger dataset sizes. The explanation-enhanced versions (LLAMA+Exp and Mistral+Exp) maintain competitive performance across different data regimes, suggesting effective integration of explanation capabilities without compromising base model performance.

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 4.1 describes our experiments with commercial LLMs in detail. Section 4.2 explains our work with open-source LLMs.

4.1 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 3.4.1. As presented in Table 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.

Subsequently, we investigated the impact of two performance-tuning parameters provided by both models: `Temperature` and `Top_p`. Figure 2c shows the performance variations of commercial models (`GPT-3.5` and `Gemini Pro 1.0`) across various `Temperature` and `Top_p` settings.

Our findings indicate that parameter tuning can enhance model performance. `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, suggest that this could be a potential caching effect or model-specific issue.

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, achieving a 95% confidence level with a 5% margin of error based on (Krejcie and Morgan, 1970) guidelines. Three experienced developers served as annotators, and we calculated the inter-annotator agreement using Cohen's Kappa (κ). The κ values between annotator pairs ranged from 0.9079 to 0.9180, with an average of 0.9135. For interpreting κ values, we used the following scale by (Landis and Koch, 1977): $\kappa \leq 0$ (less than chance agreement), $0.01 \leq \kappa \leq 0.20$ (slight), $0.21 \leq \kappa \leq 0.40$ (fair), $0.41 \leq \kappa \leq 0.60$ (moderate), $0.61 \leq \kappa \leq 0.80$ (substantial), and $0.81 \leq \kappa \leq 1$ (almost perfect agreement). According to this scale, our scores indicate "almost perfect" agreement, as they exceed the 0.81 threshold.

Tables 2 and 3 provide detailed insights into the annotation analysis. Table 2 presents the pairwise Cohen's Kappa scores, demonstrating strong consistency across all three annotators. Table 3 shows the distribution of annotations across four categories (Bug, Feature, Rating, and UserExperience), revealing similar classification patterns among annotators that further validate the high inter-annotator agreement. Based on this rigorous evaluation process, the GPT-3.5 annotated dataset achieved an average accuracy of 81.89%.

Table 2: Pairwise Cohen's Kappa Scores and Agreement Levels.

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 3: Annotation Distribution by Annotator and Category.

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

4.2 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 1 presents the base models' performance results.

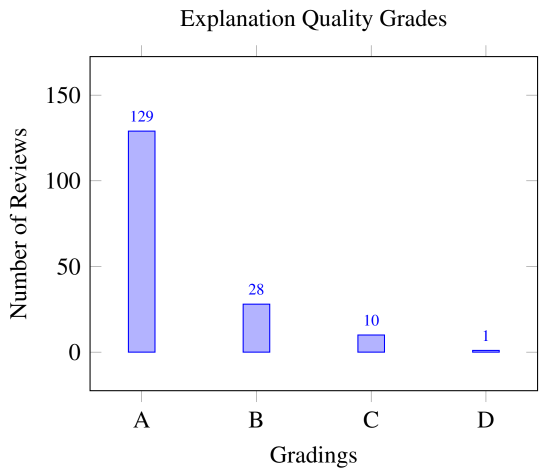


Figure 4: Distribution of grades for explanations generated by the fine-tuned Mistral 7B model on our 200-review benchmark dataset, evaluated by human experts. Grades range from A (highest quality) to D (lowest quality).

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 3.4.2.

Table 1 demonstrates that Llama 2 exhibits superior performance with prompt Template 1 from section 3.4.2, while Mistral achieves optimal performance with the *"explain-then-annotate"* prompt template (Template 2).

To determine the optimal training dataset size, we investigated the relationship between model accuracy and training data volume, maintaining a single training epoch while varying the dataset size. The results, presented in Figure 3b, demonstrate that model performance improves proportionally with the training dataset size.

We also examined the impact of training epochs using 2,500 reviews from the training set. Figure 3a illustrates these results. Our analysis reveals that models trained on smaller datasets with multiple epochs can achieve comparable performance to those trained on larger datasets with fewer epochs.

