User Feedback to Actionable Insights: A Fine-grained Multilabel Approach for Classifying App Reviews

Chirag Jain
TCS Research
Pune, India
chirag.rjain3@tcs.com

Preethu Rose Anish
TCS Research
Pune, India
preethu.rose@tcs.com

Abstract

The rapid proliferation of mobile applications has resulted in a vast volume of user-generated reviews, which often contain critical insights into bugs, feature demands, and overall user satisfaction. Efficiently organizing and interpreting this feedback is essential for developers seeking to enhance app quality and user experience. In this study, we propose a fine-grained, multilabel classification schema for app reviews, structured as a two-level taxonomy. The first level categorizes reviews into three primary classes: *Bug Report, Feature Request*, and *App Satisfaction & Feedback*. The second level further refines these categories into subtypes addressing specific concerns such as *performance, usability* and *privacy*. To automate this classification, we fine-tune several state-of-the-art large language models (LLMs), including LLaMA, Gemma, Mistral, Qwen, on a manually annotated dataset spanning 10 diverse application domains. Among these, LLaMA 3.1 achieved the highest performance, with weighted F1 scores of 0.90 and 0.79 at the first and second taxonomy levels, respectively. To assess scalability, we applied the fine-tuned LLaMA 3.1 model to label an additional 10,000 unlabeled reviews, creating a silver-standard corpus. This taxonomy-driven approach enables more nuanced analysis of user feedback, supporting targeted app improvements and fostering enhanced user satisfaction.

1 Introduction

The rapid proliferation of mobile applications has fundamentally transformed how developers collect and respond to user feedback. With millions of apps available on platforms such as Google Play and the Apple App Store, users generate vast volumes of reviews (Ceci, 2022), offering a rich yet unstructured source of insights. These reviews often contain actionable information ((Eftee et al.); (Triantafyllou et al., 2020); (Gao et al., 2022); (Martin, 2017); (Tavakoli et al., 2018)), ranging from bug reports to feature requests, that is crucial for improving app functionality and user satisfaction. However, manually analyzing this feedback is impractical due to its sheer volume, informal language, and lack of structure ((Massenon et al., 2024); (Dąbrowski et al., 2023)). For instance, consider the following review: *"The user interface is terrible! Images take ages to load, navigating through it is a hassle, and I often close the app immediately due to persistent technical issues. Among shopping apps, Amazon has the least appealing interface. Excessive and irrelevant ads constantly interrupt product searches. The filters are ineffective, merely for show. I'm done with Amazon and uninstalling it soon!"*

This review encapsulates multiple concerns, including UI design flaws (terrible interface), performance issues (slow loading) and overall user dissatisfaction (I'm done with Amazon and uninstalling it soon!). Traditional classification approaches typically assign such reviews to broad categories like *bug reports, feature requests*, or *performance issues*, often missing the nuanced context that could guide more targeted development decisions.

We argue that extracting actionable insights requires distinguishing not only the type of issue (e.g., *usability*, *performance*) but also the intent behind it (e.g., *bug report* vs. *feature request*). For instance, performance concerns may appear as bug reports ("The app crashes frequently") and as feature suggestions ("Enhance scrolling smoothness please"), while usability issues might be framed as bugs ("Navigation is frustrating") rather than enhancements. Capturing these subtleties enables developers to prioritize tasks more effectively, allocate resources appropriately, and assign issues to the correct teams, whether for backend fixes or UI redesigns. In this study, we define a bug report as a review describing an existing issue or malfunction in the app, and a feature request as a review that asks for new functionality or improvements to ex-

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 5164–5180
November 6-10, 2026 ©2026 Association for Computational Linguistics

isting features where no malfunction is present but enhancement is desired.

To address this, we propose a fine-grained, multilabel classification framework for app reviews, structured into a two-level taxonomy (illustrated in Figure 1). The first level categorizes reviews into three primary classes: *Bug Report, Feature Request*, and *App Satisfaction & Feedback*. The second level refines these into subcategories addressing specific aspects such as *performance, usability*, and *privacy*. This hierarchical structure facilitates a more comprehensive and organized approach to app improvement.

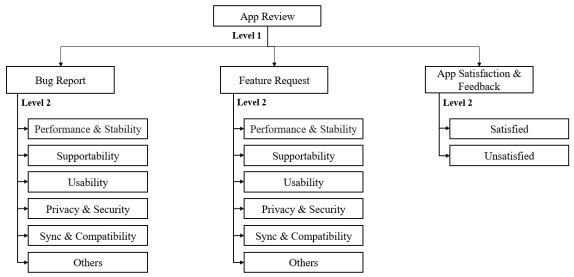


Figure 1: A Two-Level Hierarchical Schema for App Review Classification.

While traditional machine learning models have been employed for app review classification, they often struggle with the informal and context-rich nature of user generated content (Ghatora et al., 2024). For example, a review such as *"The app crashes when I try to sync data offline"* may be broadly classified as a *bug report*, overlooking more specific subcategories such as *synchronization issues*. In contrast, Large Language Models (LLMs), with their advanced semantic understanding, offer superior performance with minimal fine-tuning (Kostina et al., 2025).

We evaluated our proposed framework using a gold-standard dataset comprising 10,400 manually annotated app reviews across 10 diverse domains. The majority of these reviews (10,000) were sourced from the RARE dataset (Singh et al., 2024), which contains Google Play Store reviews. Additionally, we included 400 privacy and security-related reviews curated from (Zhang et al., 2025) (see section 3.3 for details). The dataset is split evenly into training and testing sets. We fine-tune several state-of-the-art LLMs including LLaMA 3.1 (Vavekanand and Sam, 2024), Mistral (Jiang et al., 2023), Gemma2 (Team et al., 2024), Qwen3 (Yang et al., 2025) and an encoder based transformer model BERT (Devlin et al., 2019), and compare their performance against traditional classifiers such as Random Forest, Decision Tree, Logistic Regression, SVM, XGBoost. Our results show that LLaMA 3.1 achieves the highest weighted F1 score of 0.9 at the first taxonomy level and 0.79 at the second. To further validate scalability, we apply LLaMA 3.1 to label an additional 10,000 reviews (from Apple App store) sourced from RARE dataset, creating a silver-standard corpus. Overall, this work represents an application-oriented contribution focused on task formulation, rigorous annotation, high-quality dataset construction, and systematic benchmarking for fine-grained, multi-label app review classification. Below are the key contributions.

- Development of a fine-grained multilabel classification framework for app reviews, structured as a two-level taxonomy to capture nuanced user feedback.
- Introduction of a labeled dataset of 20,400 reviews, comprising 10,400 manually annotated and 10,000 automatically labeled using LLaMA 3.1, providing a valuable resource for future research.
- Experimentation with state-of-the-art large language models and a thorough evaluation of their performance using standard metrics.
- Release of the complete labeled dataset and replication code as supplementary material to support transparency reproducibility.

The remainder of this paper is organized as follows: Section 2 reviews related work in app review classification. Section 3 presents our classification schema and dataset. Section 4 outlines the methodology and experimental setup. Section 5 discusses the results, Section 6 discusses error analysis and Section 7 concludes with future research directions. Section 8 addresses the limitations of our study.

2 Related Work

Classifying app reviews has been a long-standing research focus due to its practical relevance for developers, product managers, and software maintenance teams. User generated reviews often contain valuable insights into software bugs, feature requests, and user experience, making their automatic analysis both impactful and challenging.

Early approaches predominantly relied on pattern-based techniques ((Johann et al., 2017); (Guo and Singh, 2020); (Gu and Kim, 2015)), employing hand-crafted rules and templates curated

by domain experts. While these methods offered interpretability, they were labor-intensive, lacked scalability, and struggled to generalize across diverse app domains or evolving user language. To address these limitations, researchers adopted traditional machine learning (ML) models ((Lu and Liang, 2017); (Messaoud et al., 2019); (Araujo et al., 2020); (Al-Hawari et al., 2021); (Dave and Anu, 2022); (Deocadez et al., 2017); (Jha and Mahmoud, 2019); (Phetrungnapha and Senivongse, 2019); (Maalej and Nabil, 2015); (Haoues et al., 2023)), which learned from labeled data using features such as n-grams, TF-IDF, or bag-of-words. Despite their utility, these shallow representations often failed to capture the semantic and contextual nuances of natural language, particularly in short, ambiguous or multi-intent reviews.

Subsequent work introduced deep learning (DL) architectures, including LSTM, RNN, and ANN models ((Alsaleh et al., 2025); (Qiao et al., 2020); (Yahya et al., 2023); (Lu and Liang, 2017); (Zahoor and Bawany, 2024)), which significantly improved upon traditional ML by modeling the sequential and contextual structure of review text. These methods demonstrated notable gains, especially in processing longer and more structured feedback.

