Towards Effective Developer Communication in Open Source Software via Emotional Awareness

Mia Mohammad Imran

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Software Engineering Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/7776

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.
TOWARDS EFFECTIVE DEVELOPER COMMUNICATION IN OPEN SOURCE SOFTWARE VIA EMOTIONAL AWARENESS

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

MIA MOHAMMAD IMRAN

Doctorate in Computer Science - 2020-2024

Director: Dr. Kostadin Damevski,
Associate Professor, Department of Computer Science

Virginia Commonwealth University
Richmond, Virginia
August, 2024
Acknowledgements

I am grateful to my advisor, Dr. Kostadin Damevski, for his invaluable guidance throughout my doctoral studies. I also extend my thanks to Dr. Preetha Chatterjee, Dr. Bridget Thomson-McInnes, Dr. Rodrigo Spinola, and Dr. Shadi Rezapour for serving on my committee. My heartfelt appreciation goes to my wife, Tasneem Muneera Prottyasha, for her constant support and wisdom, as well as my family for their dedication. I acknowledge the influence of many others who have shaped my path to this point. I thank God to let me go thorough all the ups and downs. Thank you all.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>Abstract</td>
<td>x</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Contributions of the Thesis</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Thesis Structure</td>
<td>6</td>
</tr>
<tr>
<td>2 Background and Literature Review</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Emotion Models</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 Models in Psychology</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2 Capturing Emotions in Text Communication</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Emotion Recognition in Text</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Emotion Analysis in Open Source Software Development</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1 Emotion Analysis in Software Engineering</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Emotions in Text-Based Software Engineering Communication</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 Challenges</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Evaluation Metrics</td>
<td>20</td>
</tr>
<tr>
<td>3 Improving Emotion Classification in Software Engineering Using Data Augmentation</td>
<td>22</td>
</tr>
<tr>
<td>3.1 Background</td>
<td>22</td>
</tr>
<tr>
<td>3.1.1 Existing Software Artifacts</td>
<td>24</td>
</tr>
<tr>
<td>3.1.2 Data Augmentation</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Methodology</td>
<td>26</td>
</tr>
<tr>
<td>3.2.1 Data Selection</td>
<td>26</td>
</tr>
<tr>
<td>3.2.2 Pre-processing and Dataset Creation</td>
<td>27</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.2.3 Emotion Categories</td>
<td>27</td>
</tr>
<tr>
<td>3.2.4 Data Annotation</td>
<td>28</td>
</tr>
<tr>
<td>3.2.5 Studied Emotion Classification Tools</td>
<td>30</td>
</tr>
<tr>
<td>3.2.6 Metrics</td>
<td>32</td>
</tr>
<tr>
<td>3.2.7 Experiment Design</td>
<td>32</td>
</tr>
<tr>
<td>3.3 RQ1: Existing Emotion Classifiers</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1 Classification Results</td>
<td>32</td>
</tr>
<tr>
<td>3.3.2 Error Analysis of FNs</td>
<td>34</td>
</tr>
<tr>
<td>3.4 RQ2: Data Augmentation</td>
<td>39</td>
</tr>
<tr>
<td>3.4.1 Augmentation Strategies</td>
<td>39</td>
</tr>
<tr>
<td>3.4.2 Augmentation Process</td>
<td>43</td>
</tr>
<tr>
<td>3.4.3 Augmentation Results and Discussion</td>
<td>43</td>
</tr>
<tr>
<td>3.5 Threats to Validity</td>
<td>47</td>
</tr>
<tr>
<td>3.5.1 Construct validity</td>
<td>47</td>
</tr>
<tr>
<td>3.5.2 Internal validity</td>
<td>48</td>
</tr>
<tr>
<td>3.5.3 External validity</td>
<td>48</td>
</tr>
<tr>
<td>3.6 Chapter Contributions and Summary</td>
<td>49</td>
</tr>
<tr>
<td>4 Leveraging Large Language Models for Emotion Classification in Software Engineering Texts</td>
<td>50</td>
</tr>
<tr>
<td>4.1 Background</td>
<td>50</td>
</tr>
<tr>
<td>4.2 Experiment Setup</td>
<td>52</td>
</tr>
<tr>
<td>4.2.1 Compared LLMs</td>
<td>52</td>
</tr>
<tr>
<td>4.2.2 Classification Setup</td>
<td>54</td>
</tr>
<tr>
<td>4.3 Experiments</td>
<td>54</td>
</tr>
<tr>
<td>4.3.1 RQ1: Large Language Models as Emotion Classifier</td>
<td>54</td>
</tr>
<tr>
<td>4.3.2 RQ2: Integrating Polarity Features in LLMs</td>
<td>56</td>
</tr>
<tr>
<td>4.3.2.1 Procedure</td>
<td>56</td>
</tr>
<tr>
<td>4.3.2.2 Results and Discussion</td>
<td>57</td>
</tr>
<tr>
<td>4.4 Chapter Contributions and Summary</td>
<td>60</td>
</tr>
<tr>
<td>5 Understanding Figurative Language in OSS Communication</td>
<td>62</td>
</tr>
<tr>
<td>5.1 Background</td>
<td>62</td>
</tr>
<tr>
<td>5.2 Dataset</td>
<td>66</td>
</tr>
<tr>
<td>5.2.1 Data Collection</td>
<td>67</td>
</tr>
<tr>
<td>5.2.2 Data Annotation</td>
<td>68</td>
</tr>
<tr>
<td>5.2.2.1 Verifying Figurative Expressions</td>
<td>69</td>
</tr>
<tr>
<td>5.2.2.2 Rephrasing Sentences</td>
<td>70</td>
</tr>
</tbody>
</table>
5.3 Prevalence of SE-Specific Figurative Language

5.4 Experiments

5.4.1 RQ1: LLM’s figurative language interpretation capability

5.4.1.1 Compared LLMs

5.4.1.2 Procedure

5.4.1.3 Results and Discussion

5.4.2 RQ2: Performance Improvement in Affective Analysis

5.4.2.1 Compared models

5.4.2.2 Contrastive learning

5.4.2.3 Datasets

5.4.2.4 Procedure and Metrics

5.4.2.5 Results and Discussion

5.4.3 RQ3: Software Engineering Task Automation Where Affect Plays a Role

5.4.4 Implications

5.5 Threats To Validity

5.6 Chapter Contributions and Summary

6 Decoding Emotion Causes in Software Engineering Communication Using Zero-shot LLMs

6.1 Background

6.1.1 Prompt Engineering for Zero-Shot LLMs

6.1.2 Automated Emotion-Cause Extraction in NLP

6.2 Preliminary Study: Detecting Emotion Types

6.2.1 Datasets

6.2.2 Emotion Model

6.2.3 Compared Models

6.2.4 Basic Emotion Prompting

6.2.5 Granular-level Emotion Prompting

6.3 Emotion-Cause Extraction

6.3.1 Annotation

6.3.2 Model Selection

6.3.3 Prompt Design

6.3.4 Results

6.3.4.1 BLEU score

6.3.4.2 BLEU Score Interpretation

6.3.4.3 Discussion

6.3.4.4 Error Analysis
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shaver’s tree-structured emotion model</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Mapping of GoEmotions category to Shaver’s emotions</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>Comparison of emotion detection tools on GitHub data</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>Distribution of the error categories (as defined by Novielli et al.) in the FN instances</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>Emotion classification results for data augmentation strategies. For F1-score, we also show the percentage improvement over the original (unaugmented) score.</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>Model versions used from the Hugging Face library</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation of LLMs on the GitHub Dataset (F1-score).</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>Evaluation of Polarity-enhanced LLMs on the GitHub Dataset (F1-score metric, P=Polarity).</td>
<td>59</td>
</tr>
<tr>
<td>9</td>
<td>Percent of EMS with a higher similarity to the original sentence than corresponding DMS ($\text{Sim}<em>{\text{EMS}} &gt; \text{Sim}</em>{\text{DMS}}$).</td>
<td>81</td>
</tr>
<tr>
<td>10</td>
<td>Evaluation of LLMs finetuned with figurative language on the GitHub Emotion Dataset (F1-score).</td>
<td>86</td>
</tr>
<tr>
<td>11</td>
<td>Evaluation of LLMs finetuned with figurative language on the Incivility Dataset (F1-score).</td>
<td>88</td>
</tr>
<tr>
<td>12</td>
<td>Evaluation of LLMs finetuned with figurative language on the Bug Report Priority dataset (F1-score).</td>
<td>91</td>
</tr>
<tr>
<td>13</td>
<td>Extended Shaver’s tree-structured taxonomy</td>
<td>102</td>
</tr>
<tr>
<td>14</td>
<td>Micro-averaged F1-score of emotion classification models for three different datasets.</td>
<td>105</td>
</tr>
</tbody>
</table>
15 Micro averaged F1-score of emotion classification for different models on GitHub dataset. The zero-shot LLMs use the GoEmotions list of 27 emotions.

16 BLEU scores of different zero-shot LLMs.

17 Clusters of Frustration emotion causes in TensorFlow.
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Classical ML approach pipeline</td>
</tr>
<tr>
<td>2</td>
<td>Emotion Recognition Prompting using ChatGPT</td>
</tr>
<tr>
<td>3</td>
<td>Example of data augmentation using four operators</td>
</tr>
<tr>
<td>4</td>
<td>Frequency of emotions per project</td>
</tr>
<tr>
<td>5</td>
<td>Distribution of False Positive instances and False Negative instances across different tools</td>
</tr>
<tr>
<td>6</td>
<td>FNs mapped to their secondary emotions (n \geq 5)</td>
</tr>
<tr>
<td>7</td>
<td>Fine-tuning procedure using token-level attention adjustment of polarity words</td>
</tr>
<tr>
<td>8</td>
<td>Figurative language annotation procedure</td>
</tr>
<tr>
<td>9</td>
<td>Distribution of figurative language occurrence in GitHub sentences (200k) GitHub comments, (484k) sentences</td>
</tr>
<tr>
<td>10</td>
<td>RQ1 evaluation pipeline</td>
</tr>
<tr>
<td>11</td>
<td>Wrongly classified utterances among the zero-shot LLMs</td>
</tr>
</tbody>
</table>
Abstract

TOWARDS EFFECTIVE DEVELOPER COMMUNICATION IN OPEN SOURCE SOFTWARE VIA EMOTIONAL AWARENESS

By Mia Mohammad Imran

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2024.

Director: Dr. Kostadin Damevski, Associate Professor, Department of Computer Science

Emotions play an integral yet understudied role in open-source software development, profoundly shaping critical collaborative processes such as knowledge sharing, decision-making, and team dynamics. However, accurately detecting and analyzing emotions in developer communications poses significant challenges due to the lack of visual and auditory cues in text-based interactions. This dissertation investigates techniques to enhance the understanding and modeling of emotions within the textual artifacts of open-source projects. We conduct an extensive evaluation of existing emotion classification tools using a novel dataset of annotated GitHub comments. An error analysis reveals deficiencies in handling implicit emotional expressions and figurative language. We demonstrate that targeted data augmentation techniques can significantly enhance these tools, improving their performance on emotion classification tasks.

Next, we investigate the feasibility of utilizing Large Language Models (LLMs) such as BERT, RoBERTa, ALBERT, and CodeBERT for emotion classification. We
find that they surpass domain-specific tools. Targeted improvements can be achieved through fine-tuning, improving the models’ understanding of polarity and overall performance. However, our error analysis reveals that LLMs also struggle with figurative language. To address this, we conduct a study on software engineering-specific idioms and metaphors. By applying contrastive learning and task-specific fine-tuning, we enhance the models’ ability to understand the context of figurative expressions. This approach shows consistent improvement in emotion classification as well as related tasks such as incivility classification and bug report prioritization.

Additionally, we experiment with the feasibility of utilizing zero-shot reasoning to automatically extract emotion causes from developer communications. By formulating suitable prompting strategies, generative Large Language Models like ChatGPT, GPT-4, and Flan-Alpaca exhibit reasonably accurate zero-shot performance on this challenging task. A case study uncovering common triggers of Frustration in TensorFlow issues showcases potential applications for gaining actionable insights. Furthermore, our research proposes an extension of the standard emotion taxonomy to better capture the diverse range of emotions expressed in software engineering texts, facilitating more accurate and nuanced emotion analysis.
Effective communication is essential for successful collaboration and advancement in open-source software development. Communication channels, such as issue trackers, mailing lists, and online forums, facilitate knowledge-sharing and decision-making processes. Often emotion plays a role in communication, influencing how messages are conveyed and received. The expression of emotions, such as enthusiasm, empathy, or frustration, can deeply impact the tone, context, and effectiveness of the communication \cite{281,293}. While extensive research has been conducted on the technical aspects of open-source-software development \cite{260,123}, the role of emotions in text-based communication channels remains relatively unexplored \cite{153}.

Emotions often influence our attitudes, behaviors, decision-making, and interactions with others \cite{231,209,280}. They shape our perception and response to information and impact the dynamics within communication contexts. Emotions can significantly affect collaborative activities like open-source software development \cite{283}. Positive emotions, such as Joy, have been associated with increased productivity and job satisfaction in software engineering teams \cite{250,115,232,147,74}. Enthusiastic and passionate developers are more likely to convey ideas with enthusiasm, engage others in meaningful conversations, and increase productivity \cite{250,76}. Conversely, negative emotions, like Frustration, can decrease motivation and participation, leading to team attrition \cite{172}. They also hinder learning new programming languages and code comprehension \cite{149}. Emotions like Frustration or Resentment can have a negative effect on communication, impede knowledge transfer, and disrupt team
dynamics. Consequently, recent software engineering research has focused on studying developer emotions and their impact on software development activities.

Understanding emotions is complex in open-source communities like GitHub, where collaborators come from diverse backgrounds and cultures. Different individuals may have varying emotional responses to the same communication, influencing their engagement, motivation, and willingness to contribute. For instance, a critical comment on a code review might frustrate one developer while motivating another to improve their work. Therefore, recognizing and addressing emotions within text-based communication channels help promoting inclusiveness, reducing conflicts, and maximizing collaboration.

Emotions also play a vital role in knowledge sharing within the open-source community. Developers’ emotional states can influence the clarity and effectiveness of their communication during discussions or when providing support through online forums or mailing lists. By acknowledging and managing emotions, developers can improve the quality of their interactions, promote information exchange, and foster a learning culture within the open-source community.

The primary goal of this thesis is to investigate the effectiveness of existing software-engineering-specific emotion detection techniques and explore methods to enhance their efficacy. Next, we move towards using Large Language Models (LLM) to detect emotions and improving the techniques using LLMs to capture the nuances of software engineering communication. Additionally, we aim to examine the underlying causes of emotions within the context of open-source software development communication and examine how this knowledge can be practically applied. By addressing these research questions, we seek to shed light on the emotional dynamics of communication channels in the open-source community and provide insights into
enhancing communication practices to foster a supportive environment for developers.

The relevance and potential impact of this research are substantial. Firstly, understanding the effectiveness of existing software-engineering-specific emotion detection techniques is crucial for accurately identifying and analyzing emotions within open-source software development communication. By thoroughly evaluating the strengths and limitations of current approaches, we can propose improvements and advancements that enhance emotion detection and contribute to more precise assessments of emotional states in developer communication.

Secondly, uncovering the causes of emotions within open-source software development communication provides valuable insights into the factors that influence developers’ emotional experiences. By identifying these triggers, such as project challenges, conflicting opinions, or interpersonal dynamics, we can develop strategies to mitigate negative emotions and foster a positive communication climate. This research ultimately contributes to the well-being and satisfaction of developers, leading to increased productivity and better project outcomes.

Lastly, the practical application of this research holds immense potential for improving collaboration and communication practices within the open-source community. By developing guidelines and recommendations based on empirical evidence, we can facilitate more effective communication strategies that account for emotional dynamics. Strengthening knowledge-sharing and decision-making processes through these enhanced communication strategies can significantly contribute to the success of open-source software projects.

