CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

User feedback to actionable insight: Generative AI for user feedback analysis

Master's Thesis in Computer science and engineering

Divya Chandrabose

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

User feedback to actionable insight: Generative AI for user feedback analysis

Divya Chandrabose

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

User feedback to actionable insight: Generative AI for user feedback analysis
Divya Chandrabose

Supervisor: Farnaz Fotrousi, Department of Computer Science and Engineering
Examiner: Eric Knauss, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

User feedback to actionable insight: Generative AI for user feedback analysis
Divya Chandrabose
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Software teams rely heavily on user feedback to understand how people experience their applications. However, the unstructured and high-volume nature of app reviews makes manual analysis increasingly difficult. Although recent research has shown that Large Language Models are able to perform classification tasks related to user feedback, it is still unclear if it is possible to leverage such models to produce actionable insights that are applicable to software development.

This thesis explores the capability of generative AI to extract actionable insights from user feedback through a mixed-methods approach. In the qualitative phase, semi-structured interviews with five industry practitioners are conducted to establish a grounded definition of an actionable insight and identify the operational challenges of manual triage. In the experimental phase, multiple LLMs, including ChatGPT, Gemini, and Mistral AI are evaluated using prompt engineering techniques on the public REFSQ-2026 app review dataset. The research examines how well these LLMs perform when processing raw or unclassified data as opposed to classified data. Finally, the AI-generated insights are compared directly against a human analyst. This baseline was established by manually extracting insights from the dataset using the four-point criteria developed in the qualitative phase of the study, allowing for a systematic comparison to identify areas of alignment, divergence, and potential AI blind spots.

From the results it is clear that the LLMs can successfully match human expert baselines across issue recall, semantic similarity, and structural formatting, reducing processing time from hours to seconds. Furthermore, the evaluation reveals that classification labels impact each model differently: ChatGPT achieves maximum issue recall but degrades in formatting, Mistral AI performs optimally on raw text and degrades under structural constraints, and Gemini trades slight semantic precision for higher bug discovery. This research provides an evidence-based understanding of how software teams can reliably select and deploy LLMs to translate noisy user feedback into structured, developer-ready tasks.

Keywords: Requirements Engineering, User Feedback, Large Language Models, Actionable Insights, App Reviews, Prompt Engineering

Acknowledgements

I would like to express my sincere gratitude to my university supervisor, Farnaz Fotrousi, for her invaluable guidance, encouragement, and support throughout this research. Her expertise and constructive feedback have played a crucial role in shaping this work, both academically and practically.

Furthermore, I extend my appreciation to my examiner, Eric Knauss, for his time, constructive evaluation, and thoughtful suggestions, which have helped ensure the quality and rigor of this study. A special thank you also goes to the industry professionals who generously volunteered their time to participate in the expert interviews; their practical insights were foundational to the direction of this research.

Lastly, I would like to thank my family, friends, and peers for their constant support, patience, and encouragement during this journey.

Divya Chandrabose, Gothenburg, June 2026

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Problem Description . . . 2
1.2 Research questions . . . 3
1.3 Purpose of the Study . . . 4
1.4 Significance of the Study . . . 4
1.5 Thesis Outline . . . 5

2 Background **7**
2.1 User Feedback in Requirements Engineering . . . 7
2.2 From Raw Data to Actionable Insights . . . 7
2.3 Generative AI and Large Language Models (LLMs) . . . 8

3 Related Work **11**

4 Methods **15**
4.1 Expert Interviews . . . 16
4.1.1 Data Collection: . . . 16
4.1.2 Data Analysis: . . . 17
4.2 Experimental Method . . . 20
4.2.1 Data Collection and Preparation . . . 21
4.2.2 Data Analysis . . . 23

5 Results **27**
5.1 Defining an Actionable Insight (Answer to RQ1) . . . 27
5.1.1 Theme 1: Characteristics of an Actionable Insight . . . 28
5.1.2 Theme 2: Filters for Actionability (Decision Making) . . . 29
5.1.3 Theme 3: Challenges in Manual Processing . . . 30
5.2 LLM Performance in Insight Generation and Comparison to Human Baseline . . . 30
5.2.1 LLM Performance in Insight Generation (Answer to RQ2) . . . 31
5.2.2 Comparing LLM-Generated Insights to a Human Baseline (Answer to RQ3) . . . 33
5.2.3 Statistical Hypothesis Testing . . . 35

6 Discussion **37**
6.1 Defining Insights and Bottlenecks 37
6.2 Interpreting Model Behaviors 38
6.3 Implications for Software Engineering Workflows 38
6.4 Advancing the State of the Art 39
6.5 Threats to Validity and Study Limitations 40

7 Conclusion **43**
7.1 Synthesis of Core Findings: 43
7.2 Contributions to the Field: 44
7.3 Future Research: 44
7.4 Final Thoughts: 44

Bibliography **47**

A Appendix **I**
A.1 BERT score calculation I
A.2 Code for Shapiro-Wilk test III
A.3 Code for Wilcoxon Signed-Rank Test III
A.4 Human Baseline ("Ground Truth") IV

List of Figures

4.1 Overview of the Research Methodology 15
4.2 A systematic thematic analysis process 17
4.3 Keywords identified from expert interview results 18

5.1 Issue Coverage Across Classified and Unclassified Conditions for Different Datasets and LLM Models 31
5.2 Impact of Data Classification on Issue Coverage Across LLM Models 32
5.3 Average Structural Actionability Scores 33
5.4 Quantitative Comparison of AI Outputs against Human Baseline . . 34
5.5 Summary of Statistical Testing for BERT Score (Semantic Similarity) and Actionability Score (Structural Formatting) 35

A.1 Qualitative Coding Codebook I

List of Tables

1

Introduction

Software teams rely heavily on user feedback to understand how people experience their applications. Reviews and comments on the Google Play Store, Apple App Store, social media, and online forums are full of information that can help identify bugs, missing features, or general problems with usability [13][9]. Sometimes users also point out what works well and what they enjoy about the app. All of this information is useful not only for planning future updates but also for guiding the continuous improvement of the software over time. The problem is that the amount of feedback has grown so much that developers cannot always keep up [13]. Most reviews are written in different styles, sometimes in a hurry, and often without clear structure. Because of this, it becomes difficult to process them manually. Earlier studies also point out this issue. For example, Maalej and Nabil [13] showed that even separating reviews into basic groups such as bug reports, feature requests, or praise takes a lot of manual effort. Guzman et al. [9] found similar challenges when looking at software-related posts on Twitter, since many messages are unclear or lack context.

Traditional Natural Language Processing (NLP) methods have been used in the past to support the classification and analysis of user feedback. These include keyword searches, topic models, or simple machine learning classifiers [23]. While these methods help to some extent, they usually fail when the language becomes more complex [9]. They often miss the real meaning behind the text or struggle when a review contains more than one issue. Other research also points out that these older approaches do not perform well when the dataset is small or very diverse [14]. In the last few years, Large Language Models have introduced new possibilities for dealing with this type of data. Models such as ChatGPT, Gemini, and Mistral Large can understand natural language much better than traditional NLP techniques. Recent studies show that they can recognise detailed sentiment in reviews [20] and identify different kinds of requirements directly from user feedback [3]. Some work also suggests that LLMs can organise larger sets of reviews into themes or trends, which can help teams understand long-term user concerns [12].

Definition (Actionable Insight):
Jørnø and Gynther [11] describe an actionable insight as one that can support growth or guide change, meaning it has practical value for decision-makers rather than being purely descriptive.

Bhattacharjee et al. [6] described actionable insight as an action-oriented theme extracted from user feedback that captures what users want to do, what issues they encounter, or what they request, expressed as a concise phrase that can directly guide product or engineering decisions.
Susaiyah et al. [21] proposed a schema for generating actionable insights automatically, where an insight is treated as a structured unit containing a specific issue, supporting evidence, and a suggested action.

A useful way to understand this is through an example. Consider a set of raw user reviews such as the following:

- "The app freezes every time I try to upload a photo."
- "Uploads take too long, especially on mobile data."
- "Sometimes the upload button doesn't respond at all."

A descriptive summary of these reviews says, "Users report problems during photo uploads."
In contrast, an actionable insight [11][6] would state: "Improve the photo upload process by addressing repeated app freezes, unresponsive upload controls, and slow upload performance under mobile networks."

1.1 Problem Description

Many tools and techniques exist for analysing user feedback, but they mainly cover tasks such as classification or sentiment analysis. We still do not have a clear understanding of how well Large Language Models can generate deeper insights, which is why this study focuses specifically on evaluating their insight-generation ability. Earlier work has focused mainly on classifying reviews or detecting sentiment, such as grouping feedback into bugs, feature requests, or praise [13][9]. More recent studies show that LLMs can perform these tasks with better accuracy than traditional NLP approaches [20][3][12]. However, these studies rarely look beyond classification. They do not examine whether the information produced by the model is meaningful enough to guide real decisions in software development.

At the same time, research outside software engineering shows that an insight is only valuable when it can support action or improvement [11]. Insights usually combine observations with interpretation, and sometimes they involve comparing patterns across different situations to understand what is changing or why [21][10]. This type of reasoning is very different from simply labeling text or summarising it. Because of this, it is still unclear whether LLMs can produce insights that match the depth, judgement, and usefulness expected by human analysts.

For instance, consider the user review [18]: "It won't load any of my notifications when I click on them; pictures won't load and my newsfeed won't refresh." A traditional classification would simply tag this review as a "Bug Report." A summary

might condense it to “User experiences loading issues with notifications, pictures, and newsfeed.” An actionable insight [11][6], however, goes further by identifying what the user wants and why it matters. For example, “Fix notification, picture, and newsfeed loading issues to improve app reliability and user experience.”

Another gap in the existing research is that we do not know how stable or consistent LLM outputs are. For example, we do not know whether different LLMs would interpret the same user review in the same way or whether their performance changes when the reviews come from different domains. Most studies focus on a single model or a single type of dataset [15], which leaves questions about domain knowledge and cross-domain performance unanswered.

Finally, very little is known about how LLM-generated insights compare directly to insights created by humans. Human analysts rely on experience, context, and interpretation, while LLMs rely on patterns learned from training data. Whether these two forms of analysis lead to similar or very different insights has not been studied in detail.

Because of these gaps, it remains uncertain to what extent software teams can rely on LLM-generated insights. This makes it difficult to understand when these models can support human judgement, when they might miss important information, and how they should be used in practice.

This thesis aims to address this problem by examining how LLMs identify actionable insights, how two different models perform across multiple application domains, and how their outputs compare to the insights produced by human analysts.

1.2 Research questions

RQ1: How do requirements experts characterise an actionable insight extracted from user feedback, and what challenges do they identify in this process?

This qualitative question aims to establish a practitioner-grounded definition of an "actionable insight" and challenges faced by experts in the process. By understanding how industry experts define this concept, the study can identify the specific manual bottlenecks teams face when attempting to generate these insights from raw user feedback, providing a foundation for evaluating LLM solutions.

RQ2: How do large language models perform in generating actionable insights from unclassified user feedback compared to classified user feedback?

This question investigates the computational capabilities of LLMs in handling raw data. Specifically, it seeks to determine whether preprocessed (classified) user feedback significantly improves an LLM’s ability to extract useful insights or if modern models can process unstructured feedback just as effectively.

RQ3: How do insights generated by large language models compare to insights produced by human analysts when both analyse the same user feedback?

This question evaluates the practical viability of AI adoption in requirements engineering. By benchmarking the relevance and actionability of LLM-generated insights against a human baseline, the study assesses how closely AI can replicate or augment human analytical performance.

1.3 Purpose of the Study

The purpose of this study is to explore how well different Large Language Models can generate meaningful and actionable insights from user feedback and how their results compare to those produced by human analysts. The study aims to clarify what actionable insight means in the context of app reviews and to examine whether LLMs can identify such insights in a consistent and reliable way. It also evaluates several LLMs, both closed-source and open-source models, to see how they interpret the same set of user reviews and how their outputs differ when the feedback is provided in its raw form versus when it has been classified. Finally, the study looks at how the insights created by these models differ from human-generated insights. Taken together, this work aims to give a clearer understanding of when and how LLMs can support software teams in analysing user feedback and making informed decisions about product improvement.