In the domain of app review classification, research has primarily relied on encoder-based models. Although some studies have explored the use of LLMs, their focus has largely been limited to tasks such as sentiment analysis, requirements classification, and feature extraction, rather than addressing hierarchical classification ((Shah et al., 2025); (Gunathilaka and de Silva); (Alsanousi et al., 2025); (Ghosh et al., 2024); (Zuhir et al., 2025)). Several studies have explored requirement-oriented classification of user reviews, classifying reviews into functional requirements and non-functional requirements ((Chaudhary et al., 2025); (Yang and Liang, 2015); (Memon et al., 2024); (Dave and Anu, 2022); (Deocadez et al., 2017)). Others proposed more fine-grained taxonomies, categorizing reviews into bug reports, feature requests, and quality attributes such as usability, maintainability, reliability, and performance ((Hadi and Fard, 2023); (Kaur and Kaur, 2023b); (Kaur and Kaur, 2023a); (Yahya et al., 2023); (Wang et al., 2018); (Jha and Mahmoud, 2019); (Saudy et al., 2021); (Scoccia et al., 2023); (Haoues et al., 2023); (Tasnia et al., 2022)). Most of these existing approaches rely on flat, single-stage classification schema, which fail to differentiate between the framing of an issue by the user (e.g., as a bug or feature request) and its underlying nature (e.g., performance or privacy concern).

We propose a two-level hierarchical classification framework leveraging fine-tuned LLMs. At the first level, our model identifies the user's intent - classifying reviews as *bug reports, feature requests*, or *App Satisfaction & Feedback*. At the second level, it assigns content-specific labels such as *performance, privacy*, or *maintainability* for technical feedback, and *satisfied* or *unsatisfied* for sentiment-oriented reviews (see Figure 1).

To the best of our knowledge, no prior work has adopted a hierarchical, intent-aware, and content-specific classification framework using fine-tuned LLMs for app review analysis.

3 Dataset Creation and Classification Schema Design

This section outlines the methodology employed to construct a high-quality dataset for app review classification and to design a two-level hierarchical schema to enable fine-grained categorization. The process comprises three key stages: (i) data collection and refinement, (ii) schema design, and (iii) annotation.

3.1 Data Collection and Refinement

We utilized the publicly available RARE (Repository for App review REfinement) (Singh et al., 2024), a benchmark dataset designed to enhance clarity, informativeness, and utility of user reviews for both manual annotation and automated analysis. This RARE dataset comprises 20,000 pairs of original and refined reviews, evenly split between the Google Play Store and Apple App Store.

For our study, we curated a dataset of 20,400 refined user reviews. This includes 20,000 reviews from RARE, and an additional 400 reviews from (Zhang et al., 2025) specifically related to security and privacy concerns categories that were underrepresented in the original RARE dataset (see Section 3.3 for details). The length of these reviews ranges approximately from 15 to 180 words. Each review entry in RARE includes metadata such as the app name, original review text, and its corresponding refined version.

3.2 Classification Schema Design

To develop a robust and comprehensive classification schema for app reviews, we began by survey-

ing prior literature in app review analysis and software maintenance ((Wang et al., 2018); (Haoues et al., 2023); (Rahman et al., 2020); (Budake et al., 2023); (Tasnia et al., 2022)).This review guided the initial taxonomy by identifying prevalent class labels, while revealing limitations in existing categorizations, particularly in their ability to capture nuanced user intent and content-specific feedback.

Building on this foundation, we conducted an exploratory manual analysis of 100 randomly selected app reviews (from RARE), samples across 10 different apps (10 per app), following guidelines proposed in Hoda's Grounded Theory (GT) for software engineering (Hoda, 2021). Following these guidelines, we complemented the literature-driven approach with an inductive, data-driven process to ensure the schema emerged from user feedback. We then utilized open coding and constant comparison techniques—fundamental principles of GT for deriving categories from data. This qualitative assessment conducted by authors, revealed recurring themes in user feedback, including issue reporting, feature suggestions, and sentiment expression. Quantitatively, we observed that: 92.62% of reviews conveyed *user sentiment* (*satisfaction* or *dissatisfaction*), 72.13% *reported bugs* or *functional issues*, and 35.62% included suggestions for *new features* or *enhancements*. The open coding process continued iteratively until sample saturation was reached (i.e., no new categories emerged within this subset), ensuring comprehensive coverage of user feedback patterns and grounding the schema in empirical evidence.

These findings motivated the design of a three-pronged top-level classification schema, comprising: *Bug Report, Feature Request*, and *App Satisfaction & Feedback*. Through iterative discussions among the authors and by applying the open-coded themes in accordance with GT guidelines, we refined this into a two-level hierarchy: the first level captures overall intent, while the second provides fine-grained categorization. The complete schema is shown in Figure 1, with definitions provided below.

3.2.1 Level 1: Primary Categories

Bug Report: Reviews describing issues or malfunctions that negatively affect the app's expected behavior or performance.

Feature Request: Reviews suggesting new functionality or enhancement to existing features.

App Satisfaction & Feedback: Reviews expressing overall user sentiment – either positive or negative, regarding the app experience.

Level 1: Categories	Instances Count	Level 2: Subcategories	Instances Count
Bug Report	7079	Performance & Stability	2891
		Usability	2770
		Supportability	1271
		Privacy & Security	436
		Sync & Compatibility	1542
		Others	1295
Feature Request	3596	Performance & Stability	125
		Usability	2803
		Supportability	235
		Privacy & Security	91
		Sync & Compatibility	216
		Others	526
App Satisfaction & Feedback	9595	Satisfied	1661
		Unsatisfied	7934

Table 1: Classification schema with Level 1 and Level 2 categories

3.2.2 Level 2: Subcategories

Bug Report Subcategories:

Performance & Stability: Reports of lag, crashes, or performance degradation, especially post-updates.

Supportability: Issues related to help features, documentation, or customer support.

Sync & Compatibility: Problems with data synchronization, multi-device usage, or cross-platform compatibility.

Privacy & Security: Concerns about data protection, unauthorized access, or privacy violations.

Usability: Complaints regarding layout, design flaws, navigation difficulties, or user interface issues.

Others: Reports that do not fit into the above categories.

Feature Request Subcategories:

Performance & Stability: Suggestions to improve responsiveness, efficiency or update handling.

Supportability: Requests for enhanced help tools, live chat, or troubleshooting aids.

Sync & Compatibility: Suggestions for better data synching, backup, or platform support.

Privacy & Security: Requests for stronger encryption, data control or transparency.

Usability: Recommendations for improved interface design, accessibility, or user experience.

Others: Suggestions that fall outside the defined subcategories.

App Satisfaction & Feedback Subcategories:

Satisfied: Reviews expressing positive sentiment and satisfaction with the app.

Unsatisfied: Reviews reflecting dissatisfaction, frustration, or criticism.

A comprehensive breakdown of class labels across the Level 1 and Level 2 categories is shown in Table 1. This hierarchical schema generalizes

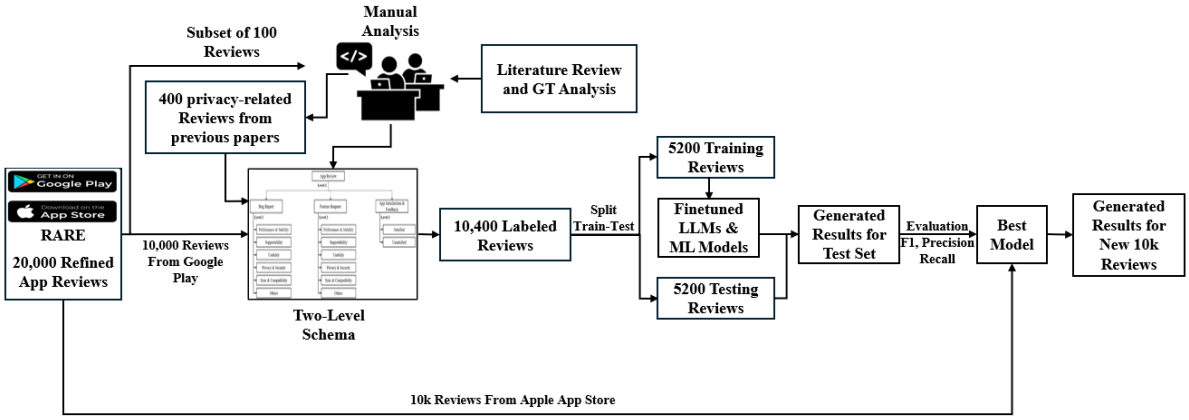


Figure 2: Pipeline for fine-grained classification of app reviews.

and extends prior efforts in app review classification by offering improved granularity and real-world applicability.