1.1 Contributions of the Thesis

This thesis makes several notable contributions to the field of emotion detection and understanding in the context of open-source software development communica-
The key contributions are outlined below:

**Evaluation of Existing Emotion Detection Tools in SE:** One primary contribution of this thesis involves conducting a comprehensive evaluation of existing emotion detection tools specifically tailored for software engineering contexts. The thesis assesses the effectiveness and accuracy of three such tools in detecting emotions within GitHub issue report and pull request comments. Through a meticulous evaluation and detailed error analysis, we aim to identify the strengths and limitations of current approaches and provide valuable insights regarding their practical applicability. The findings of this evaluation contribute to the advancement of emotion taxonomy, particularly in the context of text data, thereby enabling a more precise analysis of emotions within software developer communication. Additionally, the error analysis highlights the weaknesses of state-of-the-art models, opening avenues for further improvements in emotion detection techniques.

**Development of Data Augmentation Techniques for Improving Emotion Recognition:** Building upon the evaluation of existing emotion detection tools, this thesis introduces a novel data augmentation technique specifically designed to improve emotion recognition in software engineering communication. By leveraging generative language models such as BART [143], the thesis proposes three text augmentation techniques to generate new data instances. These techniques address the challenges posed by limited labeled data in the software engineering domain, ultimately enhancing the performance and generalizability of emotion recognition models.

**Emotion Recognition in SE Text Using LLMs:** This thesis next explores the application of Large Language Models (LLMs) for emotion recognition in software engineering texts. It compares the performance of six state-of-the-art LLMs, including BERT, RoBERTa, ALBERT, DeBERTa, CodeBERT, and GraphCodeBERT, in detecting emotions from GitHub comments. The evaluation demonstrates that
LLMs outperform traditional emotion detection tools and provides insights into the strengths and limitations of these models in the software engineering context. Error analysis reveals that LLMs struggle with recognizing implicit sentiment polarity and figurative language, suggesting areas for future enhancement.

**Improvement of LLMs in Emotion Classification Context:** The thesis also investigates the potential of integrating polarity features in training LLMs to improve their performance in emotion classification tasks. By incorporating positive and negative polarity features, the models demonstrate enhanced contextual understanding of emotions in software engineering texts. This approach results in improved accuracy and robustness in emotion classification, as evidenced by increased micro-averaged F1-scores.

**Figurative Language Understanding in SE Texts:** Recognizing the challenge that figurative language poses to emotion detection, this thesis examines the ability of LLMs to interpret metaphors and idioms commonly used in software engineering communication. By fine-tuning LLMs with a dataset specifically curated for figurative language, the research shows that enhanced models can better understand and classify emotions in texts containing figurative expressions. This contributes to more accurate affective analysis and improves the detection of incivility and prioritization of bug reports in software engineering projects.

**Uncovering the Causes of Emotions:** Going beyond just simply detecting the emotion in the text, we aim to delve into the underlying causes of emotions in software developer communication in the open-source context. We propose the utilization of Large Language Models (LLM) \[1\] as zero-shot models \[95\] for uncovering the causes of emotions. By employing advanced language models, such as ChatGPT \[16\], GPT-4 \[17\], LLaMA \[22\], Alpaca \[20\], and flan-alpaca \[10\] as “zero-shot” models without task-specific fine-tuning, the underlying factors and triggers that lead to specific emo-
tional responses in the context of open-source software development are explored. The application of zero-shot LLMs allows for the analysis of large volumes of text data, enabling the identification of patterns, topics, and contextual cues that contribute to emotional experiences. Through the utilization of these advanced models, this research reveals that emotion causes in software engineering text can be identified effectively. Additionally, a case study focusing on automatically detecting the causes of *Frustration* in a specific open-source software repository over a one-year timeline provides valuable insights into the dynamics of emotional experiences in software developer communication.

1.2 Thesis Structure

The thesis is organized as follows: Chapter 2 reviews the background and literature on emotion models in psychology and their application to text communication, alongside methodologies for emotion recognition in natural language processing, and emotion analysis in open-source software development. Chapter 3 evaluates existing emotion detection tools in software engineering and introduces a novel data augmentation technique to enhance emotion recognition performance. Chapter 4 explores the use of LLMs such as BERT, RoBERTa, and CodeBERT for emotion classification in GitHub comments, comparing their efficacy and limitations. Chapter 5 addresses the challenges of figurative language in emotion detection and assesses LLMs’ ability to interpret metaphors and idioms in software engineering communication. Chapter 6 investigates the causes of emotions in software developer communication using zero-shot LLMs, featuring a case study on detecting the causes of *Frustration* in the TensorFlow repository. Finally, Chapter 7 concludes the thesis by summarizing the key findings, discussing their implications, and suggesting future research directions.
CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter provides an overview of the background and existing literature on emotion models in psychology and on the role of emotions in open-source software development communication. We will explore various emotion models, specifically those relevant to text communication, and discuss popular models used for understanding software engineering communication. Moreover, we will delve into various methodologies employed in natural language processing (NLP) for emotion recognition. Additionally, we will go into the analysis of emotions in the context of open-source software development and the utilization of machine learning and deep learning techniques for emotion analysis. The chapter concludes by highlighting the challenges associated with emotion analysis in software engineering communication.

2.1 Emotion Models

2.1.1 Models in Psychology

Emotions have been extensively studied in the field of psychology, leading to the development of various models that seek to explain the structure and nature of emotions. On a large scale, there have been three different approaches [268, 101]:

**Categorical approach** The premise underlying this approach is that there exist a small number of emotions that are universally acknowledged [268]. One influential model in this approach is the basic emotion theory proposed by Ekman et al. [294], which suggests that there are universal and discrete emotions such as *Happiness*, *Sadness*, *Anger*, *Fear*, *Disgust*, and *Surprise* each associated with distinct facial ex-
pressions. This theory emphasizes the evolutionary significance of these emotions and their role in adaptive behavior.

Another model is the hierarchical model, as explored by Shaver et al. This approach focuses on the representation of emotions as prototypes, with certain features and characteristics defining each emotion category and further reaching out. Shaver’s tree-structured model has three layers. On the basic level, the model has six emotions: Love, Joy, Anger, Sadness, Surprise, and Fear. These six basic emotions are further branched out to Secondary and Tertiary levels totaling 135 emotions. This tree-structured model is featured in Parrott et al.’s categorization. Table 1 shows the emotion categorization of Shaver’s model.

Studies conducted by Cowen et al. identified 27 distinct categories based on 2185 short videos, 28 categories using facial expressions, and 24 using human vocalization. Based on Cowen et al.’s studies, Demszky et al. devised 27 categories for text-based emotion recognition. They also provided a mapping between these 27 categories and Ekman et al.’s six basic categories.

Although it is widely accepted that emotions are comprised of basic categories that are combined to form more complex emotions (e.g., Frustration), there is no consensus on the complete list of categories that accommodate the wide range of emotions observed in human communication text.

**Dimensional approach** The underlying premise of this approach posits that emotional states are not self-contained but rather interlinked, exhibiting a systematic relationship. This framework, often referred to as the VAD model or PAD model, provides a comprehensive assessment of emotional experiences and has been applied in various research contexts. *Valence* (V) measures the positivity or negativity of emotion, *Arousal* (A) gauges the level of activation or agitation, and *Dominance* (D) indicates the perceived control or influence in a situation. This
Table 1.: Shaver’s tree-structured emotion model

<table>
<thead>
<tr>
<th>Basic Emotion</th>
<th>Secondary Emotion</th>
<th>Tertiary Emotion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anger</td>
<td>Irritation</td>
<td>Annoyance, Agitation, Grumpiness, Aggravation, Grouchiness</td>
</tr>
<tr>
<td></td>
<td>Exasperation</td>
<td>Frustration</td>
</tr>
<tr>
<td></td>
<td>Rage</td>
<td>Anger, Fury, Hate, Dislike, Resentment, Outrage, Wrath, Hostility, Bitterness, Ferocity, Loathing, Scorn, Spite, Vengefulness</td>
</tr>
<tr>
<td></td>
<td>Envy</td>
<td>Jealousy</td>
</tr>
<tr>
<td></td>
<td>Disgust</td>
<td>Revulsion, Contempt, Loathing</td>
</tr>
<tr>
<td></td>
<td>Torment</td>
<td>-</td>
</tr>
<tr>
<td>Love</td>
<td>Affection</td>
<td>Liking, Caring, Compassion, Fondness, Affection, Love, Attraction, Tenderness, Sentimentality, Adoration</td>
</tr>
<tr>
<td></td>
<td>Lust</td>
<td>Desire, Passion, Infatuation</td>
</tr>
<tr>
<td></td>
<td>Longing</td>
<td>-</td>
</tr>
<tr>
<td>Fear</td>
<td>Horror</td>
<td>Alarm, Fright, Panic, Terror, Fear, Hysteria, Shock, Mortification</td>
</tr>
<tr>
<td></td>
<td>Nervousness</td>
<td>Anxiety, Distress, Worry, Uneasiness, Tenseness, Apprehension, Dread</td>
</tr>
<tr>
<td>Joy</td>
<td>Cheefulness</td>
<td>Happiness, Amusement, Satisfaction, Bliss, Gaiety, Glee, Jolliness, Joviality, Joy, Delight, Enjoyment, Gladness, Jubilation, Elation, Ecstasy, Euphoria</td>
</tr>
<tr>
<td></td>
<td>Zest</td>
<td>Enthusiasm, Excitement, Thrill, Zeal, Exhilaration</td>
</tr>
<tr>
<td></td>
<td>Contentment</td>
<td>Pleasure</td>
</tr>
<tr>
<td></td>
<td>Optimism</td>
<td>Eagerness, Hope</td>
</tr>
<tr>
<td></td>
<td>Pride</td>
<td>Triumph</td>
</tr>
<tr>
<td></td>
<td>Enthrallment</td>
<td>Enthrallment, Rapture</td>
</tr>
<tr>
<td></td>
<td>Relief</td>
<td>-</td>
</tr>
<tr>
<td>Sadness</td>
<td>Suffering</td>
<td>Hurt, Anguish, Agony</td>
</tr>
<tr>
<td></td>
<td>Sadness</td>
<td>Depression, Sorrow, Despair, Gloom, Hopelessness, Glumness, Unhappiness, Grief, Woe, Misery, Melancholy</td>
</tr>
<tr>
<td></td>
<td>Disappoint</td>
<td>Displeasure, Dismay</td>
</tr>
<tr>
<td></td>
<td>Shame</td>
<td>Guilt, Regret, Remorse</td>
</tr>
<tr>
<td></td>
<td>Neglect</td>
<td>Embarrassment, Insecurity, Insult, Rejection, Alienation, Isolation, Loneliness, Homesickness, Defeat, Dejection, Humiliation</td>
</tr>
<tr>
<td></td>
<td>Sympathy</td>
<td>Pity</td>
</tr>
<tr>
<td>Surprise</td>
<td>Surprise</td>
<td>Amazement, Astonishment</td>
</tr>
</tbody>
</table>
model has been widely used in research on emotions and has been applied to various domains, including affective neuroscience and psychopathology \cite{284}.

Plutchik et al.’s \cite{304} psychoevolutionary theory of emotion posits that emotions have evolved as adaptive responses to different types of environmental challenges. This is a hybrid of the categorical approach and the dimensional approach. This model identifies eight primary emotions organized in pairs of opposites, such as Joy-Sadness, Anger-Fear, Anticipation-Surprise, and Trust-Disgust. Plutchik’s theory emphasizes the functional aspects of emotions and their evolutionary significance.

**Appraisal-based approach** Building upon the dimensional approach, the appraisal-based approach adopts componential models of emotion that are rooted in appraisal theory \cite{296}. This theory suggests that individuals can experience emotions by evaluating events, and the resultant emotional experiences are shaped by their own experiences, goals, and opportunities for action. Within this perspective, emotions are perceived as changes occurring in all noteworthy components, including cognition, physiology, motivation, motor reactions, feelings, and expressions \cite{268, 261}.

These models in psychology provide a foundation for understanding emotions and serve as a basis for further exploration in the context of text communication. By incorporating these theoretical perspectives, researchers can develop models and approach specific to analyzing emotions within text-based communication channels in software engineering.

### 2.1.2 Capturing Emotions in Text Communication

Given the prevalence of text-based communication in software engineering, capturing emotions in text-based communication presents unique challenges due to the absence of non-verbal cues typically present in face-to-face interactions \cite{279}. When individuals communicate through text channels, such as emails, chat messages, or
online forums, the lack of visual and auditory cues hinders the direct observation of facial expressions, tone of voice, and body language that convey emotions \[262\].

Emotions are often conveyed implicitly or indirectly in text \[279\]. Subtle nuances, irony, sarcasm, and metaphorical expressions can all contribute to the emotional content of a message \[223, 253\]. Detecting and interpreting these implicit emotions requires a deep understanding of the context, cultural references, and the ability to recognize figurative language usage \[102, 290\]. Emotions can be highly subjective and vary among individuals \[285\]. The same text can elicit different emotional responses depending on the reader’s background, experiences, and personal interpretation \[278\]. Emotion detection models need to account for this subjectivity and incorporate a certain level of flexibility to accommodate individual differences.

Furthermore, the lack of standardized annotations for emotional labels in text datasets poses a challenge for training and evaluating emotion detection models \[178\]. Unlike sentiment analysis, which often relies on predefined sentiment polarity (positive, negative, neutral), emotions encompass a broader range of categories and intensities \[69, 85, 219\]. Annotating large-scale datasets with fine-grained emotional labels is time-consuming and requires expertise in emotion theory, making it difficult to obtain sufficiently labeled data for training robust models \[62, 110\].

To address these challenges, researchers have explored various approaches and techniques. These include leveraging linguistic features, such as sentiment words, affective lexicons, and syntactic patterns, to capture emotional content. Additionally, machine learning and deep learning algorithms have been applied to learn patterns and associations between textual features and emotional states.
2.2 Emotion Recognition in Text

Emotion recognition in the text has been the focus of extensive research, with various approaches proposed to tackle this task. These approaches can be broadly categorized into keyword-based, rule-based, classical learning-based, deep learning-based, and hybrid approaches \cite{102}. In general, all approaches involve several pre-processing steps. These steps typically include tokenization, stop-word removal, lemmatization, part-of-speech (POS) tagging, and dependency parsing. These pre-processing techniques are applied to transform the raw text into a more structured and analyzable format. Tokenization breaks the text into individual words or tokens. Stop-word removal eliminates common and insignificant words that do not carry much semantic meaning. Lemmatization reduces words to their base or dictionary form to handle variations in tense, number, or gender. POS tagging assigns grammatical labels to each word, such as noun, verb, or adjective. Dependency parsing identifies the syntactic relationships between words in a sentence.

**Keyword-based approaches** rely on predefined lists of emotion-related keywords or phrases using lexicons. These approaches assign emotions to text based on the presence or frequency of specific keywords associated with different emotions. While simple to implement, keyword-based approaches are limited by the reliance on predefined lists, which may not capture the full range of emotional expressions. One widely used lexical resource is the NRC Word-Emotion Association Lexicons \cite{252, 270, 181}.

**Rule-based approaches** utilize linguistic rules and patterns to identify emotions in text. These rules are typically developed based on linguistic, statistics, and computational concepts \cite{269, 236}. Rule-based approaches offer more flexibility and interpretability, allowing for the incorporation of linguistic nuances. However, they require manual rule creation and may not generalize well to diverse datasets.
Classical machine learning-based approaches involve training machine learning models, such as Support Vector Machines (SVMs) \cite{297} or Naive Bayes \cite{271} classifiers, using handcrafted features. These features can include lexical, syntactic, or semantic information extracted from the text. The typical features are Bag-of-Words (BoW) \cite{272}, Term Frequency-Inverse Document Frequency (TF-IDF) \cite{124}, N-grams \cite{247}, Part-of-Speech (POS) tags, sentiment lexicons, emotion lexicons, semantic features, and syntax-based features. Classical learning-based approaches require feature engineering, and their performance heavily depends on the quality of the selected features. Researchers have used classical ML models such as SVM, Naive Bayes, Vector Space Model, k-Nearest Neighbor (KNN), Bagging, Boosting, and xGBoost for automated emotion classification \cite{308,276,299,299,214,297,271}. Figure 1 shows the pipeline of the classical machine learning-based approach.