We manually reviewed the quality of explanations generated by the Mistral model, which performed better with Template 2, focusing only on correctly classified instances. The generated explanations were evaluated using a four-tier grading system: Grade A for correct, comprehensive responses containing all the relevant or important details from the user review; Grade B for correct responses with some relevant details; Grade C for generic responses lacking specific details; and Grade D for incorrect explanations inconsistent with the user review content. Figure 4 presents the results of our manual review. Out of 168 correct classifications, 129 were graded as A, 28 as B, 10 as C, and only one as D. This distribution indicates that the fine-tuned model generated satisfactory results (Grade A or B) 93.45% of the time, with 76.79% achieving the highest grade. These findings demonstrate the model's strong capability to generate accurate and detailed explanations for app review classifications. Additionally, we experimented with the effects of `Temperature` and `Top_p` parameters on all the fine-tuned models. As shown in Figures 2a and 2b, both fine-tuned models demonstrated performance variations similar to commercial LLMs across different `Temperature` settings (0-2.0) and `Top_p` values (0.0-1.0), with optimal performance typically achieved at lower `Temperatures` (0-0.5) and performance degradation observed at higher `Temperatures` (>1.5).

5 CONCLUSION

In this study, we explore the performance of commercial and open-source LLMs for app review classification. Our evaluation shows `GPT-3.5` achieving an F1 score of 0.849 in zero-shot settings, outperforming `Gemini Pro`'s 0.754, with both models performing optimally at lower `Temperature` and `Top_p` values.

Using `GPT-3.5`, we generated a balanced dataset of 10,000 reviews, validated through manual review yielding a Cohen's Kappa score of 0.913 and an average accuracy of 81.89%. These results demonstrate LLMs' effectiveness for automated annotation. Our experiments with open-source models showed instruction fine-tuning significantly improves performance, with `Llama 2` reaching an F1 score of 0.834 and `Mistral` achieving 0.821 using the *"explain-then-annotate"* approach. We found that smaller datasets (2,500 reviews) with multiple training epochs achieve comparable results to large-scale training. The *"explain-then-annotate"* approach with Mistral proved particularly effective, with 76.79% of explanations achieving Grade A in manual evaluation.

These findings demonstrate that LLMs can effectively automate app review analysis while maintaining high accuracy, with fine-tuned open-source models offering a cost-effective alternative to commercial solutions. We have published our code and dataset[9] to facilitate further research. Future work will investigate the generalizability of our results across multiple domains.

6 LIMITATIONS

While this study provides valuable insights into LLMs for app review classification, several limitations should be noted:

Our analysis focused on a limited selection of commercial (`GPT-3.5`, `Gemini Pro 1.0`) and open-source (`Llama 2`, `Mistral`) LLMs, which may not fully represent all available models. The benchmark dataset of 200 reviews, though carefully curated, could be considered small for comprehensive evaluation.

The study concentrated on English-language reviews from U.S. Google Play Store apps, potentially limiting generalizability to other regions, languages, and app categories. Due to resource constraints, we only evaluated 7 billion parameter models, though larger models may offer improved performance.

While we conducted prompt engineering experiments and selected optimal templates, further investigations could yield improvements. As noted by Chen et al. (2023), commercial models like `GPT-3.5` show temporal performance variations, making our results a snapshot of their capabilities at testing time.

7 ETHICAL CONSIDERATIONS

Our research prioritized several ethical aspects: Privacy was maintained by anonymizing app reviews and excluding personally identifiable information. We acknowledge potential biases in training data from `GPT-3.5`, which could affect classification results for underrepresented groups or app categories. Environmental impact was considered through efficiency optimization in dataset size and training epochs. However, broader discussion is needed regarding energy consumption in NLP research, especially for fine-tuning on consumer hardware. We recognize this technology's dual-use potential - while improving app development, it could be misused for manipulating rankings or spreading misinformation. To promote transparency, we've made our datasets and code pub-

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

licly available. Future research should focus on bias mitigation, energy efficiency, responsible use guidelines, and long-term effects on app ecosystems and user trust.

REFERENCES

Alizadeh, M., Kubli, M., Samei, Z., Dehghani, S., Bermeo, J. D., Korobeynikova, M., and Gilardi, F. (2023). Open-source large language models outperform crowd workers and approach chatgpt in text-annotation tasks. *arXiv preprint arXiv:2307.02179*, 101.

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

Anchiêta, R. T. and Moura, R. S. (2017). Exploring unsupervised learning towards extractive summarization of user reviews. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 217–220.

Aslam, N., RAMAY, A., KEWEN, X., and Sarwar, N. (2020). Convolutional neural network-based classification of app reviews. *IEEE Access*, 8:1–11.

Bartolo, M., Thrush, T., Riedel, S., Stenetorp, P., Jia, R., and Kiela, D. (2021). Models in the loop: Aiding crowdworkers with generative annotation assistants. *arXiv preprint arXiv:2112.09062*.