1.4 Significance of the Study

Although recent studies show that Large Language Models (LLMs) perform well when classifying user feedback, identifying requirements, or summarising large volumes of reviews [20][3][12], it is still not clear how well these models can produce actionable insights as defined above. Much of the existing research evaluates model performance using metrics such as classification accuracy or sentiment prediction [13][9][14]. Far less attention is given to whether the resulting outputs are useful for guiding decisions in software development practice [11][6].

In contrast, human analysts typically rely on interpretation, contextual knowledge, and professional judgement when turning raw feedback into concrete development actions [11].However, most prior studies do not examine whether LLM-generated insights reflect these qualities. While recent work explores automated or LLM-based analysis of user feedback [3][16], direct comparisons between insights produced by LLMs and those created by human analysts remain limited. As a result, it is still uncertain whether LLMs can move beyond largely descriptive analysis and generate insights that are genuinely actionable and comparable to human judgement. This lack of understanding makes it difficult to assess when and how LLMs can be reliably used to support decision-making in real software development settings.

Aim and Objectives:

The aim of this thesis is to understand how different Large Language Models interpret user feedback and how their insights compare to those created by human analysts. The study looks at whether these models can reliably extract useful, actionable points from real user reviews and how their results change when the feedback is structured (classified) versus unstructured. The broader goal is to see how LLMs could be used in everyday software development work, especially when teams need to make sense of large amounts of user feedback.

The objectives of this thesis are outlined below:

- Define actionable insights from requirements engineering perspective, explore how requirements experts characterise actionable insights extracted from user feedback, and identify the challenges they face during this process [RQ1].
- Examine the effectiveness of different Large Language Models including OpenAI's ChatGPT, Google Gemini, and Mistral AI Le Chat in generating actionable insights from the same set of user reviews [RQ2].
- Evaluate how model performance changes when feedback is provided in different forms, comparing insights generated from raw, unstructured reviews with insights produced when the same reviews are pre-classified into categories such as bug reports, feature requests, or usability issues [RQ2].
- Compare the insights produced by LLMs with those written by human analysts to identify areas of alignment, divergence, and potential gaps. The goal is to understand when LLMs can support or augment human judgement in requirements engineering and user feedback analysis [RQ3].

1.5 Thesis Outline

This thesis is structured into seven chapters, each addressing different aspects of the research, from background and theoretical foundations to methodology, results, and conclusions.

- Chapter 1 - Introduction: It gives a brief outline of the research topic, the problem description, the purpose, and the significance of the study. It introduces the research questions concerning the extraction of actionable insights from user feedback using generative AI and provides an overview of the thesis structure.
- Chapter 2 – Background: It deals with fundamental concepts related to requirements engineering, the hierarchy of user feedback, actionable insights, and the underlying mechanics of Large Language Models (LLMs). It provides the theoretical basis needed for proper comprehension of the subsequent research.

- Chapter 3 – Related Work: This chapter studies existing literature and tools related to automated user feedback analysis, traditional natural language processing (NLP) approaches for app review classification, and recent AI-driven requirement extraction methods. It identifies research gaps and positions this study within the context of current academic literature.

- Chapter 4 – Methods: Details the mixed-methods research design utilized in this study. It encompasses the qualitative approach used for the expert interviews and the computational experimental setup for evaluating LLMs using prompt engineering on the REFSQ-2026 user review dataset.

- Chapter 5 – Results: Presents the findings from the research. It first provides the practitioner-grounded definition of an actionable insight derived from thematic analysis, followed by the experimental results evaluating LLM performance across different data conditions (unclassified vs. classified) and comparing AI-generated insights against a human baseline.

- Chapter 6 – Discussion: Analyzes the experimental and qualitative results, interprets their implications for modern software development teams, discusses the limitations of the study, and evaluates the practical feasibility of using LLMs to triage user feedback in real-world workflows.

- Chapter 7 – Conclusion: Summarizes the essential findings of the study, emphasizes its contributions to AI-assisted software requirements engineering, and suggests possible avenues for future research.

2

Background

This chapter provides the theoretical foundation and context necessary to understand the subsequent methodology and experiments. It introduces the role of user feedback in modern software engineering, defines the concept of actionable insights, and explains the underlying mechanics of Large Language Models (LLMs).

2.1 User Feedback in Requirements Engineering

In modern software development paradigms, such as Agile and Continuous Integration/Continuous Deployment (CI/CD), software is no longer treated as a finished product upon release. Instead, it evolves continuously based on how users interact with it [23]. Consequently, traditional methods of gathering requirements, such as focus groups or formal stakeholder meetings, have been heavily supplemented by continuous, data-driven approaches.

For instance, online platforms such as Apple App Store, Google Play Store, and user forum sites produce a huge volume of unsolicited user feedback. For Requirements Engineering (RE) practitioners, this is a goldmine [13]. This is because user feedback from these sites reveals existing bugs and user needs that were not previously identified. However, the unstructured, informal, and high volume of user feedback in text form makes it difficult to analyse manually [9]. The primary challenge in modern RE is no longer acquiring user feedback, but rather filtering the noise to extract meaningful data that can guide development sprints and product roadmaps.

2.2 From Raw Data to Actionable Insights

To process large volumes of feedback, software teams often categorise data using a hierarchy of utility:

- Raw Data: The exact, unstructured text written by the user (e.g., "App crashes when I click the camera.").
- Information (or Categorisation): Data that has been organised or labelled (e.g., tagging the previous text as a "Bug Report" or "Severity: High") [13].

- Insight: A deeper understanding derived from the information that explains the root cause of a problem or a systemic pattern [16].
- Actionable Insight: The highest tier of data utility. An insight becomes "actionable" when it is structured in a way that allows decision-makers to immediately execute a change [11] [21].

Though traditional sentiment analysis or topic modelling algorithms are extremely effective at classifying data into basic "Information" categories, they are often unable to produce "Actionable Insights" [14]. The traditional approach to creating an actionable insight is to apply contextual reasoning by making a link between a user's frustration and a particular gap in a software workflow, while also suggesting a practical change to a software system [6]. Traditionally, this requires the domain expertise of software practitioners, such as product managers, requirements engineers, or developers.

2.3 Generative AI and Large Language Models (LLMs)

The conventional Natural Language Processing (NLP) techniques employed in software engineering follow statistical models. The most common ones employed are the "Bag of Words" (BOW) and the "Support Vector Machines" (SVM). Though these techniques perform well in processing the text and analyzing the frequency of words or specific syntactic structures, they lack contextual awareness and struggle with sarcasm, implicit requests, or multi-issue sentences [9] [13].

In contrast, Generative Artificial Intelligence systems are fueled by Large Language Models (LLMs), which are based on deep learning architectures of the "Transformer" type. While Generative Artificial Intelligence systems classify text based on the frequency of keywords used in the text, systems such as OpenAI's GPT, Google's Gemini, and Mistral AI are trained on vast amounts of human language. This enables them to comprehend the semantic relationships between words and produce very cohesive text [20].

For the purposes of RE, LLMs constitute a substantial step in technology development. Various studies have already started investigating their utility in several requirements engineering processes. For instance, finding ambiguities and discrepancies in formalised requirements [4] and establishing traceability links between different software artefacts [2] have been explored by researchers. With respect to the particular area of user feedback, the recent literature [19] contains examples of generative models outperforming other machine learning algorithms in classifying app reviews into bug reports or features requests.

These models have high contextual reasoning power, they are theoretically capable of overcoming the hurdle of classification and actually diagnosing the "Why" of user feedback and developing a plan of action in RE [3]. This research seeks to investigate

how well these models can bridge the gap between raw data and RE.

3

Related Work

An insight is commonly understood as a meaningful interpretation of data that reveals patterns, relationships, or behaviours and helps explain why certain phenomena occur [5]. It goes beyond simply reporting information and instead reveals something meaningful that helps explain what is happening and why. Härmä and Helaoui [10] add that an insight can often be expressed by comparing measurements taken in two different but comparable contexts. Differences between these contexts highlight important trends, changes, or issues that may not be visible through raw data alone.

One of the foundational studies in this field is the work of Maalej and Nabil [13], who explored how app reviews can be classified into meaningful categories such as bug reports, feature requests, and praise [13]. Their study showed that user feedback often contains multiple issues within a single review and that developers must spend considerable time interpreting these messages. They found that automatic classifiers based on conventional text-mining approaches often struggled when reviews were unstructured or ambiguous. This work demonstrates both the value and the difficulty of categorising feedback in a reliable way.

Guzman et al. [9] extended this understanding by analysing user discussions about software on Twitter. They showed that social media feedback presents challenges like informality, short message length, and lack of context. These characteristics make it difficult for traditional NLP techniques to correctly interpret user intent. Their study highlighted that understanding user opinions requires more than keyword matching and that context plays a crucial role in analysing feedback effectively.

Work on structured insight generation in data analytics provides a more technical perspective. Susaiyah et al. [21] approach shows that insights can be represented in a way that makes them easier to analyse and compare, especially when dealing with large amounts of text. This structured view mirrors how product teams often organise feedback into themes or categories before making decisions.

Research in natural language processing also contributes to understanding insights. Mukku et al. [16] demonstrated that customer reviews can be converted into structured "insight units" that capture aspects of a product, user sentiment, and relevant evidence from the text. Although their work focuses on general product reviews

rather than software, their definition highlights an important idea: an insight is not just a keyword or topic but a meaningful combination of what the user is talking about, how they feel about it, and why it matters. This concept aligns well with how software teams interpret user reviews to identify common issues or opportunities.

More recent research has examined how advanced neural models can improve insight extraction from reviews. One example is a study on classifying user requirements using deep learning in environments with small datasets [14]. This work reported that deep learning models can perform better than traditional models when enough labeled data is available, but they still struggle with contextual understanding and mixed-content reviews. These limitations suggest the need for methods that can interpret language in a more flexible and human-like way.

A shift toward using Large Language Models for feedback analysis has emerged in the last few years. A recent evaluation of LLMs for fine-grained sentiment analysis of app reviews showed that these models can better detect subtle emotional cues and mixed sentiments compared to earlier approaches [20]. The study demonstrated that LLMs can capture polarity at a detailed level and handle complex writing styles more effectively. This is relevant because sentiment often influences how developers prioritise issues and understand user dissatisfaction.

Another important study examined how LLMs can be used to extract requirements directly from online user feedback [3]. The authors tested several LLMs on tasks such as classification, requirement identification, and specification. The results showed that LLMs performed well in identifying functional and non-functional requirements, even in cases where the feedback was unclear or poorly structured. This indicates that LLMs may help reduce the manual effort needed to turn raw comments into actionable requirements.

Work on trend and narrative analysis has further shown that feedback can be grouped into long-term patterns that reveal emerging issues or themes [12]. The study demonstrated that combining user feedback with LLM-supported analysis can help identify repeated concerns and broader user stories rather than isolated comments. This type of analysis can guide higher-level decision-making about product evolution.

To push the definition of an actionable insight [11] further, we have to look beyond simply categorising data [13]. When a product manager looks at user reviews, they aren't just trying to sort them into buckets like bugs or feature requests. They need to know exactly what the development team should fix. A recent mapping study of over 180 automated tools by Gambo et al. [8] showed that while early sentiment classifiers were good at labelling feedback, they failed to produce backlog-ready items. In other words, they lacked actionability. To make an insight truly actionable in software engineering, it has to bridge the gap between an emotional user complaint and a concrete technical requirement.

For example, a 2025 framework [1] explored how models like GPT-4 and Gemini

can be used to pull specific usability recommendations directly out of negative app reviews [1]. Instead of just flagging a review as negative, the models were prompted to output suggestions tied to specific usability dimensions like correctness and completeness. Similarly, Sakib et al. [19] recently demonstrated that LLMs can take unstructured user comments and map them directly into Agile user stories. Their work proved that language models can match human experts when it comes to formatting requirements.

