Detecting Developer Emotions in GitHub Commits Using Large Language Models: Implications for Project Health

by

Fahmida Haque Fariha
21201483
Insaniyat Ishan
24141127
Mushfiq Zahid
21201281
Zawad Al Akram
21201589

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

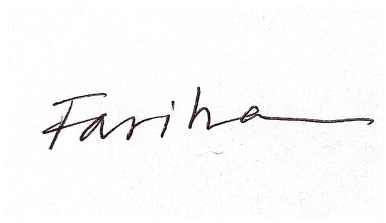
Department of Computer Science and Engineering
Brac University
October 2025

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:



Fahmida Haque Fariha
21201483

Insaniyat Ishan
24141127

Mushfiq Zahid
21201281

Zawad Al Akram
21201589

Approval

The thesis titled "Detecting Developer Emotions in GitHub Commits Using Large Language Models: Implications for Project Health" submitted by

1. Fahmida Haque Fariha (21201483)
2. Insaniyat Ishan (24141127)
3. Mushfiq Zahid (21201281)
4. Zawad Al Akram (21201589)

of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 14, 2025.

Examining Committee:

Supervisor:
(Member)

Md. Aquib Azmain
Lecturer
Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Acknowledging the emotions of developers is crucial for maintaining productivity, team morale, and project success. Developers' emotions are often reflected in their commit messages which can reveal subtle cues of satisfaction, frustration, or caution. Monitoring these signals helps project managers recognize early signs of stress and respond proactively, leading to healthier team dynamics and better project outcomes. This thesis investigates emotions in formal GitHub commit messages and their connection to long-term project health. A Project Health Dataset was built by sampling 20,000 commits from 34 repositories and aggregating them quarterly across four dimensions: bug activity, productivity, emotions, and code churn. From this, a gold-standard set of 2,000 commits was manually annotated into four categories: Satisfaction, Frustration, Caution, and Neutral. Several zero-shot models were evaluated, and a fine-tuned CodeBERT model named CommiTune was developed which was trained on both human-labeled and LLaMA-augmented data. CommiTune achieved a Macro-F1 of 0.88 and Accuracy of 86% on a held-out test set which significantly outperformed all baselines. When applied at scale, the model revealed that while short-term emotion trends show weak correlations with project metrics however, long-term patterns are strong. Frustration aligns with higher bug-fix rates, satisfaction correlates with lower churn and caution signals often precede rollback activity. This work contributes a novel dataset, a high-performing fine-tuned model and empirical evidence that developer emotions can serve as meaningful indicators of software quality and stability over time.

Table of Contents

Declaration i

Approval ii

Abstract iii

Table of Contents iv

List of Figures vii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Rationale of the Study . . . 1
- 1.3 Problem Statement . . . 2
- 1.4 Research Objectives . . . 2
- 1.5 Methodology in Brief . . . 2
- 1.6 Scopes and Challenges . . . 3

2 Literature Review 4
- 2.1 LLMs for Sentiment and Emotion Analysis . . . 4
- 2.2 Sentiment Analysis in Software Engineering . . . 5
- 2.3 Deep Learning for Emotion Detection . . . 5
- 2.4 Emotion Detection in Social Media . . . 6
- 2.5 Sentiment Analysis Methods and Challenges . . . 6
- 2.6 Biases in Pre-Trained Models . . . 7
- 2.7 Data Augmentation with Generative Models . . . 7
- 2.8 Code Churn and Project Health . . . 7
- 2.9 Literature Summary . . . 8

3 Requirements and Impacts 9
- 3.1 Final Specifications and Requirements . . . 9
- 3.2 Societal Impact . . . 9
- 3.3 Ethical Issues . . . 10

4 Dataset 12
- 4.1 Limitations in Existing Datasets . . . 12
- 4.2 Dataset Overview . . . 13
- 4.3 Data Selection and Extraction . . . 14
  - 4.3.1 Feature Engineering for Stratification . . . 14
  - 4.3.2 Proportional Sampling Strategy . . . 14

4.3.3 Bias & Limitation . . . 15

5 Data Preprocessing **16**
5.1 Objectives of Preprocessing . . . 16
5.2 Preprocessing Steps . . . 16
5.2.1 Load and Select Relevant Columns . . . 16
5.2.2 Normalize Text . . . 16
5.2.3 Remove Irrelevant Content . . . 17
5.2.4 Remove Specific Patterns . . . 17
5.2.5 Tokenization and Length Filtering . . . 17
5.2.6 Final Cleaning . . . 18
5.3 Iterative Refinement of the Emotion Model . . . 18
5.4 Label Consolidation and Reconciliation . . . 20

6 Model and Experimental Setup **22**
6.1 Model Selection Criteria . . . 22
6.2 Label Mapping and Caution Lexicon . . . 23
6.3 Experimental Protocol . . . 24
6.3.1 Dataset and Splits . . . 24
6.3.2 Evaluation Metrics and Reproducibility . . . 24
6.4 Zero-shot Baselines . . . 25
6.5 Fine-Tuning Pipeline . . . 25
6.5.1 Training and Hyperparameters . . . 26
6.5.2 Model Integration . . . 26
6.5.3 Fine-Tuning LLaMA 3.1 . . . 26
6.6 Data Augmentation . . . 27

7 Experiments Results **28**
7.1 Zero-Shot Results . . . 28
7.2 Fine-Tuning Results . . . 29
7.2.1 Experiment A: RoBERTa-large-MNLI (Generalist) . . . 29
7.2.2 Experiment B: CodeBERT (Specialist) . . . 30
7.2.3 Experiment C: DistilRoBERTa (Pattern Mimic) . . . 31
7.2.4 LLaMA 3.1 8B (PEFT / QLoRA) . . . 32
7.2.5 Comparative Summary and Practical Implications . . . 33
7.3 CommiTune Results . . . 33
7.3.1 Caveats and limits to generalization . . . 35
7.4 Ablations: Weighted vs Standard Loss . . . 36
7.4.1 Interpretation and trade-offs . . . 36

8 Project Health Dataset Construction **38**
8.1 Bug-Related Activity Metrics . . . 38
8.2 Bug-Inducing Commit Heuristic (pBIC) . . . 39
8.3 Productivity Metrics . . . 39
8.4 Code Churn Metrics . . . 40

9 Project Health Churn Correlation Analysis **42**
9.1 Methodology: Data Aggregation and Metric Definition 42
9.2 Analysis Framework 42
9.3 Results: Emotion as an Indicator of Project Health 43
9.3.1 Short-Term Analysis: A Null Finding 43
9.3.2 Long-Term Analysis: A Strategic Indicator of Culture and Risk 43
9.4 Results: Emotion as an Indicator of Code Churn Dynamics 45
9.4.1 Short-Term (Quarterly) Churn Analysis 45
9.4.2 Long-Term (Repository-Level) Churn Analysis 45
9.5 Overall Correlation Analysis 47

10 Conclusion and Future Work **48**
10.1 Conclusion 48
10.2 Contributions 48
10.3 Future Work 49

Bibliography **52**

A Sampled Repositories and Commit Counts **53**

B Preprocessing and Keyword Details **55**
B.1 Text Cleaning Regex Patterns 55
B.2 Bug-Fix and Failure Keyword Lists 55

C Annotation Process and Label Consolidation **57**
C.1 Initial Annotation Guidelines (Seven-Emotion Scheme) 57
C.2 Label Scheme Refinement and Consolidation 57
C.3 Final Label Definitions and Reconciliation Rules 58

D Reproducibility Details **60**
D.1 Fine-Tuning Hyperparameters 60
D.2 Code, Data, and Models (For Review) 60

E Caution Lexicon Used in Data Reconciliation **62**

List of Figures

3.1 Flowchart of work plan 11

4.1 Flowchart of Sampling 15

5.1 Commits before and after preprocessing 17
5.2 Flowchart of preprocessing 18
5.3 Emotion class distribution for Annotator-1 and Annotator-2 labelling 20
5.4 Workflow for Reconciling Annotated Data 21

7.1 RoBERTa-large-MNLI — training and validation loss per epoch. . . . 30
7.2 CodeBERT — training and validation loss per epoch. 30
7.3 Confusion matrix for the fine-tuned CodeBERT model at its best checkpoint. 31
7.4 Confusion matrix for the fine-tuned LLaMA 3.1 8B model. 33
7.5 Learning curve for the CommiTune model, showing F1 Macro score and losses per epoch on the validation set. 34

9.1 Contemporaneous Correlation Matrix of Emotion and Project Health Metrics. 44
9.2 Repository-Level Correlation Matrix of Emotion and Project Health Metrics. 44
9.3 Repository-Level Correlation Matrix for Code Churn. 46

Chapter 1

Introduction

1.1 Background

The fast-paced field of software development requires developers to constantly solve problems, working long hours in high-pressure environments to keep up with surging innovations. Constantly meeting tight deadlines leaves developers prone to elevated stress levels, anxiety and burnout. Since companies rely heavily on GitHub and other collaborative platforms for software development, the commit messages left by developers, serving as their digital footprints, could provide valuable insights into their emotions and state of mind. By harnessing the high-end technology of Large Language Models (LLMs), we now have the power to analyze these messages.

1.2 Rationale of the Study

A recent analysis of over 2.25 million GitHub commit messages revealed that 18% carry negative sentiment which usually signal frustration and burnout, especially when dealing with bug fixes and issues [6]. Research shows a direct correlation between the emotional tone of these commit messages with the complexity of the task at hand [6]. Moreover, another exploratory study on the relationship between developer emotions and software bugs showed that commits related to bugs tend to be more negative and they correlate with the introduction of new bugs in the software [12]. Therefore, from these findings, we uncover a crucial point—the emotions expressed in these raw commit messages can provide a deeper understanding of how they affect project health.

The intersection of workload and mental health have been the focal point of many studies. One such study found that burnout and depression are strongly related ($r = 0.52$). The same was observed for burnout and anxiety ($r = 0.460$) [9]. Software developers are especially prone to symptoms that indicate burnout like exhaustion and cynicism, these in turn can lead to project delays and a decline in both code quality and net productivity.

To overcome this, emotion detection of GitHub commit messages can serve as a strong tool to identify early signs of mental fatigue. State-of-the-art LLMs like BERT, GPT-3, and GPT-4, which can offer deep contextual understanding, detect nuanced inferences, and generate appropriate and coherent responses [28][24], should be perfect for comprehending the subtleties in developers' commit messages. Models

like Flan-UL2 are also suitable, as they outperform text-davinci-003 in emotional inference and sentiment analysis tasks [29].
However, the performance of these advanced models can be further enhanced through zero-shot and few-shot inferences, transfer learning, and fine-tuning [29]. Using these for emotion detection on GitHub commit messages can provide actionable insights to identify how developer emotions, like frustration, satisfaction, or caution, impact project outcomes. In an industry so focused on precision, potentially defining the emotional undercurrents of developers might be the missing piece to ensure project success along with developer well-being.

1.3 Problem Statement

While revolutionary models like BERT, GPT-3 and GPT-4 have made great strides in NLP, they struggle to generalize to domain-specific datasets, exhibiting significant performance degradation when applied to the terse, jargon-heavy language of software development [28][24]. This limitation is compounded by the scarcity of labeled, domain-specific data and the severe class imbalance inherent in commit messages, where most are emotionally neutral [19]. Consequently, current models fail to reliably detect the nuanced emotional signals that may precede burnout or project instability. This creates two critical research gaps: first, the absence of a highly accurate, domain-adapted emotion detection model for software engineering; and second, a lack of empirical clarity on whether developer emotions act as short-term operational predictors or long-term structural indicators of project health.

1.4 Research Objectives

This research aims to bridge these gaps through a comprehensive analysis of developer emotions. Our research objectives are to:

1. Develop a reliable labeled dataset for training, created through rigorous manual annotation to ensure high quality and consistency.

2. Develop a state-of-the-art emotion detection model for developer commit messages by leveraging domain-specific pre-training and overcoming data scarcity with synthetic data augmentation.

3. Systematically compile a dataset of objective project health metrics by defining and computing indicators for bug-fixing activity, developer productivity, and code churn.

4. Quantify the relationship between developer emotion and project health through a rigorous statistical analysis that distinguishes between short-term predictive signals and long-term structural correlations.

1.5 Methodology in Brief

To achieve these objectives, a multi-phase methodology was employed. We began by creating a “gold standard” dataset through stratified sampling and meticulous

manual annotation of 2,000 commits. A comparative evaluation of various LLMs led to the selection of CodeBERT, a domain-specific model, for fine-tuning. Its performance was then elevated to a state-of-the-art level using data augmentation. Concurrently, project health metrics were calculated from 20,000 commit messages. Finally, the outputs from our validated emotion model were correlated with these metrics using a multi-stage statistical framework to analyze both immediate and long-term relationships.

1.6 Scopes and Challenges

This study is scoped to the textual analysis of commit messages from 34 high-profile public repositories on GitHub [17]. It does not incorporate other data sources such as issue tracker comments or pull request discussions.

The primary challenge encountered was the significant scarcity of high-quality, manually labeled data for developer emotion in the academic literature. This was partly addressed through rigorous manual annotation, but the task was inherently difficult since emotions are highly subjective and annotator agreement can vary. To further mitigate data scarcity, synthetic augmentation was employed. However, this approach also faced computational limits, as generating large-scale synthetic samples with the LLaMA 3.1 (8B) model proved resource-intensive.

A secondary challenge arose in the computation of code churn metrics. Due to the high processing and storage demands, churn analysis could not be extended to the largest repositories. Instead, it was applied only to a manageable subset of small and medium-sized repositories.

Chapter 2

Literature Review

2.1 LLMs for Sentiment and Emotion Analysis

Large Language Models (LLMs) such as GPT-3, GPT-4, and BERT have emerged as strong sentiment analysis tools. In tasks such as Aspect-Based Sentiment Analysis (ABSA) and broader sentiment classification, LLMs learn from large datasets and generalize well across different domains [26][27]. However, LLMs are not without their limitations.GPT-3 and GPT-4 excel in general sentiment analysis surpassing smaller and fine-tuned models like Bidirectional Encoder Representations from Transformers (BERT) but can struggle with more specialized tasks (e.g on Aspect Based Sentiment Classification (ABSC), where ChatGPT out performs fine-tuned BERT by 10% in terms of average accuracy and 8% in terms of average F1 score) [26][27]. ChatGPT is effective at simple sentiment categorization but might suffer when a deeper knowledge is required, such as in ABSA or determining the cause of feelings. In spite of LLMs large scale and impressive capabilities, it still faces challenges when it comes to handling domain-specific subtleties which require more focused and fine-tuned approaches [26][27]. But models fine-tuned for specific datasets, such as BERT and RoBERTa routinely outperform ChatGPT [27][29]. LLMs such as Flan-UL2 and Llama 2-Chat perform well in both zero-shot and few-shot contexts where there exists tasks with limited training data; falling behind Smaller Language Models (SLMs) on larger domain specific datasets (e.g., RoBERTa outperforming bLLMs by 30.6% on GitHub) [29][28]. In text-based emotion recognition, BERT is particularly effective at identifying subtle emotional cues achieving state of the art results [15]. BERT's bidirectional context analysis allows it to surpass previous models at detecting complex emotions, when fine-tuned for specific tasks [15]. Recent improvements in EmoLLMs which focuses on emotion intensity and sentiment strength regression shows how LLMs are being tuned for more precise affective emotional analysis [31]. EmoLLMs, a suite of open-source large language models (LLMs), fine-tuned with Affective Analysis Instruction Datasets (AAID), beat GPT-4 in specific emotion-related tasks, providing more precise predictions across various domains [31][8]. While LLMs continue to push the boundaries of sentiment analysis, its success is reliant on fine tuning and task-specific adaptation for it to achieve optimal results across multiple domains [32][8].