Deep learning-based approaches such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), have achieved remarkable results in emotion recognition tasks \cite{291,173,225,165,180}. These models can automatically learn representations from text data, enabling them to capture the contextual information and dependencies that contribute to emotional expressions. However, deep learning approaches often require large labeled datasets and substantial computational resources for training.
In recent years, the field of emotion recognition has been significantly impacted by the emergence of Large Language Models (LLMs) and Prompt-based approaches \[87, 134, 161, 9, 26\]. LLMs, such as GPT \[186\], BERT \[170\], and RoBERTa \[145\], have revolutionized natural language processing by learning representations from massive amounts of text data. These models possess a deep understanding of language and have been leveraged for various emotion-related tasks \[87, 134, 161\]. LLMs excel in capturing the nuances and complexities of emotions in the text by leveraging their pre-training on large corpora. Through this pre-training, LLMs learn to encode and generate text, including emotional expressions, which enables them to provide rich and context-aware representations of emotions in textual communication.

The use of prompts has gained traction as an effective strategy for emotion recognition in text \[9, 26\]. Prompt-based approaches involve providing specific prompts or instructions to LLMs to guide their understanding and inference of emotions in the given text. By carefully designing and crafting prompts, researchers can elicit desired responses from the LLMs and enhance the accuracy of emotion recognition. Figure 2 shows a simple prompting approach using ChatGPT \[16\].

**Hybrid approaches** combine multiple techniques or models to enhance the performance of emotion recognition \[264, 274\]. For example, a hybrid approach may integrate rule-based methods with deep learning models or combine different machine learning algorithms \[239, 246, 213\]. By leveraging the strengths of different approaches, hybrid models aim to improve accuracy and robustness in emotion recognition.
2.3 Emotion Analysis in Open Source Software Development

2.3.1 Emotion Analysis in Software Engineering

Emotion analysis in the context of software engineering has emerged as a vibrant research area, attracting considerable attention from scholars across multiple disciplines. Researchers have employed various methodologies and techniques to investigate and comprehend the intricate domain of emotional aspects in developer interactions. Qualitative and quantitative analyses, sensors, visualization techniques, surveys, and diverse tools have been deployed to unravel the nuanced world of emotions in software development [153, 28, 13, 255, 232, 202, 172, 243, 182, 135, 137, 250, 76, 51, 18, 238, 240, 251, 226].

Open-source software development relies on collaboration and communication among a geographically distributed diverse community of developers with a diverse set of requirements [43, 206]. Within this context, text-based communication channels
such as issue trackers, code review platforms, commit messages, and discussion forums play a vital role in facilitating effective communication and collaboration in open-source projects [275, 259, 48, 153, 222]. This literature review will primarily focus on text-based communication channels.

2.3.2 Emotions in Text-Based Software Engineering Communication

Pletea et al. conducted sentiment analysis on security discussions on GitHub and explored the emotions expressed by developers during these discussions [245]. They analyzed the sentiment of comments in security-related discussions to gain insights into developers’ perceptions and emotions related to security issues. Guzman et al. performed sentiment analysis on commit comments in GitHub repositories to understand the emotional aspects of developers’ interactions during the software development process [241].

Ortu et al. analyzed the emotional expressions in JIRA issue comments to gain insights into developers’ emotions, such as Joy, Anger, Sadness, and Fear, and their impact on collaboration and software development outcomes [220]. Novielli et al. annotated a benchmark for emotion annotation in Stack Overflow using Shaver’s emotion model [303] to facilitate emotion analysis in software engineering contexts [183]. Calefato et al. developed Emotxt, a toolkit for emotion recognition from text, and applied it to analyze the emotional content of software developers’ comments in JIRA repositories as well as Stack Overflow comments [196, 135]. They explored the emotions expressed in the comments to provide developers with insights into their emotional states during the software development process.

Mäntylä et al. explored the possibilities of using Valence, Arousal, and Dominance (VAD) dimensions [306] for detecting burnout and productivity in software development [217]. They analyzed the emotional aspects of communication data
from issue trackers and version control systems to understand the affective states of developers and their potential impact on burnout and productivity. Islam et al. proposed DEVA, a method for sensing emotions in the Valence-Arousal space in software engineering text [176]. Later, Islam et al. developed MarValous, a machine learning-based approach for detecting emotions in the Valence-Arousal space [141]. They employed machine learning techniques to recognize emotions expressed in software engineering text, providing a quantitative representation of emotions based on the Valence-Arousal model.

Werder et al. proposed a method called MEME for extracting emotions from GitHub repositories [190]. They aimed to capture emotional data related to software development activities by analyzing issue comments and commit messages, enabling the understanding of emotions expressed in development-related text. Destefanis et al. studied the measurement of affects in 370K GitHub issue comments from 100K issues of 25K contributors to understand the emotional impact of commenters on the overall affect of issues [169]. They analyzed the affects expressed in issue comments to identify emotional patterns and their potential influence on issue resolution and collaboration. Ortu et al. (2018) conducted a mining analysis of communication patterns in software development using GitHub repositories [185]. They explored the communication patterns and emotional aspects of developers’ interactions through 650K comments from 130K issues of 64K contributors. Later, Ortu et al. empirically analyzed the affect of merged issues on GitHub repositories to understand the emotional impact of issue resolution [150]. They examined the emotions expressed in merged issues to gain insights into the affective states associated with issue resolution and the collaboration process.

Yang et al. conducted a case study on analyzing emotion words to predict the severity of software bugs in open-source projects [210]. They examined the relation-
ship between emotion words used in bug reports and the severity of reported software bugs to understand the emotional factors contributing to bug severity. Umer et al. proposed an emotion-based automated priority prediction model for bug reports [189]. They analyzed emotions expressed in bug reports and leveraged emotion-based features to predict the priority of bug reports, aiming to improve bug triage and resolution processes.

Huq et al. (2019) explored the effect of developer sentiment on fix-inducing changes in GitHub pull requests [140]. They investigated the relationship between developer sentiment expressed in pull request comments and the likelihood of inducing fixes, aiming to understand how emotions impact software development outcomes. Later, Huq et al. investigated the relationship between developer sentiment and software bugs in GitHub commits [119]. They analyzed the sentiment expressed in commit comments to explore the connection between developer sentiment and the occurrence of software bugs.

Deepak et al. investigated the interaction processes and emotions in a collaborative online network, focusing on software development communities [168]. They analyzed the emotional dynamics of interactions and examined how emotions influence collaboration and engagement within the developer community. Venigalla and Chimalakonda explored the emotions of the developer community towards software documentation [94]. Neupane et al. presented EmoD, an end-to-end approach for investigating emotion dynamics in software development [148]. They analyzed the emotional dynamics in developers’ communication data to understand how emotions change over time and their relationship with the software development process.

Chen et al. leveraged emoji usage to recognize sentiment and detect emotions expressed in developers’ communication using deep neural network [68]. Rong et al. conducted an empirical study on emoji use in software development communi-
They investigated the frequency and diversity of emoji usage in various software development communication channels to understand how developers utilize emojis to express emotions and sentiments. Bleyl et al. applied BERT-based emotion recognition to Stack Overflow posts to identify emotions expressed by developers.

While these studies provide valuable insights into the emotional aspects of software development and shed light on the profound impact of emotions on collaboration, teamwork, and overall software development outcomes, they also highlight several challenges in this area.

### 2.3.3 Challenges

Despite the growing interest and research efforts in emotion analysis in software engineering, several challenges persist in this field. These challenges include:

1. **Data Availability and Annotation**: There are limited manually annotated datasets available in software engineering emotion mining. Acquiring and annotating large-scale emotion-labeled datasets in the software engineering domain can be time-consuming and resource-intensive. The limited availability of labeled data hinders the development and evaluation of emotion analysis models.

2. **Subjectivity and Context Sensitivity**: Emotions are inherently subjective and context-dependent, varying from person to person and context to context, which poses a challenge in developing universally applicable emotion analysis techniques. The presence of subjectivity further affects the available manually annotated datasets.

3. **Linguistic Complexity**: Software engineering communication often involves technical jargon, abbreviations, and code-specific terminology.
research found that domain-specific lexicon helps to understand emotions and adapting a lexicon for a domain is challenging [177, 193, 192].

4. **Generalizability**: Achieving generalizability in emotion analysis poses a significant challenge as models developed and trained in one software development context may struggle to perform well in different software development contexts. Research suggests that optimal emotion and sentiment classification performance is achieved when models are trained and evaluated separately for specific software engineering channels such as GitHub issue comments and Stack Overflow comments [86, 184, 166].

5. **Real-time Analysis**: Real-time analysis of emotions in software engineering, particularly the integration of text-based emotion analysis techniques into real-time software development processes, remains a relatively unexplored area. Although Neupane et al. developed EmoD [148] using the EmoTxt [196] tool, there is a lack of research on the integration of text-based emotion analysis methods within the OSS community. Therefore, further investigation and research are necessary to explore and understand the potential integration of emotion analysis methods in real-time software development environments, with a specific emphasis on the OSS community.

### 2.4 Evaluation Metrics

Evaluation metrics are essential for assessing the performance of emotion analysis models in software engineering. The following metrics are commonly employed to evaluate the effectiveness of emotion analysis techniques:

**Precision**: Precision measures the proportion of correctly predicted emotions among the emotions predicted by the model. It indicates the model’s ability to accurately
identify true emotions without false positives. The Precision is calculated using the following equation:

\[
Precision = \frac{TP}{TP + FP}
\]

where TP represents the number of true positive predictions (correctly predicted emotions) and FP represents the number of false positive predictions (incorrectly predicted emotions).

**Recall**: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted emotions among all the actual emotions present in the data. It reflects the model’s ability to capture all the true emotions without false negatives. The recall is calculated using the following equation:

\[
Recall = \frac{TP}{TP + FN}
\]

where TP represents the number of true positive predictions (correctly predicted emotions) and FN represents the number of false negative predictions (missed emotions).

**F1-score**: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of the model’s performance by considering both precision and recall. F1-score is particularly useful when there is an imbalance between emotion classes in the dataset. The F1-score is calculated using the following equation:

\[
F1-score = 2 \times \frac{Precision \times Recall}{Precision + Recall}
\]

As emotion classification is usually multi-class, to calculate the average score across all classes, i.e., emotions, we use the micro-averaged variant which has been widely used in related tasks [144, 277].
CHAPTER 3

IMPROVING EMOTION CLASSIFICATION IN SOFTWARE ENGINEERING USING DATA AUGMENTATION

3.1 Background

Numerous approaches have been proposed for the detection of emotions in the written text by software developers [135, 182, 137, 176, 141]. However, the extensive evaluation of these approaches has been hindered by the limited availability of manually annotated ground truth data [48]. Manual annotation of emotions is a time-consuming process that requires minimizing annotator subjectivity [175, 48]. Moreover, emotion classifiers trained on general-purpose data perform poorly in the software engineering domain due to the specific vocabulary and characteristics found in the software-related text, such as the occasional presence of code snippets [86, 197]. Optimal performance in emotion classification is observed when the classifiers are trained and evaluated on specific software engineering channels, such as GitHub issue comments and Stack Overflow comments [184].

This chapter aims to address the limitations of current emotion classification in software engineering written communication and enhance its effectiveness through the use of data augmentation techniques. The following research questions are explored:

RQ1: How effective are existing emotion classifiers in detecting emotions in GitHub comments? Which types of emotions are most likely to be misclassified?

To answer this question, we create a new dataset for emotion classification based on GitHub issue and pull request discussions. The dataset is annotated with six primary emotion categories and further divided into secondary and tertiary emotions.
We evaluate three commonly used tools for software engineering emotion classification (ESEM-E [182], EMTk [135], SEntiMoji [137]) using our dataset. The results show that their accuracy is lower compared to the original datasets they were built and evaluated on. We conduct an error analysis focusing on instances where all three tools misclassify emotions, aiming to understand the limitations of current approaches. The analysis reveals that a significant number of errors are due to the tools’ inability to recognize explicit lexical cues present in the text.

RQ2: *Can automatic data augmentation techniques enhance the effectiveness of existing emotion classifiers?*

To answer this question, we explore three different data augmentation strategies. Each strategy significantly increases the size of the training set, generating ten times more training data than initially available. The strategies differ in terms of unconstrained augmentation, where the original text is modified randomly without additional checks, and augmentation with constraints to preserve the original emotion. Experimental results demonstrate that the most effective strategy focuses on preserving the emotional polarity of the text, whether positive (e.g., Joy and Love) or negative (e.g., Anger). Augmenting individual emotions directly is less effective due to the lack of sufficiently large software engineering-specific lexicons to generate diverse augmented instances.

In software engineering research, various emotion models have been utilized in different degrees. In next section, we provide an overview of the existing annotated datasets and introduce data augmentation as a technique to address the issue of data scarcity in machine learning by expanding the training set size.
3.1.1 Existing Software Artifacts

In order to investigate the emotions of software developers across different communication channels and software artifacts, researchers have curated manually annotated datasets, also known as gold sets. These datasets make use of predefined emotion categories. For instance, Ortu et al. [220] annotated a JIRA dataset containing 5992 issue samples, grouped as follows: group 1 consisted of 392 issue comments labeled with emotions such as Love, Joy, Surprise, Anger, Fear, and Sadness; group 2 contained 1600 issue comments labeled with emotions Love, Joy, and Sadness; and group 3 comprised 4000 issue sentences labeled with emotions Love, Joy, Anger, and Sadness. Novielli et al. [183] annotated a gold set comprising 4800 Stack Overflow questions, answers, and comments, with sentences labeled using Shaver et al. [303]’s six basic emotion categories. Venigalla et al. [94] analyzed 10996 commit messages related to software documentation updates from 998 GitHub projects and mapped them into Plutchik’s eight emotion categories [304]. Islam et al. [176] created a ground truth dataset of 1795 JIRA issue comments using the VAD model [306, 300].

3.1.2 Data Augmentation

Data augmentation (DA) is a technique employed to enhance the diversity of training data by modifying existing data in a targeted manner [72, 47]. Inadequate availability of high-quality training data is a common challenge in machine learning, particularly with the advent of complex models like deep learning. The concept of data augmentation originated in the domain of image processing, where researchers observed that performing operations such as rotating an image by 90 degrees produces a new training instance for image classification tasks, thereby improving the robustness of the model. Recently, the application of data augmentation to written
text has gained popularity, although it is generally a more challenging problem due to the intricate relationships among written words. In text data augmentation, it is crucial to maintain label invariance, which means that the augmented instance should retain the same label as the original instance.

Early techniques of interest in data augmentation included synonym replacement [237], which involves replacing a word with its synonym, and BackTranslation [234], a method that paraphrases a text by translating it to another language and then translating it back to the original language. Inspired by similar approaches in computer vision, researchers also introduced MixUp augmentation [211, 91], which combines existing examples to create new augmented instances. Wei et al. proposed a method - Easy Data Augmentation (EDA) [157] which worked surprisingly well for smaller datasets despite being a simple technique that uses straightforward operators, e.g., synonym replacement using WordNet [301], random insertion, random swap, and random deletion. As software artifact datasets are often small [89], we were inspired by EDA in devising our data augmentation technique. A recent research trend in data augmentation is contextual augmentation [91], which utilizes various large language models such as CBERT [159], BART [122], GPT-2 [103], and others. Kumar et al. [122] demonstrated that for classification tasks, BART outperforms other models due to its ability to generate longer sequences of text in context. In this work, we leverage the BART model as part of our augmentation operators.

Figure 3 illustrates an example of data augmentation applied to an utterance from a GitHub issue comment using the aforementioned operators. Research indicates that operators specifically tailored to the classification task at hand, such as emotion detection, are significantly more effective than generic ones [81].
3.2 Methodology

3.2.1 Data Selection

We selected four popular GitHub repositories, with each repository containing at least 50K stars. The repositories are Flutter/flutter\(^1\), Webpack/webpack\(^2\), Microsoft/TypeScript\(^3\), and Angular/angular\(^4\). From each repository, we collected the last 10K comments (until 11 November 2021) from pull requests and issues (5K pull request comments and 5K issue comments).