However, even with these advances in formatting, there is still a missing piece in the literature. We know LLMs can write user stories [19], but we do not fully understand how adding constraints like pre-classified labels affects their ability to actually diagnose the core problems hidden in the text. This study builds on the existing literature by testing whether these classification labels help or hinder different AI architectures when they attempt to extract fully structured, developer-ready insights.

4

Methods

This research follows a mixed-methods approach, combining qualitative explorations with computational experimental design methods [24] to investigate how actionable insights can be generated from user feedback and how Large Language Models (LLMs) perform in this task compared to human analysts. The study examines both the nature of actionable insights and the conditions under which such insights are produced, using established principles from experimental software engineering.

- Qualitative Analysis: Semi-structured interviews with domain experts to define "actionable insights" and understand current challenges in feedback analysis [RQ1].
- Computational Experiment: A computational experiment using Large Language Models (LLMs) to generate insights from public datasets, evaluating the impact of classification labels [RQ2] and comparing performance against human baselines [RQ3].

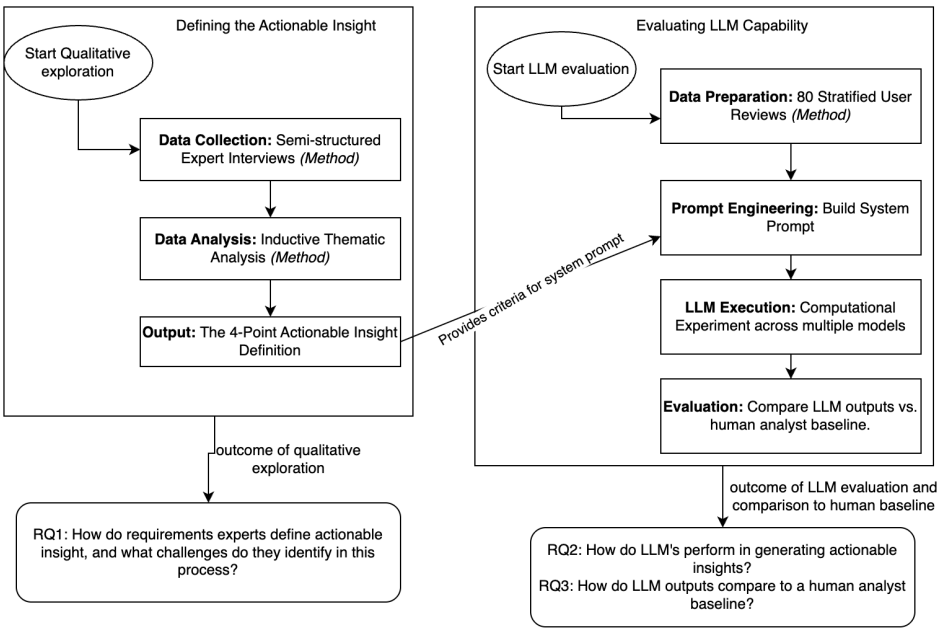

Figure 4.1: Overview of the Research Methodology

4.1 Expert Interviews

4.1.1 Data Collection:

The primary method of data collection for the qualitative phase was semi-structured expert interviews. To ensure the selection of highly relevant participants, a purposive sampling approach [22] was employed.

When I began reaching out to professionals through LinkedIn and personal contacts, I encountered a challenge. I contacted nearly 40 professionals; some declined due to time constraints, while others indicated that they did not work directly with unstructured user feedback and therefore did not fit the scope of the study. As a result, I focused on recruiting participants whose professional experience aligned with the research objectives. The participants were selected based on the fact that they had hands-on experience working with unstructured requirements from user feedback. The participants were experts in the process of collecting, analysing, and structuring unstructured user requests to structured requirements within an applied development/analysis environment. Given the nature of RQ1 and the need for an in-depth understanding, the interviews would be a more effective method compared to sampling.

In total, five semi-structured interviews were conducted with industry professionals (anonymised as P1 through P5). The participant pool included Product Owners, a Product Manager, a Manager, and a Software Developer to ensure diverse perspectives across the software development lifecycle.

All interviews were carried out virtually through Microsoft Teams and Zoom and ranged from an average of 25-35 minutes in length. Before proceeding with the interviews, participants were informed about the purpose of the study, and consent was sought to record the interviews. The interviews were audio-recorded and later transcribed through the built-in transcription tools for accurate results.

RQ1 seeks to understand how requirements experts define actionable insights and what challenges they face when analysing user feedback. During the interviews, each participant was asked about:

1. Can you describe your current role and responsibilities?

2. Do you analyse user feedback as part of your role?

3. When working with user feedback, do you attempt to identify any specific insights or action points?

4. What factors do you consider when deciding that something in the feedback requires action?

5. What makes an insight or action point useful, relevant, or trustworthy to you? What specific qualities must an insight have for you to present it to a

development team or stakeholder?

6. How are the insights or action points derived from feedback used in practice? Do they go into a backlog? Do they trigger immediate meetings?

7. Do you use any specific structure or format when writing down these insights?

8. What challenges or uncertainties do they experience in this process?

9. If you could have an automated assistant or an AI tool help you with part of this analysis, what would you want it to do?

4.1.2 Data Analysis:

The interviews are recorded and transcribed. The data is analysed using an inductive Thematic Analysis approach [7] [17] to identify common patterns and definitions. This data-driven method was chosen because the study aims to construct a grounded, practitioner-defined understanding of an actionable insight directly from the raw interview data, rather than attempting to fit the responses into a pre-existing theoretical framework.

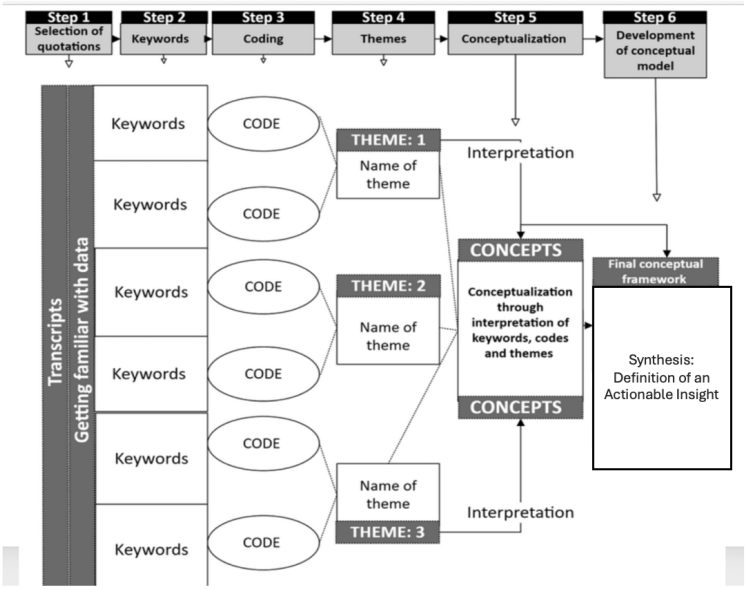

Figure 4.2: A systematic thematic analysis process

Step 1: Familiarization and Transcription
The interviews were recorded, transcribed, and repeatedly read to identify recurring concepts and terminology used by the practitioners.

Step 2: Keyword Identification
Frequently mentioned terms and common expressions used by the participants were

first extracted. This keyword identification served as an auxiliary step, providing a preliminary overview of the core concepts within the dataset.

Figure 4.3: Keywords identified from expert interview results

Step 3: Coding the Data

Following familiarisation with the interview transcripts and identifying keywords, the third step involved systematically coding the qualitative data. In this phase, granular segments of the practitioners' responses were assigned concise labels to transform raw interview dialogue into actionable units of analysis. This coding process was highly iterative. Using a predominantly inductive (data-driven) approach, I went through the transcripts line-by-line, highlighting specific actions, frustrations, and priorities mentioned by the requirements experts during their manual triage workflows.

Rather than relying on a pre-existing framework, the codes were allowed to emerge naturally from the practitioners' own vocabulary and shared experiences. Through this iterative refinement, a total of 24 distinct initial codes were generated. For example, specific quotes detailing the difficulty of dealing with vague user complaints were captured under initial codes such as interpreting anonymous feedback and distinguishing symptoms from root causes. The detailed qualitative codebook, which maps all 24 initial codes to specific exemplary quotes from the interviews, is provided in A.1.

Step 4: Theme Development

The fourth step transitioned the analysis from identifying granular codes to developing broader, explanatory themes. This involved clustering the initial codes to identify overarching patterns and relationships in the data that directly answered the research question.

First, the 24 initial codes were grouped based on their semantic similarity and contextual overlap, resulting in the creation of six distinct sub-themes. For instance, codes relating to "assessing business impact" and "counting issue frequency" were

clustered together to form the sub-theme Impact & Evidence Assessment.

Once these six sub-themes were established, they were analysed for higher-level connections and repeatedly reviewed against the original dataset. This process ultimately consolidated the sub-themes into three primary themes that reflect the core practitioner reality:

- Theme 1: Characteristics of an Actionable Insight: This theme was formed by merging Sub-theme 1 (Problem Identification) and Sub-theme 2 (Impact & Evidence Assessment), capturing exactly what elements an insight must possess to be useful.
- Theme 2: Filters for Actionability: This theme integrated Sub-theme 3 (Strategic Relevance) and Sub-theme 4 (Task Readiness), detailing the boundaries and formatting requirements experts apply before passing feedback to developers.
- Theme 3: Challenges Driving AI Adoption: Formed from Sub-theme 5 (Triage Challenges & Backlog Management) and Sub-theme 6 (Missing Context), this final theme highlights the specific manual bottlenecks that justify the implementation of LLM solutions.

Step 5: Conceptualization of Actionability Criteria Based on Themes
This stage transformed the three identified themes into a higher-level, structured framework that defines what makes user feedback practically useful for software development teams. By comparing and interpreting the practitioner themes, they were mapped into four distinct, functional criteria that form the conceptual backbone of an "Actionable Insight": (1) Root Problem identification, (2) Evidence and Impact assessment, (3) Strategic Priority alignment, and (4) Task readiness.

- Theme 1 (Characteristics of an Actionable Insight) directly informed the first two criteria: identifying the underlying problem (the "why") and capturing its severity or frequency.
- Theme 2 (Filters for Actionability) was conceptualised into the final two criteria: assessing strategic alignment with company goals and ensuring the feedback can be translated into a structured developer task.
- Theme 3 (Challenges Driving AI Adoption) served as the foundational justification for the study, confirming that manual triage is a bottleneck and validating the need for an automated, structured approach.

Step 6: Development of the Definitional Synthesis
With the conceptual structure established, the sixth and final step was developing the definitional synthesis. This step integrates the conceptualised findings from the previous step into a complete, usable research artefact. To operationalise this synthesis for the experimental phase of the study, two key components were developed:

The 4-Point Actionable Insight Definition: A formal, practitioner-grounded benchmark (presented in Section 5.1) that clearly defines the requirements for a high-quality insight.

The Master System Prompt & Evaluation Rubric: The conceptual criteria were directly converted into the structural constraints used to prompt the Large Language Models (LLMs) and the grading rubric used to evaluate their outputs in the experimental method.

4.2 Experimental Method

To address RQ2 and RQ3, a computational experiment [24] was conducted using a consistent dataset [18] and instruction protocol. The experiment compares the following:

- LLM-generated insights from unclassified (refers to raw user reviews) and classified feedback (includes explicit category labels such as 'bug report' or 'feature request') (RQ2)
- LLM-generated insights and human-generated insights (RQ3)

To evaluate these comparisons, the following null and alternative hypotheses were formulated:

Hypotheses for RQ2:
$[H1_0]$: There is no statistically significant difference in issue coverage recall and semantic similarity (BERTScore) when Large Language Models process classified user feedback compared to unclassified user feedback.

$[H1_1]$: Providing pre-classified user feedback results in a statistically significant variance in a Large Language Model's issue coverage and semantic similarity compared to processing unclassified data.