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

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

Cegin, J., Simko, J., and Brusilovsky, P. (2023). Chatgpt to replace crowdsourcing of paraphrases for intent classification: Higher diversity and comparable model robustness. *arXiv preprint arXiv:2305.12947*.

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

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

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., et al. (2023). Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Choi, J., Yun, J., Jin, K., and Kim, Y. (2024). Multi-news+: Cost-efficient dataset cleansing via llm-based data annotation. *arXiv preprint arXiv:2404.09682*.

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

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.

Dhinakaran, V., Pulle, R., Ajmeri, N., and Murukannaiah, P. (2018). App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. pages 170–181.

Di Sorbo, A., Panichella, S., Alexandru, C., Shimagaki, J., Visaggio, C. A., Canfora, G., and Gall, H. (2016). What would users change in my app? summarizing app reviews for recommending software changes.

Ding, B., Qin, C., Liu, L., Chia, Y. K., Joty, S., Li, B., and Bing, L. (2022). Is gpt-3 a good data annotator? *arXiv preprint arXiv:2212.10450*.

Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. (2013). Why people hate your app: Making sense of user feedback in a mobile app store. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Groen, E., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., Hosseini, M., Marco, J., Oriol, M., Perini, A., and Stade, M. (2017). The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 34.

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

Guo, H. and Singh, M. P. (2020). Caspar: Extracting and synthesizing user stories of problems from app reviews. page 628–640.

Guzman, E., El-Haliby, M., and Bruegge, B. (2015). Ensemble methods for app review classification: An approach for software evolution (n). pages 771–776.

Guzman, E., Ibrahim, M., and Glinz, M. (2017). A little bird told me: Mining tweets for requirements and software evolution.

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

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

Hamilton, K., Longo, L., and Bozic, B. (2024). 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*, pages 1431–1440.

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

Hosseini, M., Snijders, R., Dalpiaz, F., Brinkkemper, S., Ali, R., and Ozum, A. (2015). Refine: A gamified platform for participatory requirements engineering.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. (2023). Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*.

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

Krejcie, R. V. and Morgan, D. W. (1970). Determining sample size for research activities. *Educational and psychological measurement*, 30(3):607–610.

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *biometrics*, pages 159–174.

Li, H., Zhang, L., Zhang, L., and Shen, J. (2010). A user satisfaction analysis approach for software evolution. *2010 IEEE International Conference on Progress in Informatics and Computing*, 2:1093–1097.

Maalej, W., Kurtanović, Z., Nabil, H., and Stanik, C. (2016). On the automatic classification of app reviews. *Requirements Engineering*, 21.

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

Maalej, W., Nayebi, M., Johann, T., and Ruhe, G. (2015). Toward data-driven requirements engineering. *IEEE Software*, 33:48–56.

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

Mekala, R. R., Razeghi, Y., and Singh, S. (2024). Echoprompt: Instructing the model to rephrase queries for improved in-context learning.

Møller, A. G., Dalsgaard, J. A., Pera, A., and Aiello, L. M. (2023). The parrot dilemma: Human-labeled vs. llm-augmented data in classification tasks. *arXiv preprint arXiv:2304.13861*.

Pagano, D. and Maalej, W. (2013). User feedback in the appstore: An empirical study.

Pérez, A., Fernández-Pichel, M., Parapar, J., and Losada, D. E. (2023). Depresym: A depression symptom annotated corpus and the role of llms as assessors of psychological markers. *arXiv preprint arXiv:2308.10758*.

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Reiter, L. (2013). Zephyr. *Journal of Business & Finance Librarianship*, 18(3):259–263.

Restrepo, P., Fischbach, J., Spies, D., Frattini, J., and Vogelsang, A. (2021). Transfer learning for mining feature requests and bug reports from tweets and app store reviews.

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

Stanik, C., Haering, M., and Maalej, W. (2019). Classifying multilingual user feedback using traditional machine learning and deep learning. pages 220–226.

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

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

Vu, P. M., Nguyen, T. T., Pham, H. V., and Nguyen, T. T. (2015). Mining user opinions in mobile app reviews: A keyword-based approach. *arXiv preprint arXiv:1505.04657*.

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

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

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

Zhang, R., Li, Y., Ma, Y., Zhou, M., and Zou, L. (2023a). Llmaaa: Making large language models as active annotators. *arXiv preprint arXiv:2310.19596*.

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