---

\(^1\)https://github.com/Flutter/flutter  
\(^2\)https://github.com/Webpack/webpack  
\(^3\)https://github.com/Microsoft/TypeScript  
\(^4\)https://github.com/Angular/angular
3.2.2 Pre-processing and Dataset Creation

We pre-processed each issue and pull request comment to replace the URL, user mentions, and code with ‘<url>’, ‘<username>’, and ‘<code>’ respectively. Consistent with previous research [197, 90], we did not remove stop words. We also did not include any additional pre-processing, since the emotion classification tools we use (ESEM-E [182], EMTk [135], SEntiMoji [137]) have their own pre-processing steps.

In a preliminary analysis, we observed that many instances in the GitHub comments are neutral, i.e., do not contain any emotion [243]. Our goal is to avoid creating a sparse dataset with mostly neutral instances, which would be inefficient and time-consuming to annotate. Novielli et al. also made a similar observation and performed selective sampling in creating their dataset of Stack Overflow comments [183]. Hence, to avoid including too many neutral instances, we applied a software engineering-specific sentiment analysis tool [104] to label the pre-processed instances into ‘positive’, ‘negative’, and ‘neutral’. For each of the four repositories, we randomly selected 250 pull request comments (125 ‘positive’ comments and 125 ‘negative’ comments), 250 issue comments (125 ‘positive’ comments and 125 ‘negative’ comments), to reach a total of 2000 instances.

3.2.3 Emotion Categories

As our primary emotion model, we use Shaver’s framework [303] of emotion which has been commonly used in several software engineering studies [135, 182]. As shown in Table 1, Shaver’s framework is a hierarchical (tree-structured) emotion representation model which has three levels. At the top level, there are 6 basic emotion categories: Love, Joy, Surprise, Anger, Fear, and Sadness. For each of the basic emotions, there are secondary and tertiary-level emotions, which refine the
granularity of the previous level. For example, *Optimism* and *Hope* are the secondary and tertiary level emotions for *Joy*, respectively.

We observed that some commonly expressed emotions in developer communications, such as Approval, Disapproval, Confusion, Curiosity, etc., are missing from Shaver’s framework (as shown in Table 1), which was not designed for emotions expressed in text. For example, the following GitHub comment can be categorized with the emotion Curiosity, “*I’m curious about this - can you give more context on what exactly goes wrong? Perhaps if that causes bugs this should be prohibited instead?*”, but it does not clearly fit into any of Shaver’s existing categories. Therefore, to accommodate a wider range of emotions observed in our dataset, we use the recent text-based emotion classification framework by Demszky et al., known as GoEmotions [110]. GoEmotions uses 27 emotions to annotate Reddit comments, which are mapped to Ekman’s [294] 6 basic emotion categories. Note that, 5 of Shaver’s basic emotions (Anger, Fear, Joy, Sadness, and Surprise) are the same as Ekman’s basic categories. Ekman’s basic category Disgust is a secondary emotion of Anger in Shaver’s categories, and Shaver’s basic category Love is a subcategory of Joy in Ekman’s basic categories.

To enhance Shaver’s categories, we include selected six emotion categories from GoEmotions [110] and mapped them to basic emotions (as shown in Table 2). We adopted GoEmotions’s definitions and mapping only when an emotion is missing from Shaver’s emotion list and does not conflict with Shaver’s framework. We consider these additional six emotion categories as secondary emotion categories.

### 3.2.4 Data Annotation

Pull request and issue comments in GitHub usually consist of multiple sentences. While sometimes each sentence may express different emotions, more often, the com-
Table 2.: Mapping of GoEmotions category to Shaver’s emotions.

<table>
<thead>
<tr>
<th>GoEmotions Category</th>
<th>Mapped Basic Emotion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disapproval</td>
<td>Anger</td>
</tr>
<tr>
<td>Approval</td>
<td>Joy</td>
</tr>
<tr>
<td>Admiration</td>
<td>Joy</td>
</tr>
<tr>
<td>Confusion</td>
<td>Surprise</td>
</tr>
<tr>
<td>Curiosity</td>
<td>Surprise</td>
</tr>
<tr>
<td>Realization</td>
<td>Surprise</td>
</tr>
</tbody>
</table>

ment as a whole conveys a unique emotion. Therefore, we consider comment-level granularity for data annotation.

Two human judges were provided a set of GitHub comments with annotation instructions as follows:

“You will use the coding schema reported in Table 1 and Table 2. For each row in the spreadsheet, please indicate what emotion it conveys (if any) among the basic emotions (first column in the table), which are: Anger, Love, Fear, Joy, Sadness, and Surprise. Multiple emotion labels from the basic emotions are allowed but you should try to avoid them if possible. You can use the second and third levels in the schema as a reference for choosing the primary emotion, but the annotation should be only for the primary emotions. Please mention the second and third-level emotions whenever they are prevalent, and provide a rationale for each annotation. Make sure you consider the emotion(s) of the entire comment and not of individual sentences.”. Note that we provided a combined table of Table 1 and Table 2. The annotation instructions contained more details of the schema including definitions and examples for the basic six emotions.

The judges initially annotated a shared set of 400 comments. The sample size of 400 is sufficient to compute the annotator agreement measure with high confidence. The two annotators manually labeled the comments and measured Co-
hen’s Kappa inter-rater agreement for the six basic emotions. For each of the emotions, they found agreement greater than 0.8, which is considered to be sufficient (> 0.6) [287]. The annotators further discussed their annotations until all disagreements were resolved. Afterward, the annotators separately annotated 800 instances each to reach a total of 2000 utterances. Figure 4 shows the distribution of basic emotion categories per project in the final annotated set. In total, our dataset consists of 340 instances of Anger, 220 instances of Love, 198 instances of Fear, 422 instances of Joy, 274 instances of Sadness, and 328 instances of Surprise.

3.2.5 Studied Emotion Classification Tools

We investigate three existing software engineering-specific emotion classification tools, which we describe as follows:

**ESEM-E [182]**: Murgia et al. proposed an emotion classification tool, which was later referred to as ESEM-E. ESEM-E used Parrott’s emotion categories as classi-
fication targets, which are also featured in Shaver et al.’s model. While the source code of ESEM-E is not publicly available, we carefully read the descriptions detailed in the paper and implemented them. ESEM-E uses uni-gram and bi-gram features and machine learning models such as Support Vector Machine (SVM), Random Forest, K-Nearest Neighbours (KNN), etc. As recommended by the authors, we use the SVM model.

EMTk: Calefato et al. proposed EMTk (also known as EmoTxt), which was designed to identify developer emotions from textual communication channels. EMTk identifies six primary emotions according to Shaver’s framework. The implemented tool is publicly available on GitHub. EMTk provides two types of data sampling, ‘NoDownSampling’ and ‘DownSampling’; ‘DownSampling’ randomly samples the majority class to balance the number of instances between the majority and minority class, while ‘NoDownSampling’ does not change the training data. We use ‘NoDownSampling’ to ensure that all three tools use the same training data set. They use SVM as the machine learning classifier.

SEntiMoji: Chen et al. proposed SEntiMoji, a transfer learning approach for emotion detection in software engineering text based on emojis. SEntiMoji is developed based on DeepMoji which is an existing deep learning-based emoji representation model. The SEntiMoji model can identify various different emotion categorization schema including Shaver’s framework. Their study showed that SEntiMoji can significantly outperform existing emotion detection methods (e.g., DEVA, EMTk, Mar Valous, ESEM-E) in software engineering. The SEntiMoji source code is publicly available on GitHub.

https://github.com/SEntiMoji/SEntiMoji
3.2.6 Metrics

We choose popular metrics used to evaluate a classification task: Precision, Recall, and F1-score, which aggregates the prior two. In places where we present combined results across all emotions, we use the micro-averaged variants of each of the metrics, as they are responsive to the frequency of occurrence of each constituent emotion.

3.2.7 Experiment Design

Using random stratified sampling [194] for each basic emotion, we divide our annotated dataset into the train (80%) and test (20%) sets. The test set contains a total of 400 data points including 68 instances of Anger, 44 instances of Love, 40 instances of Fear, 84 instances of Joy, 55 instances of Sadness, and 65 instances of Surprise.

3.3 RQ1: Existing Emotion Classifiers

RQ1: How effective are existing emotion classifiers in detecting emotions in GitHub comments? What types of emotions are most likely to be misclassified?

3.3.1 Classification Results

To answer this RQ, we evaluate three well-known tools for software engineering emotion classification (ESEM-E [182], EMTk [135], SEntiMoji [137]), on our dataset of GitHub issue and pull request discussions (described in Section 3.2). The per-emotion performance of the emotion detection tools is summarized in Table 3. The overall trend among all tools is for precision to be significantly higher than recall. In other words, the tools are acting conservatively, choosing to predict more utterances
Table 3.: Comparison of emotion detection tools on GitHub data.

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESEM-E</td>
<td>0.405</td>
<td>0.250</td>
<td>0.309</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.571</td>
<td>0.118</td>
<td>0.200</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.600</td>
<td>0.265</td>
<td>0.367</td>
</tr>
<tr>
<td>Anger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Love</td>
<td>ESEM-E</td>
<td>0.651</td>
<td>0.636</td>
<td>0.644</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.786</td>
<td>0.500</td>
<td>0.611</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.733</td>
<td>0.500</td>
<td>0.595</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fear</td>
<td>ESEM-E</td>
<td>0.533</td>
<td>0.200</td>
<td>0.291</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>1.00</td>
<td>0.200</td>
<td>0.333</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.714</td>
<td>0.125</td>
<td>0.213</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Joy</td>
<td>ESEM-E</td>
<td>0.458</td>
<td>0.321</td>
<td>0.378</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.640</td>
<td>0.190</td>
<td>0.294</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.609</td>
<td>0.167</td>
<td>0.262</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sadness</td>
<td>ESEM-E</td>
<td>0.759</td>
<td>0.400</td>
<td>0.524</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.778</td>
<td>0.382</td>
<td>0.512</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.857</td>
<td>0.327</td>
<td>0.474</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surprise</td>
<td>ESEM-E</td>
<td>0.596</td>
<td>0.431</td>
<td>0.500</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.823</td>
<td>0.446</td>
<td>0.580</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.846</td>
<td>0.338</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>ESEM-E</td>
<td>0.553</td>
<td>0.365</td>
<td>0.440</td>
</tr>
<tr>
<td></td>
<td>EMTk</td>
<td>0.759</td>
<td>0.292</td>
<td>0.422</td>
</tr>
<tr>
<td></td>
<td>SEntiMoji</td>
<td>0.723</td>
<td>0.278</td>
<td>0.402</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>0.678</td>
<td>0.312</td>
<td>0.421</td>
</tr>
</tbody>
</table>

as negative (lacking a certain emotion) than positive, which leads to lower recall. Based on the aggregate measure, F1-score, ESEM-E performed best for Love, Joy, and Sadness, EMTk performed best for Fear and Surprise, and SEntiMoji performed best for Anger.

Results across all emotions are summarized in the bottom part of Table 3. Here, on the micro-averaged F1-score metric, ESEM-E improves over the next best tool EMTk by 0.018 (4.3%). On micro-averaged precision, EMTk improves over SEntiMoji by 0.036 (5.0%), while on micro-averaged recall, ESEM-E improves the next best tool EMTk by 0.073 (25.0%). While ESEM-E performs best by far on recall, its performance on precision is worse than EMTk by 0.206 (37.3%) and SEntiMoji by
0.17 (30.7%). Overall, across three tools, the average micro F1-score is 0.421.

To examine whether the tools tend to struggle in the same instances or if they have complementary strengths, we plot Venn diagrams of the false positive and false negative instances in Figure 5. While the false positive instances seem to be broadly spread across different tools, the vast majority of false negative instances are shared among the three tools. In total, 176/301 (58%) false negative instances across all emotions were misclassified by all three tools.

### 3.3.2 Error Analysis of FNs

Because of the unusually high agreement between the tools on the false negative (FN) instances, we focus our error analysis there, i.e., on the 176 FN instances.

First, we examine the secondary emotions (as listed in Table 1 and Table 2) that are present in the FNs with the goal of understanding if specific emotions are particularly difficult to classify. We create a visualization in Figure 6 to understand the distribution of FN instances, i.e., to create the mapping of secondary emotion...
Fig. 6.: FNs mapped to their secondary emotions \( n \geq 5 \).

categories (right side of image) to the six basic level emotions (left side of image) in the FN instances. In this visualization, we only consider secondary emotion categories which have at least 5 FNs. The width of the ribbons of top-level emotions represents their proportions in the dataset, while the width of the ribbons on the right side represents their proportion of FNs. We observe that some secondary emotions like Irritation, Nervousness, and Zest have a significant number of FNs and represent the majority of FNs for basic emotions like Anger, Fear, and Joy. For instance, Irritation
expressed via comments like “oh my god, explanation of official documents waste eight hours of me. Why isn’t there a case to explain this”, was misclassified (as FN) 23 out of the 34 times (67.6%) it appeared in the test set. Nervousness, e.g., “I guess my concern is that it sets a precedent where somebody could see it and think that it would be fine to use in core”, was also difficult to recognize as it was misclassified 21 out of 32 times (65.6%), while Zest was also mistaken often, with a misclassification frequency of 14/18 (77.8%). Relative to these hard-to-recognize emotions, Affection, which is part of the Love basic emotion, was an FN only 5 out of 35 times (14.3%).

Second, to understand the specific difficulties that the tools encountered, we performed a manual qualitative analysis of the 176 FNs. To perform this analysis, we use the error categories defined by Novielli et al. [184] to understand sentiment classification errors in software engineering text.

For each of the FN instances in our dataset, one of the authors of this paper performed the initial error mapping, while another author reviewed it and indicated disagreements that were resolved via a discussion. In Table 4, we report the distribution of error categories assigned to our FN instances. During the analysis, we observed that multiple error categories can be assigned to some of the FN instances in our dataset. Hence, we chose more than one (i.e., two) error categories for 16 FN instances, while the rest 160 (176 - 16) instances were assigned one error category each.

The most frequent error category found in the FNs is General error, which indicates an inability to recognize lexical cues that occur in the text. For instance, in the comment “that’s awesome, I’ve been needing this for a while”, annotated as Joy, the tools likely missed clear lexical cues (e.g., the word ”awesome”). Similarly for “oh my god, explanation of official documents waste eight hours of me, Why isn’t there a case to explain this”, annotated as Anger, the tools likely missed the idiomatic expression
"waste N hours of me". In other cases, the classifiers miss due to misspellings or broken syntax, as in "It’s annoying me specifically when I want to set it as default value in constructors", which is annotated as Anger. General error is also the most prevalent in 10 of the 13 secondary categories that are shown in Figure 6.

In 61 cases, the tools failed because of the presence of Implicit sentiment polarity in texts. Often, humans use common knowledge to recognize emotions that the tools miss. Consider the following example “Specifically, I'm less confident in the second commit than the first. AFAICT, it could only return true if a recursive call to itself returned true and all of the recursive base cases returned false.”. This was annotated as Fear (annotators perceived it as an expression of Worry – a 3rd-level emotion that maps to Fear, see Table 1), but the emotion is not present explicitly. Sometimes, annotators inferred an emotion based on external knowledge. For instance, “In that case I would advise you to please file a separate issue with the exact steps and logs to reproduce the issue. Because this issue is about existing apps. Thanks”, was annotated as Anger since the speaker is expressing a violation of the community rules (Irritation as the secondary emotion). Implicit sentiment polarity is the most prevalent in 4 out of the 13 secondary emotions in Figure 6 (one category was a tie with General error). As reported by Novielli et al. [86], a hostile attitude is often implicit and indirect, which we observe in the error for Anger’s secondary emotions like Exasperation and Irritation. Demszky et al. [110] noted that the Nervousness emotion is likely expressed implicitly, which we also observe in our data; the top error category of comments marked with Nervousness is Implicit sentiment polarity.

In 15 cases, we notice that the tools were not able to correctly classify utterances due to Pragmatics. This type of error occurs when the annotators consider the context of the comment. In the comment "hmm, even after a push I still see this test on github, but not locally", the author seems to have encountered something unexpected, which
the annotators marked as Surprise.