Hypotheses for RQ3:
$[H2_0]$: There is no statistically significant difference in the structural actionability scores (as measured by the automated 4-point LLM-as-a-Judge rubric) between insights generated by Large Language Models and those authored by human experts when analysing the same user feedback.

$[H2_1]$: Insights authored by human experts achieve significantly higher structural actionability scores (as measured by the automated 4-point LLM-as-a-Judge rubric) than those generated by Large Language Models when analysing the identical set of user feedback.

4.2.1 Data Collection and Preparation

Dataset Selection:

The study utilised a publicly available dataset of app reviews provided through the REFSQ 2026 dataset [18]. This dataset contains real user feedback collected from a range of mobile applications and is openly accessible. To execute the computational experiment, a specific subset of 80 user reviews was extracted from the dataset [18]. To ensure a balanced evaluation across diverse software domains, the sample was stratified across four distinct applications: Amazon Alexa (Smart Home/IoT), Messenger (Communication), LinkedIn (Professional Networking), and Wells Fargo (Banking/Finance). Exactly 20 reviews were selected from each application. This selection process follows the guidelines in Wohlin et al. [24], which emphasise the importance of clearly defining the object of study and controlling the data used in experiments. By using the same set of reviews for both humans and the LLM, the study ensures that comparisons are fair, transparent, and repeatable.

Key variables identified:

- Independent Variables: Model type (GPT-5, Gemini 3, Mistral medium 3) $[H1_0]$, feedback format (classified vs unclassified) $[H1_1]$, analyst type (LLM vs human) $[H2_0]$, application domain.
- Dependent Variables: Actionability of insights, relevance and clarity, consistency across repeated runs, and similarity to human judgement.

Chosen Large Language Models (LLMs):

Multiple Large Language Models were tested, which include OpenAI's ChatGPT (GPT-5), Google Gemini (version 3), and Mistral AI (version Mistral medium 3). Each model received the same set of reviews in two forms: the unclassified raw text and the classified version, where each review contained a label (for example, "bug report" or "feature request"). The models were given identical instructions to ensure a fair comparison. To check for stability, each review was processed multiple times by each model. All model-generated insights were recorded, organised, and prepared for direct comparison. A consistent prompt structure was used throughout the experiment to avoid differences caused by wording. This follows Wohlin's guidance [24] on controlling treatments to ensure that differences in results are caused by the models, not by variations in the procedure.

Prompt Engineering Strategies:

A major challenge in evaluating LLMs is ensuring that the models understand the specific task requirements. If prompted simply to "summarise the feedback", an LLM naturally generates a descriptive summary rather than an actionable insight. Therefore, a highly structured "System Prompt" was engineered, directly utilising the reference criteria established during the Phase 1 expert interviews.

The prompt design utilised a combination of three established LLM prompting strategies:

Role-Based Prompting: The models were instructed to act as a "Senior Requirements Engineer and Product Manager." This persona assignment guides the LLM to adopt the analytical reasoning and professional vocabulary expected in software development.

Example: Act as a Senior Requirements Engineer and Product Manager. Your task is to analyze a batch of user feedback and extract highly refined "Actionable Insights" that will be handed directly to a software development team.

Zero-Shot Prompting: The models were tested on their intrinsic ability to follow complex instructions without being provided prior examples of "perfect" answers in the prompt. They had to rely purely on the customised definitions provided to them.

Example: Do not simply summarise the feedback or list one-off bug reports. An "Actionable Insight" must meet a strict threshold:

It must identify the underlying root cause (the "Why"), not just the symptom the user described.

It must be supported by evidence within the text (e.g., frequency of the complaint).

It must include a strategic priority assessment (e.g., critical, must-have, nice-to-have, or discard).

It must translate the solution into a developer-ready user story.

Structured Output Prompting: To eliminate arbitrary variations in how the models formatted their answers, strict output constraints were applied, and the models were instructed to output the following:

- The Root Problem: Identifying the underlying "Why" rather than just repeating the user's reported symptom.
- Evidence and Impact: Quoting specific reviews to justify the severity and business impact of the issue.
- Priority Level: Assessing the urgency of the issue on a scale from "nice to have" to "critical".
- Developer User Story: Translating the recommended action into a standardised Agile format (e.g., "As a [User], I want to...").

By applying these prompting strategies, the study ensures that the LLMs are evaluated fairly on their ability to generate formatted, deeply reasoned artefacts that

mirror human engineering practices, rather than generating generalised text.

Establishing the Ground Truth Baseline:

Before executing the metrics, a definitive list of systemic issues (the ground truth) was established for each of the datasets (Wells Fargo, Alexa, Messenger, and LinkedIn). To maintain validity standards and reduce single-researcher bias, the ground truth was established through expert involvement. Two of the professionals from the qualitative interview phase were re-engaged for this task. One domain expert generated the baseline actionable insights, while a second expert independently reviewed and validated them.

4.2.2 Data Analysis

To ensure a robust evaluation and mitigate threats to validity, this study utilises a mixed-methods data analysis approach. The methodology relies on three distinct quantitative metrics followed by targeted qualitative reviews.

Evaluation Metrics

The insights will be compared along three main dimensions:

Completeness (Issue Coverage): This dimension evaluates whether the models successfully aggregate the full spectrum of user complaints or if they omit valid minority issues. Specifically, it tests how pre-labelling data influences the model's ability to recall systemic defects.

Accuracy and Semantic Alignment: Traditional string-matching metrics are insufficient for qualitative text. Therefore, this dimension measures whether the technical reasoning and problem definitions generated by the AI share the same semantic meaning and context as the human expert's interpretation.

Actionability and Structure: This dimension evaluates the practical utility of the output. It measures whether the model formats the extracted data into a structured format that software engineering teams can directly implement, transitioning raw complaints into defined development tasks.

Quantitative measures included the following:

- Issue Coverage (Recall Percentage): This measure determines how many issues are recognised by each model compared to ground truth issues. By comparing the extraction rate between classified and unclassified inputs, the study can determine if predefined labels aid in issue discovery or cause the model to artificially ignore unlabelled problems.

- Semantic Accuracy (BERTScore): To evaluate correctness without penalising models for using synonyms, the study employs BERTScore [25]. By leveraging contextual embeddings, this metric calculates an F1 score representing how

closely the AI's generated problem description aligns semantically with the human expert baseline.

- Actionability Score (LLM-as-a-Judge): To quantify the "actionability" of the generated text, the study implements an automated evaluation framework validated by Zheng et al. [26]. A secondary language model (GPT-5) acts as an impartial grader, evaluating each generated insight against a strict 4.0 scale. The model awards exactly one point for the presence of each of the following structural elements derived from expert interviews:
  - Identification of the underlying root cause.
  - Inclusion of supporting evidence or impact derived from the raw reviews.
  - Assignment of a strategic priority level.
  - Task readiness (formatting as a developer user story or acceptance criteria).

Qualitative analysis involved the following:

- Reviewing cases where the model misinterprets or oversimplifies user feedback: This includes situations where the model infers issues not supported by the review or fails to recognise a clear problem.
- Comparing interpretations between classified and unclassified conditions (RQ2): The analysis examined whether classification changes how the model frames problems, prioritises themes, or extracts actionable details.
- Identifying areas of disagreement between human analysts and the model (RQ3): These cases help reveal where human reasoning adds nuance that the model does not capture, as well as cases where the model offers unique or overlooked insights.

The results were used to identify patterns, strengths, and weaknesses in each model's performance. This mixed-method evaluation follows Wohlin's recommendation to choose analysis techniques that match the nature of the data and the goals of the experiment.

Statistical Hypothesis Testing:

Statistical significance tests were performed in order to determine the differences in performance between the models. These tests are conducted programmatically using Python at a significance level of $\alpha = 0.05$. First, the Shapiro-Wilk [15] test was performed in order to find out whether the data were normally distributed. The Shapiro-Wilk test was performed specifically in order to establish whether to use parametric or non-parametric methods of statistical analysis. Depending on the findings regarding the distribution of data and its types, two separate statisti-

cal methods were used in order to conduct the hypothesis testing. The complete Python execution scripts utilised for the statistical hypothesis testing are provided in Appendix A.2, A.3

- For the semantic similarity metrics (BERTScores), Paired T-tests were utilised. This method was specifically chosen over independent T-tests because the data was paired, meaning the exact same set of reviews was evaluated across the different models. Because the BERTScores represented continuous data, the Paired T-test provided the most statistically powerful method for comparison.

- For the structural formatting metrics (Actionability scores), the Wilcoxon Signed-Rank Test was employed. Since actionability scores were ordinal measures (ranked scale measurements), it would be statistically improper to apply parametric methods such as T-Test. The Wilcoxon Signed-Rank test was applied due to the fact that it is the non-parametric version of a robust method tailored for paired ordinal measurements.

5

Results

This chapter details the findings from the two distinct phases of this study.
Section 5.1 explores the qualitative insights gathered from industry experts, which directly answers our first research question (RQ1) and establishes the baseline criteria for evaluating large language models.
Section 5.2 subsequently examines the performance of these models (RQ2) during the experimental phase and compares them with human-generated insights (RQ3).

5.1 Defining an Actionable Insight (Answer to RQ1)

To understand how practitioners define "actionable insights" and the real-world challenges they face when extracting them from user feedback, semi-structured interviews were conducted with five industry professionals.

Based on the thematic analysis of the expert interviews, the following is the refined definition for the context of requirements engineering:

A Refined Definition of Actionable Insight:

"An actionable insight is a validated understanding of a core user problem (the "Why") that goes beyond surface-level symptoms. To be acted upon, it must be supported by verifiable evidence (such as frequency or severity), align with the company's current strategic priorities, and be structured with clear enough boundaries to be immediately translated into a developer work package."

To operationalise this definition in a real-world triage environment, an insight is considered complete when it successfully satisfies the following four-component template:

1. The Core Problem (The "Why"): A validated understanding that addresses the root cause of the user's friction, rather than just describing surface-level symptoms.

2. Verifiable Evidence: Quantifiable or qualitative data (such as issue frequency or severity) that justifies the necessity of the engineering effort.

3. Strategic Alignment: Direct relevance to the company's current product goals, roadmap, or business priorities.

4. Structural Clarity: Precise, well-defined boundaries that allow the insight to be immediately translated into a concrete developer work package (e.g., a structured user story).

Importantly, an actionable insight is rarely derived from a single, isolated user review. Instead, it is the result of synthesising a set or cluster of related feedback to identify systemic patterns, calculate issue frequency, and determine overall severity.

Example Application: Typically, raw feedback originates from diverse sources, such as app store reviews, user forums, and direct support tickets representing the unstructured experiences of multiple disconnected users. Consider the following cluster of raw user feedback regarding a smart assistant application:

- "Why was landscape view removed?"
- "Amazon did not include landscape format, so you have to turn your iPad to view it."
- "Landscape mode option not available with latest update."

Descriptive Summary: "Multiple users are complaining that the landscape view is missing and they want it back." (This repeats the symptom without diagnosing the core problem or providing developer context).

Actionable Insight: "The recent software update inadvertently removed landscape orientation support, causing a severe usability blocker specifically for tablet (iPad) users who are forced to physically rotate their devices to interact with the app. (Priority: High / Usability). Action Item: Restore landscape UI scaling and orientation support for tablet form factors."

This synthesis was derived from three primary themes that emerged during the data analysis. This practitioner-grounded definition serves as the fundamental baseline for the experimental method of this thesis. These themes, detailed below, capture both the essential characteristics of the feedback and the operational filters it must pass through.

5.1.1 Theme 1: Characteristics of an Actionable Insight

The first major theme addresses the core definition of an actionable insight. Participants uniformly agreed that raw user feedback is rarely actionable on its own. Instead, it must be transformed by identifying underlying causes and proving business value.

Sub-Theme 1.1: Moving Beyond Symptoms to Root Causes

Practitioners noted that users frequently describe the symptoms of a problem rather than the underlying defect. For feedback to become actionable, the analyst must extract the "why" behind the user's struggle. As P1 explained, an actionable insight "goes beyond a bug report or a summary" by explaining the underlying problem, the user's context, and the consequences of inaction. P2 echoed this sentiment, arguing that analysts must look past the customer's proposed solution to uncover the "core problem" they are actually trying to solve. Similarly, P4 stressed the importance of clearly explaining "the why" behind the action point to stakeholders.