3.3 Annotation

To construct a high-quality annotated dataset, we engaged five software engineering professionals with over five years of experience in mobile app development. All annotators had prior experience to app review analysis for bug detection, feature prioritization, and product decision-making, ensuring familiarity with domain-specific terminology and user concerns. These professionals were colleagues within our organization (recruited on a voluntary basis) who agreed to contribute their expertise as part of an internal research initiative aimed at advancing legal AI applications. Participation was optional, and while no monetary compensation was provided, each participant was clearly informed about the task, its purpose, and expected time commitment before agreeing to participate.

Prior to annotation, the annotators were provided with a comprehensive document detailing the app review classification schema. To calibrate label interpretation and ensure consistency, a training round was conducted using 500 randomly selected reviews from a pool of initial 10,000 Google Play Store reviews (from RARE). These reviews were independently annotated by all five software engineering professionals. Discrepancies, particularly in ambiguous cases, were resolved through group discussion, resulting in a shared understanding of class definitions and aligned interpretations for recurring ambiguity patterns (e.g., distinguishing bug reports from feature requests). This process focused on aligning annotators' interpretations and establishing shared calibration through collective discussion. For instance, the review *"The app crashes when I try to sync data. Please fix this."* was initially misclassified by some annotators as a *'Feature Request: Sync & Compatibility'*. Following deliberation, it was correctly relabeled as a *Bug Report: Sync & Compatibility*, reflecting the user's report of a malfunction rather than a request for new functionality. Similar discussions were used to align interpretations across comparable cases.

Inter-annotator agreement was quantified using Fleiss' Kappa, yielding a score of 0.76, indicative of substantial agreement and validating both the schema clarity and annotation reliability. Category-wise agreement is shown in Appendix B Table 6. Following the calibration round, the remaining reviews were annotated independently by individual annotators. Given that app reviews often exhibit recurring linguistic patterns, the shared calibration facilitated consistent interpretation across the dataset, and although no further overlap sampling was performed, annotators consistently applied the aligned interpretations established during these discussions. Subsequently, these initial 10,000 reviews from RARE were evenly distributed among the annotators (2,000 reviews each, comprising 200 reviews per app), ensuring balanced domain coverage. Manual annotation was efficient, with each review requiring approximately 30 seconds to 1 minute for annotation. On an average, each review received 2 to 3 labels, reflecting the multi-faceted nature of user feedback. Further dataset statistics and corresponding distribution plots, including review length, label co-occurrence patterns, and label cardinality are provided in Appendix A (Figure 3, 4, 5)

During post-annotation analysis, we observed underrepresentation of privacy and security-related

reviews. To address this, an additional 400 reviews specifically focusing on these themes were sourced from (Zhang et al., 2025). These reviews were refined using the methodology described in (Singh et al., 2024) (see Section 3.1). These additional 400 reviews were evenly distributed among the annotators (80 reviews each) and manually annotated using the same schema. The combined set of manually annotated reviews constitute the final gold corpus (10,400 reviews). Examples of app reviews and their annotated hierarchical classes (Level 1 and Level 2) are provided in Appendix B Table 9.

In parallel, the additional 10,000 reviews from the Apple App store were annotated automatically using our top-performing fine-tuned LLaMA model (see section 4), trained (on training split) on the manually annotated dataset comprising of 10,400 reviews. These reviews form the silver corpus, enabling scalable annotation while preserving high-quality label assignments. To assess its scalability and reliability, a subset of 1000 reviews from 10,000 silver corpus reviews was independently evaluated by the same five software professionals involved in Section 3.3. Their expert assessment indicated that 90% of silver-standard labels were consistent with the manually annotated gold- standard labels, demonstrating reasonable reliability and suitability for downstream analysis.

The complete annotation pipeline, including dataset construction, schema development, and model training, is illustrated in Figure 2.

4 Experiments

We evaluated the classification framework using the gold standard corpus comprising 10,400 manually annotated app reviews. The dataset was evenly split into training and testing subsets (5,200 reviews each). This split was stratified by app domain, with reviews from each of the 11 domains divided in the same ratio into training and test sets. This ensured proportional representation of every domain in both splits. To perform multi-label sequence classification, we fine-tuned several state-of-the-art LLMs, including LLaMA 3.1-8B, Mistral-7B, Gemma 2-9B, and Qwen 3-8B and encoder-based transformer model BERT large. To enhance training efficiency and reduce computational overhead, we applied LoRA (Low-Rank Adaptation) and 4-bit quantization, enabling model fine-tuning on NVIDIA V100 GPU (32 GB) in approximately 10 hours without significant performance degradation. To mitigate class imbalance, particularly in underrepresented categories such as privacy-related sublabels, we incorporated custom label weighting during training. Our implementation uses a standard inverse-frequency (balanced) class weighting scheme (i.e. BCE with inverse frequency class weighting), where class weights are computed from the training data using scikit-learn's compute_class_weight (class_weight="balanced"). These class-specific weights are incorporated into the training loss so that errors on minority classes contribute more heavily during optimization than errors on majority classes. Instead of treating all classes equally, we assigned higher weights to minority classes and lower weights to majority classes. This ensured that misclassifications on underrepresented categories contributed more significantly to the overall loss, thereby encouraging the model to learn balanced decision boundaries. The class-specific weights were inversely proportional to their frequencies in the training set,

$$w_c = 1 - \frac{n_c}{\sum_{j \in C} n_j}, \tag{1}$$

where n_c denotes the number of positive labels in category c and $\sum_{j \in C} n_j$ represents the total number of positive labels across all categories. This weighting strategy was incorporated into the loss function (binary cross-entropy), ensuring that rare categories received proportionally higher attention during optimization.

In addition to fine-tuned models, we incorporated a prompted LLM baseline by evaluating LLaMA3.2-3B-Instruct under a zero-shot setting, with preliminary results presented in Section 5. All LLMs were initialized from their official Hugging Face checkpoints and fine-tuned for our task adapted to each model's architecture to ensure consistent embedding alignment. Given that the majority of prior work on app review classification has employed ML models, we included them to establish robust baselines and facilitate direct comparison. To this end, we trained traditional multi-label classifiers, including Random Forest, Decision Tree, Logistic Regression, Support Vector Machine (SVM), and XGBoost, on the same dataset. For this, the refined review texts were transformed into dense vector embeddings using pre-trained sentence transformers before classification. Specifically, we used the sentence transformer model `all-MiniLM-L6-v2` to generate these fixed-length sentence embeddings, which served as input fea-

tures for all traditional classifiers. For all models, multi-label predictions were obtained by applying a sigmoid activation to the output logits, followed by a fixed per-label decision threshold of 0.5 to determine positive label assignments. This value of 0.5 is selected as threshold because it is the standard and most commonly adopted setting in multilabel classification and provided stable performance across labels in our preliminary experiments. Model performance was evaluated using weighted-average F1 score, precision, and recall, capturing both overall effectiveness and class-specific accuracy. Here, the weighted-average F1 score is computed by weighing each label's F1 by its support (i.e., the number of true instances of that label), and is computed separately for Level-1 and Level-2. While our primary experiments used refined reviews from the RARE dataset to reduce annotation ambiguity, we also conducted a supplementary evaluation on raw reviews. Using LLaMA 3.1-8B, performance decreased from 0.90 $\rightarrow$ 0.83 (Level-1 weighted F1) and 0.79 $\rightarrow$ 0.71 (Level-2 weighted F1), with macro-F1 dropping from 0.87 $\rightarrow$ 0.80 (Level-1) and 0.64 $\rightarrow$ 0.57 (Level-2). This highlights the impact of noisy language, typos, and slang in real app reviews, and suggests a practical refine-then-classify pipeline for deployment. Detailed results are presented in Section 5. Cross-domain generalization was evaluated using a domain-held-out setup with LLaMA-3.1-8B, in which each domain was excluded from training and used only for testing (see Appendix B Table 7). Appendix B Table 8 summarizes the model-specific hyperparameters and training configurations used in our experiments.

	Model Name	P	R	Weighted-F1	Macro-F1
Language Models	LLaMA 3.1-8B-Instruct	0.92	**0.88**	**0.90**	**0.87**
	Mistral-7B-Instruct-v0.3	0.91	**0.88**	0.89	**0.87**
	Gemma-2-9B-it	0.92	0.87	0.89	0.86
	Qwen 3-8B	**0.93**	0.83	0.88	0.85
	BERT large	0.79	0.74	0.76	0.71
	LLaMA 3.2 (Prompting)	0.82	0.76	0.72	0.68
Traditional ML Models	Random Forest	0.90	0.62	0.68	0.60
	Decision Tree	0.77	0.80	0.79	0.72
	SVM	0.88	0.78	0.82	0.77
	Logistic Regression	0.88	0.76	0.81	0.75
	XGBoost	0.88	0.75	0.80	0.74

Table 2: First Level Classification Performance.