Sometimes, the use of *Figurative language*, such as humor, irony, sarcasm, or metaphors, causes difficulties for the classifiers to identify emotions correctly. Often this type of utterance uses neutral words to express an emotion. For example, "Well, *if you tried it you’d know". In other cases, the lexical presence of *Politeness*, such as “thank you”, “please”, etc., may cause misclassification. For instance, consider the following example, “Hi, thanks for your contribution, but we can’t review this because you didn’t follow the contributing instructions [...]”, which is marked as Anger due to the violation of community rules (secondary emotion - *Irritation*). In other cases, the utterances involve *Polar facts*, that is the utterance invokes an emotion for most people, i.e., the annotators consider the reported situation to invoke an emotion. For instance, in “I blame the autoformatter.”, the annotators marked this comment as Anger as they considered the author was irritated for facing the same problem (second level emotion - *Irritation*). Overall, we observe that *Figurative language*, *Politeness* and *Polar facts* usually occur in negative emotions (i.e., *Anger*, *Sadness*, *Fear*). The annotation in emotion and sentiment is a subjective task as the perception of emotions varies depending on personality traits and personal relevant dispositions. We observe this in 3.1% cases in our error distribution. For instance, “Do you understand that it is impossible in some cases or can lead to increase size of bundle?” was considered as *Regret* (3rd level *Sadness*) by one annotator, however, the other annotator considered it “Neutral”.

RQ1 Takeaway: Towards improving or designing new emotion classification tools, some types of errors could be more difficult to address than others. We hypothesize that tools should be able to improve on the most prevalent category of *General error* the most, as the lexical cues can be introduced via better training data, which could then be recognized by the tools. Towards that goal, we next investigate if *Data Augmentation* can be an
Table 4.: Distribution of the error categories (as defined by Novielli et al. [184]) in the FN instances.

<table>
<thead>
<tr>
<th>Error category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>General error</td>
<td>77</td>
</tr>
<tr>
<td>Implicit sentiment polarity</td>
<td>61</td>
</tr>
<tr>
<td>Pragmatics</td>
<td>15</td>
</tr>
<tr>
<td>Figurative language</td>
<td>15</td>
</tr>
<tr>
<td>Politeness</td>
<td>10</td>
</tr>
<tr>
<td>Polar facts</td>
<td>8</td>
</tr>
<tr>
<td>Subjectivity in annotation</td>
<td>6</td>
</tr>
</tbody>
</table>

effective strategy to automatically build better training data.

3.4 RQ2: Data Augmentation

RQ2: Can automatic data augmentation techniques be used to improve the effectiveness of existing emotion classifiers?

3.4.1 Augmentation Strategies

We explore three different data augmentation strategies that target emotion classification in software engineering. For each strategy, we use augmentation operators that transform each instance from the training set into a number of augmented instances, each introducing a slightly different vocabulary or idioms into the training set. In fact, we augment by applying a randomly chosen set of our augmentation operators, one after the other in a “stacked” fashion. In some of the augmentation operators, we rely on recently-introduced generative techniques that are capable of introducing realistic word spans [122, 167].

Via the different augmentation strategies we propose, we explore unconstrained vs. constrained choices of augmented data. For instance, we examine software-specific vs. generic choices of words to augment with, and how to ensure the original emo-
tions are preserved (or enhanced) by the augmentation. Specifically, we introduce the following three data augmentation strategies: Unconstrained, Lexicon-based, and Polarity-based.

**Unconstrained Strategy.** The unconstrained strategy uses augmentation operators that have been previously shown to be effective in NLP and apply them at a randomly chosen location in the text. Inspired by Kumar et al.’s [122] work where they found that a BART-based [143] generative model outperformed other strategies, we use BART to create generative augmentation operations such as Word Insertion and Word Substitution. The Unconstrained Strategy uses the following four operators:

- **Word Insertion using BART:** We insert a word at any position in the original utterance.

- **Word Substitution using BART:** We substitute a word at any position.

- **Word Deletion:** We randomly delete a word at any position.

- **Sentence Shuffling:** When an utterance has more than one sentence, we randomly shuffle the sentences.

**Lexicon-based Strategy.** We observe that sometimes the Unconstrained Strategy produces utterances that may not preserve the original emotion. Note that one of the primary requirements of data augmentation is label invariant, i.e., for the original label to be preserved through the transformation. To deal with this problem, we leverage a software engineering-specific emotion lexicon [205] in order to validate the augmented words generated through the Unconstrained Strategy. Specifically, for each augmented utterance produced by the Unconstrained Strategy, we check if the augmented words exhibit emotion and if the word’s emotion does not match the original emotion. In that case, we replace the word with a software engineer-
ing emotion-specific word that preserves the original emotion of the instance. If an utterance is annotated as Joy, and an augmented word exhibits a different emotion (and not Joy), we replace the word with a word from the Joy category of a software engineering-specific lexicon. For example, the Unconstrained Strategy augments the following utterance, which is annotated as Love, “This looks good, thanks for clarifying the docs.” to “This looks worse, thanks for reviewing the docs.”. Here the introduction of the word “worse” changes the emotion of the original. However, if “worse” is replaced with a Love-specific word, i.e., “wonderful”, the text becomes “This looks wonderful, thanks for reviewing the docs.”, which preserves the original label.

As a lexicon, we use NRC’s emotion lexicon combined with the software engineering-specific emotional lexicon from Määttäniemi et al.’s work, which contains a total of 428 words. Since Määttäniemi et al.’s lexicon is not annotated with Shaver’s basic emotion categories, we use NRC’s emotion lexicon to map each word from Määttäniemi et al.’s lexicon to Shaver’s basic categories. For example, Määttäniemi et al.’s lexicon contains the word afraid, which is also available in the NRC emotion lexicon. Since each word in the NRC lexicon is annotated with a specific category (e.g., the word afraid is annotated under the emotion category Fear), we map these words to Määttäniemi et al.’s lexicon to get a lexicon that is software engineering-specific and also has associated emotion categories.

Note that, as the NRC emotion lexicon uses Plutchik’s emotion categories which have 8 basic emotions, we make two adjustments. First, their basic emotion Disgust is a subcategory in Shaver’s basic emotion Anger. Therefore, we combine NRC’s Disgust module with Anger module and use it in our Anger lexicon. Second, Plutchik’s categories do not contain Shaver’s basic category Love, therefore we use NRC’s positive module instead as Love; the positive module contains words with positive polarity. The Lexicon Strategy uses the following four operators:
- **Word Insertion**: We insert a word at any position in the original utterance. The inserted word comes from the above mention lexicon.

- **Word Substitution**: We substitute a word at any position. The substituted word comes from the above mention lexicon.

- **Word Deletion**: Same as the Unconstrained Strategy.

- **Sentence Shuffling**: Unconstrained Strategy.

**Polarity-based Strategy.** While the Lexicon-based Strategy removes some of the noise that is introduced by the Unconstrained Strategy, we believe the process can be streamlined and the augmentation quality further improved. For instance, a significant problem with the Lexicon-based Strategy is that it uses a lexicon with a very limited number of words. To overcome this constraint, instead of specific emotions, we focus on increasing the polarity of words in the augmented instances. Inspired by GoEmotions’ grouping of emotions with sentiment polarity, we formulate three rules that augmented instances have to follow: 1) preserve (or increase) positive polarity words when the annotated utterance is Love and Joy, 2) preserve (or increase) negative polarity words when the annotated utterance is Anger, Fear, and Sadness, and 3) preserve the original utterance polarity when the annotated utterance is Surprise.

To identify words that exhibit positive polarity, negative polarity, or no polarity, we use SentiWordNet 3.0. While ensuring that each valid instance follows the above criteria, we generate new augmented instances using the same operators as for the Unconstrained Dataset. The only modification is that for **Word Deletion**, we only randomly delete a word if it does not exhibit sentiment polarity.
3.4.2 Augmentation Process

For all three of the above data augmentation strategies, for each instance in our training set, we generate 10 augmented instances, which is considered a reasonable augmentation ratio in the literature [151]. For each generated instance, if Sentence Shuffling is used, we only apply it once. We apply $n$ augmentation operations to each instance, where $n = \max(2, 20\%\text{ of the length (i.e., number of words in the instance)})$ [97]. We use nlpaug [146] for the generative operations and use bart-base [143] as our BART model’s weights. To further ensure that the augmented instance does not change the meaning of the original instance, we added an additional quality check where we ensure that the cosine similarity of BERTOverflow [131] vectors computed from the augmented and original instance is ($\geq 0.9$) apart [127]. BERTOverflow [131] is a software engineering-specific version of BERT [170], pre-trained on the Stack Overflow data dump. We load BERTOverflow using the huggingface library [7].

3.4.3 Augmentation Results and Discussion

Overall, across all three tools, all of the augmentation strategies improved performance over the original results (Table 5). The average micro F1-score improvement with the Unconstrained Strategy is 4.8% (0.441), with the Lexicon Strategy, is 7.8% (0.455), and with the Polarity Strategy 9.3% (0.461). Considering the three tools separately, we observe improved F1-score across the board, with EMTk benefiting the most using the Polarity Strategy with an improvement in F1-score of 13.7%.

The Unconstrained Strategy worked best with SENtiMoji by improving the F1-score by 7.7%. Both Lexicon Strategy and Polarity Strategy improved most with EMTk by 10.7% and 13.7% respectively. The reasons likely lie behind the feature extraction methods of EMTk, as its classification features are based on an emotion
Table 5.: Emotion classification results for data augmentation strategies. For F1-score, we also show the percentage improvement over the original (unaugmented) score.

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Strategy</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ESEM-E</td>
<td>0.567</td>
<td>0.250</td>
<td>0.347 (12.3%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.571</td>
<td>0.235</td>
<td>0.333 (66.5%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.630</td>
<td>0.250</td>
<td><strong>0.358</strong> (-2.5%)</td>
</tr>
<tr>
<td></td>
<td>Unconstrained</td>
<td>ESEM-E</td>
<td>0.581</td>
<td>0.265</td>
<td>0.364 (17.8%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.531</td>
<td>0.250</td>
<td>0.340 (70.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.625</td>
<td>0.221</td>
<td>0.326 (-11.2%)</td>
</tr>
<tr>
<td></td>
<td>Anger</td>
<td>ESEM-E</td>
<td>0.500</td>
<td>0.235</td>
<td>0.320 (3.6%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.609</td>
<td>0.206</td>
<td>0.308 (54.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.615</td>
<td>0.235</td>
<td>0.340 (-7.4%)</td>
</tr>
<tr>
<td></td>
<td>Lexicon</td>
<td>ESEM-E</td>
<td>0.596</td>
<td>0.636</td>
<td>0.615 (-4.5%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.703</td>
<td>0.591</td>
<td>0.642 (5.1%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.719</td>
<td>0.523</td>
<td>0.605 (1.7%)</td>
</tr>
<tr>
<td></td>
<td>Polarity</td>
<td>ESEM-E</td>
<td>0.630</td>
<td>0.659</td>
<td>0.644 (0.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.659</td>
<td>0.614</td>
<td>0.635 (3.9%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.710</td>
<td>0.500</td>
<td>0.587 (-1.3%)</td>
</tr>
<tr>
<td></td>
<td>Love</td>
<td>ESEM-E</td>
<td>0.667</td>
<td>0.682</td>
<td><strong>0.674</strong> (4.7%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.727</td>
<td>0.545</td>
<td>0.623 (2.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.733</td>
<td>0.500</td>
<td>0.595 (0.0%)</td>
</tr>
<tr>
<td></td>
<td>Unconstrained</td>
<td>ESEM-E</td>
<td>0.545</td>
<td>0.150</td>
<td>0.235 (-19.2%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.600</td>
<td>0.225</td>
<td>0.327 (-1.8%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.700</td>
<td>0.175</td>
<td>0.280 (31.5%)</td>
</tr>
<tr>
<td></td>
<td>Fear</td>
<td>ESEM-E</td>
<td>0.600</td>
<td>0.150</td>
<td>0.231 (-20.6%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.818</td>
<td>0.225</td>
<td>0.353 (6.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.636</td>
<td>0.175</td>
<td>0.275 (29.1%)</td>
</tr>
<tr>
<td></td>
<td>Polarity</td>
<td>ESEM-E</td>
<td>0.500</td>
<td>0.150</td>
<td>0.231 (-20.6%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.867</td>
<td>0.325</td>
<td><strong>0.473</strong> (42.0%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.600</td>
<td>0.150</td>
<td>0.240 (12.7%)</td>
</tr>
<tr>
<td></td>
<td>Joy</td>
<td>ESEM-E</td>
<td>0.456</td>
<td>0.310</td>
<td>0.369 (-2.4%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.486</td>
<td>0.214</td>
<td>0.298 (1.4%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.477</td>
<td>0.250</td>
<td>0.328 (25.2%)</td>
</tr>
<tr>
<td></td>
<td>Unconstrained</td>
<td>ESEM-E</td>
<td>0.500</td>
<td>0.321</td>
<td>0.391 (3.4%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.590</td>
<td>0.274</td>
<td>0.374 (27.2%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.526</td>
<td>0.238</td>
<td>0.328 (25.2%)</td>
</tr>
<tr>
<td></td>
<td>Polarity</td>
<td>ESEM-E</td>
<td>0.492</td>
<td>0.345</td>
<td><strong>0.406</strong> (7.4%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.613</td>
<td>0.226</td>
<td>0.330 (12.2%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEntiMoji</td>
<td>0.575</td>
<td>0.274</td>
<td>0.371 (41.6%)</td>
</tr>
<tr>
<td></td>
<td>Unconstrained</td>
<td>ESEM-E</td>
<td>0.767</td>
<td>0.418</td>
<td>0.541 (3.2%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMTk</td>
<td>0.909</td>
<td>0.364</td>
<td>0.519 (1.4%)</td>
</tr>
</tbody>
</table>
ESEM-E performed best with the Lexicon Strategy, outperforming the original dataset by 6.8%. As ESEM-E directly uses unigrams and bigrams as its features, it is likely that the repetition of lexical cues produced by the Lexicon Strategy significantly helped this tool. EMTk performed most effectively with the Polarity Strategy (13.7% improvement) as positive and negative sentiment polarity scores are one of its features. SEntiMoji performed best with the Polarity Strategy as well, outperforming the original dataset by 8.0%; however, SEntiMoji’s performance did not vary significantly over all three augmentation approaches.
The emotions that are improved most with data augmentation strategies are *Sadness* and *Joy*, which is consistent with Murgia et al.'s findings that they are easier to identify compared to other basic emotions. The reason is likely because data augmentation helped to introduce more lexical cues that were missing in the original dataset. In our analysis for RQ1, we observed that the most prevalent error category for *Sadness* and *Joy* FNs was *General Error*. *Sadness* achieved a maximum F1-score of 0.557 in Unconstrained Strategy with SEntiMoji, and *Joy* achieved a maximum F1-score of 0.406 with the Polarity Strategy and ESEM-E. *Surprise* performed best with the Polarity Strategy where all three tools improved the F1-score with EMTk producing the best result, an F1-score of 0.630. Previous research shows that *Surprise* in SE is generally hard to detect since it is not very frequent. However, with the addition of GoEmotions’s categories and data augmentation, the detection of *Surprise* improved significantly. As noted in previous research, *Anger* and *Fear* are difficult to predict, as they often depend on the message context. During our error analysis, we saw that most *Implicit sentiment polarity* errors occur with *Anger* and *Fear*. These types of errors were difficult to identify even after data augmentation. SEntiMoji did not improve *Anger* performance in any of the strategies; while EMTk improved, its performance with the initial dataset was very low. In the case of *Fear*, ESEM-E did not improve with any of the strategies. However, *Fear* performed significantly better with the Polarity Strategy in EMTk, achieving an F1-score of 0.473. Further research on what caused EMTk to perform better for *Fear* may help to pinpoint how to further improve classifying this emotion.

One interesting case is that with the original dataset, *Love* performed best across all three tools, however, with data augmentation, the performance of *Love* did not improve significantly. This points to a limitation of data augmentation in that it can only be of a limited benefit, i.e., useful only in cases where sufficient lexical cues are
not already present in the data.

**RQ2 Takeaway:** Overall, we observe that Data Augmentation generally improves emotion classification performance across different emotions and tools. We observe improvements especially when the initial dataset has insufficient lexical cues for a specific emotion. Out of the three augmentation strategies we experimented with, the Polarity Strategy worked really well, as it provided a balance between completely unconstrained augmentation (which introduces noise) and highly constrained augmentation (which fails to increase size and diversity). Data augmentation is likely only able to improve performance up to point, as our current augmentation operators do not seem to help in identifying implicit emotions, such as Sarcasm.