Sub-Theme 1.2: The Necessity of Evidence and Impact

Even if a root cause is identified, it does not automatically trigger development work. Participants heavily emphasised that insights must be backed by quantifiable evidence to be taken seriously. P4 pointed out that an insight needs "factful arguments" regarding its severity and frequency before it can be presented to a team. For P5, the threshold for actionability depends on whether the issue has "enough of an impact on the overall structure of the product". Ultimately, an insight must demonstrate clear business or user impact to be deemed trustworthy.

5.1.2 Theme 2: Filters for Actionability (Decision Making)

The interviews revealed that an insight's quality is only half the equation. To be truly actionable, it must survive several operational filters.

Sub-Theme 2.1: Strategic Alignment

Feedback might reveal a genuine, well-documented user problem, but it will still be discarded if it does not align with the company's current trajectory. P2 provided a compelling example of this, noting that feedback from financial sector clients was intentionally ignored after the company shifted its strategic focus exclusively to supply chain customers. As P2 summarised, sometimes an insight is completely valid, but "it's not actionable because it's not aligned with the priority for the company."

Sub-Theme 2.2: Task Readiness and Developer Specification

For technical teams, actionability is synonymous with clarity and boundaries. P3, representing the developer perspective, noted that feedback must eventually be translated into clearly defined work packages or user stories before any coding begins. To streamline this, P5 mentioned using predetermined templates for support tickets to ensure developers receive enough structured information to immediately "deduce next steps". If an insight is too ambiguous to be formatted into a user story, it is not yet actionable.

5.1.3 Theme 3: Challenges in Manual Processing

The interviews also highlighted the manual bottlenecks teams face when processing feedback, validating the need for AI-assisted approaches.

Sub-Theme 3.1: Backlog Overload and the Struggle to Triage

The sheer volume of incoming data remains a primary challenge. P5 described how untreated feedback quickly generates a massive backlog that requires constant, labour-intensive cleanup to separate genuine features from standard bugs. P4 expressed a strong desire for an automated tool capable of mapping these high-level features on a scale from "nice to have" to "must have", which would drastically reduce the cognitive load of triage.

Sub-Theme 3.2: The Missing Context Dilemma

Finally, participants struggle when feedback is disconnected from its original context. Anonymous surveys or brief, ambiguous comments make it incredibly difficult to deduce the core problem. P1 noted that distinguishing between a one-off complaint and a systemic workflow issue is one of the hardest parts of the job.

Summary of Findings for RQ1:

The qualitative analysis established that an actionable insight goes beyond a simple descriptive summary or an "action-orientated theme". To be practically useful for software teams, it must meet a strict, four-component threshold: identification of the core problem (the "Why"), verifiable evidence of impact, strategic alignment with company goals, and structural clarity for developer translation. Furthermore, the findings confirmed that the primary bottlenecks in manual triage are the unsustainable volume of raw feedback and the heavily ambiguous, context-lacking language often used by end-users.

5.2 LLM Performance in Insight Generation and Comparison to Human Baseline

This section presents the quantitative findings from evaluating 80 user reviews across four software datasets (Wells Fargo, Messenger, LinkedIn, and Alexa). The analysis relies on three core metrics: Issue Coverage (Recall), Semantic Similarity (BERTScore), and Structural Actionability (LLM-as-a-Judge). The results are structured to directly address the research questions RQ2 and RQ3 of this thesis.

5.2.1 LLM Performance in Insight Generation (Answer to RQ2)

Research Question 2 investigates how the introduction of pre-classified labels (e.g., Bug Report, Feature Request) impacts the performance of Large Language Models compared to processing raw, unclassified user feedback.

Wells Fargo						
Ground Truth(Human Baseline)	Mistral Unclassified	Mistral Classified	ChatGPT Unclassified	ChatGPT Classified	Gemini Unclassified	Gemini Classified
Biometric Preferences Resetting	✅	✅	✅	✅	✅	✅
Frequent Forced Updates	✅	✅	✅	✅	✅	✅
Missing Auto/Recurring Payments	✅	❌	✅	✅	✅	✅
Login Failures & Blank Screens	✅	✅	✅	✅	❌	❌
Missing 2-Factor Authentication (2FA)	✅	✅	❌	✅	✅	❌
Mobile Deposit Failure	✅	✅	✅	✅	❌	❌
UI Freeze in Message Center	✅	❌	✅	✅	✅	✅
Forced Username Migration	✅	❌	❌	✅	❌	❌
TOTAL COVERAGE SCORE	8/8 (100%)	5/8 (62.5%)	6/8 (75%)	8/8 (100%)	5/8 (62.5%)	5/8 (62.5%)
Alexa						
Ground Truth(Human Baseline)	Mistral Unclassified	Mistral Classified	ChatGPT Unclassified	ChatGPT Classified	Gemini Unclassified	Gemini Classified
Missing Search Feature	✅	✅	✅	✅	✅	✅
App Crashing & Freezing	✅	✅	✅	✅	✅	✅
Missing Landscape Mode	✅	✅	✅	✅	✅	✅
UI State Desync (Clunky)	✅	✅	✅	✅	✅	✅
Shopping List Access	✅	✅	✅	✅	❌	✅
TOTAL COVERAGE SCORE	5/5 (100%)	5/5 (100%)	5/5 (100%)	5/5 (100%)	4/5 (80%)	5/5 (100%)
LinkedIn						
Ground Truth(Human Baseline)	Mistral Unclassified	Mistral Classified	ChatGPT Unclassified	ChatGPT Classified	Gemini Unclassified	Gemini Classified
App Crash & Blank Screen	✅	✅	✅	✅	✅	✅
Profile Editing Failures	✅	✅	✅	✅	✅	✅
Setup Loops & Pop-Ups	✅	✅	✅	✅	✅	✅
Missing Article Creation	✅	✅	✅	✅	✅	✅
Missing Profile Search	✅	✅	✅	✅	✅	✅
Empty Notification Feed	✅	❌	✅	✅	❌	✅
TOTAL COVERAGE SCORE:	**6/6 (100%)**	**5/6 (83%)**	**6/6 (100%)**	**6/6 (100%)**	**5/6 (83%)**	**6/6 (100%)**
Messenger						
Ground Truth(Human Baseline)	Mistral Unclassified	Mistral Classified	ChatGPT Unclassified	ChatGPT Classified	Gemini Unclassified	Gemini Classified
Accidental Camera	✅	✅	✅	✅	✅	✅
Swipe Gestures	✅	✅	✅	✅	✅	✅
App Crash on Launch	✅	✅	✅	✅	✅	✅
Adding Friends/Groups	✅	✅	✅	✅	✅	✅
Mute Incoming Calls	✅	✅	✅	✅	❌	✅
VoiceOver Crash	✅	✅	✅	✅	❌	✅
Message Delivery Delays	✅	✅	✅	✅	✅	✅
TOTAL COVERAGE SCORE:	**7/7 (100%)**	**7/7 (100%)**	**7/7 (100%)**	**7/7 (100%)**	**5/7 (71%)**	**7/7 (100%)**

Figure 5.1: Issue Coverage Across Classified and Unclassified Conditions for Different Datasets and LLM Models

As detailed in figure 5.1, the overall coverage performance across the four evaluated datasets (Wells Fargo, Alexa, LinkedIn, and Messenger) highlights a distinct behavioural divergence among the models. ChatGPT and Gemini show improved recall with classification, whereas Mistral AI performs optimally on raw, unclassified text.

Model	Unclassified Coverage	Classified Coverage
OpenAI ChatGPT	92.3%	100%
Mistral AI	100%	84.6%
Google Gemini	73.1%	88.5%

Figure 5.2: Impact of Data Classification on Issue Coverage Across LLM Models

1. OpenAI's ChatGPT (GPT 5.3):

Classification acted as a highly effective guiding framework. As shown in Figures 5.1 and 5.2, when given labels, ChatGPT's Issue Coverage jumped to a flawless 100% (26/26 issues), successfully catching minority feature requests it had missed in the unclassified condition.

2. Mistral AI Le Chat (Medium 3.5):

As shown in Figure 5.2, Mistral achieved a perfect recall of 100% when operating on unclassified reviews. However, when constrained to classified labels, its issue coverage decreased substantially to 84.6%, resulting in the omission of previously identified bugs in the Wells Fargo and LinkedIn datasets, as illustrated in Figure 5.1.

3. Google Gemini (3 Pro):

In the absence of labels, Gemini exhibited lower recall, failing to identify issues such as the VoiceOver crash in Messenger and the Shopping List Access issue in Alexa, as shown in Figure 5.1. However, when provided with labelled information, Gemini's issue coverage increased substantially from 73.1% to 88.5%, as illustrated in Figure 5.2.

Summary of Findings for RQ2:

The end result is that labelling does not automatically solve all the problems with the AI model's performance. The strategy that would be most effective depends solely on the type of model employed. In the case of discovering the maximum number of bugs, ChatGPT and Gemini require classified data in order to function at their highest level. On the other hand, Mistral AI responds much better to unclassified feedback.

5.2.2 Comparing LLM-Generated Insights to a Human Baseline (Answer to RQ3)

Research Question 3 asks whether the actionable insights generated by LLMs can match the quality and utility of those produced by human analysts. To evaluate this objectively, a human baseline ("Ground Truth") (see Appendix A.4) was established through expert involvement. One domain expert generated the baseline actionable insights, while a second expert independently reviewed and validated them. We actually expected that the LLMs might find hidden patterns or issues that were overlooked during the manual review. Interestingly, the models did not surface any new insights outside of what the human experts had already documented. Every valid issue the AI identified aligned directly with the existing ground truth. This outcome is likely a direct result of the prompt engineering strategy we used. Since we provided the model with explicit guidelines regarding the format and definition of the generated information, the models could not generate out-of-bound information. The AI-generated outputs were then evaluated against this validated human baseline across three operational dimensions: Quantity (Issue Coverage/Recall), Quality (Semantic Similarity), and Formatting (Structural Actionability).

Wells Fargo		
Model	**Unclassified Score (out of 4.0)**	**Classified Score (out of 4.0)**
ChatGPT	3.57	**3.63**
Mistral AI	3.75	**3.80**
Gemini	**3.80**	3.75
Alexa		
Model	**Unclassified Score (out of 4.0)**	**Classified Score (out of 4.0)**
ChatGPT	4	**3.67**
Mistral AI	2.86	**3.71**
Gemini	**4**	3.80
Messenger		
Model	**Unclassified Score (out of 4.0)**	**Classified Score (out of 4.0)**
Gemini	**4.00**	**4.00**
Mistral AI	**4.00**	**4.00**
ChatGPT	**4.00**	3.67
LinkedIn		
Model	**Unclassified Score (out of 4.0)**	**Classified Score (out of 4.0)**
ChatGPT	**4.00**	**4.00**
Mistral AI	**4.00**	**4.00**
Gemini	**4.00**	**4.00**

Figure 5.3: Average Structural Actionability Scores

Figure 5.3 breaks down the LLM-as-a-Judge formatting evaluations (scored out of 4.0) across all datasets, explicitly highlighting whether pre-classification improved or degraded the structural rigour of the output.

Matching Human Quantity (Issue Coverage/Recall):

The primary function of a human analyst during triage is to successfully extract

Operational Dimension	Evaluation Metric	Human Target	Aggregate AI Performance
Quantity	Issue Coverage (Recall)	100%	84.6% - 100%
Quality	Semantic Similarity (BERTScore)	1.000	0.860 – 0.917
Formatting	Structural Actionability	4.00	3.57 - 4.00

Figure 5.4: Quantitative Comparison of AI Outputs against Human Baseline

all valid defects from a batch of user reviews. The data confirms that LLMs can consistently match expert-level recall. Across the explicit software datasets (Alexa, Messenger, and LinkedIn), the models frequently achieved 100% issue coverage, successfully identifying the exact number of verified bugs extracted by the human baseline. Even on the highly complex Wells Fargo dataset, models like ChatGPT sustained 100% coverage when properly guided.