5 Results & Discussion

The experimental results clearly demonstrate that LLMs substantially outperform traditional ML models in hierarchical text classification tasks, encompassing both first level (broad categories) and second level (fine-grained subcategories) classifications. Table 2 and 3 summarize the weighted average precision, recall, F1 and Macro-F1 scores for each of the evaluated models.

	Model Name	P	R	Weighted-F1	Macro-F1
Language Models	LLaMA 3.1-8B-Instruct	0.81	**0.78**	**0.79**	**0.64**
	Mistral-7B-Instruct-v0.3	0.79	**0.78**	0.78	0.63
	Gemma-2-9B-it	0.81	0.77	0.78	0.64
	Qwen 3-8B	**0.82**	0.73	0.76	0.58
	BERT large	0.50	0.85	0.62	0.40
	LLaMA 3.2 (Prompting)	0.82	0.74	0.71	0.55
Traditional ML Models	Random Forest	0.72	0.48	0.50	0.29
	Decision Tree	0.50	0.53	0.51	0.32
	SVM	0.71	0.63	0.66	0.45
	Logistic Regression	0.73	0.62	0.65	0.43
	XGBoost	0.71	0.60	0.63	0.43

Table 3: Second Level Classification Performance.

Among the evaluated LLMs, LLaMA 3.1 achieves the highest performance, with weighted F1 scores of 0.90 at the first level and 0.79 at second level, indicating robust overall accuracy. This is closely followed by Mistral and Gemma, both attaining F1 scores of 0.89 (first level) and 0.78 (second level), and Qwen, which scores 0.88 and 0.76, respectively. Notably, Qwen exhibits the highest precision at both classification levels (0.93 and 0.82), reflecting its strong ability to minimize false positives. However, its relatively lower recall suggests a tendency to miss relevant instances compared to its counterparts. Moreover, LLaMA3.2 (Prompting) delivers competitive results (0.72 and 0.71 at the first and second level, respectively). This suggests that while prompting improves adaptability, the observed gap relative to finetuning likely stems from differences in both model size, variants and training setup.

In contrast, traditional models such as SVM and Logistic Regression perform moderately well at the first level but exhibit a marked decline in performance at the second level. This highlights their limitations in capturing the nuanced, context-dependent patterns present in informal app reviews.

These findings underscore the superior contextual understanding of LLMs, particularly LLaMA 3.1, in distinguishing complex, multi-intent user feedback, such as differentiating between *bug reports* and *feature requests* within a single review (e.g., *"The app crashes when syncing data, please add a backup option!"*). At Level-2, macro-F1 is notably lower than weighted F1, reflecting weaker performance on minority feature-request subcategories such as *supportability*, *performance & stability*, *privacy & security*, and *others*. This gap highlights that fine-grained classification remains

Levels	Classes	LLaMA 3.1			Mistral			Gemma2			Qwen3			BERT			LLaMA 3.2 (Prompting)		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Level1	App Satisfaction & Feedback	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.97	0.96	0.96	0.93	0.94	0.47	0.80	0.59	0.72	0.99	0.83
	Bug Report	0.92	0.87	0.89	0.91	0.87	0.89	0.92	0.85	0.89	0.93	0.80	0.86	0.92	0.77	0.84	0.98	0.45	0.62
	Feature Request	0.85	0.71	0.77	0.84	0.73	0.78	0.85	0.66	0.75	0.87	0.67	0.75	0.64	0.63	0.64	0.95	0.35	0.51
Level2	App Satisfaction & Feedback_Satisfied	0.95	0.80	0.87	0.93	0.82	0.87	0.90	0.89	0.90	0.93	0.80	0.86	0.77	0.91	0.84	0.85	0.91	0.88
	App Satisfaction & Feedback_Unsatisfied	0.93	0.97	0.95	0.93	0.96	0.94	0.92	0.96	0.94	0.94	0.93	0.93	0.59	0.97	0.73	0.72	0.93	0.80
	Bug Report_Supportability	0.89	0.77	0.83	0.90	0.80	0.85	0.90	0.79	0.84	0.91	0.75	0.82	0.13	0.75	0.22	0.96	0.33	0.49
	Bug Report_Performance & Stability	0.75	0.81	0.78	0.78	0.82	0.80	0.76	0.84	0.80	0.82	0.77	0.79	0.29	0.91	0.44	0.99	0.47	0.64
	Bug Report_Usability	0.69	0.68	0.69	0.67	0.67	0.67	0.72	0.64	0.68	0.72	0.57	0.64	0.34	0.49	0.40	1.00	0.38	0.55
	Bug Report_Privacy & Security	0.77	0.96	0.85	0.75	0.93	0.83	0.78	0.95	0.86	0.79	0.90	0.84	0.39	0.99	0.56	1.00	0.27	0.43
	Bug Report_Sync & Compatibility	0.76	0.67	0.71	0.72	0.69	0.71	0.71	0.68	0.70	0.75	0.65	0.69	0.44	0.72	0.54	1.00	0.42	0.59
	Bug Report_Others	0.45	0.24	0.32	0.43	0.33	0.37	0.52	0.23	0.31	0.44	0.14	0.22	0.15	0.11	0.13	1.00	0.26	0.41
	Feature Request_Supportability	0.36	0.04	0.06	0.50	0.03	0.05	0.69	0.08	0.14	1.00	0.04	0.08	0.00	0.00	0.00	1.00	0.25	0.40
	Feature Request_Performance & Stability	0.55	0.11	0.19	0.35	0.11	0.17	0.43	0.17	0.24	0.33	0.04	0.07	0.00	0.00	0.00	0.50	0.17	0.26
	Feature Request_Usability	0.78	0.76	0.77	0.77	0.80	0.78	0.80	0.72	0.76	0.79	0.75	0.77	0.24	0.83	0.37	1.00	0.24	0.39
	Feature Request_Privacy & Security	0.44	0.32	0.37	0.38	0.40	0.39	0.33	0.28	0.30	0.20	0.04	0.07	0.17	0.33	0.22	0.50	0.14	0.22
	Feature Request_Sync & Compatibility	0.59	0.41	0.48	0.64	0.24	0.35	0.58	0.30	0.40	0.67	0.14	0.23	0.00	0.00	0.00	1.00	0.31	0.48
	Feature Request_Others	0.53	0.26	0.35	0.43	0.24	0.31	0.53	0.25	0.34	0.60	0.22	0.32	0.00	0.00	0.00	1.00	0.34	0.51

Table 4: Detailed Class-wise Performance of Language Models at Level 1 and Level 2

Levels	Classes	Random Forest			Decision Tree			Logistic Regression			SVM			XGBoost		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Level1	App Satisfaction & Feedback	0.92	0.97	0.94	0.92	0.92	0.92	0.93	0.96	0.95	0.93	0.98	0.95	0.93	0.95	0.94
	Bug Report	0.94	0.30	0.46	0.72	0.81	0.76	0.88	0.62	0.73	0.89	0.63	0.74	0.88	0.62	0.73
	Feature Request	0.78	0.29	0.42	0.47	0.50	0.48	0.77	0.49	0.60	0.77	0.53	0.63	0.76	0.46	0.57
Level2	App Satisfaction & Feedback_Satisfied	0.88	0.27	0.41	0.43	0.44	0.44	0.83	0.64	0.72	0.83	0.63	0.72	0.79	0.59	0.68
	App Satisfaction & Feedback_Unsatisfied	0.80	0.98	0.88	0.81	0.81	0.81	0.87	0.96	0.91	0.86	0.97	0.91	0.86	0.95	0.90
	Bug Report_Supportability	0.80	0.25	0.38	0.39	0.39	0.39	0.74	0.54	0.63	0.76	0.56	0.65	0.77	0.53	0.62
	Bug Report_Performance & Stability	0.81	0.47	0.60	0.43	0.58	0.49	0.74	0.63	0.68	0.78	0.64	0.70	0.74	0.62	0.67
	Bug Report_Usability	0.73	0.05	0.09	0.30	0.37	0.33	0.60	0.34	0.43	0.63	0.34	0.44	0.59	0.32	0.41
	Bug Report_Privacy & Security	0.97	0.35	0.51	0.31	0.53	0.39	0.89	0.91	0.90	0.89	0.95	0.92	0.85	0.88	0.86
	Bug Report_Sync & Compatibility	0.94	0.04	0.08	0.30	0.32	0.31	0.76	0.38	0.51	0.81	0.38	0.52	0.69	0.35	0.47
	Bug Report_Others	0.00	0.00	0.00	0.13	0.18	0.15	0.29	0.00	0.01	0.00	0.00	0.00	0.35	0.03	0.05
	Feature Request_Supportability	0.00	0.00	0.00	0.04	0.04	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Feature Request_Performance & Stability	0.00	0.00	0.00	0.01	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Feature Request_Usability	0.74	0.35	0.48	0.43	0.47	0.45	0.70	0.58	0.63	0.71	0.63	0.67	0.70	0.54	0.61
	Feature Request_Privacy & Security	0.00	0.00	0.00	0.02	0.08	0.04	1.00	0.04	0.08	1.00	0.04	0.08	0.20	0.04	0.07
	Feature Request_Sync & Compatibility	0.00	0.00	0.00	0.08	0.07	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.01	0.02
	Feature Request_Others	0.00	0.00	0.00	0.05	0.05	0.05	1.00	0.00	0.01	0.00	0.00	0.00	0.33	0.00	0.01

Table 5: Detailed Class-wise Performance of Traditional Machine Learning Models at Level 1 and Level 2

more challenging for rare labels. Tables 4 and 5 provide a detailed comparison of the class-wise performance of language models (LMs) and traditional machine learning (ML) models at both Level 1 and Level 2 categories. While weighted F1 reflects overall performance, it is dominated by frequent labels. Therefore, we report macro-F1 and per-label scores, which show that rare Level-2 categories remain harder to classify.