### 3.5 Threats to Validity

Several limitations may impact the interpretation of our findings. We categorize and list each of them below.

#### 3.5.1 Construct validity

Construct validity concerns the relationship between theory and observation. Shaver’s emotions model [303] and the GoEmotions model by Demszky et al. [110] are two different schemas. We use a combination of the two, which may violate their original design. To mitigate this risk, we carefully read both of the original research and used Shaver’s model as our primary schema, integrating GoEmotions’ categories only when they are complementary and do not conflict with Shaver’s in any part of their definitions. Furthermore, the error analysis in RQ1 shows that none of the emotion categories that integrated GoEmotions are among the worse performing. Instead, the addition of GoEmotions secondary emotion categories specifically improves the performance of the basic category Surprise, which has exhibited a relatively low
F1-score in previous research in software engineering emotion classification [48].

3.5.2 Internal validity

Internal validity concerns the study design factors that may influence the results. One such threat to our study is not doing cross-validation, which would have improved the reliability of the results. We mitigate this threat by using stratified sampling and a reasonable train-test data split of 80%-20% respectively. Another threat is that to conduct our experiments we use existing tools and their released code, except for ESEM-E [182]. It is possible that we have incorrectly implemented ESEM-E, although, we explicitly followed the authors’ instructions to mitigate this threat. The subjectivity of annotating emotions (and in the error analysis) presents another threat to internal validity. However, the use of a three-tiered emotion structure and high inter-rater agreement (\( > 0.8 \)) ensures the reliability of the annotation procedure.

3.5.3 External validity

External validity concerns the generalization of our findings. Our study shows that data augmentation improves emotion classification across the three tools we experimented with. However, the specific augmentation strategies may not generalize beyond the three tools we studied and our dataset extracted from GitHub comments. More specifically, our findings may not generalize over other types of artifacts in software engineering, such as StackOverflow, JIRA, etc. While our results introduce the potential of data augmentation for emotion classification, further investigation is needed in other to validate our results beyond the tools and the data used in our study.
3.6 Chapter Contributions and Summary

This chapter presents the findings of our study on the limitations of existing machine learning-based tools for classifying emotions in software engineering-related text. We conducted a qualitative analysis of three tools, namely ESEM-E [182], EMTk [135], and SEntiMoji [137], using a curated dataset of 2000 GitHub pull requests and issue comments.

Our evaluation revealed that while some types of errors were more challenging to address than others, there is potential for improving the performance of existing tools through better training data. To this end, we explored three data augmentation strategies to enhance emotion detection in software-related text. Our results demonstrated that augmentation operators focusing on words with specific polarity yielded significant improvements compared to generic augmentation operators. Specifically, the use of polarity-based augmentation resulted in an average increase of 9.3% in the micro F1-Score across the three emotion classification tools.

Moving forward, our next objective is to assess the performance of Large Language Models (LLMs) in recognizing emotions. Additionally, we aim to investigate whether addressing the figurative elements present in the text can enhance emotion recognition.
CHAPTER 4

LEVERAGING LARGE LANGUAGE MODELS FOR EMOTION CLASSIFICATION IN SOFTWARE ENGINEERING TEXTS

4.1 Background

In the previous chapter, we highlighted the limitations of software engineering-specific emotion detection tools, which struggled to achieve satisfactory performance even after applying data augmentation techniques. This observation underscores the need for more advanced and robust approaches to emotion classification in software engineering texts. Large Language Models (LLMs) have emerged as a promising solution, demonstrating remarkable success in various natural language processing tasks, including emotion detection in the general domain [14].

However, compared to open-domain, LLMs rarely have been used in software engineering text for emotion classification. A recent study by Li et al. [11] introduced the use of BERT [170] as part of a two-stage framework for ambiguity detection, surpassing the performance of State-of-the-art SE tool in our curated GitHub emotion dataset curated. Another recent study by Bleyl et al. [29] found that BERT outperforms EMTk on a Stack Overflow dataset curated by Novielli et al. [183]. This shift towards leveraging LLMs prompts a critical inquiry into the overall efficacy of such models in the nuanced task of emotion classification within software engineering contexts. Notably, LLMs like BERT [170] and RoBERTa [145] have already demonstrated state-of-the-art results across various software engineering domains, including code completion, code review, bug localization, sentiment analysis, and toxicity detection [34, 70, 179, 19, 133, 129, 31].
In light of these developments, we embark on a comprehensive exploration of Large Language Models in emotion classification in our curated GitHub dataset (Chapter 3). Through rigorous evaluation and comparison, we aim to shed light on the strengths, limitations, and potential implications of leveraging LLMs in this nuanced domain. Our study utilizes the GitHub dataset we curated. We assess the effectiveness of six state-of-the-art LLMs, including four general domain models (BERT, RoBERTa, ALBERT and DeBERTa) and two specifically tailored for software engineering (CodeBERT and GraphCodeBERT).

To harness the potential of LLMs for emotion classification in software engineering, we formulate two key research questions:

**RQ1: How accurately can Large Language Models classify emotions in software engineering texts?**

In this research question, we aim to evaluate the performance of LLMs in classifying emotions in software engineering texts. We specifically focus on six state-of-the-art LLMs: four general domain models (BERT, RoBERTa, ALBERT, and DeBERTa) and two software engineering-specific models (CodeBERT and GraphCodeBERT). To assess their performance, we apply these LLMs to our curated GitHub dataset, which contains software engineering texts labeled with emotions. By comparing the predicted emotions from the LLMs with the ground truth labels, we can measure their f1-score in emotion classification. The results suggest that the LLMs generally outperform state-of-the-art models, EMTk, ESEM-E, and SEntiMoji by a margin. LLMs also outperform these domain-specific tools on their results after data augmentation. However, we observe that some errors still persists, namely, implicit polarity errors, figurative language, emoji, etc.

**RQ2: Can integrating polarity features in the training improve Large Language Models’...**
In our previous chapter, we observe that one of the major error category for emotion classification is the failure to understand polarity. We also observe in RQ1 that implicit polarity is still one of the major error categories. In this RQ, we focus on the potential benefits of incorporating polarity features during transformer model training. Specifically, we examine the extent to which positive and negative polarity features enhance the contextual understanding of LLMs for emotion classification in software engineering texts. We incorporate polarity features in attention layer of the LLMs. Our results suggest that they, indeed, improve the baseline LLMs further 1.75% to 9.16% in micro-averaged f1-score metric.

4.2 Experiment Setup

4.2.1 Compared LLMs

In this study, we compare six encoder-based LLMs: BERT, RoBERTa, DeBERTa, ALBERT, CodeBERT, and GraphCodeBERT. Previous research has shown that BERT, RoBERTa, and ALBERT perform well in software engineering affect analysis tasks [133, 65], while DeBERTa has demonstrated promising results in sentiment analysis [59]. Additionally, we include two domain-specific models, CodeBERT and GraphCodeBERT, which have been widely used and proposed by Microsoft for various software engineering tasks [100, 78]. Below, we provide a brief overview of each model:

BERT [170]: Introduced by Google in 2018, BERT is a bidirectional transformer-based model pre-trained on a diverse corpus, including Wikipedia and the Book Corpus. Its bidirectional architecture allows BERT to capture context from both the left and right contexts of a given word, making it highly effective for a wide range of NLP
tasks.

**RoBERTa [145]**: Developed by Meta AI, RoBERTa is a refined version of the transformer-based architecture. By modifying key hyperparameters and leveraging an extensive training dataset, RoBERTa enhances its performance on benchmark NLP tasks, demonstrating improved language representation understanding.

**DeBERTa [118]**: Developed by Microsoft Research, DeBERTa is an evolution of BERT that focuses on enhancing the decoding process in language understanding tasks. It incorporates directional masks during training to effectively capture bidirectional dependencies, excelling in tasks that require sequential reasoning.

**ALBERT [142]**: Designed by Google Research, ALBERT introduces efficiency improvements to the BERT architecture without compromising representational power. This is achieved through parameter reduction techniques during training, resulting in resource-efficient yet highly effective language representations.

**CodeBERT [113]**: Developed by Microsoft Research and tailored for programming languages and code understanding, CodeBERT is pre-trained on a large-scale dataset of programming tasks. This specialization allows CodeBERT to excel in source code-related tasks, such as code summarization and variable naming.

**GraphCodeBERT [116]**: Also developed by Microsoft Research, GraphCodeBERT is a transformer-based model designed for comprehending programming languages and code. Pre-trained on a vast dataset of programming tasks, it effectively captures the complex structures and semantics of code, serving as a proficient solution for source code-related tasks.

We utilize the publicly available versions of each model from the Hugging Face library [7]. The specific model versions used in our experiments are listed in Table 6.
Table 6.: Model versions used from the Hugging Face library

<table>
<thead>
<tr>
<th>Model</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>bert-base-uncased</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>roberta-base</td>
</tr>
<tr>
<td>ALBERT</td>
<td>albert-base-v2</td>
</tr>
<tr>
<td>DeBERTa</td>
<td>microsoft/deberta-v3-base</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>microsoft/codebert-base</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td>microsoft/graphcodebert-base</td>
</tr>
</tbody>
</table>

4.2.2 Classification Setup

As our dataset is multi-label, we employ one-vs-all settings for all models. A one-vs.-all solution consists of N (here, N = 6) separate binary classifiers, each answering a separate classification question during training [15]. We divide the datasets into stratified train (80%) and test (20%) splits based on emotions using random sampling [194]. For all 6 models, we will use the same train and test sets.

4.3 Experiments

4.3.1 RQ1: Large Language Models as Emotion Classifier

RQ1: How accurately can Large Language Models classify emotions in software engineering texts?

The results for emotion analysis on GitHub dataset, utilizing the LLMs are presented in Table 7. The table includes F1-scores for individual emotion classes and micro-averaged F1-scores. From the table, it is evident that all models demonstrate strong performance compared to the results we observed in Chapter 3 (Table 3 and Table 5).

DeBERTa attains the highest micro-averaged F1-score of 0.610 (which is 27.08% higher than the best result we observed in table 5), followed by RoBERTa. ALBERT performed worse of all models with a micro-averaged F1-score of 0.538 (which is still
12.08% higher compared to the best result in Table 5. The two SE-specific models perform similar. For individual emotions, Deberta performed best for Anger (F1-score of 0.578), Joy (0.605) and Sadness (0.642), ALBERT for Love (0.753), CodeBERT for Fear (0.548) and RoBERTa for Surprise (0.673).

**Error Analysis:** To identify model mistakes, we conduct an error analysis. We focus exclusively on the GitHub benchmark for error analysis. To understand where the LLMs commonly make mistakes, we examine 67 cases where all six models agreed, yet the ground truth differed (9 false positives, 58 false negatives), distributed across emotions. For assessment, we employ Novielli et al.’s [184] error categorization, using a thematic approach as described in Chapter 3.

Echoing the result’s of our previous chapter, the prevalent categories are ‘General Error’ and ‘Implicit Sentiment Polarity.’ General errors, occurring 29/67 times, manifest when models misinterpret or struggle to comprehend lexical cues conveying emotions. For instance, the text “Nice, this is more slick” is annotated as Joy. Another example, “I’m actually surprised this didn’t get flagged by the analyzer...,” annotated as Surprise, remains mispredicted by all models. The majority of Surprise (10/15), Joy (8/14), and Love (4/5) errors fall into the general errors category.

Implicit sentiment polarity errors occur 18/67 times. An example - “Patiently waiting for any updates. [...].” - is annotated as Sadness, with none of the models predicting it correctly. This category is particularly noticeable in Joy (6/14) and Sadness (5/11).

The third major error category is ‘Figurative Language’ (9/67), which occurs when users use humor, irony, sarcasm, metaphors, etc to convey emotion. This category is noticeable among negative emotions (Anger 5/13 and Fear 3/9). For example, the following utterance “[...] I understand what you mean by “takeover” however it doesn’t hurt to be a little more explicit.” - annotated as Anger but none of the models
Table 7.: Evaluation of LLMs on the GitHub Dataset (F1-score).

<table>
<thead>
<tr>
<th>Model</th>
<th>Anger</th>
<th>Love</th>
<th>Fear</th>
<th>Joy</th>
<th>Sadness</th>
<th>Surprise</th>
<th>Micro Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>0.506</td>
<td>0.712</td>
<td>0.536</td>
<td>0.579</td>
<td>0.636</td>
<td>0.594</td>
<td>0.588</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.525</td>
<td>0.683</td>
<td>0.492</td>
<td>0.500</td>
<td>0.613</td>
<td><strong>0.673</strong></td>
<td>0.592</td>
</tr>
<tr>
<td>ALBERT</td>
<td>0.446</td>
<td>0.753</td>
<td>0.357</td>
<td>0.447</td>
<td>0.631</td>
<td>0.602</td>
<td>0.538</td>
</tr>
<tr>
<td>DeBERTa</td>
<td>0.578</td>
<td>0.736</td>
<td>0.476</td>
<td><strong>0.605</strong></td>
<td><strong>0.642</strong></td>
<td>0.611</td>
<td><strong>0.610</strong></td>
</tr>
<tr>
<td>CodeBERT</td>
<td>0.446</td>
<td>0.653</td>
<td><strong>0.548</strong></td>
<td>0.518</td>
<td>0.591</td>
<td>0.574</td>
<td>0.545</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td>0.476</td>
<td>0.632</td>
<td>0.507</td>
<td>0.552</td>
<td>0.551</td>
<td>0.578</td>
<td>0.549</td>
</tr>
</tbody>
</table>

were able to detect it correctly.

Of note, in 13/67 cases, the utterances contain emojis which contribute in expressing emotions. The models possibly fail to capture them. For instance, “And yes, there should be tests 🎉🎉” - annotated as Fear, none of the models predict it accurately.

**RQ1 Takeaway**: The transformer-based LLMs outperform the state-of-the-art models. DeBERTa and RoBERTa emerge as top performers. However, they still fail to capture the nuances of textual features in some cases.

### 4.3.2 RQ2: Integrating Polarity Features in LLMs

**RQ1: Can integrating polarity features in the training improve Large Language Models’ ability for emotion classification?**

In open-domain research, leveraging word polarity has proven effective for sentiment analysis [132, 27, 120]. We aim to explore its applicability to emotion classification within the SE domain.

#### 4.3.2.1 Procedure

To integrate polarity features, we enhance LLMs through token-level attention, focusing on tokens associated with polarity words.
Initially, we employ natural language processing techniques, utilizing the NLTK and SentiWordNet to extract word polarity. This involves a series of steps such as tokenization, part-of-speech tagging, and identification of words with discernible sentiment, resulting in a concise list of polarity words for each utterance.

Subsequently, the model architecture is fine-tuned at the attention level, with a focus on polarity words. Attention weights are adjusted to assign greater significance to tokens linked with polarity words. This involves a strategic blending of attention weights related to the primary input text and those corresponding to polarity words. The adjustment ensures a heightened emphasis on the embeddings of polarity words, achieved by multiplying attention weights for primary input text’s hidden states by 0.75 and those for polarity words’ hidden states by 0.25, followed by concatenation. This modification enhances the model’s sensitivity to sentiment-carrying terms during training, allowing for improved discernment of subtle variations in sentiment expression. Refer to Figure 7 for a visual representation of the token-level attention adjustment procedure.

4.3.2.2 Results and Discussion

The results of GitHub dataset’s emotion classification using polarity-enhanced LLMs are presented in Table 8 along with the percentage difference in performance against baseline LLMs. The findings show that polarity-enhanced LLMs outperform baseline LLMs. The improvement on LLM baseline ranges from 1.86% to 9.16% on micro-averaged F1-score. The most improved LLM is CodeBERT in micro-averaged F1-score. Enhancement is observed consistently across individual emotions, with Joy, Fear and Sadness improving for all six models, Anger and Love for four models, and Surprise for four models. Noteworthy improvements include a 26.61% enhancement in Anger by CodeBERT, a 15.23% improvement in Love by RoBERTa, a 25.37%
Fig. 7.: Fine-tuning procedure using token-level attention adjustment of polarity words.

improvement in Fear by ALBERT, a 31.19% improvement in Joy by ALBERT, an 8.47% improvement in Sadness by GraphCodeBERT, and a 11.54% improvement in Surprise by CodeBERT. The average LLM improvement in GitHub is similar to the general domain sentiment analysis results [132, 27].