2.2 Sentiment Analysis in Software Engineering

In software engineering sentiment analysis of GitHub commit messages and comments is used to assess developer emotions offering insights into productivity, team dynamics, and issues like bug introduction and burnout. Earlier studies conducted using tools like SentiStrength found most of 60,425 GitHub commit comments to be neutral; with significant emotional trends directly linked to factors like programming language, time of day, and team distribution. Java projects received more negative comments than other languages, with Mondays showing the highest negative sentiment, while afternoons received more positive feedback [4]. Another analysis of 2.25 million commits found that the majority of the commits (74.74%) were neutral, while negative feelings were higher (18.05%) than positive ones (7.19%). Interestingly, Tuesdays tend to have the most negative sentiment overall [6]. There exists a link between developer sentiment and software bugs, categorizing commits into Fix-Inducing Changes (FICs), Parents of FICs (pFICs), Fixing Changes (FCs), and Fix-Inducing Fixes (FIFs). It finds that FICs, pFICs, and FCs have more emotional content, with negative sentiment being higher in bug-related commits and leading to further FIFs [12]. On smaller or imbalanced datasets larger LLMs like Llama 2-Chat and Vicuna have performed well in zero shot and few-shot learning scenarios. However, fine-tuned smaller LLMs like BERT and RoBERTa perform better on bigger and balanced datasets indicating a trade-off between model size, job difficulty, and dataset features [28]. Valence, Arousal, and Dominance (VAD) has the potential to diagnose burnout and assess developer productivity but its effectiveness is limited by general-purpose lexicons. In spite of this, VAD measures open up opportunities for improved monitoring of developer well-being and project health [5].

2.3 Deep Learning for Emotion Detection

Emotion detection from text using deep learning is vital in applications like customer feedback analysis, social media monitoring, and human-computer interaction [21][18]. Research has found that traditional approaches face challenges such as managing complex emotions, sarcasm, and informal language, leading to inconsistent outcomes. On the other hand, deep learning models, including Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNNs), achieve over 90% accuracy by effectively capturing context and sentence structure [21]. Furthermore, a similar study showed that LSTM combined with CNN yields consistently high results across diverse datasets [21]. Additionally, BERT models reach similar accuracy, particularly when fine-tuned for emotion detection, outperforming lexicon-based techniques that typically achieve only 60-70% accuracy [18][14]. Research has also demonstrated that attention mechanisms and recurrent neural networks like DialogueRNN enhance performance, achieving high accuracy in detecting emotions within conversations; however, they still encounter challenges distinguishing similar emotions, such as happiness and excitement, leading to a precision drop of about 5-10% [10]. Despite these advancements, challenges like sarcasm detection, short text processing, and limited emotion-labeled datasets remain [19][11]. Moreover, multimodal data integration and enhancing models like GPTs are proposed to push accuracy rates beyond 98% when optimized for diverse contexts and datasets [22][11].

2.4 Emotion Detection in Social Media

Emotion detection in social media has emerged as an essential area of research, especially as platforms like Twitter and Facebook serve as major outlets for expressing emotions. Machine learning (ML) and deep learning techniques are widely applied to identify emotions in textual data. A key study examined seven ML algorithms for recognizing emotions from social media conversations and found that the Generalized Linear Model (GLM) outperformed other models with an accuracy of 92% and an F1 score of 0.901,making it the most ideal in this situation for detecting emotions. Conversely, though, Random Forest and Decision Trees models showed reduced effectiveness, demonstrating that for social media text analysis, certain algorithms are better suited [16]. In order to bring out features, by integrating Natural Language Processing (NLP), this research showed that integrating these techniques greatly increased the precision of depression detection. On the other hand, for a more comprehensive analysis, it also highlighted the need for future research to include biometric and multimodal data [20].Both research highlight how machine learning can improve the identification of emotions in social media, but they also highlight difficulties in interpreting sarcasm and informal language, as well as the requirement for more reliable, multimodal methods.

2.5 Sentiment Analysis Methods and Challenges

Sentiment analysis, also known as opinion mining, is a branch of NLP that focuses on extracting and analyzing subjective information from textual data, such as social media posts, product reviews, and blogs. Research has found that methods are generally categorized into Lexicon-Based, Machine Learning (ML) techniques, and Hybrid approaches that combine both [24][19]. On the other hand, Lexicon-Based techniques use predefined sentiment dictionaries but often struggle with informal language, sarcasm, and domain-specific meanings [24][3]. A similar study showed that ML models like Support Vector Machines (SVM), Naive Bayes, Long Short-Term Memory (LSTM) networks, and BERT have demonstrated greater flexibility and accuracy, particularly in handling larger datasets [24][19][3]. However, challenges such as sarcasm detection, polysemous word interpretation, and the need for domain-specific lexicons persist [24][19]. Research has indicated that Hybrid approaches, employing TF-IDF for word weighting and combining it with models like Naive Bayes, have shown improved performance; however, they still face issues in managing irony and diverse language patterns [24][3]. Furthermore, emotion detection, a subset of sentiment analysis, leverages models such as Ekman's six basic emotions and Plutchik's Wheel of Emotions, utilizing deep learning models like BERT for granular analysis [19]. While BERT achieves a high F1 score of 89% in specific cases, a similar research effort has highlighted that limited domain-specific datasets and challenges in word-sense disambiguation remain significant obstacles [19][3]. Additionally, multilingual sentiment analysis complicates tasks, requiring models to address translation issues and cultural differences, as noted in various studies [24][3]. Transfer learning techniques, such as GPTs, show potential for improving sentiment and emotion detection across various domains, yet research suggests that enhancements for real-time and accurate context handling are still necessary to meet evolving challenges [19][3].

2.6 Biases in Pre-Trained Models

Pre-trained language models (PLMs) like BERT and RoBERTa are effective in sentiment analysis and emotion detection but are prone to biases. These biases often arise from the selection of emotion categories, prompt phrasing, and training data. Studies have shown that PLMs perform well in basic sentiment classification but struggle with fine-grained emotion detection, particularly for complex emotions like joy or fear. The accuracy of these models can fluctuate depending on prompt wording. To mitigate biases, improving label selection and designing better prompts are essential for enhancing emotion detection accuracy [23] [16].

2.7 Data Augmentation with Generative Models

A critical challenge in sentiment analysis of commit messages is class imbalance, i.e., most commits are neutral, with relatively few expressing frustration or caution. Data augmentation has emerged as a solution. Recent studies explore generative models for augmenting scarce emotion data. Back-translation and paraphrasing with transformer models [7] increase diversity, while more recent approaches leverage GPT-style models to synthesize realistic training samples. Domain-specific emotional data can be created with the help of conditional text generation [31], improving balance across categories. In software engineering, synthetic commits generated with contextual cues could mitigate underrepresentation of rare emotions like "satisfaction" or "caution." However, generative augmentation is not without risks: synthetic data may introduce noise or amplify biases. Nonetheless, engineering prompts carefully along with filtering strategies have shown improvements in downstream emotion detection tasks [31]. Thus, incorporating augmentation with LLMs represents a promising direction for addressing imbalance in commit message sentiment analysis.

2.8 Code Churn and Project Health

Code churn, which is typically measured as the number of lines added, deleted, or modified in commits, has been widely recognized as an indicator of project stability and long-term health [30]. High churn often reflects instability, frequent rework, or evolving requirements, all of which correlate with increased defects [1]. One research demonstrated that churn was highly predictive of post-release failures in large-scale systems when combined with dependency metrics [1]. Similarly, a related research found churn to be a strong predictor of software vulnerabilities, which linked frequent modifications with higher security risks [2]. Beyond defects, churn is also linked to corrective effort and maintainability. One study showed that projects with high churn tend to have a greater proportion of corrective commits, which makes it a marker of reduced project quality [13]. These findings establish churn as both an operational signal of short-term instability and a structural indicator of long-term project health. Despite this, churn has largely been studied in isolation from human-centric factors such as developer sentiment. Existing research frames churn as a purely technical metric, leaving a gap in understanding how emotional dynamics within development teams interact with churn patterns. This work extends prior studies by examining churn alongside developer emotions, bridging technical and

socio-emotional indicators of project health.

2.9 Literature Summary

The literature shows that while LLMs like GPT-3/4 excel in general sentiment analysis, fine-tuned models such as BERT and RoBERTa consistently perform better on domain-specific tasks like commit message analysis. Within software engineering, sentiment analysis has revealed clear links between developer emotions and project outcomes, with negative emotions often tied to bug-related activity. Deep learning models (LSTM, CNN) achieve high accuracy in emotion detection, but challenges remain with sarcasm, short texts, and data imbalance. This promotes the use of data augmentation techniques such as back-translation and generative models.
On the project health side, metrics like bug activity, productivity, and code churn are widely recognized as indicators of software quality. In particular, churn is strongly associated with defects, vulnerabilities, and corrective effort. However, existing studies treat these technical metrics separately from socio-emotional factors. Together, the literature points to a key gap: while emotions and health metrics are individually well-studied, their integration remains unexplored. This opens the opportunity for our work, which combines emotional and technical signals to provide a holistic view of project health.

Chapter 3

Requirements and Impacts

3.1 Final Specifications and Requirements

Building on the objectives defined in Chapter 1.3, the system requirements were derived to ensure both research validity and practical applicability. The central goal of this work is to develop a state-of-the-art emotion detection model for developer commit messages, and to integrate these outputs with project health metrics such as bug-fix activity, productivity, and code churn. The model should be capable of classifying four emotions: *Frustration, Satisfaction, Caution,* and *Neutral*, with a target Macro F1 ≥ 0.85 on a gold-standard dataset. Beyond classification, the system must detect short-term predictive and long-term structural correlations between emotions and project health metrics (bug-fixing, churn, productivity) with statistical significance ($p < 0.05$).
The workflow follows five stages (see 3.1): (1) dataset preparation through collection, filtering, annotation, and augmentation; (2) model training with fine-tuned domain-specific LLMs such as CodeBERT and RoBERTa; (3) evaluation against zero/few-shot baselines; (4) integration of outputs into repository–quarter datasets; and (5) statistical analysis of correlations.
Non-functional requirements emphasize efficiency and scalability: the framework is designed to be modular and reproducible, enabling substitution of models and augmentation methods. Constraints include limited labeled data, reliance on public repositories, and English-only commit messages. Compliance standards were upheld through the IEEE Code of Ethics, ODC-By licensing for datasets, and GDPR anonymization of all commit data.

3.2 Societal Impact

The implications of this research extend well beyond technical boundaries. By detecting emotional cues in developer communications, it becomes possible to identify early signs of burnout or frustration before they escalate. This helps teams maintain healthier work environments and more stable software projects. On a broader cultural level, this work encourages a more empathetic approach to engineering—one that values emotional awareness as part of effective collaboration, especially for globally distributed teams. Such awareness can reduce miscommunication and increase inclusion across linguistic and cultural divides.

However, with these benefits come important responsibilities. If emotional data were ever traced back to individual developers, it could create privacy or reputational risks. These concerns are mitigated in this study through strict anonymization and aggregation, ensuring that insights are always drawn at the group or project level and not the individual. Still, a potential negative consequence lies in the misuse of such technology—for example, using emotional analytics in hiring or performance evaluations. This possibility demands the need for strong ethical safeguards and clear policies to ensure that emotion detection promotes human well-being and does not undermine it.

3.3 Ethical Issues

This research is guided by ethical considerations. Pre-trained models such as CodeBERT, trained primarily on English and code-heavy corpora, may carry linguistic and cultural biases that underrepresent non-English-speaking developers. While privacy concerns exist in emotion inference from commit messages, these were mitigated through strict anonymization and aggregation. Reproducibility was prioritized by releasing the full processing pipeline and training scripts on GitHub. Any future human-involved annotation would require ethical review or IRB approval. Academic integrity was ensured through transparent citation and responsible data augmentation. However, models may still misinterpret subtle cues such as sarcasm or humor, which is an inherent limitation that emphasizes cautious interpretation and humility in deployment.

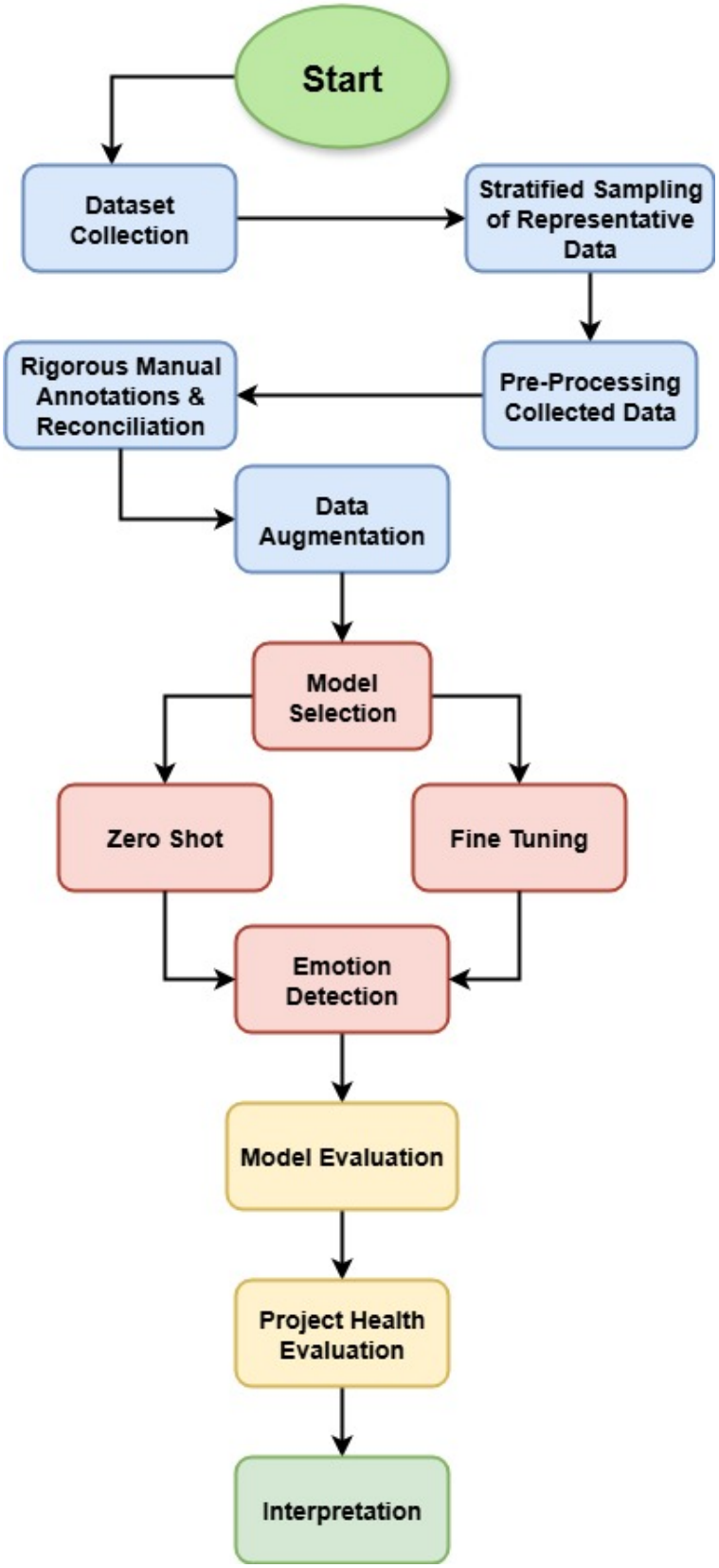


Figure 3.1: Flowchart of work plan

Chapter 4

Dataset

Today GitHub operates as the worldwide leading hub for collaborative software development, as more than 100 million programmers participate in millions of projects across every field [25]. Developers utilize its extensive coverage which makes the platform an important research venue for understanding programmer sentiments via commit messages. Currently available datasets that are used to perform sentiment analysis in software engineering have many limitations which reduce their effectiveness in identifying subtle emotional expressions in highly technical commit messages.

4.1 Limitations in Existing Datasets

Lack of High-Quality and Representative Data: Professional and personal projects typically exist within the same datasets which produce inconsistent commit formats while reducing the records of actual professional software sentiments [6]. The datasets regularly display small sample sizes of repositories as well as inadequate labelling annotation [4][28]. The technical complexity of commit messages makes manual annotation difficult, thus researchers depend on heuristic methods which create labeling errors and inhibit detection of delicate emotional fluctuations [28][3].

Domain Adaptation Challenges: Sentiment analysis datasets lack engineering focus since they originated from social media and product review contexts which makes broad algorithm applications problematic [29][16]. BERT alongside GPT present limited effectiveness in domain adaptation work since they perform poorly without executing fine-tuning on domain tailored datasets[15].

Class Imbalance in Sentiment Labels: A majority of commit messages fall into a neutral category, therefore, models struggle to identify genuine emotional variances in unbalanced distributions [6][3]. Dead zones emerge from datasets that fail to tell apart neutral content from messages with borderline sentiment, thereby diminishing their ability to accurately classify text [28].

Extensive Preprocessing Requirements: Data from GitHub scrapes typically lacks predefined metadata features for commit aims along with developer position information and issue relationship tracking [5]. These datasets need extensive cleaning procedures and the brief nature of commit messages also hinders sentiment detection algorithms [29][15].