6 Error Analysis

Although our quantitative evaluation showed strong performance across hierarchical levels, a deeper qualitative analysis revealed several failure patterns. We conducted this analysis on a curated subset of misclassified reviews, comparing LLMs and traditional ML models. This analysis highlighted the inherent challenges of fine-grained, multi-label classification in informal app reviews.

A major source of error was overlapping semantics, where common words like "issue", "problem", or "update" appeared in multiple contexts such as bug, dissatisfaction, usability, and feature related reviews. For example, reviews mentioning *"progress becomes difficult after a year"* were often misclassified as performance related bugs instead of usability issues. Similarly, monetization-related complaints like *"ads are annoying if you don't pay for Premium"* frequently misclassified as bug report instead of dissatisfaction or feature requests.

Another frequent challenge was multi-intent phrasing, where a single review expressed multiple concerns such as usability, feature requests, etc. within one statement. Instruction-tuned LLMs like LLaMA 3.1, Gemma 2, etc. handled these cases better than ML baselines but still produced errors when intent cues were unevenly distributed. Models using prompting often missed fine-grained subcategories. They usually predicted the main category correctly but failed to include specific subcategories like *Performance & Stability*. We also found cases where the model focused too much on the tone of the review instead of the actual problem described. For example, a review that sounded positive but mentioned a serious bug was sometimes classified as *"Satisfied"*. In addition, there were

errors where related subcategories were confused, such as predicting *Sync & Compatibility* instead of *Usability*. We performed this analysis with expert annotators, and it confirmed that overlapping semantics, multi-intent phrasing, and subcategory ambiguity accounted for most errors across both LLMs and ML models. Performance is strongest at Level-1 and frequent subcategories, while minority feature-request labels remain less robust.

7 Conclusion and Future Work

In this study, we propose a two-level hierarchical taxonomy for fine-grained app review classification, enabling developers to distinguish between high-level user intents (e.g. *bug reports, feature requests*) and specific content themes (e.g., *performance, usability, privacy*). Our experimental results demonstrate that LLaMA 3.1 outperforms traditional models and other LLMs, particularly in achieving a balanced trade-off between precision and recall across nuanced subcategories.

To support reproducibility and future research, the two annotated datasets comprising a gold-standard set of 10,400 manually labeled reviews and a silver-standard set of 10,000 automatically labeled reviews, are included in the supplementary material. These resources form a strong foundation for automated app review analysis.

Future work will explore three key directions. First, we will build on the zero-shot prompting results reported in this study by exploring diverse prompt engineering strategies and additional LLMs. Second, we plan to integrate our framework into software development pipelines to enable real-time feedback triage and task routing (e.g., directing usability issues to UI/UX teams). Third, we aim to extend the framework to support multilingual app reviews, broadening its applicability across diverse user groups. We also plan to explore explainable AI methods to improve model transparency and foster trust in automated classification systems. In addition, we will investigate label-specific threshold tuning and validation-based calibration to improve minority-class performance.

8 Limitations

As with any empirical research, our work has practical limitations: (1) The classification schema was developed using data from ten app domains, which may not adequately represent niche categories such as enterprise software or finance. This limitation raises concerns about the schema's adaptability across diverse application areas; we will expand the taxonomy in future work to include additional domains. (2) While we achieved a substantial inter-annotator agreement, the absence of post-calibration overlap sampling limits our ability to empirically assess annotation drift in the long-tail. Although annotators followed shared guidelines and benefited from recurring linguistic patterns in app reviews, subtle inconsistencies particularly for low-frequency or long-tail labels cannot be fully ruled out. Ambiguous cases, particularly in distinguishing between usability related bugs reports and feature requests, may lead to inconsistencies in the classification process. (3) We opted not to utilize advanced models like GPT-4 as the current token-based pricing model for GPT-4 Turbo is prohibitively expensive for large-scale experiments, rendering it unfeasible. However, we plan to explore similar open-source models that are comparable to GPT-4 in the future. (4) App reviews often contain informal language, slang, and typo errors, which complicate interpretation. While we are using a refined dataset, some reviews may still be difficult to understand, and the overall quality of the refined reviews remains uncertain. We will continue to improve our dataset to address these challenges.

References

Assem Al-Hawari, Hassan Najadat, and Raed Shatnawi. 2021. Classification of application reviews into software maintenance tasks using data mining techniques. *Software Quality Journal*, 29(3):667–703.

Nahed Alsaleh, Reem Alnanih, and Nahed Alowidi. 2025. Hybrid deep learning approach for automating app review classification: Advancing usability metrics classification with an aspect-based sentiment analysis framework. *Computers, Materials & Continua*, 82(1).

Bassam Alsanousi, Stephanie Ludi, and Hyunsook Do. 2025. Mobile app quality assessment: Leveraging llm-driven data augmentation for enhanced user reviews analysis. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, pages 1754–1756.

Adailton Araujo, Marcos Golo, Breno Viana, Felipe Sanches, Roseli Romero, and Ricardo Marcacini. 2020. From bag-of-words to pre-trained neural language models: Improving automatic classification of app reviews for requirements engineering. In *Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, pages 378–389. SBC.

Rajendra Budake, Sudhakar Bhoite, and Kabir Kharade. 2023. Identification and classification of functional and non-functional software requirements using machine learning. In *AIP Conference Proceedings*, volume 2946. AIP Publishing.

Laura Ceci. 2022. Global reviews of top android apps by category 2022.

Mohit Chaudhary, Chirag Jain, and Preethu Rose Anish. 2025. Exploring zero-shot app review classification with chatgpt: Challenges and potential. *arXiv preprint arXiv:2505.04759*.

Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2023. Mining and searching app reviews for requirements engineering: Evaluation and replication studies. *Information Systems*, 114:102181.

Dev Dave and Vaibhav Anu. 2022. Identifying functional and non-functional software requirements from user app reviews. In *2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–6. IEEE.

Roger Deocadez, Rachel Harrison, and Daniel Rodriguez. 2017. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th international requirements engineering conference workshops (REW)*, pages 367–371. IEEE.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Sadik Yasin Eftee, Maman Yusuf Khan, Raiyan Noor, Hasan Mahmud, and Md Kamrul Hasan. Extraction of app problems and its corresponding user action from user review of apps using few-shot learning. *Available at SSRN 5093717*.

Cuiyun Gao, Yaoxian Li, Shuhan Qi, Yang Liu, Xuan Wang, Zibin Zheng, and Qing Liao. 2022. Listening to users' voice: Automatic summarization of helpful app reviews. *IEEE Transactions on Reliability*, 72(4):1619–1631.

Pawanjit Singh Ghatora, Seyed Ebrahim Hosseini, Shahbaz Pervez, Muhammad Javed Iqbal, and Nabil Shaukat. 2024. Sentiment analysis of product reviews using machine learning and pre-trained llm. *Big Data and Cognitive Computing*, 8(12):199.

Tanmai Kumar Ghosh, Atharva Pargaonkar, and Nasir U Eisty. 2024. Exploring requirements elicitation from app store user reviews using large language models. *arXiv preprint arXiv:2409.15473*.

Xiaodong Gu and Sunghun Kim. 2015. "what parts of your apps are loved by users?" (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 760–770.

Sadeep Gunathilaka and Nisansa de Silva. Automatic analysis of app reviews using llms.

Hui Guo and Munindar P. Singh. 2020. Caspar: Extracting and synthesizing user stories of problems from app reviews. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 628–640.

Mohammad A Hadi and Fatemeh H Fard. 2023. Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews. *Empirical Software Engineering*, 28(4):88.