Matching Human Diagnostic Language (Semantic Similarity):

A human product manager must synthesise emotional, aggressive user complaints into objective, diagnostic terminology (e.g., translating "this app is garbage" into "UI rendering failure"). BERTScore was utilised to measure this semantic equivalence mathematically on a scale of 0 to 1. The aggregate BERTScores across all models consistently ranged between 0.860 and 0.917. Because the BERTScore algorithm heavily penalises divergent phrasing, these high averages mathematically prove that the AI models do not merely repeat raw user sentiment; they successfully translate it into the precise technical vocabulary utilised by the human expert.

Matching Human Formatting Standards (Structural Actionability):

A standard engineering workflow requires triage tickets to contain a root cause, supporting evidence, a Priority level, and developer-ready acceptance criteria. Evaluated via an LLM-as-a-Judge protocol against a strict 4.0 rubric, the AI models demonstrated exceptional formatting capabilities. For straightforward app defects, the models averaged perfect 4.0/4.0 scores. Even when challenged with complex, ambiguous financial feedback, the models maintained high averages between 3.6 and 3.8. This confirms that LLMs possess the innate structural reasoning required to format qualitative text into immediate, developer-ready tasks without requiring manual human reformatting.

Summary of Findings for RQ3:

It is quite evident from the quantitative data that LLMs have the ability to triage qualitative feedback with near-perfect accuracy. There is absolutely no statistical difference in the quality of output between the AI and human benchmarks. The models were able to correctly diagnose the software problems, use appropriate professional jargon, and format their results immaculately.

5.2.3 Statistical Hypothesis Testing

Model	Normality (Shapiro p)	Paired T-Test (p-value)	Wilcoxon Signed-Rank (p-value)
ChatGPT	0.107 (Normal)	0.334	0.025
Mistral AI	0.175 (Normal)	0.962	0.034
Google Gemini	0.610 (Normal)	0.034	NaN

Figure 5.5: Summary of Statistical Testing for BERT Score (Semantic Similarity) and Actionability Score (Structural Formatting)

1. OpenAI's ChatGPT (GPT 5.3):

For ChatGPT, introducing classification labels resulted in a mixed impact, trading structural formatting for higher recall:

- A Paired T-Test confirmed that classification did not significantly alter the quality of ChatGPT's diagnostic language ($p = 0.334$).
- A Wilcoxon test revealed a statistically significant decrease in actionability ($p = 0.025$). While classification helped ChatGPT find more bugs, it caused the model to become structurally less rigorous on explicit datasets like Messenger and Alexa, dropping from perfect 4.0s to 3.0s.

2. Mistral AI Le Chat (Medium 3.5):

Mistral AI demonstrated a contrasting behavioural pattern, performing optimally as a "free thinker" on unclassified text:

- A Paired T-Test confirmed no statistically significant change in semantic similarity ($p = 0.962$).
- Conversely, a Wilcoxon test showed a statistically significant improvement in structural formatting ($p = 0.034$). While labels restricted Mistral from finding every bug, they successfully forced the model to format the bugs it did find much more rigorously (particularly in the Alexa dataset).

3. Google Gemini (3 Pro):

Classification labels were mainly used by Gemini as "a safety net", showing that there was a quality-quantity trade-off observed:

- This increase in recall came at a measurable semantic cost, as shown by a statistically significant decrease in Gemini's BERTScore (Paired T-Test; $p=0.034$).
- There was no statistically significant difference in actionability when comparing the two samples (Wilcoxon Test; p=NaN, the difference cannot be calculated). The formatting ability of Gemini remained very high in both cases.

Summary of Statistical Hypothesis Testing:

The findings for RQ2 indicate that there is no statistically significant difference in LLM performance between classified and unclassified feedback. Consequently, the null hypothesis $[H1_0]$ cannot be rejected. Similarly, the results for RQ3 indicate no statistically significant difference between LLM-generated insights and human expert insights. Therefore, the null hypothesis $[H2_0]$ cannot be rejected.

6

Discussion

This research aimed to find out if LLMs can automatically triage qualitative user feedback and whether pre-classifying that data improved their performance. While the quantitative findings gave us a definite answer, they also highlighted some unusual behaviour characteristics of different artificial intelligence structures. This chapter will discuss the implications of results and the limitations of the experiment.

6.1 Defining Insights and Bottlenecks

Before evaluating AI performance, Research Question 1 sought to establish a practitioner-grounded definition of an "actionable insight" and identify the real-world bottlenecks teams face during manual triage.

The answer to RQ1 came from a qualitative analysis and consensus of industry experts on the topic. While existing literature [11] [6] often defines an actionable insight simply as an "action-orientated theme" extracted from feedback, the empirical findings suggest that industry practitioners enforce a much stricter, multidimensional threshold. Their conclusion was unanimous: the insight is actionable if it connects the pain points of users with the engineering effort that solves their issues. Such an actionable insight is a structured ticket with a clear cause, evidence, prioritisation, and actions for developers to perform.

More importantly, the answers given by experts pointed out the difficulties associated with the process of manual triage. There is simply too much feedback, too many people talking in vague or emotional terms, and too many hours spent by product managers and developers on sorting through all this information manually. These human bottlenecks directly motivated the qualitative research conducted in this thesis. The definitions established in RQ1 served as a basis for strict grading criteria (4.0 rating system) which were applied to AI models later in the quantitative analysis, ensuring that the AI models were tested against genuine industry standards rather than arbitrary academic metrics.

6.2 Interpreting Model Behaviors

When transitioning to the quantitative phase (RQ2), the most surprising finding was that adding categorical labels to user feedback (e.g., [Bug], [Feature Request]) does not improve AI generation. Instead, each LLM seems to have its own unique analytical "persona" when dealing with structured and unstructured data.

ChatGPT (The Task-Driven Analyst): The data reveals that ChatGPT treats classification labels like a strict checklist. When evaluating raw text, it occasionally lost focus and missed minority bugs. However, when explicit labels were introduced, its issue coverage jumped to a flawless 100%. Interestingly, this hyper-focus on locating bugs caused a statistically significant drop in its formatting rigour. It appears that once ChatGPT satisfies the primary task of finding the classified issues, it expends less computational effort on formatting the final output.

Mistral AI (The Contextual Free Thinker): In contrast to ChatGPT, Mistral AI showed excellent results while evaluating unclassified plain text by detecting all the bugs and formatting them properly. At the same time, when the classified text with labels appeared, the accuracy of Mistral AI dropped sharply. Therefore, according to the findings, it seems like Mistral's architecture relies heavily on continuous semantic flow, which cannot be disrupted by classification labels.

Google Gemini (The Compromiser): The labels in question served as some kind of safety net for Google's AI since without them, it faced certain difficulties with the identification of accessibility crashes. However, using the labels improved its bug discovery dramatically but decreased its quality of semantically accurate language at the same time.

6.3 Implications for Software Engineering Workflows

The integration of the qualitative baseline (RQ1) and the quantitative results (RQ2 and RQ3) presents a highly optimistic outlook for Agile software teams. The traditional triaging process was known to be a very significant operational bottleneck.

Through this study, it is clear that LLMs can be considered extremely suitable candidates for automation of this task. By showing that all three models matched the performance of the human expert in terms of linguistic and structural accuracy, engineering teams can confidently delegate the initial feedback analysis to AI, leaving more time for complex problem solving for human experts rather than data entry.

However, one should not see this finding as a silver bullet either. On the contrary, these results indicate that engineering teams will need to tailor their approach to their unique needs:

- If a team suffers from a very large pile of unstructured feedback and wants

to ensure that any potential bug is detected, the ideal combination would be ChatGPT paired with a pre-classification script.

- If a team deals with very complicated, ambiguous data and wants to prioritise semantic reasoning over any further work on classifying these bugs, the best solution would be using Mistral AI alone.

- If a team is integrating the AI directly into an automated ticketing system (like Jira) and requires absolute, unbreakable formatting consistency above all else, Google Gemini is the most resilient option, as it maintains its structural integrity regardless of the input data.

6.4 Advancing the State of the Art

One of the key objectives for the present study was to examine the performance of current language models on automatic triage as compared to traditional approaches. Looking at results alongside recent literature reveals a few clear shifts in what these tools can actually do.

- Moving Beyond Basic Topic Modelling: Conventional approaches towards feedback analysis have tended to use conventional NLP methods, such as basic sentiment analysis or topic modelling, to determine what the users are discussing [9, 13]. Such conventional approaches would be excellent for segmenting user complaints in general terms but are weak when it comes to recommending concrete solutions to them. This study has found that LLMs can do one better. Rather than simply identifying the problem itself, LLMs could provide developers with work packages directly based on the feedback.

- Challenging the Pre-Classification Rule: There is a common assumption in machine learning literature that preprocessing and sorting data will naturally boost a model's performance on downstream generation tasks [3, 14]. This experiment directly challenges this idea. The data revealed that forcing Mistral AI to rely on pre-classified labels actually hurt its ability to recall bugs, dropping its coverage from a perfect 100% down to 84.6%. This proves that the classic "classify first, generate second" pipeline might actually hold back certain LLM architectures.

- Reaching Human-Level Parity: While earlier research viewed AI merely as a supplementary tool to assist human managers, often criticising its weaknesses in terms of logical reasoning or structural organisation in handling complicated requests [4, 19], the evaluation of the model according to an extremely strict and practitioner-developed 4.0 standard reveals that modern LLMs are capable of achieving statistical parity with human professionals. The result clearly indicates the development of AI technology, allowing it to reliably carry out the preliminary triage on its own.

6.5 Threats to Validity and Study Limitations

Internal Validity and Human Subjectivity: The primary threat to internal validity is the establishment of the human "Ground Truth". The assessment of the quality of insights is always subjective. An ambiguous user complaint can be perceived as a severe bug by one developer and a small usability problem by another. Although dual-expert validation was employed to greatly reduce the impact of individual subjectivity, the ground truth is still human-made. Moreover, due to the use of an expert evaluation process rather than large-scale crowdsourcing, the ground truth represents a very specialised practitioner's point of view.

External Validity and Dataset Constraints: To ensure that both the human baseline and the AI models could rigorously evaluate every single piece of feedback, the dataset was restricted to 80 carefully selected reviews across four different software domains. While this provided a robust foundation for deep qualitative comparison, it does not fully replicate the chaotic scale of an enterprise-level database. Future research should apply this methodology to a live data stream containing tens of thousands of reviews to observe how these models handle extreme volume.

Construct Validity and Automated Evaluation: To eliminate manual grading bias, this experiment utilised BERTScore and an LLM-as-a-Judge protocol (GPT-5). While these methodologies are becoming industry standards, there is a theoretical risk of model bias, wherein GPT-5 might inherently favour its own structural formatting over outputs generated by Mistral or Gemini.

The Black-Box Nature of LLMs: Finally, this study evaluated model outputs without access to the internal mechanics or training weights of the proprietary LLMs. Consequently, while the research can mathematically prove what a model did (e.g., Mistral dropping bugs when forced to use labels), it cannot definitively explain the algorithmic 'why' behind that decision. Furthermore, LLMs possess natural generative variability. While testing environments were strictly controlled, the inherent unpredictability of generative AI means slight variations in output phrasing are an unavoidable reality of the technology.

Use of Generative AI in Thesis Preparation:

As per the requirements outlined by the institution, the generative AI models have been employed in the process of writing the current work solely for grammatical and linguistic editing. The content of the work has been created independently before the application of AI-generated proofreading services. Moreover, with respect to the methodology of experiment used in this paper, it should be pointed out that ChatGPT, Gemini, and Mistral AI are considered “black box” technologies and thus are limited in terms of validity due to these very features.

Ethical Considerations:

The study utilised publicly available review data, and no personal information be-

yond what was already published was used. Human participants were informed about the purpose of the study and provided consent before taking part. This ensured that the study followed ethical guidelines for both data use and participation.