Informal and Noisy Language in Commit Messages: Large datasets of commit messages usually contain both professional and casual language which makes it difficult to classify sentiments [6]. Furthermore, open-source repositories usually contain a large number of commit messages written by students, inexperienced developers, and hobbyists who use unstructured, highly informal language. This leads to inconsistencies that affect model performance. Model accuracy suffers because emotion recognition algorithms trained on informal datasets show difficulties in processing corporate commit messages within real-world industrial contexts [15]. Preprocessing experimental datasets for message cleaning takes up excessive time while demanding substantial work before analysis [5].

Sentiment	**Commit Message**
Negative	"I effed up my commit, sorry...."
Neutral	"This was pulled from the old makefiles and gitian descriptors. It's required for deterministic builds."
Positive	"I am geeking out over the bug and the fix. srsly awesome."

Table 4.1: Example of informal language in labeled GitHub dataset [28]

Achieving better sentiment classification requires datasets to have balanced mixtures of large size, professional content, and well-structured, diverse elements.

4.2 Dataset Overview

The dataset used in this research, available on Kaggle, has 4.3 million commit messages extracted from 34 of the most popular projects (based on number of forks and star count) available on GitHub[17]. This dataset is thus representative of only high-quality professional software development projects instead of casual or hobbyist projects. The Open Data Commons Attribution License (ODC-By) v1.0 strengthens dataset credibility through policies which enable full transparency and compliance while ensuring free unrestricted use. The dataset being publicly accessible, allows researchers the chance to verify its origin and reproduce studies; even contributing to enhancements making it more reliable for sentiment analysis. The ODC-By license offers both academic and commercial usage permissions, preventing limitations found in restrictive datasets, making it quintessential for this research. Five key columns were used to structure this dataset which were the commit (unique identifier for every commit message), Author (name or alias of the programmer), Date (timestamp at which change was carried out), Message (a short description of the change) and Repo (repository) And these are the attributes that enables trend analysis for all projects/contributors. The dataset consists of 34 popular repositories with more than 89,000 different programmers, resulting in a broad view on the software development workflows. Given its scale and diverse nature, it is a valuable source of commit behavior, project evolution, and collaborative coding practices studies.

4.3 Data Selection and Extraction

To ensure efficiency, scalability and data quality a 20,000 commit message subset was used as a representative sample of the 4.3 million dataset in this research. Analyzing the entire dataset would have been computationally expensive and time-consuming. On the contrary, creating a well-structured subset would maintain statistical representativeness without overwhelming resource constraints. This sample was designed to preserve key distributions so that findings can be reasonably generalized to the full dataset, subject to the limitations discussed in Section 4.3.3. To make sure that the research is both practically feasible and methodologically sound while maintaining data richness, this approach had to be taken.

4.3.1 Feature Engineering for Stratification

Project Size Calculation: To ensure project diversity, the commit count of each repository had to be measured. Each commit entry in the repository was assigned a project size value based on the total number of commits that are present in the repository. This measure functioned as a stratification factor to ensure equal representation of projects at different scales.

Temporal Grouping: Timestamp data was converted into uniform structured time parameters to apply temporal segmentation. The timestamp for each commit determined which quarter it belonged to so sentiment shifts could be tracked over time. Sample measurements were also tracked evenly through different time periods. For example: Mon Jan 11 20:17:10 2021 +0100 in 'Date' will be labeled as "2021Q1"

Stratification Label Formation: The following factors were combined to construct a stratification label to ensure proportional sampling:

- Project size (total commits per repository)
- Repository name
- Quarterly time period

All commits received specific stratification labels to represent their repository features while showing associations between timeline periods. This approach of sampling enables proper representation of the original dataset in our sampled dataset. It ensures projects from all sizes, repos and time periods are included in our sampled data.

Example: A row in the dataset having the stratification label of 100_python_2020Q1 means: Project size = 100 commits, Repo = python, Time period = 2020 Q1.

4.3.2 Proportional Sampling Strategy

Each stratification label represents a single group. So, if 100 commits have the same strat_label, that would mean they belong to the same group.

1. Group proportions were calculated to determine a suitable number of samples per group for reaching a total of 20,000 commit messages. This allocation method prevented underrepresentation without compromising the diversity and temporal structure of project sizes.

2. Precision adjustments were made to the sample distribution after accounting for errors in order to reach the intended dataset size. The biggest group experienced adjustments which protected proportional distribution and prevented sampling issues.

3. A random sampling process for commit messages followed the allocations determined for each stratification group. The selected samples were aggregated to form a representative balanced commit message dataset that maintained the characteristics found in the original full dataset.

4.3.3 Bias & Limitation

Even though the sampling strategy preserves key distributions across projects and time periods, potential biases have to be acknowledged. By focusing exclusively on 34 highly popular repositories, the dataset may underrepresent smaller or hobbyist projects, whose commit practices and emotional tone can differ substantially. Similarly, filtering down to 20,000 commits, though statistically representative, may smooth out rare edge cases of developer sentiment. These limitations mean that results should be interpreted as reflective of professional large-scale projects, rather than the broader ecosystem of all open-source contributions.

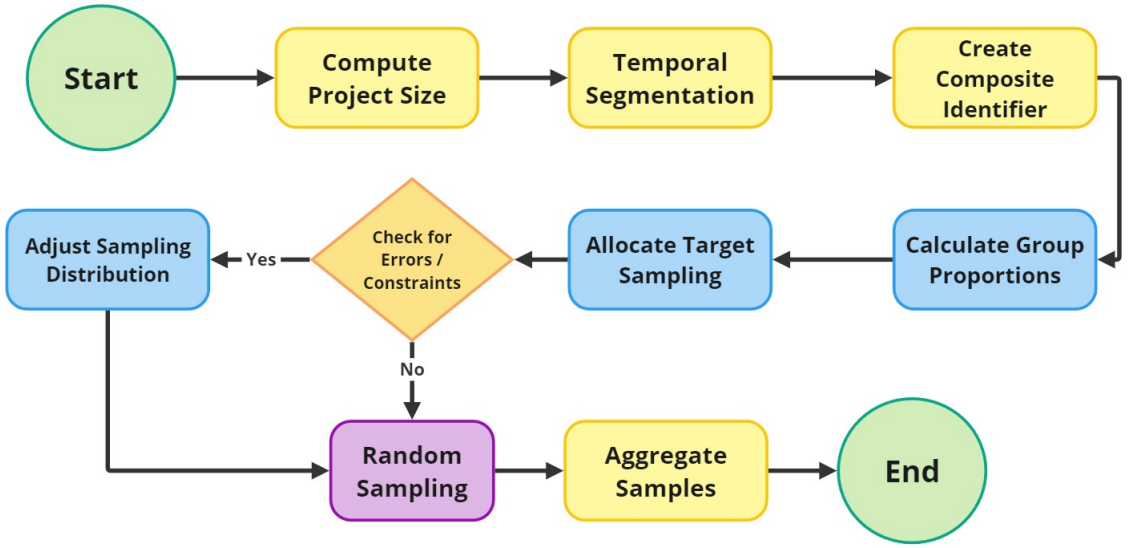


Figure 4.1: Flowchart of Sampling

Chapter 5

Data Preprocessing

5.1 Objectives of Preprocessing

Preprocessing is the task of cleaning up raw commit messages and preparing them for machine learning and analysis. The data contained in the commit message is often unprocessed or inconsistent, noisy, and redundant which blocks effective data utilization. This issue is addressed through preprocessing, where formats are standardized, irrelevant content is removed, special characters are handled, duplicates are eliminated, and any non meaningful parts of the commits are pruned out. And this process makes sure that we keep only high quality commit messages thus improving the accuracy and efficiency of the downstream machine learning tasks like text classification, sentiment analysis, and trend detection.

5.2 Preprocessing Steps

5.2.1 Load and Select Relevant Columns

The dataset was imported, keeping only the critical columns: author, date, repository, and message. The elimination of extraneous columns from the original dataset improved processing efficiency while maintaining data relevance to research targets. The dataset required refinement through selective retention of essential features including author information, timestamp and repository alongside commit messages. This refined the dataset into a format suitable for the next analysis steps.

5.2.2 Normalize Text

Normalization procedures were applied to commit messages for uniformity. We converted all commit messages to lowercase, in order to ignore case-sensitive variations. The beginning and end of each message were stripped of whitespace, and multiple spaces or newlines in the text were replaced with a single space. These steps made sure the text data was standardized without additional formatting problems and ready for the next step such as tokenization.

5.2.3 Remove Irrelevant Content

Redundant text from the commit messages was removed from the dataset to improve its quality. Messages of pull requests, as well as of automated issue labels (e.g. (#12345)) and auto-generated texts such as “merge branch” lacked significant sentiment information. Also commit messages that consist of fewer than three words or are contentless were excluded. Sentiment analysis was performed on the dataset, after transforming it to only contain pertinent, human-authored commit messages.

5.2.4 Remove Specific Patterns

The dataset was cleaned further to remove unwanted patterns and content unrelated to sentiment analysis. Removing of URLs, email addresses, UUID-like IDs, metadata fields such as reviewed_on and commit_queue: their presence added noise but didn’t bring much in terms of value added. To keep the emphasis on that primary message, the expressions like “bug = 123” were reduced to “bug”. To not clutter things, non essential special characters like ”@”, ”#” and ”!” were erased, redundant punctuation or letters were removed to keep things clear. This made the dataset better as only the content important to sentiment analysis was preserved.

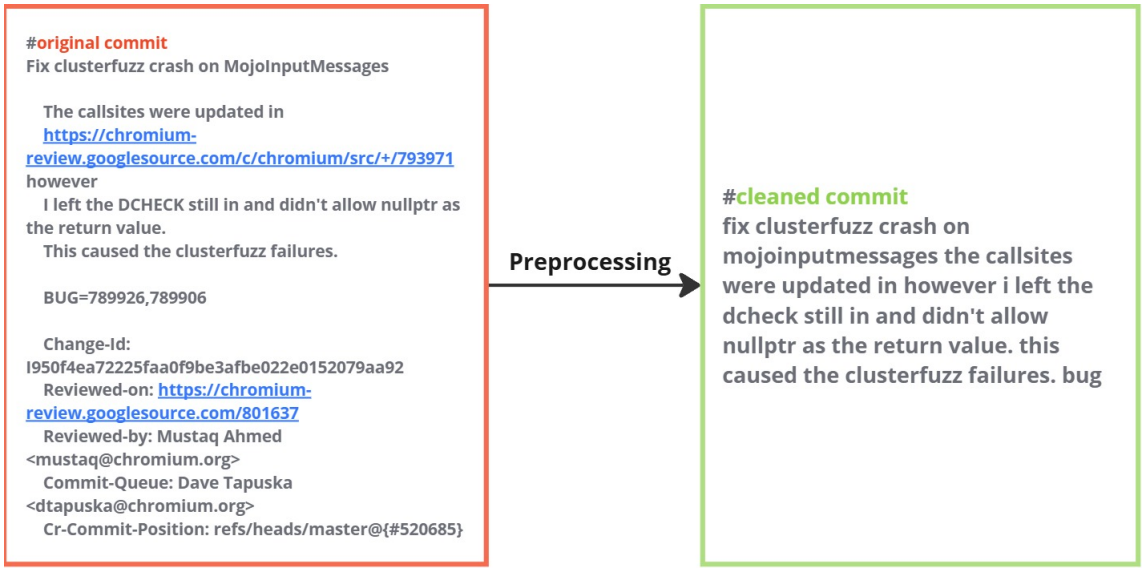


Figure 5.1: Commits before and after preprocessing

5.2.5 Tokenization and Length Filtering

To make commit messages suitable for Large Language models, the commit messages were decomposed into discrete tokens. The messages were segmented into a format compatible with model processing using the DistilBERT tokenizer. To work with the model’s input limitations, a length filter that retained only those messages that had 512 tokens or less was implemented. To maintain the brevity and manageability of the text messages beyond this threshold were eliminated.This procedure preserved the contextual integrity of the data, which meant that not a lot of information was lost that might sufficiently affect the accuracy of the model’s output.

5.2.6 Final Cleaning

At this stage, gaps from previous processing stages were eradicated to ensure consistent formatting. This cleanup was necessary to fix any discrepancies that could have come up in previous phases.

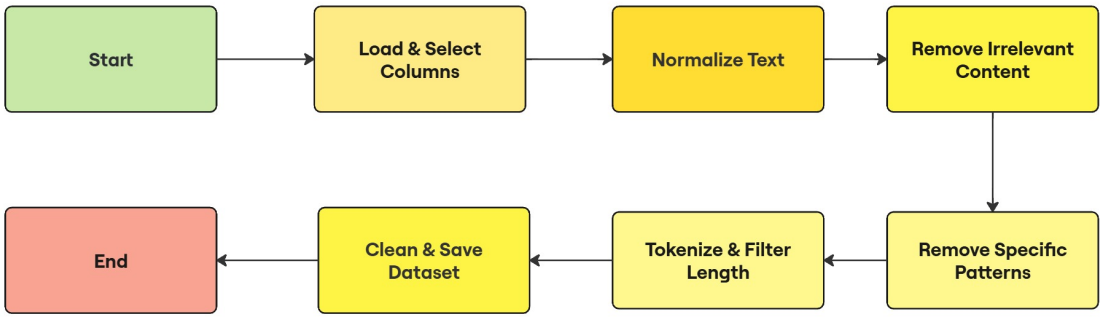


Figure 5.2: Flowchart of preprocessing

5.3 Iterative Refinement of the Emotion Model

Initially, a seven-emotion scheme (Joy, Excitement, Satisfaction, Frustration, Anger, Sadness, Neutral) was used for annotation, which is common in the literature. However, the first pass by two independent annotators yielded a Cohen's κ of 0.596, indicating poor agreement for the formal, technical commit text. Qualitative error analysis revealed recurring confusions, particularly between Joy and Excitement, and Anger and Frustration.
In software development, signals of risk (warnings, prevention, fallbacks) are often more actionable than raw affect. This led to the explicit introduction and precedence of a new category, **Caution**. Accordingly, the label space was consolidated into categories that the data could reliably support:

- **Satisfaction:** This category combines joy, excitement, and gratification expressed by developers. It represents feelings of accomplishment, positive outcomes, or enthusiasm.
  Example commit messages: "Implemented new feature without errors," "Breakthrough in performance optimization," "Improved search tags and reduced risk of bugs."

- **Frustration:** This category merges anger, sadness, and frustration into a single negative-emotion class. It reflects irritation, disappointment, or annoyance when facing challenges or failed attempts.
  Example commit messages: "Fix silly bug in login flow," "Revert failing changes," "Remove the horrible line of code."

- **Caution:** This newly introduced category represents defensive or risk-oriented signals in commit messages. It captures warnings, preventive actions, or fallback solutions identified via keyword patterns.
  Example commit messages: "Temporary workaround until API limitation resolved," "Add warning for potential null values," "Introduce fallback mechanism for timeout errors." The full lexicon to caution is provided in Appendix E

- **Neutral:** Commit messages with no clear emotional or risk-related content fall into this category. They typically describe straightforward updates or maintenance tasks without affective tone.
  Example commit messages: "Update README.md," "Fix typo," "Refactor code for clarity."

Inter-Annotator Agreement	
Metric	**Value**
Cohen's κ (7 labels)	0.596
Cohen's κ (4 labels)	0.710

Table 5.1: Inter-annotator agreement before and after consolidation of emotion labels.

Emotion Intensity. In addition to assigning categorical emotion labels, annotators were also asked to score the intensity of emotions on a 0–7 scale. Following prior work, these scores were divided into four conceptual levels: Minimal (0–1), Low (2–3), Medium (4–5), and High (6–7). Intensity was intended to capture the strength of emotional expression in a commit message, based on cues such as emotional words, repetition, and strong adjectives.

In practice, however, the assigned intensity scores showed considerable variance across annotators, suggesting low inter-annotator reliability and limited reproducibility. For this reason, intensity was not formally evaluated for agreement metrics (e.g., Cohen's κ, Krippendorff's α). Instead, intensity information was used in a supplementary role, serving as contextual evidence to help resolve disagreements about the primary four emotion categories during reconciliation.