Mariem Haoues, Raouia Mokni, and Asma Sellami. 2023. Machine learning for mhealth apps quality evaluation: An approach based on user feedback analysis. *Software Quality Journal*, 31(4):1179–1209.

Rashina Hoda. 2021. Socio-technical grounded theory for software engineering. *IEEE Transactions on Software Engineering*, 48(10):3808–3832.

Nishant Jha and Anas Mahmoud. 2019. Mining non-functional requirements from app store reviews. *Empirical Software Engineering*, 24:3659–3695.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Timo Johann, Christoph Stanik, Alireza M. Alizadeh B., and Walid Maalej. 2017. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 21–30.

Kamaljit Kaur and Parminder Kaur. 2023a. Bert-rcnn: an automatic classification of app reviews using transfer learning based rcnn deep model.

Kamaljit Kaur and Parminder Kaur. 2023b. Mnor-bert: multi-label classification of non-functional requirements using bert. *Neural Computing and Applications*, 35(30):22487–22509.

Arina Kostina, Marios D Dikaiakos, Dimosthenis Stefanidis, and George Pallis. 2025. Large language models for text classification: Case study and comprehensive review. *arXiv preprint arXiv:2501.08457*.

Mengmeng Lu and Peng Liang. 2017. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st international conference on evaluation and assessment in software engineering*, pages 344–353.

Walid Maalej and Hadeer Nabil. 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.

William J Martin. 2017. *App store analysis for software engineering*. Ph.D. thesis, UCL (University College London).

Rhodes Massenon, Ishaya Gambo, Roseline Oluwaseun Ogundokun, Ezekiel Adebayo Ogundepo, Sweta Srivastava, Saurabh Agarwal, and Wooguil Pak. 2024. Mobile app review analysis for crowdsourcing of software requirements: a mapping study of automated and semi-automated tools. *PeerJ Computer Science*, 10:e2401.

Zulfiqar Ali Memon, Nida Munawar, and Maha Kamal. 2024. App store mining for feature extraction: analyzing user reviews. *Acta Scientiarum. Technology*, 46(1).

Montassar Ben Messaoud, Ilyes Jenhani, Nermine Ben Jemaa, and Mohamed Wiem Mkaouer. 2019. A multi-label active learning approach for mobile app user review classification. In *Knowledge Science, Engineering and Management: 12th International Conference, KSEM 2019, Athens, Greece, August 28–30, 2019, Proceedings, Part I 12*, pages 805–816. Springer.

Kittisak Phetrungnapha and Twittie Senivongse. 2019. Classification of mobile application user reviews for generating tickets on issue tracking system. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, pages 229–234. IEEE.

Zhilei Qiao, Alan Wang, Alan Abrahams, and Weiguo Fan. 2020. Deep learning-based user feedback classification in mobile app reviews.

Md Abdur Rahman, Oishi Mahmud, Nishat Tasnim Niloy, and Md Saeed Siddik. 2020. Prioritize android app reviews for effective version release. *ASM Sci. J.*, 13:21–30.

Rabab Emad Saudy, Alaa El Din El-Ghazaly, Eman S Nasr, and Mervat H Gheith. 2021. A proposed keyword-based feature extraction approach for labeling and classifying egyptian mobile apps arabic slang user requirements reviews. In *National Conference on Big Data Technology and Applications*, pages 24–37. Springer.

Gian Luca Scoccia, Romina Eramo, and Marco Autili. 2023. Studying users' perception of iot mobile companion apps. *Pervasive and Mobile Computing*, 92:101786.

Faiz Ali Shah, Ahmed Sabir, Rajesh Sharma, and Dietmar Pfahl. 2025. How effectively do llms extract feature-sentiment pairs from app reviews? In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 123–138. Springer.

Amrita Singh, Chirag Jain, Mohit Chaudhary, and Preethu Anish. 2024. Refining app reviews: Dataset, methodology, and evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 595–608.

Khandaker Rifah Tasnia, Nafisa Mehjabin, Md Ashif Aziz, Abu Raihan Mostofa Kamal, Afrina Khatun, and Lutfun Nahar Lota. 2022. Dcnr: A transfer learning approach for detecting and classifying non-functionalrequirements from user reviews.

Mohammadali Tavakoli, Liping Zhao, Atefeh Heydari, and Goran Nenadić. 2018. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. *Expert Systems with Applications*, 113:186–199.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Ioannis Triantafyllou, Ioannis C Drivas, and Georgios Giannakopoulos. 2020. How to utilize my app reviews? a novel topics extraction machine learning schema for strategic business purposes. *Entropy*, 22(11):1310.

Raja Vavekanand and Kira Sam. 2024. Llama 3.1: An in-depth analysis of the next-generation large language model.

Tianlu Wang, Peng Liang, and Mengmeng Lu. 2018. What aspects do non-functional requirements in app user reviews describe? an exploratory and comparative study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 494–503. IEEE.

Abdulsamad E Yahya, Atef Gharbi, Wael MS Yafooz, and Arafat Al-Dhaqm. 2023. A novel hybrid deep learning model for detecting and classifying non-functional requirements of mobile apps issues. *Electronics*, 12(5):1258.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Hui Yang and Peng Liang. 2015. Identification and classification of requirements from app user reviews. In *SEKE*, pages 7–12. Citeseer.

Kanwal Zahoor and Narmeen Zakaria Bawany. 2024. Explainable artificial intelligence approach towards classifying educational android app reviews using deep learning. *Interactive Learning Environments*, 32(9):5227–5252.

Jianzhang Zhang, Jialong Zhou, Jinping Hua, Nan Niu, and Chuang Liu. 2025. Mining user privacy concern topics from app reviews. *Journal of Systems and Software*, page 112355.

Najla Zuhir, Amna Mohammad Salim, Parvathy Premkumar, and Moshiur Farazi. 2025. Beyond stars: Bridging the gap between ratings and review sentiment with llm. *arXiv preprint arXiv:2509.20953*.

A Dataset Statistics

In Appendix A, we display the distribution of review lengths for the gold and silver corpora in Figure 3, ensuring clarity and conciseness while preserving the key information.

Following this, we present the label co-occurrence heatmap for the gold-standard corpus in Figure 4. From Figure 4, it can be observed that darker regions indicate frequent label co-occurrences, such as App Satisfaction & Feedback_Unsatisfied with Bug Report subcategories, highlighting the multi-faceted nature of user feedback where dissatisfaction often accompanies specific technical complaints.

Finally, we show the label cardinality distribution for both corpora in Figure 5. From Figure 5, it is clear that approximately 60% of gold-standard reviews receive 1–2 labels, while 30% have 3 labels, demonstrating the prevalence of multi-label cases and validating the need for a multi-label classification framework.

B Hyperparameters and Examples

In Appendix B, we report the results of the cross-domain generalization evaluation. Specifically, we assess the robustness of the proposed framework by performing a domain-held-out evaluation, where reviews from one application domain are excluded during training and used exclusively for testing, while the model is trained on the remaining domains. All experiments are conducted using our best-performing model, LLaMA-3.1-8B. The performance of the domain-held-out experiments is summarized in Table 7, which reports weighted-F1 scores at both Level-1 and Level-2 of the taxonomy. From Table 7, it can be observed that Level-1 weighted-F1 scores remain consistently high across all eleven held-out domains (ranging from 0.85 to 0.93), while Level-2 scores remain stable (ranging from 0.73 to 0.82) despite the increased granularity of the classification. This suggests that performance degradation under domain shift is modest, indicating that the model captures generalizable semantic patterns in app reviews rather than relying on domain-specific cues. Additionally, Table 9 illustrates representative examples of app reviews from the dataset together with their corresponding classifications. Next, the specifications of hyperparameters and configurations for LLMs are given in Table 8. Finally, detailed category-wise inter-annotator agreement statistics are reported in 6.