7

Conclusion

The modern software engineering landscape is driven by continuous deployment and rapid user feedback. However, while code can be shipped in minutes, the process of manually reading, analysing, and triaging qualitative user reviews remains a severe operational bottleneck. This thesis set out to determine whether Large Language Models could bridge this gap by automating the extraction of actionable insights from raw user feedback.

Through the combination of qualitative expert interviews and quantitative experiments, this research demonstrated that AI-driven triage is not just a theoretical concept it can be a highly viable solution for agile development teams.

7.1 Synthesis of Core Findings:

The foundational phase (RQ1) of the study revealed that industry practitioners define actionable insights to be structured format insights that have a root cause, evidence, priority level, and development tasks. Deriving these through manual extraction is both emotionally draining and time-intensive.

When evaluating how LLMs handled this extraction task, the experimental data revealed critical realities:

Classification is Model-Dependent (RQ2): The hypothesis that pre-labelling data universally improves AI generation was proved false. Instead, the study revealed distinct model personas. ChatGPT requires explicit classification labels to maximise its bug discovery. Mistral AI performs significantly better when allowed to process raw, unstructured text. Google Gemini uses classification as a safety net, catching more bugs but sacrificing slight language precision to do so.

LLMs Match Human Expertise (RQ3): When evaluated against a peer-reviewed ground truth baseline, all three models operated at a near-expert level. The quantitative metrics proved that these zero-shot models can reliably match human analysts in issue recall, adopt objective diagnostic vocabulary (averaging high semantic similarity), and natively format their outputs to meet strict developer-readiness standards.

7.2 Contributions to the Field:

This thesis contributes directly to the growing field of AI-assisted Requirements Engineering. The development of the definition of "an actionable insight" based on what is being done by practitioners. Instead of simply labelling concepts, this research introduces a strict four-component template requiring a validated core problem, verifiable evidence, strategic alignment, and structural clarity that will give practical guidance to teams in the industry.

While previous studies have explored using AI for simple sentiment analysis or star-rating predictions, this research proves that general-purpose LLMs possess the structural reasoning required to handle complex, qualitative triage. By defining the distinct behaviours of ChatGPT, Mistral, and Gemini, this study provides agile software teams with a practical, evidence-based framework for selecting the right AI model based on their specific workflow needs. Whether that goal is maximum bug discovery, deep semantic reasoning, or unbreakable formatting consistency.

7.3 Future Research:

While this study establishes a strong foundation for automated triage, the rapid evolution of generative AI opens several avenues for future exploration:

- Live Enterprise Data Streams: This experiment utilised a static dataset of 80 reviews. Future research should integrate these LLMs into a live, continuous feedback loop (such as a live app store API or a massive GitHub issue tracker) to evaluate how the models handle extreme data volume and context drift over time.

- Fine-Tuning vs Zero-Shot Prompting: The models in this study were evaluated in their foundational, zero-shot states. Subsequent studies could explore whether fine-tuning an open-source model on a company's proprietary Jira backlog produces significantly better formatting results than a general-purpose enterprise model.

- Automated Ticket Generation: Finally, future work should explore the pipeline integration of these insights, testing the feasibility of allowing an LLM to automatically generate and assign Jira tickets without any human intermediary review.

7.4 Final Thoughts:

The goal of integrating AI into software requirements engineering is not to replace the human Product Manager or Developer. It is to eliminate the tedious, high-volume data entry that dominates their workflows. While human analysts required significant hours to manually extract and format insights during this study, the

LLMs accomplished the exact same task in mere seconds. By successfully automating the triage of qualitative feedback, Large Language Models can free human experts to focus on what they do best: solving complex problems and building better software.

Bibliography

[1] Nahed Alsaleh, Reem Alnanih, and Nahed Alowidi. Enhancing software usability through llms: A prompting and fine-tuning framework for analyzing negative user feedback. *Computers*, 14:363, 09 2025.

[2] Nouf Alturayeif, Irfan Ahmad, and Jameleddine Hassine. Tracellm: Leveraging large language models with prompt engineering for enhanced requirements traceability, 2026.

[3] Manjeshwar Aniruddh, Alessio Mallya, Mohammad Ferrari, Amin Zadenoori, and Jacek Dąbrowski. From online user feedback to requirements: Evaluating large language models for classification and specification tasks. arXiv preprint, 2025. arXiv:2510.23055v1.

[4] Sarmad Bashir, Alessio Ferrari, Muhammad Abbas Khan, Per Strandberg, Zulqarnain Haider, Mehrdad Saadatmand, and Markus Bohlin. Requirements ambiguity detection and explanation with llms: An industrial study. pages 620–631, 09 2025.

[5] Leilani Battle and Alvitta Ottley. What exactly is an insight? a literature review. *arXiv preprint arXiv:2307.06551*, 2023.

[6] Kasturi Bhattacharjee, Rashmi Gangadharaiah, Kathleen McKeown, and Dan Roth. What do users care about? detecting actionable insights from user feedback. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Industry Track)*, pages 239–246, Seattle, Washington, USA, 2022. Association for Computational Linguistics.

[7] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

[8] Ishaya Gambo, Roseline Ogundokun, Ezekiel Ogundepo, Sweta Srivastava, Saurabh Agarwal, and Wooguil Pak. Mobile app review analysis for crowdsourcing of software requirements: a mapping study of automated and semi-automated tools. *PeerJ Computer Science*, 10:e2401, 11 2024.

[9] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do twitter users say about software? In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 96–105, 2016.

[10] Aki Härmä and Rim Helaoui. Probabilistic scoring of validated insights for personal health services. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6. IEEE, 2016.

[11] R. L. Jørnø and K. Gynther. What constitutes an 'actionable insight' in learning analytics? *Journal of Learning Analytics*, 5(3):198–221, 2018.

[12] Z. S. Li. Leveraging user feedback for requirements through trend and narrative analysis. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*, pages 386–390, Hannover, Germany, 2023. IEEE.

[13] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 116–125, 2015.

[14] R. R. Mekala, A. Irfan, E. C. Groen, A. Porter, and M. Lindvall. Classifying user requirements from online feedback in small dataset environments using deep learning. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 139–149. IEEE, 2021.

[15] Nornadiah Mohd Razali and Bee Yap. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *J. Stat. Model. Analytics*, 2, 01 2011.

[16] S. S. Mukku, M. Soni, J. Rana, C. Aggarwal, P. Yenigalla, R. Patange, and S. Mohan. Insightnet: Structured insight mining from customer feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics, 2023.

[17] Muhammad Naeem, Wilson Ozuem, Kerry Howell, and Silvia Ranfagni. A step-by-step process of thematic analysis to develop a conceptual model in qualitative research. *Qualitative Market Research: An International Journal*, 26(1):123–146, 2023.

[18] REFSQ Steering Committee. REFSQ-2026 user review dataset. `https://github.com/jsdabrowski/REFSQ-2026/blob/main/dataset/dataset.txt`, 2026. Accessed: 2026-02-10.

[19] Shadman Sakib, Oishy Fatema Akhand, Tasnia Tasneem, and Shohel Ahmed. From reviews to requirements: Can llms generate human-like user stories?, 2026.

[20] Faiz Ali Shah, Ahmed Sabir, and Rajesh Sharma. A fine-grained sentiment analysis of app reviews using large language models: An evaluation study. arXiv

preprint, 2024. arXiv:2409.07162v1.

[21] A. Susaiyah, M. Premkumar, A. Zasha, and N. Azaria. Schema-driven actionable insight generation and smart recommendation. arXiv preprint, 2023. arXiv:2307.13176.

[22] Omid Tajik, Jawad Golzar, and Shagofah Noor. Purposive sampling. *Research Education/ Non-empirical Article Purposive Sampling*, 2:1–9, 12 2024.

[23] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 14–24, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, latest edition, 2024.

[25] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. arXiv preprint, 2019. arXiv:1904.09675.

[26] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. arXiv preprint, 2023. arXiv:2306.05685.

A

Appendix

Sub-Themes	Initial Code	Example Quote from Practitioner Transcript
1. Problem Identification	Looking past proposed solutions	"Just because they say something, you don't take it at face value. You have to go find out, okay, why is it that you think that?"
	Identifying underlying problem	"A step before that is you first have to convert what they say into a...break it down into what is the problem."
	Distinguishing symptoms from root causes	"An insight explains the underlying problem, the context, and the consequences if nothing changes."
	Differentiating bugs vs. features	"First what I would look into is if support ticket is relevant for backlog or if it is a case of misuse... then it would be considered if user feedback is bug fix or new feature."
	Grouping vague/iterative feedback	"Maybe what we wrote down is not exactly what is needed. So, it it sometimes it takes a couple of iterations to to reach to exactly what is needed."
2. Impact & Evidence	Assessing business/revenue impact	"Because if it's really good for them, but maybe it's not really driving revenue for us... then there is no revenue component."
	Counting occurrences/frequency	"Other factor considered is the number of users affected and then it is the frequency of the issue."
	Assessing workflow severity	"So if it's stopping a customer from being able to do their job, it's critical that we stop our work and fix it for them."
	Requiring multiple sources	"It is based on multiple sources, not a single opinion."
3. Strategic Relevance	Alignment with company focus	"The insight then, it's not that it's not valid, it's just it's not actionable because it's not aligned with the priority for the company or the focus."
	Defining strategic boundaries	"If what is required is outside the scope of the project, we can we can debate, I would say... Can this outside of the scope of the project?"
	Aligning with roadmap themes	"Strategic relevance: Does it align with current goals, roadmap themes, or market direction?"
	Weighing customer size/context	"So then it might be worth doing certain things for the $1 million contract just because it's a huge amount. So the insight is then also takes into consideration the context of it
4. Task Readiness	Ensuring reproducibility	"What makes it relevant is if the statements given in the tickets are reproduceable and if there are enough details to describe the issue."
	Formatting into user stories	"We formulate work packages or like user stories or features and user stories."
	Defining clear context & acceptance	"Most are translated into well-scoped backlog items with context and acceptance criteria."
	Using predetermined templates	"There are templates for support tickets with predetermined fields that users can fill in order to get enough information to deduce next steps."
	Providing factful arguments	"I believe relevancy and explaining the Why behind action point with factful arguments and having a clear target of what problem it will solve."
5. Triage Challenges & Backlog Mgmt	Managing backlog volume	"The challenge it can create over time is that a big backlog of reported issues and features can be created."
	Distinguishing one-off vs systemic issues	"A constant challenge is Distinguishing between one-off issues and systemic problems."
	Cleaning up outdated feedback	"Some of these might not be relevant in the near future, so it requires proper clean up of backlog from time to time."
6. Missing Context	Interpreting anonymous or ambiguous feedback	"If a survey is anonymous we cannot derive what the problem is about if we have further questions."
	Lacking the "why" behind surface-level data	"You might have thousands of customers, but then you don't know why... you have to dig a little deeper beyond the surface layer to really understand."
	Interpreting symptom descriptions	"A constant challenge is Users describing symptoms rather than root causes."