Annotation Challenges. Several challenges arose during annotation that are typical when applying fine-grained emotion schemes to technical text. Messages sometimes conveyed multiple emotions simultaneously, requiring annotators to identify the dominant one in context. In ambiguous cases, where the emotional content was unclear, resolution was attempted collaboratively; if no agreement was possible, the message was classified as *Neutral*. Sarcasm and humor also posed difficulties, as these required interpretation beyond literal phrasing.
To improve reliability, annotation was conducted in a two-pass process: an initial reading with a tentative label and intensity score, followed by a more detailed review to refine both. Nevertheless, agreement under the original seven-label scheme remained modest (Cohen's $\kappa = 0.596$), with frequent confusion between closely related categories such as *Joy* and *Excitement* or *Anger* and *Frustration*. Intensity scores also exhibited considerable variance across annotators, limiting their reproducibility. These challenges motivated the consolidation of labels into four categories—*Satisfaction*, *Frustration*, *Caution*, and *Neutral*—supported by a structured reconciliation procedure. This approach emphasized interpretability, reduced category overlap, and enabled the construction of a high-quality gold dataset of 2,000 commit messages.

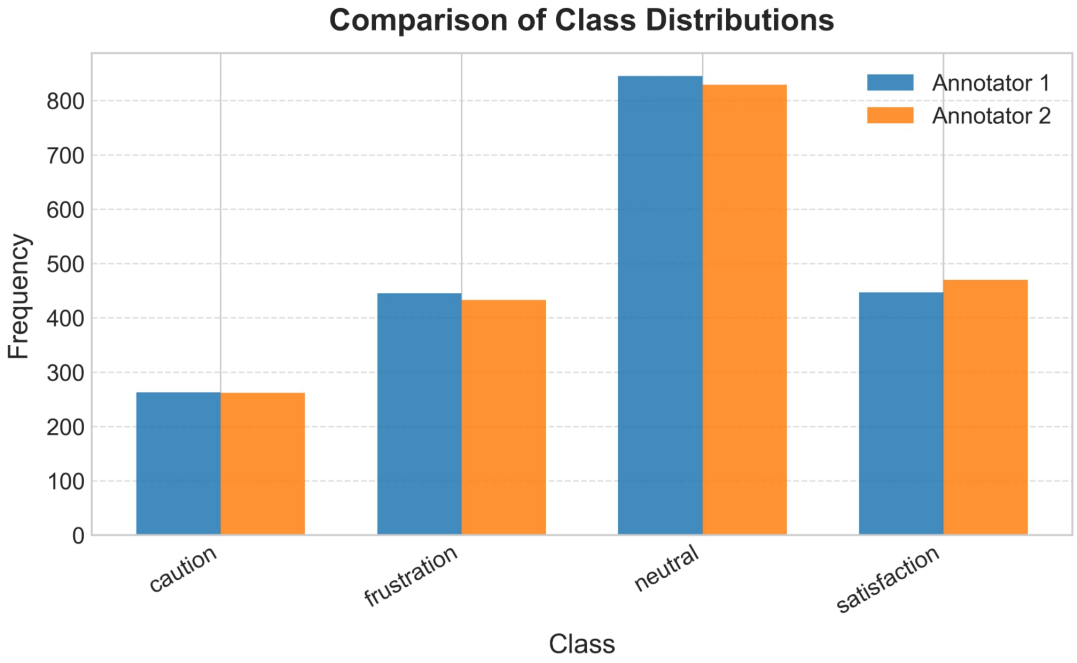


Figure 5.3: Emotion class distribution for Annotator-1 and Annotator-2 labelling

5.4 Label Consolidation and Reconciliation

After mapping the initial seven categories to the four consolidated labels, a single gold label for each commit was derived through a transparent, rule-based reconciliation procedure. Precedence was first given to *Caution*: if either annotator selected this label, the final assignment was also *Caution*. When both annotators agreed on *Caution*, the intensity was computed as the rounded mean (nearest integer, 0–7); otherwise, the *Caution* intensity was carried forward.
In cases of agreement on *Satisfaction*, *Frustration*, or *Neutral*, the shared label was retained, with intensity calculated as the rounded mean. When one annotator selected *Neutral* and the other an emotional category, the emotional label was adopted and its intensity preserved, since *Neutral* intensity was fixed at 0. Finally, in cases of direct conflict between *Satisfaction* and *Frustration*, the label associated with the higher intensity was chosen, and ties were flagged for manual review.
Applying this procedure resulted in a reconciled gold dataset of 2,000 commits.

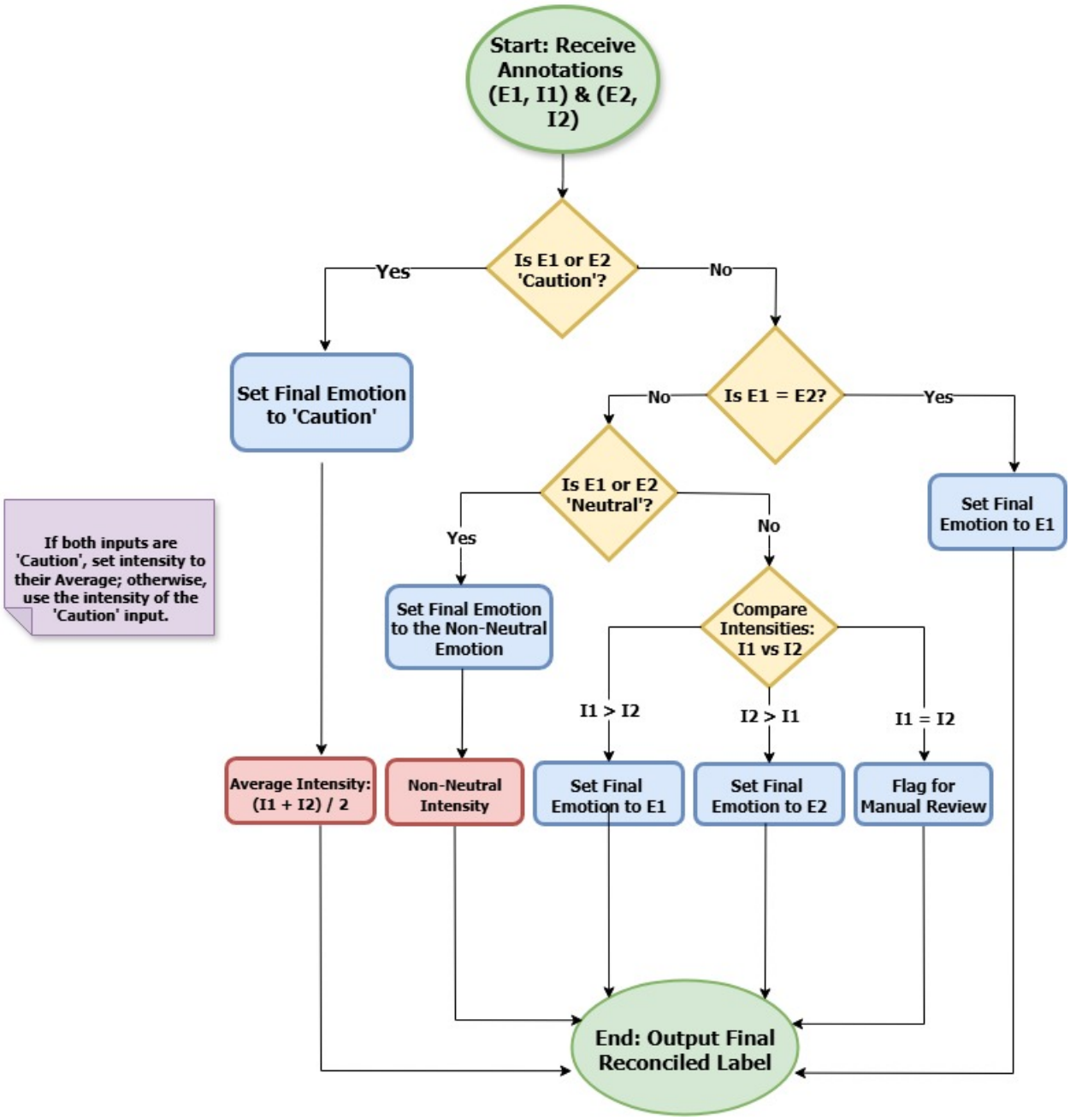


Figure 5.4: Workflow for Reconciling Annotated Data

Chapter 6

Model and Experimental Setup

6.1 Model Selection Criteria

The experimental design required a balanced roster of models spanning different paradigms: general-purpose language models, code-aware encoders, task-specific emotion classifiers, and generative annotators. Models were included if they satisfied two primary conditions: (i) compatibility with the four-label classification scheme of this study (*Satisfaction, Frustration, Caution, Neutral*), and (ii) availability of reproducible implementations. Models lacking a Neutral category were excluded to prevent structural bias. Each selected model contributes complementary perspectives: code-awareness, language generalization, emotion specialization, or generative reasoning.

Table 6.1: Experimental Model Roster

Model	**Type**	**Training Domain**	**Label Space**	**Role in Study**
RoBERTa-large-MNLI	Transformer (NLI)	Multi-domain natural text	Entailment scores	General-purpose zero-shot baseline
CodeBERT	Transformer (code+NL)	Source code + natural language	Embedding similarity / NLI	Code-aware zero-shot baseline
DistilRoBERTa (emotion)	Transformer classifier	GoEmotions (short text)	Emotion labels (GoEmotions)	Compact, efficiency-oriented baseline
RoBERTa-base GoEmotions	Transformer classifier	Social media text (GoEmotions)	Fine-grained emotions	Social-media-tuned emotion detector
EmoLLaMA-7B	Instruction-tuned LLM	Multi-domain text	Free-text emotion labels	Generative probe

Exclusion note: distilbert-base-uncased-emotion was excluded because it lacks a Neutral category, rendering it incompatible with the four-label taxonomy.

6.2 Label Mapping and Caution Lexicon

Each model outputs labels in its own taxonomy. To enable comparison, all outputs were mapped to the four ground-truth categories: *Satisfaction, Frustration, Caution, Neutral*.

Table 6.2: Mapping from Model-Native Outputs to Four Categories

Native Label Examples	**Target Category**
joy, love, optimism, admiration, excitement	Satisfaction
anger, sadness, disgust, annoyance, disappointment	Frustration
risk, warning, fallback	Caution
neutral (model-native)	Neutral

Generative models such as EmoLLaMA produce free-text labels. These were normalized (lowercased, stripped punctuation) and matched against the same mapping table.

6.3 Experimental Protocol

The experimental protocol was designed to provide a transparent and reproducible framework for evaluating emotion detection in commit messages. It establishes a clear separation between zero-shot baselines, which serve as diagnostic reference points, and fine-tuned models, which incorporate domain adaptation and augmentation. By standardizing inference recipes, evaluation metrics, and statistical checks across all models, we ensure that any observed improvements can be attributed to the modelling choices themselves rather than experimental artifacts.

6.3.1 Dataset and Splits

All experiments were conducted on the reconciled four-label gold-standard dataset introduced in section 5.4, comprising *Satisfaction*, *Frustration*, *Caution*, and *Neutral*. This dataset was drawn from a stratified sample of 20,000 commits across 34 high-profile GitHub repositories, ensuring both temporal and project diversity. Within this collection, 2,000 manually annotated commits formed the gold-standard subset used for supervised training and evaluation.

Of these 2,000 commits, 80 percent (1,600) were reserved for model development and 20 percent (400) were held out as a final test set to ensure an unbiased measure of generalization. Within the 1,600 development subset, 80 percent (1,280) were used for training and 20 percent (320) for validation. This partitioning maintained consistency across all fine-tuning experiments and ensured that the same 400 commits were never exposed during model selection or optimization.

For data-augmentation experiments, the 1,600 training–validation samples were expanded to 4,800 through generative paraphrasing (6.6). *CodeBERT* was fine-tuned on this augmented corpus using an 80/20 split for training and internal testing, with 20 percent of the training portion further used for validation. Final results were always reported on the original 400-commit held-out test set to preserve comparability and avoid data leakage.

6.3.2 Evaluation Metrics and Reproducibility

All experiments were conducted under a unified evaluation protocol to ensure comparability across zero-shot and fine-tuned models. To address class imbalance, Macro-F1 was used as the primary metric, giving equal weight to minority emotions such as *Satisfaction* and *Caution*. Complementary metrics like per-class precision, recall, and F1 were reported to capture disparities in model sensitivity and specificity.

In addition to Macro-F1, we employed an *Intensity-Weighted F1* score to provide a more nuanced evaluation of model performance. This metric accounts for the varying "emotional strength" or prominence assigned to each label during the manual annotation process (as detailed in Section 5.4).

During annotation, each commit message was not only assigned a primary emotion label (*Caution*, *Neutral*, *Satisfaction*, *Frustration*) but also an associated intensity score. This intensity score reflects the annotator's judgment of how strongly that particular emotion was expressed in the commit message.

The *Intensity-Weighted F1* metric incorporates these individual intensity scores by adjusting the contribution of each class's F1 score to the overall weighted average.

Specifically, classes with higher average emotional intensity in the gold-standard dataset contribute more significantly to the final *Intensity-Weighted F1* score.
Confusion matrices were analyzed to reveal systematic misclassifications (e.g., Neutral misread as Frustration, or rollback-related commits misinterpreted as Caution).
To contextualize performance, random and majority-class baselines were computed as lower bounds for expected accuracy.
All experiments were run on identical hardware (Tesla P100 GPU) under controlled environments, with the random seed fixed at 42 for reproducibility. Each model was evaluated across three shuffled runs of the evaluation split to measure variability. Results are reported as mean performance with standard deviation, and statistical significance was tested using paired bootstrap confidence intervals and t-tests (details in Chapter 7).

6.4 Zero-shot Baselines

Zero-shot baselines serve as a reference point for how well models perform without any task-specific adaptation. For NLI-based models such as RoBERTa-large-MNLI and CodeBERT (NLI variant), a consistent hypothesis template was used to generate prediction:

```
Assistant: The emotion is: LABEL
```

where `LABEL` $\in$ {*Satisfaction, Frustration, Caution, Neutral*}. Entailment probabilities were normalized across all four hypotheses, and the label with the highest probability was selected.
For classifiers pre-trained on fine-grained emotion taxonomies (e.g., GoEmotions RoBERTa, DistilRoBERTa emotion models), native categories are collapsed deterministically into the four-label space via the mapping in 6.2. Generative inference (EmoLLaMA-7B) uses the constrained prompt:

```
Human: Task: Categorize the developer's emotion in this
git commit message. Choose ONLY from: Satisfaction,
Frustration, Caution, Neutral. Message: port 'missing
optional unwrap' diagnostic to new abstraction
```

Outputs were parsed and mapped to the four-label space (see C.1). Any displayed confidence values were wrapper artifacts and were not treated as probabilities. Evaluation followed the unified protocol described in 6.3.2.

6.5 Fine-Tuning Pipeline

While zero-shot baselines establish a reference point, the central methodological contribution of this research lies in adapting large language models to the domain of software engineering. Fine-tuning was conducted on the reconciled four-label dataset described in 5.4, using domain-specific augmentation to address data scarcity and imbalance. This pipeline was deliberately structured to ensure transparency, reproducibility, and fairness across models, such that improvements could be attributed to domain adaptation rather than extraneous factors.

6.5.1 Training and Hyperparameters

All fine-tuning runs used the 1,600-example training split, with hyperparameters tuned against the 400-example validation split. Early stopping was employed with a patience of three epochs, monitoring Macro-F1 on the validation set to avoid overfitting.

Table 6.3: Fine-tuning configuration used across all models

Parameter	Value	Description
Learning rate	2e-5	AdamW optimizer with linear decay
Batch size	8	Mini-batch size during training
Epochs	Up to 10	With early stopping (patience = 3)
Gradient accumulation	Enabled	To stabilize small-batch training
Precision	Mixed (FP16)	To reduce memory usage
GPU	Tesla P100	Standardized across all experiments

Dynamic padding was implemented using Hugging Face's DataCollatorWithPadding, which automatically pads sequences to the longest input within each batch rather than to a global maximum length. These parameters were standardized across all candidate models (CodeBERT, RoBERTa, DistilRoBERTa) to preserve comparability.

6.5.2 Model Integration

During fine-tuning, all outputs were mapped to the four-label taxonomy introduced earlier (see 6.2). This ensured that evaluation remained aligned with the zero-shot baselines. Augmented datasets (see 6.6) were integrated only into the training split, leaving validation and test data untouched to preserve the integrity of comparisons. Together, this fine-tuning pipeline provides a robust methodological backbone for examining the benefits of domain-specific adaptation and augmentation over zero-shot inference.

6.5.3 Fine-Tuning LLaMA 3.1

Unlike the transformer-based discriminative models that were fine-tuned through full-parameter updates, the generative LLaMA 3.1 8B model was adapted using Parameter-Efficient Fine-Tuning (PEFT) via Quantized Low-Rank Adaptation (QLoRA). This technique enabled lightweight fine-tuning on limited GPU resources while retaining the model's broad linguistic competence.