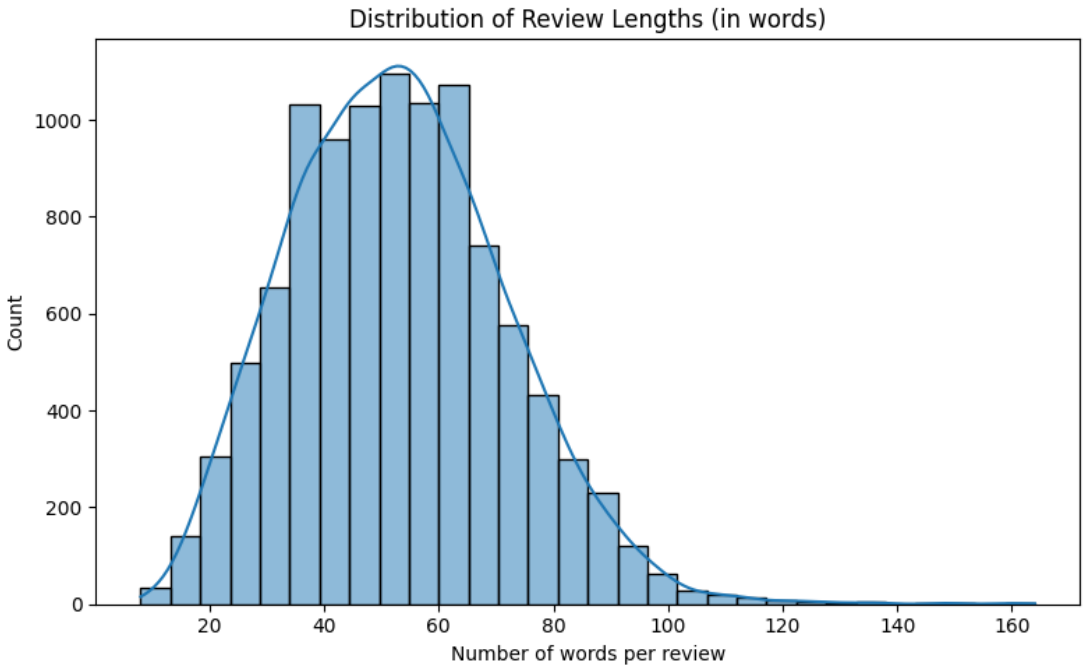
Level 1 Class	Level 2 Class	Agreement
Bug Report (0.75)	Performance & Stability	0.82
	Supportability	0.76
	Sync & Compatibility	0.71
	Privacy & Security	0.73
	Usability	0.77
	Others	0.68
Feature Request (0.70)	Performance & Stability	0.73
	Supportability	0.70
	Sync & Compatibility	0.72
	Privacy & Security	0.69
	Usability	0.68
	Others	0.67
App Satisfaction & Feedback (0.84)	Satisfied	0.82
	Unsatisfied	0.85
	Overall Agreement: 0.76	

Table 6: Classwise Fleiss' Kappa Agreement by Level 1 and Level 2 Classes

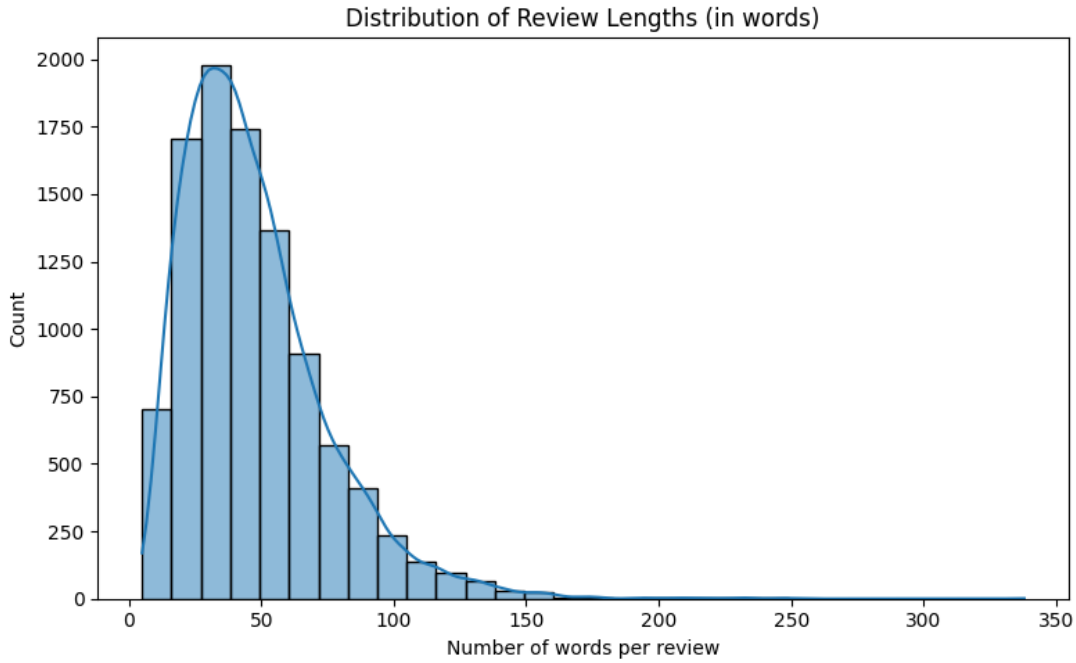
Held-out Domain	Level-1 Weighted-F1	Level-2 Weighted-F1
YouTube	0.92	0.81
WhatsApp	0.89	0.74
Uber	0.88	0.75
Twitter	0.92	0.78
Spotify	0.90	0.80
Privacy Reviews	0.87	0.76
Netflix	0.89	0.77
Google Fit	0.90	0.74
Duolingo	0.90	0.74
Candy Crush Saga	0.85	0.73
Amazon Shopping	0.93	0.82

Table 7: Cross-domain generalization results using LLaMA-3.1-8B under a domain-held-out evaluation setting.

s



(a) Gold Corpus

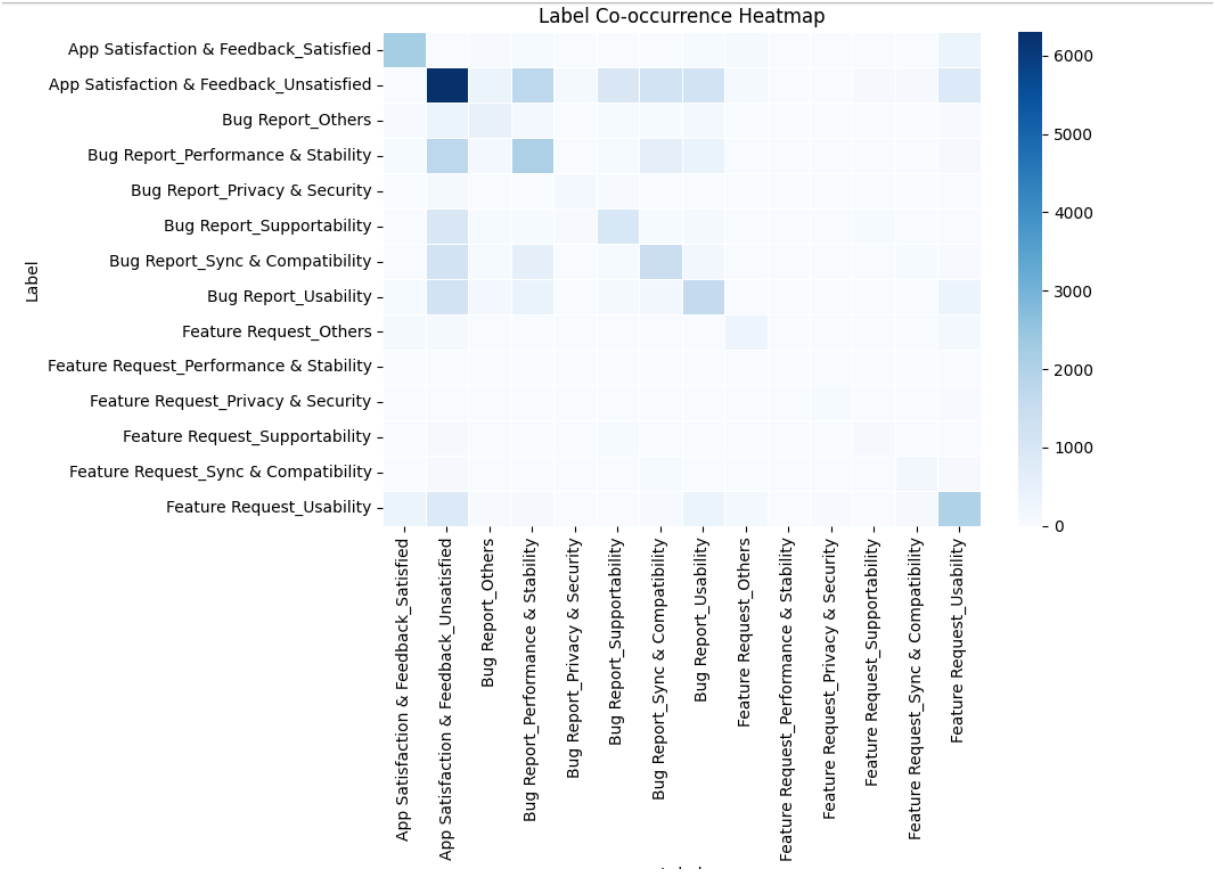


(b) Silver Corpus.

Figure 3: Distribution of Review Lengths (Gold and Silver Corpus).