Figure A.1: Qualitative Coding Codebook

A.1 BERT score calculation

```python
from bert_score import score

def calc_and_print(issue_name, human, cgpt_u, cgpt_c, mist_u, mist_c, gem_u, gem_c):
    print(f"\n========================================")
    print(f" Calculating BERTScore: {issue_name}")
    print(f"========================================")
    _, _, f1_cu = score(cgpt_u, human, lang="en", verbose=False)
    _, _, f1_cc = score(cgpt_c, human, lang="en", verbose=False)
    _, _, f1_mu = score(mist_u, human, lang="en", verbose=False)
    _, _, f1_mc = score(mist_c, human, lang="en", verbose=False)
    _, _, f1_gu = score(gem_u, human, lang="en", verbose=False)
    _, _, f1_gc = score(gem_c, human, lang="en", verbose=False)

    print("--- ChatGPT ---")
    print(f"Unclassified: {f1_cu.mean().item():.3f}")
    print(f"Classified:   {f1_cc.mean().item():.3f}")
    print("--- Mistral AI ---")
    print(f"Unclassified: {f1_mu.mean().item():.3f}")
```

```
    print(f"Classified:   {f1_mc.mean().item():.3f}")
    print("--- Gemini ---")
    print(f"Unclassified: {f1_gu.mean().item():.3f}")
    print(f"Classified:   {f1_gc.mean().item():.3f}")

# ==========================================
# ISSUE 2: App Crashing & Freezing
# ==========================================
hb_2 = ["A severe stability defect causes the app to freeze, hang on a blank screen
cu_2 = ["The app suffers from severe performance and stability issues likely due to
cc_2 = ["The app suffers from severe performance and stability issues (likely memor
mu_2 = ["Frequent crashes, freezes, and white screens indicate severe technical deb
mc_2 = ["Users report frequent crashes, lags, and white screens, suggesting: Memory
gu_2 = ["The app is likely suffering from memory leaks or inefficient background sy
gc_2 = ["The app is suffering from severe technical debt or poorly optimized resour

calc_and_print("Issue #2 (App Crashing)", hb_2, cu_2, cc_2, mu_2, mc_2, gu_2, gc_2)

# ==========================================
# ISSUE 3: Missing Landscape Mode
# ==========================================
hb_3 = ["A recent update removed support for landscape orientation, severely degrad
cu_3 = ["The removal of landscape support indicates a regression in feature parity,
cc_3 = ["The app no longer supports landscape orientation, likely due to a design o
mu_3 = ["The removal of landscape mode suggests a design decision to simplify the U
mc_3 = ["Users report that landscape mode is no longer available, forcing them to r
gu_3 = ["A recent update appears to have forced a portrait-only orientation lock. T
gc_3 = ["A recent update likely locked the orientation to Portrait. This ignores th

calc_and_print("Issue #3 (Landscape Mode)", hb_3, cu_3, cc_3, mu_3, mc_3, gu_3, gc_

# ==========================================
# ISSUE 4: UI State Desynchronization
# ==========================================
hb_4 = ["The mobile application state fails to refresh or sync in real-time with th
cu_4 = ["The interface suffers from poor usability due to unclear navigation struct
cc_4 = ["The interface redesign introduced usability regressions-likely due to poor
mu_4 = ["The app's interface is described as clunky, not intuitive, and failing to
mc_4 = ["The app's interface is described as clunky, not intuitive, and took a step
gu_4 = ["There is a synchronization gap between the physical Echo hardware and the
gc_4 = ["There is a disconnect between the client (App) and the hardware state (Ech

calc_and_print("Issue #4 (UI State Sync)", hb_4, cu_4, cc_4, mu_4, mc_4, gu_4, gc_4

# ==========================================
# ISSUE 5: Shopping List Accessibility
# ==========================================
```

```
hb_5 = ["The core shopping list feature is buried within the navigation hierarchy,
cu_5 = ["Users are unable to access critical features like Alexa functionality and
cc_5 = ["The workflow to access shopping lists is overly complex or buried within t
mu_5 = ["The shopping list feature is either poorly integrated into the navigation
mc_5 = ["Users describe accessing their shopping list as frustrating, suggesting: P
gu_5 = ["Users are experiencing high cognitive load and friction because the app la
gc_5 = ["The Information Architecture (IA) has become too deep and hierarchical. Us

calc_and_print("Issue #5 (Shopping List)", hb_5, cu_5, cc_5, mu_5, mc_5, gu_5, gc_5
```

A.2 Code for Shapiro-Wilk test

```
from scipy import stats

# Unclassified and Classified scores for a model
# AI BERTScore Data
unclassified_scores = [0.894, 0.884, 0.906, 0.950, 0.901, 0.894, 0.873, 0.865, 0.86
classified_scores = [0.907, 0.891, 0.906, 0.901, 0.914, 0.881, 0.867, 0.885, 0.853,

# 1. Calculate the differences between the paired scores
differences = [c - u for u, c in zip(unclassified_scores, classified_scores)]

# 2. Check for Normality (Shapiro-Wilk Test)
shapiro_stat, shapiro_p = stats.shapiro(differences)
print(f"Shapiro-Wilk p-value: {shapiro_p:.4f}")

# 3. Choose the Test based on the Shapiro p-value
if shapiro_p > 0.05:
    print("Data looks NORMAL. Running Paired T-Test...")
    t_stat, p_val = stats.ttest_rel(unclassified_scores, classified_scores)
    print(f"T-Test p-value: {p_val:.4f}")
else:
    print("Data is NOT NORMAL. Running Wilcoxon Signed-Rank Test...")
    w_stat, p_val = stats.wilcoxon(unclassified_scores, classified_scores)
    print(f"Wilcoxon p-value: {p_val:.4f}")
```

A.3 Code for Wilcoxon Signed-Rank Test

```
from scipy import stats

# --- Actionability Data ---
unclassified_scores = [4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

```
classified_scores = [4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]


# 2. Run the Wilcoxon Signed-Rank Test
w_stat, p_val = stats.wilcoxon(unclassified_scores, classified_scores)

print(f"Wilcoxon p-value: {p_val:.4f}")

# 3. Interpretation Logic:
if p_val < 0.05:
    print("Result: Statistically Significant Difference! (Reject the Null Hypothesi
else:
    print("Result: No Significant Difference. (Fail to reject the Null Hypothesis)"
```

A.4 Human Baseline ("Ground Truth")

Wells Fargo

1. Biometric Preferences Resetting

   Human Baseline Insight: The app is failing to retain user preferences for Touch ID and fingerprint authentication, forcing users to repeatedly re-enable biometrics after updates or session expirations. (Priority: High). Action: Investigate local secure storage/keychain and ensure biometric state variables persist across the app lifecycle.

2. Frequent Forced Updates

   Human Baseline Insight: Users are experiencing severe workflow interruptions due to frequent, mandatory app updates that block immediate account access. (Priority: Medium). Action: Implement flexible update policies allowing users to skip non-critical patches and consolidate minor releases to reduce overall update frequency.

3. Missing Auto/Recurring Payments

   Human Baseline Insight: The mobile application lacks feature parity with web banking, specifically missing the ability to set up, manage, and edit recurring automated payments. (Priority: High). Action: Integrate recurring payment APIs into the mobile bill pay flow, allowing users to schedule and modify automated transfers.

4. Login Failures & Blank Screens

   Human Baseline Insight: A critical authentication or rendering defect is preventing users from accessing their accounts, often resulting in a blank screen

post-login or a complete sign-in failure. (Priority: Critical). Action: Debug the session initialisation and UI rendering sequence post-authentication, and implement fallback error messaging for timeouts.

5. Missing 2-Factor Authentication (2FA)

   Human Baseline Insight: The application lacks two-factor authentication (2FA), creating a perceived security vulnerability and failing to meet modern user expectations for financial data protection. (Priority: High). Action: Introduce optional SMS or Authenticator-based 2FA within the user security settings.

6. Mobile Deposit Failure

   Human Baseline Insight: The mobile check deposit feature is systematically failing to process submissions, likely due to image capture validation errors or backend processing timeouts. (Priority: High). Action: Review mobile deposit telemetry logs to identify the drop-off point and improve error handling for camera capture and API submission.

7. UI Freeze in Message Centre

   Human Baseline Insight: Navigating to the in-app message centre causes a UI deadlock, freezing the screen and forcing users to forcibly restart the application to regain navigation. (Priority: High). Action: Fix the routing state and unhandled asynchronous loading on the messages screen to prevent UI thread blocking.

8. Forced Username Migration

   Human Baseline Insight: A recent security update enforced new username criteria without providing a seamless migration path, locking out and frustrating legacy users whose existing credentials are now deemed invalid. (Priority: Medium). Action: Implement a guided credential migration flow upon login rather than abruptly rejecting legacy usernames.

Alexa

1. Missing Search Feature: The application lacks a global search capability, significantly increasing user friction when attempting to find specific settings or skills. (Priority: Medium). Action: Implement a universal search bar component in the main navigation header.

2. App Crashing & Freezing: A severe stability defect causes the app to freeze, hang on a blank screen, or crash entirely during initialisation and general use. (Priority: Critical). Action: Profile the app's memory usage and startup sequence to identify and resolve the unhandled exceptions causing thread locks.

3. Missing Landscape Orientation: A recent update removed support for land-

scape orientation, severely degrading the UX for tablet (iPad) users who mount their devices horizontally. (Priority: High). Action: Re-enable responsive landscape layouts and ensure UI constraints adapt correctly to horizontal device rotation.

4. UI State Desynchronisation: The mobile application state fails to refresh or sync in real-time with the physical Echo device's current activities, creating a disjointed experience. (Priority: High). Action: Implement WebSockets or long-polling to ensure the mobile UI accurately reflects the live hardware state.

5. Shopping List Accessibility: The core shopping list feature is buried within the navigation hierarchy, causing user frustration when trying to access it quickly in stores. (Priority: Medium). Action: Elevate the Shopping List shortcut to the primary bottom navigation bar or home screen dashboard.

LinkedIn

1. Post-Update Crash & Blank Screen: A critical regression in the recent update causes the application to crash on launch or display a blank screen, blocking all access. (Priority: Critical). Action: Roll back the faulty routing module introduced in the latest patch and patch the app initialisation sequence.

2. Profile Editing Failures: Users are experiencing a backend synchronisation error or UI block that prevents them from saving edits or viewing their own profiles. (Priority: High). Action: Debug the PUT/POST requests on the profile editing endpoints and ensure the UI correctly parses the returned profile payload.

3. Obtrusive Pop-Ups & Setup Loops: Users are trapped in repetitive "setup" flows and bombarded with aggressive pop-ups asking for email contact syncs. (Priority: High). Action: Refactor the onboarding state flags to ensure users who dismiss or complete the contact sync are not prompted again.

4. Missing In-App Article Creation: The mobile client lacks feature parity with the desktop web version, specifically the ability to author and publish long-form blog posts/articles. (Priority: Medium). Action: Integrate a mobile-optimised rich text editor to allow users to draft and publish articles natively.

5. Missing Profile Search: The core search functionality appears to be broken or missing, preventing users from searching for specific profiles. (Priority: Critical). Action: Restore the global search bar functionality and verify the connection to the user indexing service.

6. Empty Notification Feed Bug: A data-fetching bug causes the notifications and connection requests tabs to display badges but load empty screens when clicked. (Priority: High). Action: Fix the API pagination or rendering bug in the notification feed view that is failing to populate the list components.

Messenger

1. Accidental Camera Trigger: Scrolling down vertically in the conversation list over-sensitises the gesture recogniser, accidentally triggering the camera module. (Priority: High). Action: Adjust the swipe-to-camera gesture threshold to prevent conflicts with standard vertical scrolling.

2. Swipe Gesture Regressions: A recent update altered standard swipe behaviours, removing the ability to swipe-to-delete threads and replacing it with a confusing "move to active" action. (Priority: Medium). Action: Restore the standard left-swipe context menu allowing users to delete, mute, or archive conversations.

3. App Crashing on Launch: The application fails to initialise and crashes immediately upon opening for a significant portion of users following the latest update. (Priority: Critical). Action: Identify and hotfix the fatal exception occurring during the app's startup lifecycle.

4. Social Graph Failures (Adding Friends): A critical backend API error prevents users from successfully adding new friends or modifying group chat participants. (Priority: Critical). Action: Investigate the social graph API endpoints returning errors when POST requests are made to add users to groups or friend lists.

5. Missing Mute Incoming Calls: Users lack granular control over VoIP notifications, explicitly requesting the ability to mute incoming audio/video calls without muting text messages. (Priority: Medium). Action: Add a "Mute Incoming Calls" toggle within the app's notification preference settings.

6. VoiceOver Accessibility Crash: Tapping a message while iOS VoiceOver is active causes a fatal app crash, completely blocking visually impaired users. (Priority: Critical). Action: Debug the accessibility labels and focus states on the message bubble components to prevent the VoiceOver rendering crash.

7. Message Delivery Delays: Network socket instability or background sync failures are causing severe (20+ minute) delays in message delivery and receipt. (Priority: High). Action: Optimise the background connection pooling and push notification payloads to ensure real-time message delivery.