QLoRA compresses model weights into 4-bit precision and injects small trainable adapter layers within the attention blocks. During fine-tuning, only these adapter parameters were updated, significantly reducing memory consumption and computational cost without altering the frozen base model. The fine-tuning procedure followed the same experimental protocol outlined in 6.5.1, ensuring consistency in data usage, optimization settings, and evaluation standards.

This efficient configuration allowed the LLaMA model to participate in domain adaptation under realistic hardware constraints, demonstrating that generative large language models can be specialized for software-engineering tasks even with restricted computational budgets.

6.6 Data Augmentation

A critical challenge in commit message emotion detection is the scarcity of labeled data, compounded by the heavy skew toward Neutral messages. In our gold-standard dataset, less than 15% of commits express clear affective states, making it difficult for models to generalize. To mitigate this, we introduced a targeted data augmentation strategy leveraging the generative model: LLaMA 3.1 8B.

Following the establishment of a fine-tuned CodeBERT baseline (Macro-F1: 0.59), a targeted data augmentation strategy was introduced to address residual weaknesses, particularly the underperformance on the Satisfaction class (See Section 7.X). A generative augmentation approach was employed using a previously fine-tuned LLaMA 3.1 8B model (see 6.5.3). For each of the 1,600 training samples, two synthetic paraphrases were generated while preserving the original emotional label. This expansion increased the size of the training set from 1,600 to approximately 4,800 samples, substantially enriching the diversity of emotional expressions available during training. The augmentation process was applied only to the training split. The 400-sample validation set remained untouched to ensure unbiased evaluation, and the final test set was strictly held out for reporting performance. After augmentation, CodeBERT was retrained from scratch on the expanded dataset, producing a direct comparison against the non-augmented baseline.

This augmentation strategy balanced methodological simplicity with empirical effectiveness: by leveraging a strong generative model for paraphrasing, it reduced overfitting to the original limited phrasing of commits while preserving the semantic integrity of labels. As later sections demonstrate, this approach led to measurable gains in classification performance, particularly in recognizing subtle expressions of Satisfaction that had previously eluded the baseline model.

Chapter 7

Experiments Results

7.1 Zero-Shot Results

Building on the unified evaluation protocol described in chapter 6.6, we first assessed the zero-shot performance of several off-the-shelf pretrained models on the 2,000-sample manually labeled dataset. Performance was measured using *Macro-F1*, which treats all classes equally regardless of frequency, and an *Intensity-Weighted F1*, which adjusts scores based on the emotional strength of each label as detailed in Section 6.3.2.
The results were notably poor across all models, with the highest Macro-F1 achieved by RoBERTa-NLI (0.2058), a performance only marginally above random chance. Other models including CodeBERT (0.1968), DistilRoBERTa (0.1886), GoEmotions (0.1374) and EmoLLaMA (0.1296) all exhibited similarly weak results. These findings confirm that generic pretrained models, whether optimized for natural language inference (RoBERTa-NLI), programming language semantics (CodeBERT) or general emotional text (GoEmotions), fail to capture the nuanced emotional signals present in developer communication.

Table 7.1: Zero-Shot Model Baselines (Macro-F1 and Intensity-Weighted F1)

Model	**Macro-F1**	**Intensity-Weighted F1**	**Primary Bias**
RoBERTa-NLI	0.2058	0.2148	Caution
CodeBERT	0.1968	0.1910	Satisfaction
DistilRoBERTa	0.1886	0.1922	Neutral
GoEmotions	0.1374	0.1406	Neutral
EmoLLaMA	0.1296	0.1321	Neutral

The error analysis revealed that the mistakes made by the zero-shot models were not random but systematic. RoBERTa-NLI exhibited a strong Caution Bias. In technical contexts, it frequently predicted Caution even when the message expressed ordinary satisfaction or mild frustration. In contrast, CodeBERT displayed a consistent Satisfaction Bias, often labeling neutral or warning-related commits as positive in tone. As illustrated in Table 7.2, these recurring misclassifications highlight the fundamental limitations of both general-purpose language models and domain-specific code models when applied in a zero-shot setting.
Overall, the zero-shot baselines serve as a critical reference point. Even the best-performing generalist model, RoBERTa-NLI, achieved only a Macro-F1 of about

0.21, which is barely above random chance. This reinforces the central claim of this study: that true understanding of developer communication requires domain-specific fine-tuning to overcome these systematic biases and capture the distinctive emotional language of software development.

Table 7.2: Qualitative Error Analysis of Zero-Shot Models

Model	**Primary Bias**	**Example Misclassification**	**Analysis**
RoBERTa-NLI	Caution	Message: "query tiles add a debug switch to immediately expire query tiles" Manual Label: Neutral Prediction: Caution	The model misinterprets neutral technical terms like "debug" and "expire" as signals of risk, failing to recognize them as standard developer vocabulary.
CodeBERT	Satisfaction	Message: "remove backforwardcache specific ipc and use mojom" Manual Label: Neutral Prediction: Satisfaction	The model incorrectly associates the procedural act of refactoring and removing legacy code with a positive emotional expression, confusing a functional improvement with developer sentiment.

7.2 Fine-Tuning Results

Fine-tuning was performed using the modular pipeline described in Chapter 6.6, implemented with the Hugging Face `Trainer` API for seamless backbone swapping, checkpoint management, and automated validation monitoring. This setup ensured reproducibility and consistent evaluation across all models while minimizing configuration overhead.

7.2.1 Experiment A: RoBERTa-large-MNLI (Generalist)

RoBERTa was treated as a generalist baseline to measure how well a strong NLI-pretrained language model can adapt to developer emotion detection under direct supervision.

The learning curve in Figure 7.1 plots the epoch-level training and validation losses of RoBERTa. The most notable point is that, following a first learning step in epoch 3, validation loss grows exponentially as training loss keeps declining, which is a typical manifestation of overfitting. Optimal numeric Macro-F1 was at Epoch 6 (Macro-F1 $\approx$ 0.57), though even at that checkpoint validation loss was high, meaning the model had started to memorize the training patterns over that portion of the hold-out validation set.

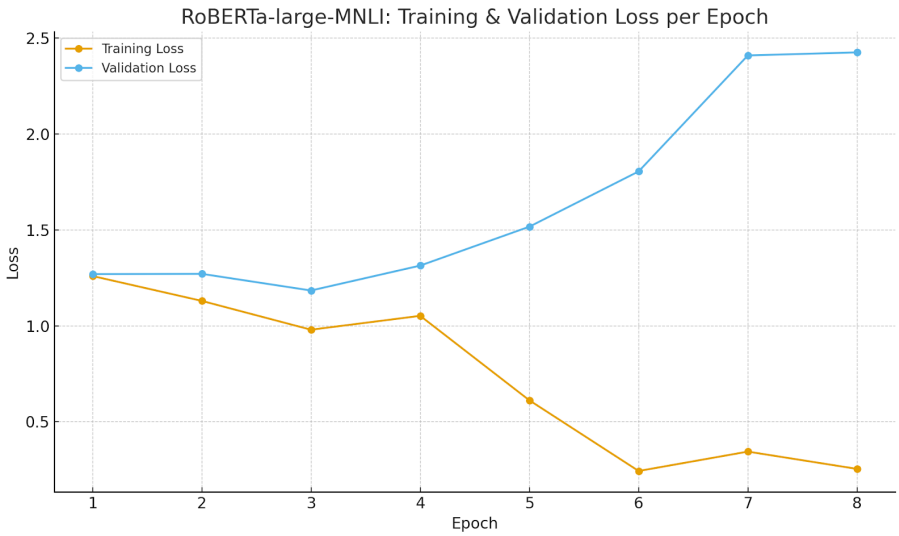


Figure 7.1: RoBERTa-large-MNLI — training and validation loss per epoch.

7.2.2 Experiment B: CodeBERT (Specialist)

CodeBERT was selected as a specialist backbone (pre-trained on code and developer text) to test whether domain inductive biases result in more robust emotion representations.

CodeBERT had the highest Macro-F1 at Epoch 3 (Macro-F1 = 0.5885) and was relatively steady in its learning behaviour during the first three epochs. The epoch curves for CodeBERT are plotted in Figure 7.2. Contrary to RoBERTa, the validation loss of CodeBERT does not grow significantly during the initial epochs but starts increasing after the peak. This suggests that its learning is more controlled by generalization on this task, in agreement with its domain-specialized pretraining.

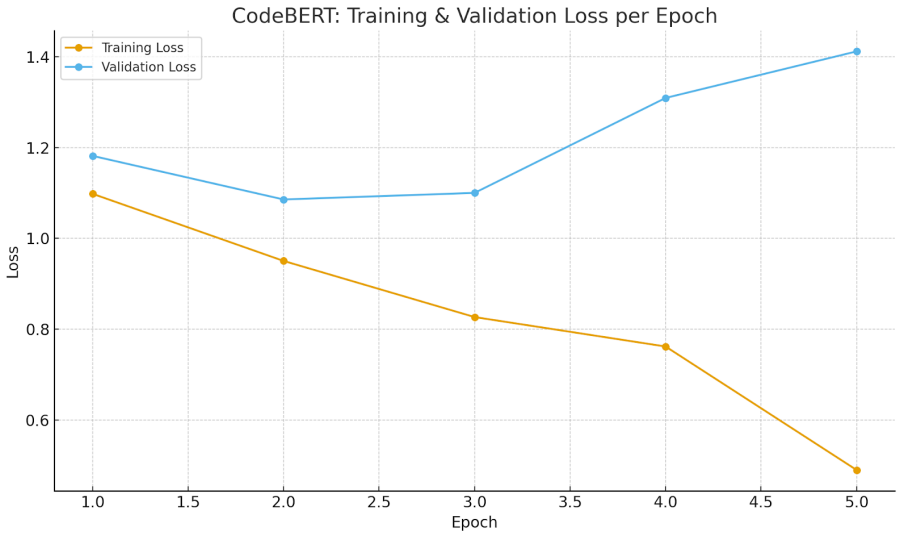


Figure 7.2: CodeBERT — training and validation loss per epoch.

This per-class breakdown indicates CodeBERT is strong on the `caution` label and its relatively weaker performance on `satisfaction`; matching the qualitative error analysis that `satisfaction` is a harder class to disambiguate from neutral/technical

text without more data. The early checkpoint behavior (peak at Epoch 3) is significant: the model learns useful features quickly and then it drifts a bit. This is because later epochs are more preoccupied with minimizing training loss than with generalisation.

Table 7.3: Full Classification Report for Fine-Tuned CodeBERT (Best Checkpoint: Epoch 3).

Class	Precision	Recall	F1-Score	Support
Caution	0.79	0.79	0.79	53
Satisfaction	0.47	0.44	0.46	107
Neutral	0.58	0.63	0.60	139
Frustration	0.52	0.50	0.51	101
Accuracy	—	—	0.56	400
Macro Avg	0.59	0.59	0.59	400
Weighted Avg	0.56	0.56	0.56	400

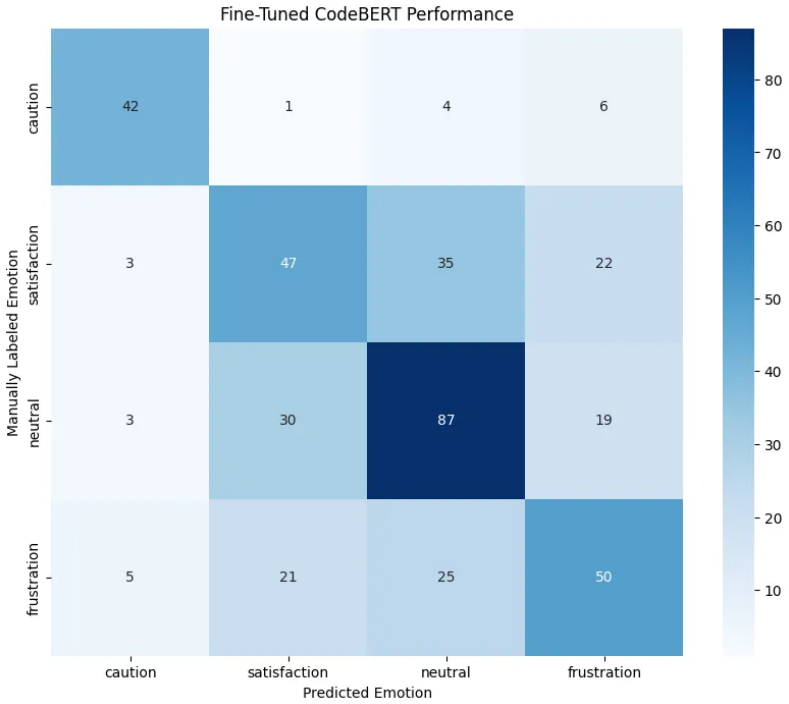


Figure 7.3: Confusion matrix for the fine-tuned CodeBERT model at its best checkpoint.

7.2.3 Experiment C: DistilRoBERTa (Pattern Mimic)

DistilRoBERTa was evaluated to understand whether a distilled model would capture semantic nuance or default to pattern matching.
DistilRoBERTa showed strong performance on the automatically generated `Caution` label (F1 = 0.87), but struggled with the manually annotated classes: `Satisfaction`, `Neutral`, and `Frustration`. This suggests that the model is highly

Table 7.4: Full Classification Report for Fine-Tuned DistilRoBERTa.

Class	Precision	Recall	F1-Score	Support
Caution	0.87	0.87	0.87	53
Satisfaction	0.44	0.49	0.46	107
Neutral	0.60	0.44	0.51	139
Frustration	0.48	0.60	0.54	101
Accuracy	—	—	0.55	400
Macro Avg	0.60	0.60	0.59	400
Weighted Avg	0.56	0.55	0.55	400

effective at detecting explicit lexical cues and surface-level patterns. However, it is less capable of capturing the subtle emotional nuances that require human-like contextual understanding. In essence, DistilRoBERTa performs well as a pattern recognizer but remains limited as a true semantic interpreter without further domain-specific fine-tuning.

7.2.4 LLaMA 3.1 8B (PEFT / QLoRA)

An advanced experiment using LLaMA 3.1 8B fine-tuned via PEFT/LoRA (QLoRA) on the 1600 sample training set to test whether a large generalist LLM fine tuned efficiently could match or exceed the domain specialist. The outcome was negative: Macro-F1: 0.31, Accuracy: 0.35 (35%). Caution recall: 0.11 (6/54 caution messages was correctly recognised).

Table 7.5: Full Classification Report for the LLaMA 3.1 8B PEFT Experiment.

Class	Precision	Recall	F1-Score	Support
Caution	0.24	0.11	0.15	54
Frustration	0.32	0.44	0.37	106
Neutral	0.38	0.38	0.38	134
Satisfaction	0.38	0.33	0.36	106
Accuracy	—	—	0.35	400
Macro Avg	0.33	0.32	0.31	400
Weighted Avg	0.34	0.35	0.34	400

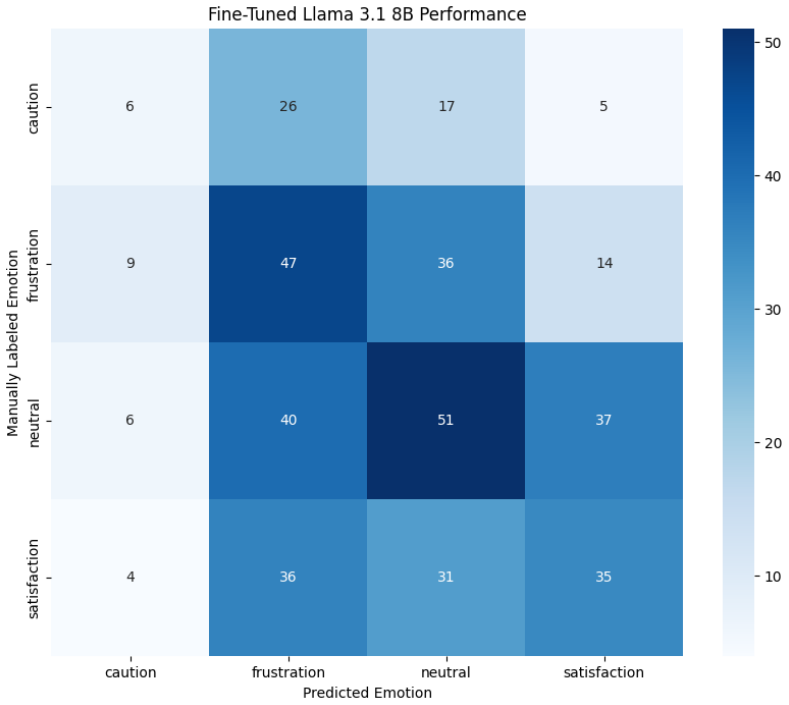


Figure 7.4: Confusion matrix for the fine-tuned LLaMA 3.1 8B model.