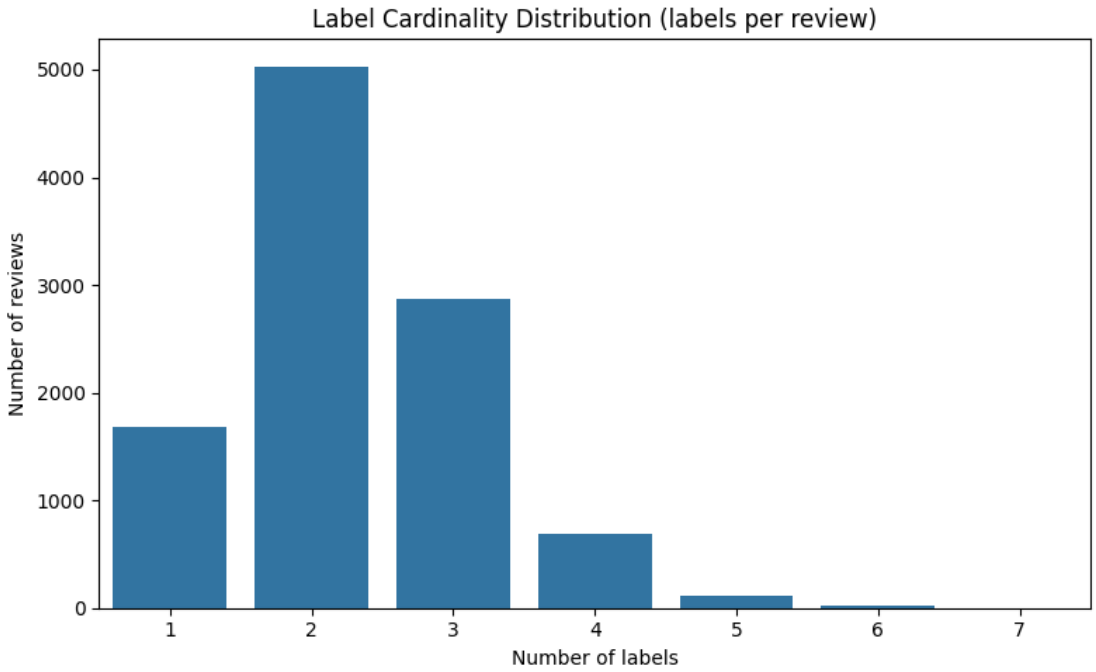


(a) Gold Corpus

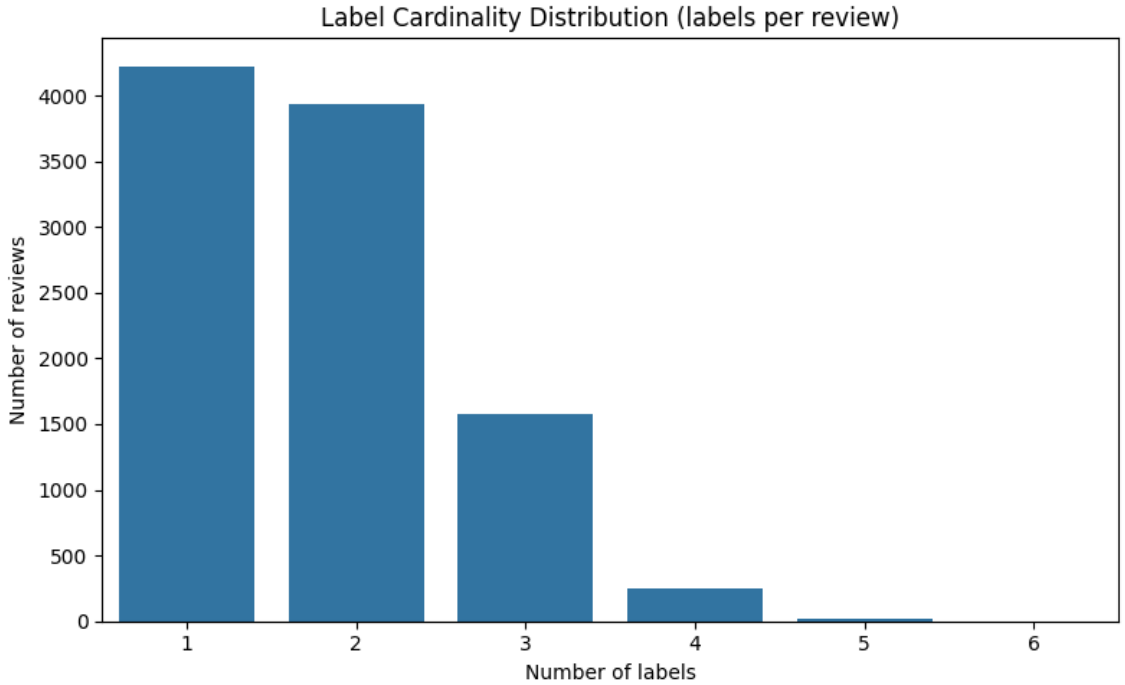


(b) Silver Corpus

Figure 4: Label Co-occurrence Heatmap (Gold and Silver Corpus)



(a) Gold Corpus



(b) Silver Corpus

Figure 5: Label Cardinality Distribution (Gold and Silver Corpus).

Model	Hyperparameters
LLaMA 3.1	per_device_train_batch_size=4
	num_train_epochs=10
	learning_rate=1e-5
	weight_decay=0.01
	save_steps=345
	logging_steps=345
Mistral	per_device_train_batch_size=4
	num_train_epochs=12
	learning_rate=3e-4
	weight_decay=0.01
	save_steps=500
	logging_steps=500
Gemma	num_train_epochs=10
	per_device_train_batch_size=4
	gradient_accumulation_steps=1
	logging_steps=345
	save_steps=345
	learning_rate=1e-4
	weight_decay=0.01
Qwen	per_device_train_batch_size=4
	num_train_epochs=10
	learning_rate=5e-5
	weight_decay=0.01
	save_steps=500
	logging_steps=500
Bert	TRAIN_BATCH_SIZE = 16
	TEST_BATCH_SIZE = 16
	EPOCHS = 65
	LEARNING_RATE = 1e-5
	THRESHOLD = 0.5
Random Forest	N_estimators=100
	Criterion='gini'
	Bootstrap=True
	Random_state=42
Decision Tree	Criterion='gini'
	Random_state=42
SVM	Kernel='rbf'
	C=1
	Gamma='scale'
	Random_state=42
Logistic Regression	Penalty='l2'
	Solver='lbfgs'
	Max_iter=1000
	Random_state=42
XGBoost	Booster='gbtree'
	Learning_rate=0.3
	Max_depth=6
	Random_state=42

Table 8: Hyperparameter Configurations of Fine-tuned LLMs and Traditional Models

Review	Classification
Generally satisfied with Amazon's service, I primarily shop on their website instead of using the app. However, upon trying the Android app, I noticed its slowness and need for optimization. Although the user interface is organized, navigating between tabs and options feels sluggish. Comparatively, the website is faster and more responsive. Hoping for improved app performance from Amazon.	App Satisfaction & Feedback: [Satisfied], Bug Report: [Performance & Stability], Feature Request: [Performance & Stability]
Struggling to get it functioning properly. The step count fails to initiate upon movement, causing frustration. The setup process is cumbersome. Upon reviewing the workout, no steps have been recorded. Disappointing experience, another subpar app.	App Satisfaction & Feedback: [Unsatisfied], Bug Report: [Usability, Performance & Stability]
Picture-in-picture isn't functioning properly. When you exit the app, the video stops, and there's no pop-up window on the home screen for Netflix with the video I was watching. This is unlike my experience with Disney+, Amazon Prime, Hulu, and HBO Max, where a pop-up window appears when I hit the home button. I attempted to reach out to Netflix about this issue, but the response I received via email was to call customer service as if they either developed the app or handle bug fixes.	App Satisfaction & Feedback: [Unsatisfied], Bug Report: [Usability, Supportability], Feature Request: [Usability]
Recently, the app has been experiencing glitches. Over the past few months, my music randomly stops, and the app crashes, clearing the queue. Additionally, the queue limit is frustrating, lacking a "play next" option similar to Google Play Music. Song radio often displays songs already in my library, deviating from the intended purpose of discovering new, similar music. The feature rarely introduces songs not already in my collection.	App Satisfaction & Feedback: [Unsatisfied], Bug Report: [Performance & Stability, Others], Feature Request: [Usability]
The Spaces feature frequently crashes or distorts speakers, requiring them to rejoin, particularly in regular users' Spaces, not affecting those deemed important. Seeking help seems futile; the help page appears ornamental. Notifications arrive extremely late, sometimes after 18 hours or days, despite settings. Space hosts shouldn't block listeners; it's unfair. If disliked, they can prevent you from speaking.	App Satisfaction & Feedback: [Unsatisfied], Bug Report: [Performance & Stability, Supportability], Feature Request: [Usability]

Table 9: Examples of App Reviews with Assigned Classifications (Level 1 and Level 2)