Error Analysis: Similar to RQ1, we look into GitHub dataset for this case as well. In RQ1 error analysis, we observe that in 67 cases all models predict incorrectly. Out of those 67 cases, we find that 40 cases still produce erroneous results. However, in rest 27 cases, at least one model predict correctly. The most prominent resolved error categories are: 13/29 (44.82%) general errors, 9/18 (50%) implicit sentiment polarity, and 2/3 politeness. For example, “i’m actually surprised this didn’t get flagged by the
Table 8: Evaluation of Polarity-enhanced LLMs on the GitHub Dataset (F1-score metric, P=Polarity).

<table>
<thead>
<tr>
<th>Model</th>
<th>Anger</th>
<th>Love</th>
<th>Fear</th>
<th>Joy</th>
<th>Sadness</th>
<th>Surprise</th>
<th>Micro Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT+P</td>
<td>0.484</td>
<td>0.733</td>
<td><strong>0.583</strong></td>
<td><strong>0.629</strong></td>
<td>0.636</td>
<td><strong>0.661</strong></td>
<td>0.619</td>
</tr>
<tr>
<td>BERT (+/-)</td>
<td>-4.35%</td>
<td>+2.95%</td>
<td>+8.77%</td>
<td>+8.64%</td>
<td>+0.0%</td>
<td>+11.28%</td>
<td>+5.27%</td>
</tr>
<tr>
<td>RoBERTa+P</td>
<td>0.475</td>
<td><strong>0.787</strong></td>
<td>0.538</td>
<td>0.583</td>
<td>0.654</td>
<td>0.598</td>
<td>0.603</td>
</tr>
<tr>
<td>RoBERTa (+/-)</td>
<td>-9.52%</td>
<td>+15.23%</td>
<td>+9.35%</td>
<td>+16.60%</td>
<td>+6.69%</td>
<td>-11.14%</td>
<td>+1.86%</td>
</tr>
<tr>
<td>ALBERT+P</td>
<td>0.471</td>
<td>0.744</td>
<td>0.448</td>
<td>0.587</td>
<td><strong>0.674</strong></td>
<td>0.561</td>
<td>0.580</td>
</tr>
<tr>
<td>ALBERT (+/-)</td>
<td>+5.45%</td>
<td>-1.16%</td>
<td>+25.37%</td>
<td>+31.19%</td>
<td>+6.83%</td>
<td>-6.82%</td>
<td>+7.86%</td>
</tr>
<tr>
<td>DeBERTa+P</td>
<td>0.588</td>
<td>0.680</td>
<td>0.507</td>
<td>0.633</td>
<td>0.654</td>
<td>0.623</td>
<td><strong>0.620</strong></td>
</tr>
<tr>
<td>DeBERTa (+/-)</td>
<td>+1.72%</td>
<td>-7.51%</td>
<td>+6.48%</td>
<td>+4.74%</td>
<td>+1.92%</td>
<td>+1.89%</td>
<td>+1.75%</td>
</tr>
<tr>
<td>CodeBERT+P</td>
<td><strong>0.565</strong></td>
<td>0.691</td>
<td>0.576</td>
<td>0.530</td>
<td>0.607</td>
<td>0.640</td>
<td>0.595</td>
</tr>
<tr>
<td>CodeBERT (+/-)</td>
<td>+26.61%</td>
<td>+5.80%</td>
<td>+5.08%</td>
<td>+2.36%</td>
<td>+2.68%</td>
<td>+11.54%</td>
<td><strong>+9.16%</strong></td>
</tr>
<tr>
<td>GraphCodeBERT+P</td>
<td>0.514</td>
<td>0.654</td>
<td>0.551</td>
<td>0.570</td>
<td>0.598</td>
<td>0.521</td>
<td>0.563</td>
</tr>
<tr>
<td>GraphCodeBERT (+/-)</td>
<td>+7.84%</td>
<td>+3.58%</td>
<td>+8.70%</td>
<td>+3.19%</td>
<td>+8.47%</td>
<td>-9.86%</td>
<td>+2.52%</td>
</tr>
</tbody>
</table>

*analyzer...*” - this utterance is now correctly predicted by CodeBERT.

The least improved error category is ‘Pragmatics’. Pragmatics is the error category when the classifiers deal with context information. That is often human annotators consider external facts for annotation. For example, consider the following utterance, “[…/ This change makes it the same as the line above and I don’t see any reason to have two lines that are showing the same thing.” - is annotated as Anger as the commentator is annoyed/disagreed with the change in code. 6/7 (85.71%) Pragmatics related errors remains unresolved. Another least improved error category is ‘Figurative Language’ - 6/9 (66%) utterances predictions remained unchanged. For example, this following utterance, “[…/ We need to add this test coverage. It’s just not ‘urgent’. 😊” - which expresses sarcasm and annotated as Anger. The models predict it incorrectly.

We observe that there still remains a considerable amount of utterances (10/13) that are misclassified contain emojis. For example, the previously mentioned utterance, “And yes, there should be tests 😞 😞 😞” - all models still predict incorrectly.
**RQ2 Takeaway:** Polarity-enhanced LLMs consistently outperform the baseline LLMs. Notable improvements are seen in micro-averaged F1-scores, along with substantial enhancements for individual metric across various models. While the findings indicate that integrating sentiment polarity improves LLM performance, it do not always help to capture context-dependent emotions.

### 4.4 Chapter Contributions and Summary

This chapter presents a comprehensive comparative analysis of state-of-the-art Pre-trained LLMs for emotion classification in software engineering texts. By evaluating six LLMs on a carefully curated GitHub dataset, we demonstrate their effectiveness in capturing emotional nuances in technical contexts.

Our findings show that LLMs, particularly DeBERTa and RoBERTa, significantly outperform existing software engineering-specific emotion detection tools, achieving improvements of up to 27.08% in micro-averaged F1-score. The error analysis reveals common challenges, such as general comprehension errors, implicit sentiment polarity, figurative language, and contextual pragmatics.

Furthermore, we propose a novel approach to integrate polarity features into the training process of LLMs, leveraging token-level attention adjustment. The polarity-enhanced LLMs consistently outperform both the baseline LLMs and existing tools, with improvements ranging from 1.86% to 9.16% in micro-averaged F1-score. This demonstrates the potential of incorporating sentiment information to enhance the models’ emotion classification capabilities.

However, our study also highlights the persistent challenges in handling figurative language and pragmatics-related errors, emphasizing the need for further research to address these complex linguistic phenomena in software engineering texts.

The contributions of this chapter are threefold:
1. We provide a comprehensive evaluation of state-of-the-art LLMs for emotion classification in software engineering texts, establishing a robust benchmark and identifying their strengths and limitations.

2. We propose a novel approach to integrate polarity features into the training process of LLMs, demonstrating significant improvements in emotion classification performance.

3. We conduct an in-depth error analysis, shedding light on the common challenges and future research directions for advancing emotion classification in software engineering.

In conclusion, this study underscores the potential of LLMs in advancing affective analysis in software engineering, paving the way for more empathetic and context-aware tools to support software development processes. The insights gained from this research can guide future efforts in refining LLMs and developing more advanced techniques to capture the nuances of emotions expressed in technical contexts.
CHAPTER 5

UNDERSTANDING FIGURATIVE LANGUAGE IN OSS COMMUNICATION

5.1 Background

Figurative language is the use of words or phrases in a way that deviates from their literal meaning, aiming to evoke specific concepts or imagery within one’s imagination. Figurative language consists of different types, such as metaphors, which use comparisons to describe something differently (e.g., “the road ahead is a long and winding journey”); idioms, which are common phrases that have alternate meanings (e.g., ‘to beat around the bush’); similes, which use ‘like’ or ‘as’ to compare two things (e.g., ‘as light as a feather’); and personification, which gives human qualities to objects or animals (e.g., ‘the leaves danced in the wind’).

Within the software engineering (SE) community, professionals often employ various distinctive figurative expressions that are not commonly used in everyday discourse. For instance, developers utilize the metaphorical term ‘anti-pattern’ to communicate the idea of a recurring problem that should be avoided. Idioms, another frequently employed form of figurative expression, play a crucial role in SE communication by succinctly and colloquially conveying common ideas or concepts. An example of this is when developers describe poorly written code as ‘spaghetti code’, implying that it is convoluted and challenging to comprehend.

Just as humans use phrases like ‘boil’ with anger or ‘a breath of fresh air’ for


62
relief, developers might say ‘a thorn in my side’ to express persistent annoyance or difficulty with an API or a feature. Use of pejorative terms like ‘garbage code’, ‘Frankencode’ can be indicators of severe negative emotions, leading to toxic discussions. Therefore, understanding the use of figurative language in software development discourse can help detect the use of offensive language and provide valuable insights into the overall health of a software project. Developers also use figurative expressions to indicate the impact and severity of a bug. For instance, while expressions like ‘a ticking time bomb’ suggests significant future problems, ‘showstopper’ and ‘critical roadblock’ emphasize the urgency of addressing the bug at the present. A recent blog article noted how an improved understanding of software engineering metaphors would mitigate the risks of misinterpretations, fostering more precise and effective communication within the software development community. While a recent study showed figurative language like Humor has positive effect on developer engagement. We found in chapter as well as recent research studies have also highlighted that flaws in SE emotion and sentiment detection tools stem from the use of figurative language. On a Stack Overflow and GitHub dataset, Novielli et al. found that 9% of the errors in sentiment analysis were due to figurative language, noting that it poses an open challenge for sentiment detection in software engineering.

---

2 https://github.com/chipsalliance/chisel/pull/3352#issuecomment-1593479230
3 https://github.com/adamit24/countdownClass/issues/1
4 https://github.com/drupal-code-builder/drupal-code-builder/issues/165
5 https://github.com/godotengine/godot/issues/77480#issue-1726201628
6 https://github.com/conwid/VSCleanBin/issues/2#issuecomment-571094076
7 https://github.com/canjs/canjs/pull/3286
Despite its potential impact on the performance of automatic tools focused on understanding SE text, there have been very limited studies on analyzing figurative language in SE, e.g., there have been some studies on SE synonyms [136, 198] and programming language-specific idioms [164]. In this chapter, we aim to go beyond the synonyms and explore the broader landscape of figurative language in SE. We aim to ‘shed light on’ or analyze the use of figurative language (specifically, metaphors and idioms) in SE communication channels and contribute to the understanding of how recently proposed language models that target software-related text can be made to recognize figurative expressions.

LLMs, such as BERT [170] and RoBERTa [145], have recently demonstrated state-of-the-art results on a variety of software engineering tasks, e.g., code completion, code review, bug localization, sentiment analysis, toxicity detection [34, 70, 179, 19, 133, 129] as well as emotion detection as we observed in previous chapter. While LLMs are not explicitly designed to detect figurative languages like metaphors and idioms, they can acquire this ability through training on large datasets such as Wikipedia and Stack Overflow [130, 36, 30, 92]. This capability is particularly beneficial in the software engineering context, as it enables a more nuanced and accurate analysis of developer communications. Without this ability, an LLM may misinterpret or misclassify text, leading to erroneous results. For instance, if an LLM cannot recognize the idiom ‘edge case’, it may interpret the phrase literally and erroneously categorize the text as being related to a specific type of physical boundary instead of grasping its figurative meaning of a rare or unusual scenario.

Through this chapter, we will examine the relevance of figurative language in GitHub communication channels, the ability of LLMs to detect figurative language in the SE context, and the impact of figurative language on affect analysis and bug report priority detection. By gaining a deeper understanding of the role and effects of
figurative language in SE, we aim to contribute to the development of more effective and accurate NLP-based systems for SE tasks, specifically in automated recognition of developer emotions and incivility on GitHub, and bug report priority detection.

We focus on answering the following three research questions:

**RQ1:** *How well can existing LLMs interpret figurative language (i.e., metaphors and idioms) used in software engineering?*

To answer this RQ, we collect a set of 2000 sentences containing figurative language and create *rephrased* sentences, i.e., sentences with similar meanings but without figurative expressions. We also create *altered* sentences that share as many words as the original sentence but convey different meanings, e.g., using metaphors in their literal sense or using idioms in a different context other than software engineering. This procedure of creating, so-called, entailed and non-entailed text from premise text has been widely used in NLP [58, 227, 105]. Using this data triple of original, rephrased, and altered meaning sentences, we investigate whether LLMs can recognize the semantics of figurative sentences by computing how often the models identified the semantic dissimilarity of the rephrased sentence with the altered sentence. Our results suggest that LLMs have a limited ability to interpret figurative language, with higher performance for general figurative expressions than software engineering-specific ones.

**RQ2:** *Can the performance of software engineering-specific affective analysis be improved by a better insight into figurative language?*

Affect expressions are the means to convey emotions, feelings, and attitudes to others [219]. For some time now, researchers have been exploring automatic affect analysis, which encompasses tasks such as emotion analysis, sentiment analysis, and incivility analysis. To answer RQ2, we fine-tune several LLMs using contrastive learning [121] with our dataset of figurative language in order to improve their ability to
interpret figurative language. We then compare the performance of the fine-tuned LLMs to the original models in two affect datasets: the emotion dataset curated from GitHub by us, and an incivility dataset curated from GitHub. Our results indicate that fine-tuned LLMs perform better in both cases.

**RQ3: Can a better understanding of figurative language enhance software engineering automation where affect plays a role?**

A number of research tasks in SE indirectly involve affective natural language text, e.g., app review analysis, opinion mining [67, 48]. Specifically, in this RQ we investigate how a better understanding of figurative language can impact bug report priority detection, which is a significant area of interest in open-source software research [23, 235, 189, 156]. Umer et al. observed that emotions influence bug report priority detection [189]. To address this problem, recently researchers have employed Language Models (LLMs) [23]. In this study, we explore LLMs fine-tuned with contrastive learning using our figurative language dataset, similar to the approach in RQ2, and conducted experiments on the publicly available Bugzilla dataset. Our results indicate that fine-tuning with our figurative language dataset improves bug report priority detection.

### 5.2 Dataset

To conduct our experiments, we curate a dataset of developer communications containing figurative language. Towards that goal, we first collect data from GitHub issues and pull requests and identify the occurrences of idioms and metaphors. To inquire whether language models understand figurative language, we manually rephrase the original sentences containing figurative language to generate: 1) sentences that
are similar in meaning to the original but do not contain idioms or metaphors; and 2) sentences that contain similar words as the original sentences but are semantically dissimilar, i.e., have a different meaning. In this section, we detail each step involved in constructing our dataset.

5.2.1 Data Collection

We selected nine popular GitHub repositories, each with a minimum of 50k stars: skylot/jadx, laravel/laravel, microsoft/PowerToys, rails/rails, redis/redis, facebook/react, tensorflow/tensorflow, huggingface/transformers, and microsoft/vscode. We collected 10k comments from each repository (5k PR comments and 5k issue comments) between February 2022 and May 2023. We split the comments into sentences using NLTK and filtered out sentences with fewer than 5 words, resulting in a total of 202k sentences.

One of our study’s end goals is to examine figurative language’s impact on affective expressions in a software engineering context. Previous research has shown that most comments on GitHub are neutral, lacking any detectable emotions or sentiments. Therefore, we excluded neutral sentences by using a software engineering-specific sentiment analysis tool.

10https://github.com/skylot/jadx
11https://github.com/laravel/laravel
12https://github.com/microsoft/PowerToys
13https://github.com/rails/rails
14https://github.com/redis/redis
15https://github.com/facebook/react
16https://github.com/tensorflow/tensorflow
17https://github.com/huggingface/transformers
18https://github.com/microsoft/vscode
19https://www.nltk.org/
In addition, to avoid including sentences that do not contain any figurative expressions, we applied a popular metaphor detection [130] and an idiom detection tool [36] to identify candidate metaphors and idioms in each sentence. This model-in-the-loop approach is popular in Natural Language Inference (NLI) research, e.g., figurative language interpretation [125, 32], as it maximizes the value of annotation effort, which requires tedious human labor. We discarded sentences that do not contain any candidate idioms or metaphors. We randomly selected 1000 sentences containing metaphors from the remaining sentences. We also randomly chose 1000 sentences containing idioms (different from the metaphor set). This process resulted in a dataset of 2000 sentences.