The PEFT/LoRA model using 4-bit quantization (QLoRA) failed to effectively adapt the general-purpose LLM to the narrow, domain-specific emotion labels. While its performance on most classes was moderate, it performed disastrously on `Caution`. This outcome highlights a key insight: parameter-efficient fine-tuning of large, generalist models does not automatically transfer domain-specific understanding, especially when the available in-domain training data is limited.

7.2.5 Comparative Summary and Practical Implications

Among all fine-tuned models, CodeBERT (Epoch 3) emerged as the most stable and performant, achieving a Macro-F1 of 0.59 with balanced behavior across all emotion classes. RoBERTa trained rapidly but showed signs of overfitting after early epochs. Whereas, DistilRoBERTa excelled at detecting explicit caution-related patterns but failed to generalize to subtler, human-judged emotions. The LLaMA PEFT model, despite its efficient parameter optimization, struggled to internalize domain-specific emotional cues, emphasizing that scaling or parameter efficiency alone cannot substitute for targeted domain adaptation.
Collectively, these results reinforce two central insights: first, that domain-aligned representations matter more than model scale, and second, that generalist LLMs require explicit domain supervision to handle the technical and emotional nuances of developer communication effectively.

7.3 CommiTune Results

This section reports the definitive outcomes of the experiment in which CodeBERT was fine-tuned on a synthetically augmented training set and evaluated on a com-

pletely held-out, manually-labelled test set. The experiment aimed to address the question of whether augmentation with generative outputs of the LLM can address the issue of the lack of data in the original dataset and yield robust and generalizable emotion classification results. This model was trained on an augmented corpus of about 4,800 samples with a held-out test partition of 400 original human-labelled samples. The typical fine-tuning steps were applied (no class-weighted loss) and an internal validation split was utilized in the early stopping and the selection of hyperparameters.
Retraining of CodeBERT on an augmented dataset and then testing on a clean test set led to substantial improvement in all metrics, creating a new state-of-the-art performance in this project:

- Macro F1: 0.88
- Accuracy: 86%

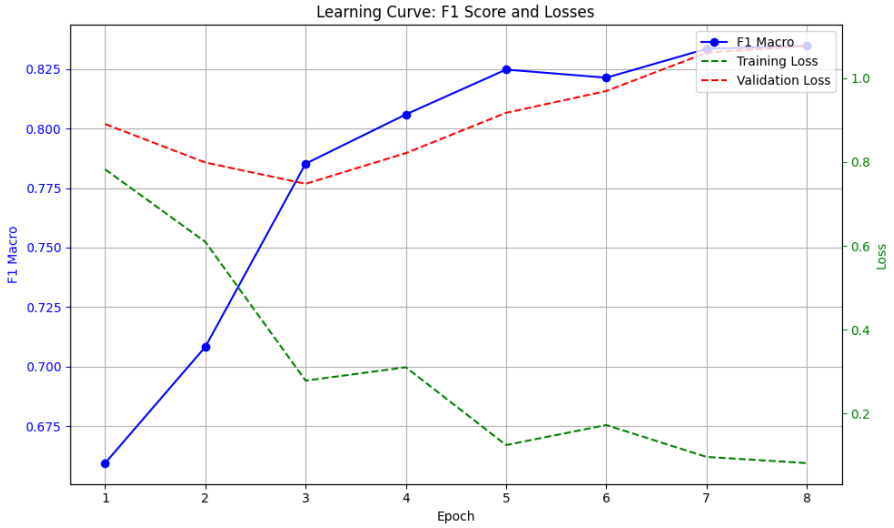


Figure 7.5: Learning curve for the CommiTune model, showing F1 Macro score and losses per epoch on the validation set.

Table 7.6: Final per-class breakdown for the CommiTune model on the held-out test set.

Emotion	**Precision**	**Recall**	**F1-Score**
caution	0.98	0.92	0.95
neutral	0.90	0.86	0.88
satisfaction	0.77	0.92	0.83
frustration	0.89	0.79	0.84

These results represent a substantial improvement relative to prior fine-tuning on the original, non-augmented dataset, most notably the recovery of the satisfaction class from a previously low F1-score (0.46) to 0.83.

Table 7.7: Performance comparison between Baseline and CommiTune (Augmented).

Metric	Class	Baseline	CommiTune	Improvement
F1-Score	Caution	0.87	0.95	+0.08
	Frustration	0.54	0.83	+0.29
	Neutral	0.51	0.86	+0.35
	Satisfaction	0.46	0.85	+0.39
Accuracy	Overall	0.55	0.86	+0.31
Macro F1	Overall	0.59	0.87	+0.28

As shown in Table 7.7, the per-class and overall performance metrics demonstrate that the model effectively learned discriminative representations for each emotion category. The exceptionally high F1-score for `caution` (0.95) and the strong performance on `neutral` (0.86) indicate that the augmented data helped the model capture both domain-specific lexical cues and broader pragmatic patterns. The improvement in `satisfaction` (+0.39) is particularly notable: recall in this class increased substantially, suggesting fewer false negatives and enhanced sensitivity to positive sentiment expressions that were previously under-represented. Overall, accuracy and macro F1 also rose significantly (+0.31 and +0.28), underscoring the robustness and generalization capability of the fine-tuned model.
Fine-tuning CodeBERT on a synthetically augmented dataset yielded a high-performing classifier with a Macro F1 of 0.88 and 86% accuracy on a 400-sample held-out test set. The confusion matrix supports the per-class metrics, revealing minimal off-diagonal mass and concentrated improvements where the original model had failed. While results are compelling and justify the augmentation approach, we explicitly limit claims to improved transfer within the evaluated domain and recommend further evaluation for strong generalization claims.

7.3.1 Caveats and limits to generalization

Even though the held-out evaluation contains original, manually-labeled samples and hence provides a more powerful test than an in-domain synthetic split, there are various restrictions to assertions of wide generalization:

- **Sample size:** The held-out set contains 400 examples; this is large enough to illustrate any significant improvements, but too small to claim the ability to generalize to different real-world distributions.
- **Labeler and domain bias:** The project team (or contracted annotators) manually labelled the test set. Any mutual annotation policies, cultural biases or domain coverage between training/augmentation pipelines and the held-out set may partially inflate estimated performance.
- **Distributional shift:** Augmented data can cover many realistic variants but may still fail to represent edge cases or domain shifts (e.g., different user populations, platforms, or temporal language drift).

- **Single-model evaluation:** Results are reported for one architecture (CodeBERT) and one augmentation strategy; broader claims would benefit from cross-model and cross-method replication.

7.4 Ablations: Weighted vs Standard Loss

Ablation evaluates the effect of applying a class-weighted loss during fine-tuning versus the standard (unweighted) cross-entropy objective used in the definitive experiment. The aim is to determine whether explicit cost sensitivity benefits minority classes without unduly harming performance on the majority classes.
Two configurations were compared while holding all other training settings constant (same augmented training set, validation scheme, early stopping, and hyperparameters):

- **Standard training (No weights):** vanilla cross-entropy loss.
- **Weighted loss training:** class-weighted cross-entropy that increases the loss contribution from the smallest class (caution) to counterbalance its relative scarcity.

The table below summarizes the held-out test performance for both settings:

Table 7.8: Comparison of Standard and Weighted Loss Training on the Held-Out Test Set.

Metric	Standard Training	Weighted Loss	Analysis
Macro F1	0.88	0.86	Standard is higher. The overall balanced performance is slightly better without weights.
Accuracy	0.86	0.85	Standard is higher. It gets more predictions correct overall.
F1: frustration	0.84	0.80	Standard is better.
F1: satisfaction	0.83	0.83	Tie. No significant difference.
F1: neutral	0.88	0.85	Standard is better.
F1: caution	0.95	0.96	Weighted is slightly better.

7.4.1 Interpretation and trade-offs

The ablation shows a clear trade-off: applying class weights yields a modest gain on `caution`, the smallest class (F1: 0.95 $\rightarrow$ 0.96), but at the cost of reduced performance on the rest, larger classes, with `frustration` and `neutral` being the most notable. Since Macro F1 is a means of performance across classes, the overall net

effect is that the weighted Macro F1 (0.86) is lower than the objective Macro F1 (0.88). Mechanically, increasing the loss weight for the minority class amplifies gradient signals for examples of that class, causing the optimizer to prioritize reductions in their classification error. In a finite-capacity model where the augmented dataset already offers coverage of previously under-represented patterns, such a redistribution of learning activity can actually increase the representation accuracy of other classes which are similar in lexical or pragmatic aspects. The outcome is that the sensitivity to the minority-class is better at the cost of the balanced performance of the entire label set.

The ablation study shows that naive class weighting is not a solution: although it can progressively enhance the ability to detect the smallest class (with caution), it may degrade overall balanced performance when applied atop an already-augmented training set. For this project's dataset and objectives, the standard (no-weight) training regime is the preferred choice because it attains the highest Macro F1 and more consistent performance across the core emotion categories.

Chapter 8

Project Health Dataset Construction

8.1 Bug-Related Activity Metrics

In addition to measuring productivity and code churn, project health can also be assessed through bug-related activity. This study uses two straightforward but informative indicators: the rate of potential bug-fixing commits (*pBFC_rate*) and the rate of potential revert commits (*pRevert_rate*). Both metrics are derived by analyzing the textual content of commit messages.
A commit is considered a potential bug-fixing commit if its message contains words such as *fix*, *bug*, *error*, *issue*, *patch*, *resolve*, or *correct*, including common variations of these terms. Similarly, a commit is regarded as a potential revert commit if the message includes words like *revert*, *rollback*, or *backed out*. These keyword-based heuristics allow automated identification of bug-related activity within the repository.
Once each commit has been labeled, the bug-related metrics are aggregated at the repository–quarter level to produce proportional measures of activity. For a given repository r and calendar quarter t, the bug-fixing and revert commit rates are defined as follows:

$$pBFC_{r,t} = \frac{\text{Number of bug-fixing commits in repository } r \text{ during quarter } t}{\text{Total number of commits in repository } r \text{ during quarter } t}$$

$$pRevert_{r,t} = \frac{\text{Number of revert commits in repository } r \text{ during quarter } t}{\text{Total number of commits in repository } r \text{ during quarter } t}$$

Here, the numerators count commits identified as bug-fixing or revert based on keyword heuristics, while the denominators normalize by the total number of commits in that repository–quarter. This proportional formulation ensures that trends reflect relative activity rather than absolute commit volume, reducing scale bias between high-traffic and low-traffic projects.
By combining textual heuristics with normalized metrics, these indicators provide a clear, comparable measure of bug-related activity and contribute to a comprehensive assessment of project health.

8.2 Bug-Inducing Commit Heuristic (pBIC)

Bug-inducing commits represent changes that are likely to introduce defects rather than resolve them, offering a complementary perspective to bug-fixing and revert metrics. In this study, we operationalize this concept through the *Potential Bug-Inducing Commit* (pBIC) heuristic, which identifies commits that may lead to subsequent bug fixes.

A commit is classified as a *Potential Bug-Inducing Commit (pBIC)* if it is not a bug-fixing or revert commit (as defined in Section 8.1) and its message contains explicit failure-related keywords such as *broke*, *crash*, or *fail*. This heuristic identifies commits that signal potential problems without being direct fixes, leveraging both linguistic cues and the emotional tone of the commit message. This definition allows us to isolate changes that are likely to introduce subtle defects, offering a complementary perspective to explicit bug-fixing and revert metrics.

The quarterly pBIC rate for a repository is calculated as the proportion of pBIC commits relative to the total number of commits in that quarter:

$$pBIC_{r,t} = \frac{\text{Number of pBIC commits in repository } r \text{ during quarter } t}{\text{Total number of commits in repository } r \text{ during quarter } t}$$

The final project health dataset integrates *pBIC* along with other metrics such as productivity, code churn, bug-fixing, revert rates, and emotion distributions. Each repository–quarter record thus captures developer activity, maintenance behavior, and potential risk factors, enabling a holistic analysis of how emotional states, code changes, and project practices interact to influence long-term project stability and quality.

8.3 Productivity Metrics

Productivity serves as a quantitative indicator of development activity within a given period and provides essential context for understanding the relationship between developer emotions and project health. This study employs three complementary indicators that collectively capture both the magnitude and distribution of contributions across time.

The first indicator, denoted as *total_commits*, represents the total number of commits made in a repository during a given calendar quarter. This metric reflects the overall volume of development work. The second indicator, *unique_authors*, measures the number of distinct contributors active within the same quarter, thereby capturing the breadth of participation. Finally, the third indicator, *avg_commits_per_author*, estimates the distribution of workload among contributors by dividing the total number of commits by the number of unique authors.

Formally, for a repository r and quarter t, the metric is defined as:

$$avg\_commits\_per\_author_{r,t} = \begin{cases} \dfrac{total\_commits_{r,t}}{unique\_authors_{r,t}} & \text{if } unique\_authors_{r,t} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Each commit is assigned to a calendar quarter based on its timestamp, after which aggregation is performed at the (*repository*, *quarter*) level. This approach enables a consistent temporal comparison of productivity across repositories of different sizes and activity levels.

Table 8.1: Summary of Productivity Metrics

Metric	**Definition**
total_commits	Total number of commits per quarter
unique_authors	Number of distinct contributors per quarter
avg_commits_per_author	Mean number of commits per author per quarter

8.4 Code Churn Metrics

While productivity metrics capture the overall volume of development activity, churn metrics quantify the magnitude of changes introduced to the codebase, offering valuable insights into project stability and volatility. Code churn is derived from commit-level statistics extracted using Git's `--numstat` option, which reports the number of files modified, as well as the lines added and deleted for each commit.
For each commit, three values are recorded: *files_changed*, *lines_added*, and *lines_deleted*. To enable cross-project comparison, these metrics are aggregated at the repository–quarter level by computing the average values for all commits within that period. This yields three primary measures: the average number of files changed, the average number of lines added, and the average number of lines deleted per quarter. Formally, for each repository r and quarter t, the churn metrics are defined as:

$$avg\_files\_changed_{r,t} = \frac{\sum_{i=1}^{C} files\_changed_i}{C}$$
$$avg\_lines\_added_{r,t} = \frac{\sum_{i=1}^{C} lines\_added_i}{C}$$
$$avg\_lines\_deleted_{r,t} = \frac{\sum_{i=1}^{C} lines\_deleted_i}{C}$$

where C denotes the number of commits in the repository during quarter t.
The extraction process begins by retrieving commit history using the command `git log --numstat`. The resulting output is parsed to identify commit-level blocks, from which file-level changes are computed. Each commit is then assigned to its corresponding calendar quarter, and its metrics are recorded. To handle large repositories and long histories efficiently, the extraction pipeline is designed to be crash-safe and resumable: partial results are stored incrementally, allowing interrupted runs to resume without data loss. The scripts used for extracting these churn metrics will be made publicly available to support reproducibility and facilitate future research.
It should be noted, however, that not all repositories could be processed due to computational scaling limitations. The final dataset includes churn data for 84 quarters, representing approximately 30% of the total sample. Consequently, large repositories with long development histories are underrepresented, introducing a potential

sampling bias. Therefore, any analysis of churn metrics should be interpreted in light of this limitation.

Chapter 9

Project Health Churn Correlation Analysis

9.1 Methodology: Data Aggregation and Metric Definition

To prepare the data for statistical analysis, raw, emotion-labeled commits were first transformed into quantitative project health metrics through a rule-based heuristic approach, followed by temporal aggregation.
Three key health metrics were defined using keyword patterns within commit messages. A commit was classified as a Probably Bug-Fixing Commit (pBFC) if its message contained terms associated with bug fixing, such as "fix," "bug," "issue," or "patch." A commit was flagged as a Probably Revert Commit (pRevert) if its message included terms indicating a rollback of changes, like "revert" or "rollback." Finally, the Probably Bug-Inducing Commit (pBIC) metric was designed to identify potentially problematic commits that are not explicit bug fixes. A commit was flagged as a pBIC if it was not a pBFC and its message contained either a negative emotion (frustration or caution) or explicit failure-related keywords like "broke," "crash," or "fail." This definition isolates commits that signal problems without being direct fixes.
Following the assignment of these boolean flags to all 20,000 sampled commits, the data was aggregated into a structured time-series dataset suitable for analysis. Commits were grouped by their source repository and the calendar quarter in which they occurred. Within each repository-quarter group, rate-based metrics (e.g., pBFC_rate) were calculated by computing the mean of each boolean flag, effectively representing the proportion of each commit type within that period.

9.2 Analysis Framework

The investigation proceeded through four distinct analytical stages, each designed to answer a specific research question regarding the interplay between emotion ratios and the derived health and churn metrics.
Contemporaneous Correlation: A Pearson correlation matrix was computed on the quarterly data to identify immediate, same-period relationships.
Lagged Correlation: To test for short-term predictive power, correlations were

calculated between emotion ratios in a given quarter and the health metrics of the subsequent quarter.
Granger Causality: A more rigorous time-series analysis was performed to determine if the historical time-series of an emotion ratio could statistically predict future values of the health metric time-series.
Repository-Level Correlation: To assess long-term structural relationships, all quarterly data for each repository were aggregated into a single set of mean values for a final correlation analysis. While many of the contemporaneous and lagged correlations appear weak, they still offer practical significance by highlighting subtle trends in developer behavior and project dynamics that can inform early warning signals, risk assessment, and management strategies over time.

9.3 Results: Emotion as an Indicator of Project Health

9.3.1 Short-Term Analysis: A Null Finding

The time-series analyses consistently revealed that developer emotions are not effective as short-term operational indicators or predictors of project health. The contemporaneous correlation analysis yielded only weak relationships. For instance, the strongest positive correlation was between `frustration_ratio` and `pBFC_rate` ($r = 0.19, p < 0.001$), while the strongest negative correlation was between `neutral_ratio` and `pBFC_rate` ($r = -0.26, p < 0.001$). The lagged correlation analysis, designed to test predictive power, found no meaningful signals, as all correlation coefficients were negligible (ranging between -0.07 and 0.07) and not statistically significant.
These findings were further substantiated by the Granger causality tests, which provided stronger statistical evidence for the lack of predictive power. For all combinations of emotion ratios predicting `pBFC_rate` and `pBIC_rate`, the resulting p-values (ranging from 0.15 to 0.93) were substantially higher than the 0.05 significance threshold, with corresponding F-statistics consistently low. The combined results, visualized in the correlation matrix in Figure 9.1, lead to the clear conclusion that developer emotion metrics are not reliable for forecasting short-term changes in project health.

9.3.2 Long-Term Analysis: A Strategic Indicator of Culture and Risk

In stark contrast, the aggregated repository-level analysis revealed moderate and significant correlations, suggesting that the long-term, prevailing emotional tone of a project is a meaningful indicator of its overarching engineering culture and health profile. The analysis, detailed in the matrix in Figure 9.2 and summarized in Table 9.1, uncovered a distinct split in how different emotions correlate with different types of bugs.
A moderate positive correlation was found between a repository's average `frustration_ratio` and its `pBFC_rate` ($r = 0.39, p = 0.023$), as well as between its `caution_ratio` and `pBFC_rate` ($r = 0.37, p = 0.030$). This suggests that

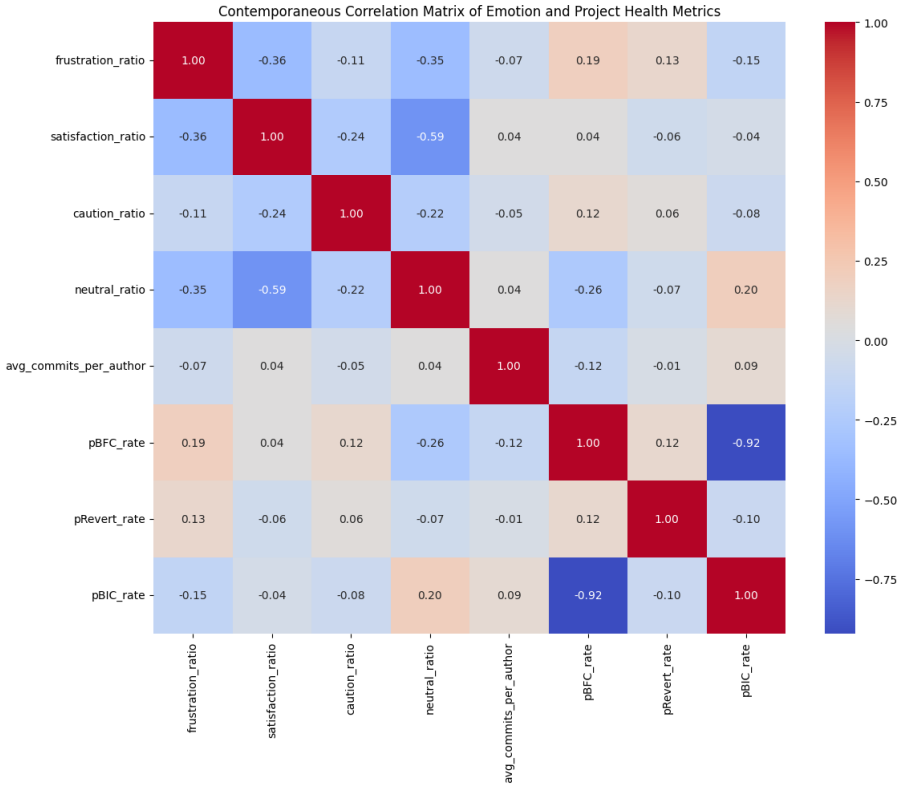


Figure 9.1: Contemporaneous Correlation Matrix of Emotion and Project Health Metrics.

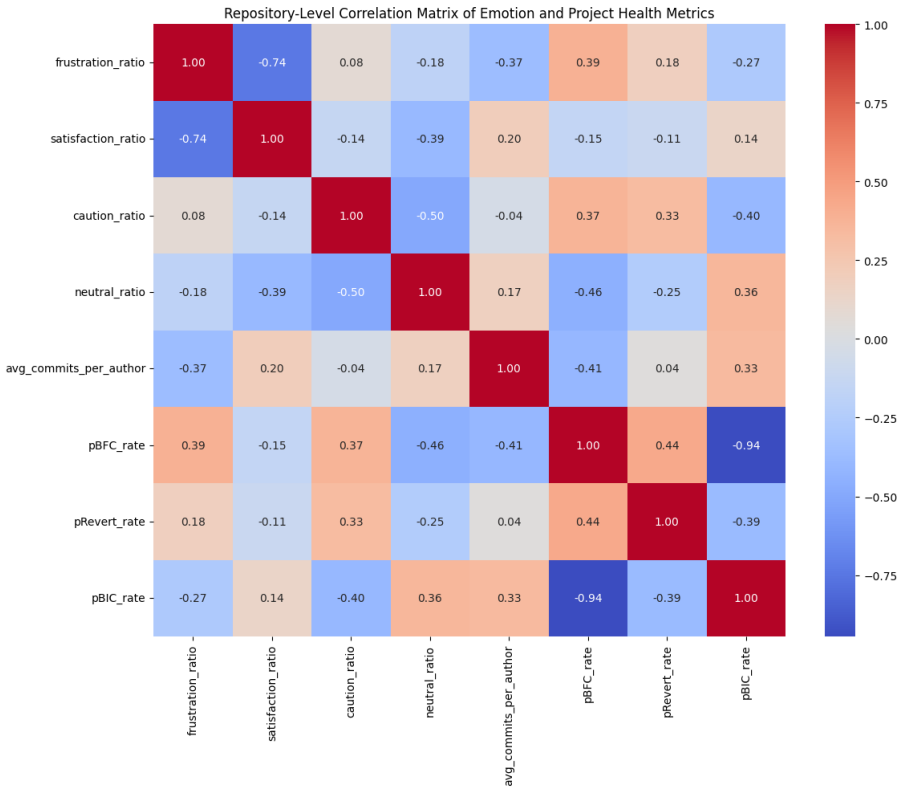


Figure 9.2: Repository-Level Correlation Matrix of Emotion and Project Health Metrics.

projects with a higher prevalence of "loud" emotions like frustration and caution tend to also have a higher rate of explicit bug-fixing activity. Conversely, a different pattern emerged for subtler issues. There was a moderate positive correlation between a repository's `neutral_ratio` and its `pBIC_rate` ($r = 0.36, p = 0.036$) and a moderate negative correlation between its `caution_ratio` and `pBIC_rate` ($r = -0.40, p = 0.019$). This indicates that projects dominated by a "quiet," neutral emotional tone may be more susceptible to introducing subtle, bug-inducing changes.

Table 9.1: Comparison of Quarterly and Repository-Level Correlations for Project Health Metrics.

Correlated Pair	**Quart. (r)**	**Quart. (p)**	**Repo. (r)**	**Repo. (p)**
Frustration vs. pBFC Rate	0.19	¡ 0.001	0.39	0.023
Satisfaction vs. pBFC Rate	0.04	0.097	-0.15	0.400
Caution vs. pBFC Rate	0.12	¡ 0.001	0.37	0.030
Neutral vs. pBFC Rate	-0.26	¡ 0.001	-0.46	0.007

9.4 Results: Emotion as an Indicator of Code Churn Dynamics

A specialized analysis was conducted to investigate the relationship between developer emotions and code churn, defined by metrics such as the average number of files changed, lines added, and lines deleted. The analysis was performed on a filtered subset of the data (84 quarters) for which churn metrics were available, ensuring greater accuracy.

9.4.1 Short-Term (Quarterly) Churn Analysis

Consistent with the project health findings, the contemporaneous quarterly analysis revealed only weak and inconsistent correlations between emotion ratios and churn metrics. As summarized in Table 9.2, the most notable signal was a minor positive correlation between the `neutral_ratio` and the average number of lines deleted ($r = 0.14$). Other correlations were negligible, such as `frustration_ratio` with `avg_files_changed` ($r = -0.04$) and `satisfaction_ratio` with `avg_lines_added` ($r = 0.02$). It is important to note that these results are based on a subset of 84 quarters, representing approximately 19.79% of the total sample (327 out of 1652 rows), due to computational constraints in churn extraction (Section 8.4). This sampling bias may affect the generalizability of the findings, as large repositories with long histories are underrepresented. Overall, the results indicate that emotional tone is not a strong short-term predictor of the volume of code change.

9.4.2 Long-Term (Repository-Level) Churn Analysis

The aggregated repository-level analysis again provided a contrasting and much clearer narrative, revealing strong and highly significant correlations. The findings

Table 9.2: Quarterly Correlations: Emotions vs. Code Churn Metrics.

Emotion	Files Changed (r)	p-value	Lines Added (r)	p-value	Lines Deleted (r)	p-value
Frustration	-0.04	0.453	-0.05	0.358	-0.05	0.388
Satisfaction	-0.06	0.258	0.02	0.720	-0.09	0.121
Caution	0.00	0.949	-0.04	0.474	-0.04	0.516
Neutral	0.09	0.095	0.03	0.533	0.14	0.012

indicate that the long-term emotional culture of a repository is a powerful indicator of its typical churn profile. The most compelling result is the very strong negative correlation between a repository's average `satisfaction_ratio` and all three churn metrics: `avg_files_changed` ($r = -0.89$), `avg_lines_added` ($r = -0.73$), and `avg_lines_deleted` ($r = -0.87$). This strongly suggests that repositories with a more positive emotional culture tend to experience significantly less code churn. Conversely, the `frustration_ratio` exhibited a strong positive correlation with both `avg_files_changed` ($r = 0.65$) and `avg_lines_deleted` ($r = 0.63$). This indicates that a sustained culture of frustration is associated with larger, more disruptive changes, as visually detailed in the correlation matrix in Figure 9.3.

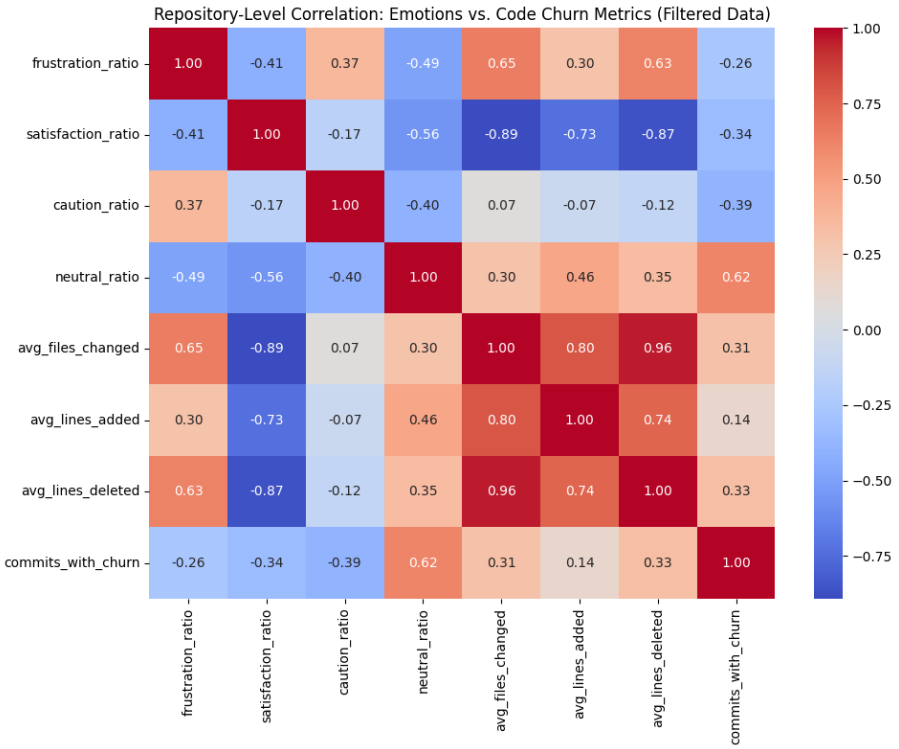


Figure 9.3: Repository-Level Correlation Matrix for Code Churn.

Table 9.3: Repository-Level Correlations: Emotions vs. Code Churn Metrics.

Emotion	Files Changed (r)	p-value	Lines Added (r)	p-value	Lines Deleted (r)	p-value
Frustration	0.65	¡ 0.001	0.30	0.084	0.63	¡ 0.001
Satisfaction	-0.89	¡ 0.001	-0.73	¡ 0.001	-0.87	¡ 0.001
Caution	0.07	0.694	-0.07	0.688	-0.12	0.490
Neutral	0.30	0.085	0.46	0.006	0.35	0.043

9.5 Overall Correlation Analysis

This comprehensive analysis reveals a nuanced, dual-faceted relationship between developer emotion and project health. While emotions lack the granularity to serve as effective short-term predictors of operational metrics, they emerge as a powerful barometer of a project's long-term engineering culture and risk profile. The findings suggest that a high prevalence of "loud" negative emotions like frustration is an indicator of a project struggling with overt, traditional bugs that require frequent fixes. In contrast, an overwhelmingly "quiet" or neutral emotional landscape, while correlated with fewer explicit bug fixes, is linked to a different risk: the introduction of subtle, bug-inducing changes that may be overlooked in a culture that lacks expressed vigilance.

Similarly, the churn analysis demonstrates that a long-term culture of satisfaction is strongly associated with smaller, incremental changes—a hallmark of stable, healthy projects. A culture of frustration, however, is linked to large, sweeping changes that may signify instability or major refactoring efforts. Ultimately, this work demonstrates that analyzing the long-term emotional profile of a software project provides valuable, high-level insights into its overall health, development style, and the specific nature of its technical debt.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

This thesis explored how developers' emotions, reflected in their commit messages, relate to the overall health of software projects. By combining large language models with domain adaptation and empirical analysis, it examined emotion as both a linguistic and socio-technical signal within software engineering. The study introduced the Project Health Dataset, linking 20,000 commits from 34 repositories with project metrics like churn, productivity, and bug activity, and a gold-standard subset of 2,000 manually annotated commits labeled as satisfaction, frustration, caution or neutral. It also proposed `CommiTune`, a fine-tuned CodeBERT model that achieved a Macro-F1 of 0.88 and Accuracy of 86%, substantially outperforming all baselines. The model's success showed that domain-specific pre-training and targeted generative augmentation are more effective for emotion detection than model scale alone. Applied at scale, `CommiTune` revealed that while short-term emotional shifts carry little predictive power, long-term patterns of frustration, caution, and satisfaction correlate strongly with project stability, bug-fix rates, and code churn. This positioned developer emotion as a meaningful indicator of team culture and project resilience.

10.2 Contributions

The key contributions of this work are threefold. First, we introduce **CommiTune**, a fine-tuned `CodeBERT` model enhanced with LLM-based data augmentation, which achieves a Macro-F1 score of 0.88 for emotion detection in developer text. Second, we present the **Project Health Dataset**, comprising 20,000 labeled commit messages mapped to bug-fix activity, code churn, and productivity indicators, providing a valuable resource for studying affect in software engineering. Finally, through empirical analysis, we demonstrate that **long-term aggregated emotional trends serve as reliable structural indicators of project health**, while short-term emotional fluctuations tend to be noisy and less predictive.

10.3 Future Work

Future research should extend this work through broader and more diverse datasets to reduce sampling bias and validate generalizability across organizations and languages. Models should move beyond simple emotion labels and capture mixed or context-driven feelings, like sarcasm or conflict. Tracking emotions over time and exploring cause-and-effect could help predict project outcomes rather than just explain them afterward. Ethical safeguards, like anonymizing data and using it responsibly, are essential. Combining multiple sources: pull requests, issue comments, and chat logs could give a fuller picture of how developers feel while working.
This approach helps us understand not just the code developers write, but the emotions that shape how they write it.

Bibliography

[1] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007, pp. 364–373. DOI: 10.1109/ESEM.2007.13.

[2] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, USA: ACM, 2007, pp. 529–540. DOI: 10.1145/1315245.1315311.

[3] R. Feldman, "Techniques and applications for sentiment analysis," *Communications of the ACM*, vol. 56, no. 4, pp. 82–89, 2013.

[4] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 352–355.

[5] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance: Possibilities for detecting burnout and productivity?" In *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 247–258.

[6] V. Sinha, A. Lazar, and B. Sharif, "Analyzing developer sentiment in commit logs," in *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 520–523.

[7] M. Fadaee, A. Bisazza, and C. Monz, "Data augmentation for low-resource neural machine translation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, R. Barzilay and M.-Y. Kan, Eds., Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 567–573. DOI: 10.18653/v1/P17-2090. [Online]. Available: https://aclanthology.org/P17-2090/.

[8] V. Blinov, V. Bolotova-Baranova, and P. Braslavski, "Large dataset and language model fun-tuning for humor recognition," in *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019, pp. 4027–4032.

[9] P. Koutsimani, A. Montgomery, and K. Georganta, "The relationship between burnout, depression, and anxiety: A systematic review and meta-analysis," *Frontiers in psychology*, vol. 10, p. 284, 2019.

[10] N. Majumder, S. Poria, D. Hazarika, R. Mihalcea, A. Gelbukh, and E. Cambria, "Dialoguernn: An attentive rnn for emotion detection in conversations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 6818–6825.

[11] F. A. Acheampong, C. Wenyu, and H. Nunoo-Mensah, "Text-based emotion detection: Advances, challenges, and opportunities," *Engineering Reports*, vol. 2, no. 7, e12189, 2020.

[12] S. F. Huq, A. Z. Sadiq, and K. Sakib, "Is developer sentiment related to software bugs: An exploratory study on github commits," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2020, pp. 527–531.

[13] A. Mockus, "The corrective commit probability code quality metric," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 704–719, 2020. DOI: 10.1109/TSE.2018.2884955.

[14] A. Yadav and D. K. Vishwakarma, "Sentiment analysis using deep learning architectures: A review," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 4335–4385, 2020.

[15] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: A review of bert-based approaches," *Artificial Intelligence Review*, vol. 54, no. 8, pp. 5789–5829, 2021.

[16] A. Chowanda, R. Sutoyo, S. Tanachutiwat, *et al.*, "Exploring text-based emotions recognition machine learning techniques on social media conversation," *Procedia Computer Science*, vol. 179, pp. 821–828, 2021.

[17] D. Dave, *Github commit messages dataset*, 2021. DOI: 10.34740/KAGGLE/DSV/2143532. [Online]. Available: https://www.kaggle.com/dsv/2143532.

[18] P. Nandwani and R. Verma, "A review on sentiment analysis and emotion detection from text," *Social network analysis and mining*, vol. 11, no. 1, p. 81, 2021.

[19] S. Zad, M. Heidari, H. James Jr, and O. Uzuner, "Emotion detection of textual data: An interdisciplinary survey," in *2021 IEEE World AI IoT Congress (AIIoT)*, IEEE, 2021, pp. 0255–0261.

[20] N. V. Babu and E. G. M. Kanaga, "Sentiment analysis in social media data for depression detection using artificial intelligence: A review," *SN computer science*, vol. 3, no. 1, p. 74, 2022.

[21] S. K. Bharti, S. Varadhaganapathy, R. K. Gupta, *et al.*, "Text-based emotion recognition using deep learning approach," *Computational Intelligence and Neuroscience*, vol. 2022, no. 1, p. 2 645 381, 2022.

[22] J. Guo, "Deep learning approach to text analysis for human emotion detection from big data," *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 113–126, 2022.

[23] R. Mao, Q. Liu, K. He, W. Li, and E. Cambria, "The biases of pre-trained language models: An empirical study on prompt-based sentiment analysis and emotion detection," *IEEE transactions on affective computing*, vol. 14, no. 3, pp. 1743–1753, 2022.

[24] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A survey on sentiment analysis methods, applications, and challenges," *Artificial Intelligence Review*, vol. 55, no. 7, pp. 5731–5780, 2022.

[25] T. Dohmke, "100 million developers and counting - the github blog," *The GitHub Blog*, 2023.

[26] M. U. Hadi, R. Qureshi, A. Shah, *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.

[27] Z. Wang, Q. Xie, Y. Feng, Z. Ding, Z. Yang, and R. Xia, "Is chatgpt a good sentiment analyzer? a preliminary study," *arXiv preprint arXiv:2304.04339*, 2023.

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

[29] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," *arXiv preprint arXiv:2305.15005*, 2023.

[30] H. Hovhannisyan, Y. Zhou, A. Durdyev, A. Bosu, and M. Greiler, "Linking code and documentation churn: Preliminary analysis," *arXiv preprint arXiv:2410.05992*, 2024. [Online]. Available: https://arxiv.org/abs/2410.05992.

[31] Z. Liu, K. Yang, Q. Xie, T. Zhang, and S. Ananiadou, "Emollms: A series of emotional large language models and annotation tools for comprehensive affective analysis," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 5487–5496.

[32] J. Yang, H. Jin, R. Tang, *et al.*, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, pp. 1–32, 2024.

Appendix A

Sampled Repositories and Commit Counts

Below is the complete list of the 34 open-source repositories analyzed in this study. The table specifies the number of commits randomly sampled from each repository's history (2010–2023) to form the 20,000-commit dataset used for emotion analysis.

Table A.1: List of Sampled Repositories and Commit Counts

No.	Repository Owner	Repository Name	Commits Sampled
1	apache	hadoop	712
2	apache	hive	650
3	apache	kafka	821
4	apache	spark	945
5	elastic	elasticsearch	1102
6	facebook	react	880
7	facebook	react-native	765
8	flutter	flutter	998
9	gin-gonic	gin	420
10	golang	go	788
11	google	guava	350
12	kubernetes	kubernetes	1250
13	laravel	laravel	680
14	microsoft	vscode	915
15	moby	moby	730
16	nodejs	node	812
17	numpy	numpy	450
18	pandas-dev	pandas	620
19	prettier	prettier	380
20	prometheus	prometheus	510
21	pytorch	pytorch	855
22	rails	rails	710
23	redis	redis	490
24	scikit-learn	scikit-learn	580
25	spring-projects	spring-boot	615
26	spring-projects	spring-framework	690
27	storybookjs	storybook	430
28	tensorflow	tensorflow	1050
29	torvalds	linux	1420
30	webpack	webpack	550
31	pallets	flask	390
32	django	django	635
33	jupyter	notebook	352
34	rust-lang	rust	910
Total			**20,000**

Appendix B

Preprocessing and Keyword Details

This appendix provides the exact regular expression (regex) patterns and keyword lists used for data cleaning and for identifying specific commit types (e.g., bug fixes, reverts) in the project health analysis.

B.1 Text Cleaning Regex Patterns

The following regex patterns were applied sequentially to each commit message to normalize the text before analysis.

B.2 Bug-Fix and Failure Keyword Lists

The following keyword lists and regex patterns, were used to calculate the project health metrics, including the Potential Bug-Fixing Commit (pBFC) rate.

Bug-Fix Keywords (BUG_KEYWORDS) Used to identify commits that are likely fixing a bug.

- *Keywords:* fix, fixes, fixed, fixing, bug, bugs, issue, issues, error, errors, patch, patched, patches, resolve, resolves, resolved, resolving, correct, corrected, correcting, correction.
- *Regex Pattern:* `\b(fix`fixes—...—correction)_ (case-insensitive)

Failure-Inducing Keywords (FAILURE_KEYWORDS) Used as part of the heuristic to identify Potential Bug-Introducing Commits (pBICs).

- *Keywords:* oops, broke, broken, breakage, regression, regressed, fail, failed, failure, crash, crashed, crashing, introduce bug, introduced bug, caused bug.
- *Regex Pattern:* `\b(oops`broke—...—caused bug)_ (case-insensitive)

Table B.1: Regex Patterns for Text Cleaning

Step	Description	Regex Pattern
1	Normalize Whitespace & Newlines	`\s+` and `\s*\n\s*`
2	Remove Bracketed Content	`\[.*?\]` and `<.*?>`
3	Remove Preceding Words	`\b\w+\s+\w+\s+<.*?>`
4	Remove Phrase Separators	`\b\w+:\w+\b, \b\w+=\w+\b, ...`
5	Remove Email Addresses	`\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b`
6	Remove URLs & UUIDs	`?://§+|www§+|` `git-svn-id:.*?|svn://§+|...|`
7	Remove Metadata Fields	`reviewed-on:*.*,|` `commit-queue:*.*, ...|`
8	Isolate "bug" Keyword	Replaces `\bbug\b\s*=\s*\S+` with "bug"
9	Remove Angle Brackets	`[<>]`
10	Remove Metadata Signatures	`(signed-off-by|reviewed-by|acked-by):|` `+.*?(|)|`
11	Remove Alphanumeric Strings	`\b[A-Za-z]*\d+[A-Za-z]*\b`
12	Remove All Digits	`\d+`
13	Remove Words with ¿2 Repeated Letters	`\b\w*(\w)\1{2,}\w*\b`
14	Remove Repeated Punctuation	`[.,]{2,}`
15	Remove Special Characters	`[^,!?']`—

Revert Keywords Used to identify commits that revert previous changes.

- *Keywords:* revert, reverted, reverting, revert of, rollback, back-out, back out.
- *Regex Pattern:* `\b(revert(?:ed`ing)?—revert of—rollback—back[-]?out)_— (case-insensitive)

Appendix C

Annotation Process and Label Consolidation

This appendix details the full methodology used to create the gold-standard dataset for this study. It covers the initial guidelines provided to annotators, the data-driven refinement of the label scheme, and the final reconciliation process.

C.1 Initial Annotation Guidelines (Seven-Emotion Scheme)

The manual annotation was first conducted using a seven-emotion scheme common in sentiment analysis literature. Two independent annotators were tasked with assigning one of the following labels to each of the 2,000 commit messages:

- Joy/Excitement: Expressions of high-arousal positive emotion, often indicating enthusiasm for a new feature or breakthrough.
- Satisfaction: Expressions of contentment or accomplishment, related to successful fixes or implementations.
- Anger/Frustration: Expressions of negative emotion resulting from obstacles, difficult bugs, or problematic code.
- Sadness: Expressions of disappointment or low-arousal negative emotion.
- Neutral: Messages with no discernible emotional content, typically describing routine tasks.

C.2 Label Scheme Refinement and Consolidation

An initial inter-annotator agreement analysis on the seven-label scheme yielded a Cohen's of 0.596, indicating only modest agreement. A qualitative review of the disagreements revealed recurring confusion between semantically adjacent categories, particularly Joy versus Excitement and Anger versus Frustration. To build a more reliable and reproducible dataset, a data-driven consolidation was performed. In addition, a new, pragmatically important category, Caution, was introduced to capture

signals of risk and defensive programming, which are highly relevant in a software engineering context. The initial labels were mapped to a final, four-category scheme. This consolidation significantly improved agreement to a Cohen's of 0.710.
The final mapping was as follows:

Table C.1: Mapping of Initial Labels to Consolidated Labels

Initial Label(s)	**Consolidated Label**	**Rationale**
Joy, Excitement, Satisfaction	Satisfaction	Merged to form a single, robust positive emotion category.
Anger, Sadness, Frustration	Frustration	Merged to create a single negative emotion category.
Keywords (e.g., "workaround", "warning")	Caution	A new, high-precedence category for risk signals.
Neutral	Neutral	Remained unchanged.

C.3 Final Label Definitions and Reconciliation Rules

The final gold-standard dataset, used for all model training and evaluation, is based on the following four consolidated categories.

Satisfaction: Combines joy, excitement, and gratification. Represents feelings of accomplishment, positive outcomes, or enthusiasm. Example: "Implemented new feature without errors."

Frustration: Merges anger, sadness, and frustration. Reflects irritation, disappointment, or annoyance when facing challenges. Example: "Fix silly bug in login flow."

Caution: Represents defensive or risk-oriented signals. Captures warnings, preventive actions, or fallback solutions. Example: "Temporary workaround until API limitation resolved."

Neutral: For commits with no clear emotional or risk-related content, describing straightforward updates or maintenance. Example: "Update README.md."

A single gold label for each commit was then derived using a transparent, rule-based reconciliation procedure:

- Caution Precedence: If either annotator selected Caution, the final label was set to Caution.
- Agreement: If both annotators agreed on Satisfaction, Frustration, or Neutral, the shared label was retained.
- Neutral vs. Emotion: If one annotator chose Neutral and the other chose an emotional label, the emotional label was adopted.
- Direct Conflict: In cases of direct conflict between Satisfaction and Frustration, the label associated with the higher annotator-assigned intensity score was chosen. Ties were flagged for manual review.

Appendix D

Reproducibility Details

This appendix provides the necessary details to reproduce the key findings of this study, including fine-tuning hyperparameters and links to the code and model artifacts.

D.1 Fine-Tuning Hyperparameters

All fine-tuning experiments were conducted using a standardized configuration to ensure comparability. Table D.1 details the hyperparameters used for the main models evaluated in this work.

Table D.1: Hyperparameters for all fine-tuning experiments.

Parameter	**CodeBERT**	RoBERTa-large -MNLI	LLaMA 3.1 8B (QLoRA)
Optimizer	AdamW	AdamW	AdamW
Learning Rate	2×10^{-5}	2×10^{-5}	2×10^{-4}
Batch Size	8	8	4
Max Epochs	10	10	3
Early Stopping	Yes (patience=3)	Yes (patience=3)	No
Warmup Steps	500	500	0
Weight Decay	0.01	0.01	0.0
Precision	Mixed (FP16)	Mixed (FP16)	4-bit Quantization
GPU	Tesla P100	Tesla P100	Tesla P100
Random Seed	42	42	42

D.2 Code, Data, and Models (For Review)

To comply with the double-blind review process, all artifacts have been made available in an anonymized format. The final, de-anonymized links will be provided in the camera-ready version upon acceptance.

GitHub Repository: The complete codebase, including scripts for data preprocessing, model training, evaluation, and statistical analysis, is available at: https://github.com/anonymous-commit-emotion/reproducibility-pack-2025

Hugging Face Model: The final fine-tuned CommiTune model checkpoint is available on the Hugging Face Hub at: https://huggingface.co/anonymous-researcher-73/commiTune

Dataset: The 2,000-commit gold-standard annotated dataset and the final 20,000-commit dataset with emotion labels are included in the GitHub repository.

Appendix E

Caution Lexicon Used in Data Reconciliation

This appendix provides the lexicon used to programmatically flag and override commit messages as belonging to the Caution category during the data reconciliation phase. During the process, if a commit message contained any of the keywords from the list below (matched case-insensitively), its emotion label was automatically set to 'Caution'. This served as a high-precedence rule to standardize the labeling of risk-oriented commits before the final reconciliation logic was applied.

Table E.1: Explicit 'Caution' Keyword List from Reconciliation Script

Category	**Keywords**
Direct Warnings & Alerts	caution, warning, risk, careful, beware, cautious, heed, watch out, danger, prudent, advisory, alert
Preventive & Defensive Actions	avoid, prevent, ensure, disable, check for, validate, sanitize
Risk Mitigation & Recovery	recoverable, safe path, fallback, gracefully handle