5.2.2 Data Annotation

First, we recruited four annotators (two graduate students and two senior undergraduate students) who were each given 500 sentences to annotate (250 with metaphors and 250 with idioms). Due to the nature of the task and difficulties with crowd-sourcing [105], we opted for a small number of annotators that are native speakers/professionally fluent in English with a strong computer science background. Along with the 2000 sentences in total, the annotators were provided with a set of candidate figurative expressions marked by the above-mentioned tools. We instructed them to: 1) verify the candidates as metaphors or idioms and judge whether each metaphor or idiom is specific to software engineering or general purpose; and 2) create rephrased sentences from the original. We also held a short training session in which we reviewed the annotation process for a few representative examples with each annotator. Below, we describe these data annotation steps in detail (see also Figure 8).
5.2.2.1 Verifying Figurative Expressions

For verifying metaphors and idioms, we followed best practices from existing literature. More specifically, to verify the metaphors we asked the annotators to carefully read the Metaphor Identification Procedure (MIP) guideline by the Pragglejaz Group [282]. The MIP guideline is a well-known procedure for identifying metaphors. Based on the guideline, the annotators marked the correct metaphoric expressions from the candidate set. For example, the annotators confirmed that ‘nasty bug’ is a valid metaphor for a difficult fault in the sentence, “Otherwise, this could give us a nasty bug.”

We noted in the annotation instructions that most metaphors are conventional, i.e., metaphors that are often used in everyday language [171]. For example, in the following sentence: “I see your point”, ‘see’ and ‘point’ both are metaphors [171]. Often such cases can be observed in software engineering communication. For instance, ‘pinging’ in the following sentence is a metaphor: “Hi @[USER], thanks for pinging me on this issue.” Here, ‘pinging’ is a colloquial way of saying ‘contacting someone’, while the literal meaning of ‘pinging’ comes from computer networking terminology [20]

For verifying idioms, we followed the guideline provided by Stowe et al. [58], which asked the annotators to look up idioms in popular dictionaries (such as the Oxford English Dictionary[21], the Webster Dictionary[22] and the Longman Dictionary of Contemporary English [23] and popular search engines (e.g., Google). We instructed the annotators to consider an expression as likely to be an idiom if its dictionary definition is: 1) applicable in the context; and 2) a good syntactic fit in the same

---

21 https://www.oxfordlearnersdictionaries.com/
22 https://www.merriam-webster.com/
23 https://www.ldoceonline.com/
environment. For example, in the sentence, “I will also be keeping an eye on you”, ‘keeping an eye’ is an idiom which means ‘to watch someone or something or stay informed about the person’s behavior, especially to keep someone out of trouble.’ Conversely, when the meaning of the candidate idiomatic expression is literal in the context of the sentence and the dictionary definition is not applicable, it is likely not to be an idiom. For example, in the sentence, “It was cold, so cold in the jeep that it was with difficulty that Alexei kept his eyes open”, ‘kept his eyes open’ is not an idiom. Since software-specific words have distinct meanings from conventional terms (e.g., bug, issue, error, function), we supplied annotators with established software engineering glossary terms from the FDA and Google.

Once annotators verified the candidate set, we asked them to mark whether the figurative expressions were software engineering-specific or general-purpose. The annotators identified 752 sentences with metaphors and 909 with idioms, totaling 1661 sentences. The remaining 339 sentences did not contain any figurative words. These 1661 sentences had a total of 1741 unique figurative expressions, with 445 being SE-specific and 1296 general.

5.2.2.2 Rephrasing Sentences

The process of rephrasing sentences was divided into two phases: creating semantically-equivalent rephrased sentences and constructing altered-meaning sentences. We refer to the semantically equivalent rephrased sentences as Equivalent Meaning Sentence (EMS) throughout the paper. These sentences retain the original meaning of the

---

24 https://dictionary.cambridge.org/
26 https://developers.google.com/machine-learning/glossary
Fig. 8.: Figurative language annotation procedure.

sentence, but the figurative expressions are replaced with literal terms. We refer to the altered-meaning sentences as Different Meaning Sentence (DMS). These sentences are modified so that they significantly differ in meaning from the original sentences.

a) EMS Construction: The annotators were tasked with rephrasing each sentence on their list, i.e., removing the (verified) figurative expressions while maintaining the original semantics of the sentence as much as possible. In other words, the replaced figurative expression should entail its literal counterpart. For example, in the sentence, “[USER] Thanks for your help, what you said may be a hidden bug.”, the figurative expression ‘hidden bug’ is replaced with ‘unseen error’ resulting in the EMS:
“[/USER] Thanks for your help, what you said may be an unseen error.” This approach is inspired by previous research by Stowe et al. on figurative language in NLP.\textsuperscript{58} It is worth noting that for EMS we did not employ multiple annotators to annotate the same set or calculate inter-annotator agreement as Stowe et al. found that this method does not yield significantly different quality compared to the conventional approach.\textsuperscript{58}

b) DMS Construction: Different Meaning Sentences are variations of metaphorical or idiomatic sentences that convey a different meaning than the original sentence and do not entail it.\textsuperscript{58} Two strategies were employed to construct DMS: a) using figurative expressions in a literal sense; and b) replacing the figurative expressions and their context with different words. These strategies are inspired by previous research on figurative language in natural language processing.\textsuperscript{58 99 117} We also apply a model-in-the-loop approach for DMS generation.\textsuperscript{125 32} More specifically, we first generated four candidate DMSs for each sentence, two each for each strategy using GPT-4 API (‘gpt-4’\textsuperscript{27}), and, second, we recruited human annotators to select the best-generated candidate (or to create one of their own if none is available).

Candidate DMS Generation. ChatGPT\textsuperscript{17} has shown promising results in data annotation tasks, including text generation, in some cases outperforming human crowd-workers.\textsuperscript{4 6} Following recent literature, we create two GPT-4 prompts for the two different strategies for DMS generation.\textsuperscript{9 6} The prompts were carefully devised by using the existing literature on this topic.\textsuperscript{58 99 117} The prompts for generating DMS are as follows:

\textit{Generating DMS by using the figurative language in a literal manner:}

\textsuperscript{27}https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo/
DMS Selection. Two additional annotators (one of the authors and one senior undergraduate student) were responsible for the candidate selection of the DMS. We
provided the annotators the original sentence, the figurative expressions, and the list of candidate DMSs with the following instructions: “You will be provided with 4 candidate sentences, two of which come from Type 1 and two come from Type 2. Choose the best 1 out of the 4 candidates, with a preference towards choosing from Type 1. If none of these 4 are good candidates, write None. When choosing, try to choose a sentence that has 1) similar semantic order to the original sentence, and 2) a different meaning than the original sentence.”

We instructed the annotators to write their own DMS when their selection is ‘None’. Once they completed an annotation pass over the entire dataset, the two annotators met in person in order to discuss the 310 cases where they disagreed (i.e., selected different DMS candidates or ‘None’) and resolved them in order to achieve 100% agreement. This human-in-the-loop methodology helps with the more difficult task of DMS generation, enhancing the overall quality and efficiency the process. The iterative resolution of differences ensured a high quality of annotated data.

5.3 Prevalence of SE-Specific Figurative Language

In order to understand if SE-specific figurative language appears frequently in the wild, we examine the frequency of occurrence of figurative language in a large sample of developer communication on GitHub. More specifically, we collected 1,000 issue comments and 1,000 pull request comments for each of the top 100 repositories by star count on GitHub, i.e., a total of 200k comments. We analyzed comments made from September 1, 2022, to January 1, 2023, spanning 4 months, and excluded repositories with fewer than 1,000 issue and pull request comments during this time. The collected 200k comments were split into a total of 484k sentences using NLTK28.

28https://www.nltk.org/
Leveraging our annotated dataset consisting of 1741 unique figurative expressions (445 SE-specific and 1296 general), we searched for matches in the set of 484k sentences after applying standard NLP pre-processing (removing punctuation and non-alphabet characters, and lemmatization using SpaCy) since some of the figurative expressions have different spelling variations (e.g., ‘root cause’, ‘root-cause’, and ‘root causes’). To ensure that the matches were not spuriously identifying figurative language (due to polysemy), we also executed the metaphor and idiom detection tools [130, 36], the same ones that we use for candidates generation, selecting only the matches that were also confirmed by one of these tools.

Of the examined 484k sentences, the 445 SE-specific figurative expressions that annotators identified occurred in 44k sentences (9%), while the 1296 general figurative expressions occurred in 107k sentences (22%). Some sentences (2.67%) had both SE-specific and general figurative expressions.

The distribution of general and SE-specific figurative expressions in different GitHub repositories is shown in Figure 9a. SE-specific figurative language does appear in non-trivial amounts in most repositories we examined (i.e., in between 3.69% and 16.08% of sentences), but much less often than general figurative expressions, which occurred in between 13.2% and 38.62% of sentences. In Figure 9b, we present the frequency of SE-specific figurative expressions identified within our corpus of 200k GitHub comments. Of the 445 SE-specific figurative expressions we investigated, 324 (72.8%) appear no more than 10 times, as indicated by the red dotted line, suggesting that most such expressions are infrequent. Among these, 193 expressions are absent from our dataset. This absence can be attributed to two main factors: a) expressions that are specific to particular projects (e.g., ‘ghost highlight’ and ‘ghost monitor’ in UI-related projects); and b) unique expressions used to describe highly specific scenarios (e.g., ‘dead fork,’ ‘magic code’).
Fig. 9.: Distribution of figurative language occurrence in GitHub sentences (200k GitHub comments, 484k sentences).

Note that our study provides only a lower bound, as it matches using an incomplete set of figurative language expressions. Therefore, the likely presence of figurative language is even higher than we report. This exploratory study highlights the importance of understanding figurative language in the SE context, as it can provide insight into the daily communications of developers.

5.4 Experiments

Using the assembled dataset, we created specific experiments for each of our three research questions. In this section, we describe the experiments and discuss the corresponding results.

5.4.1 RQ1: LLM’s figurative language interpretation capability

**RQ1: How well can existing LLMs interpret figurative language (i.e., metaphors and idioms) used in software engineering?**

In order to determine how well popular LLMs understand metaphors and idioms, we examine whether they can understand the semantic relationship between the original sentence, the equivalent sentence (i.e., EMS), and the different-meaning sentence (i.e., DMS). The task of differentiating EMSs from DMSs of the original sentences
Fig. 10.: RQ1 evaluation pipeline.
can be thought of as Recognizing Textual Entailment (RTE) [66]. RTE involves determining whether a statement, called the hypothesis, can be inferred from a given text, called the premise. In our context, the premise is the original sentence, and the hypotheses are the EMS and DMS. We evaluate whether an LLM can infer the EMS from the original sentence, and if it is a DMS, the model should not deduce it.

One way to approximate the RTE task is to posit that the model should recognize the original sentence as semantically closer to the EMS than to the DMS. By comparing the embedding vectors of the original sentence and the sentence in question, we can measure whether the two sentences are similar or dissimilar and, therefore, whether the most similar sentence is an EMS or a DMS.

5.4.1.1 Compared LLMs

We compare four LLMs; three general domain models – BERT [170], RoBERTa [145], and ALBERT [142], which are popular in NLP and SE tasks [65, 104, 14]. BERT [170] is a transformer-based model pre-trained on extensive text data from Wikipedia and BooksCorpus. RoBERTa [145] is an improved version of BERT, while ALBERT [142] enhances efficiency through parameter reduction techniques. Additionally, we evaluate a popular SE-specific LLM – CodeBERT [113], which is pre-trained on natural language – programming language pairs. We use bert-base-uncased,roberta-base,albert-base-v2 and microsoft/codebert-base available from Hugging Face [7].

5.4.1.2 Procedure

We generate embedding vectors using each of the LLMs for each pair of sentences, i.e., <original sentence, EMS> and <original sentence, DMS>. Next, we compute the similarity between the vectors in each pair and then compare the resulting similarity scores [152]. We perform standard software engineering text-specific preprocessing
operations such as URL removal, username removal, stack trace removal, etc.  

Since the LLMs in our cohort produce word-level embedding vectors, there are several possibilities for aggregating these into sentence-level embedding vectors. Reimers et al. 152 noted that mean pooling (the mean of all per-word output vectors generated by the LLM) is one of the best strategies. While we opt for mean pooling when generating each sentence’s embedding vector, we also note that this strategy is still error-prone due to the anisotropy problem, i.e., the difference in the scale of the embedding vectors 139. For this reason, we apply the normalization proposed by Yan et al. 60, which is based on Singular Value Transformation (SVT). SVT uses singular value decomposition and a threshold using the soft-exponential function by Godfrey et al. 229

Following normalization, we compute vector pair similarity with the cosine similarity metric. Cosine similarity measures vector alignment, with values from 1 (identical) to 0 (orthogonal) to -1 (opposite) 249. Then, we compare the two similarities in order to determine if the \(<\text{original sentence}, \text{EMS}\rangle\) similarity ($\text{Sim}_{\text{EMS}}$) is higher than the \(<\text{original sentence}, \text{DMS}\rangle\) similarity ($\text{Sim}_{\text{DMS}}$). Figure 10 summarises the entire procedure. To evaluate the RQ, we compute the percentage of instances where $\text{Sim}_{\text{EMS}}$ is greater than $\text{Sim}_{\text{DMS}}$. We examine three sentence categories: a) those containing SE-specific figurative language only (n=371); b) those containing general figurative language only (n=1179); and c) overall, containing either (a) and (b), or both (n=1661). For each model and category, we measure the statistical significance of the difference between the two cosine similarities using the one-tailed Wilcoxon signed-rank test (i.e., testing if $\text{Sim}_{\text{EMS}} > \text{Sim}_{\text{DMS}}$ with statistical significance). We apply the Benjamini-Hochberg correction to control the false discovery rate. A small $p$-value (e.g., $p$-value $< 0.05$) indicates that the difference is unlikely to be due to chance and that there is a statistically significant difference between the two sam-
ples. We also compute the effect size, which measures the magnitude of the difference between the two samples, using Cliff’s Delta ($\delta$) \[302\], where $|\delta| > 0.147$, $0.33$, and $0.474$ indicate small, medium, and large effects respectively.

5.4.1.3 Results and Discussion

Table 9 shows the SE-specific, General, and Overall (i.e., combined) results. The higher the percentage of $<\text{original sentence, EMS}>$ pairs with larger cosine similarity, i.e., $\text{Sim}_{EMS} > \text{Sim}_{DMS}$, the better the model is at recognizing figurative language. The results table shows that the BERT and RoBERTa models have the highest percentage of correctly understood pairs for all categories. BERT, RoBERTa, and ALBERT models correctly recognize 84.51%, 83.70%, and 81.79% of sentences containing SE-specific figurative expressions, 87.40%, 85.21%, and 85.80% of General figurative expressions, and 86.57%, 84.95%, and 85.0% of the Overall figurative expressions, respectively. In the case of CodeBERT, which exhibits the poorest results out of all models, there is no significant difference between SE-specific and General results (77.99% and 79.63% respectively). This is likely because the model is pre-trained with programming-specific data enabling it recognize some software engineering figurative language terms. However, it also likely loses the ability to capture General figurative language, which is present in the other LLMs. From this study, we observe that all of the models can understand figurative language to a reasonable degree (i.e., ranging between 77.99% to 87.40%).

This is also evident from $p$-value and Cliff’s $|\delta|$. In each case, the statistically significant is with a $p$-value $< 0.01$ and a $|\delta|$ greater than 0.474 is considered a large effect for all models. The p-value less than 0.01 indicates that the observed difference in similarity between the two groups is highly unlikely to be due to chance and we conclude that there is a statistically significant difference in similarity between
Table 9.: Percent of EMS with a higher similarity to the original sentence than corresponding DMS ($\text{Sim}_{EMS} > \text{Sim}_{DMS}$).

<table>
<thead>
<tr>
<th>Model</th>
<th>SE-specific $\text{Sim}<em>{EMS} &gt; \text{Sim}</em>{DMS}$</th>
<th>p-value</th>
<th>Cliff’s $\delta$</th>
<th>General $\text{Sim}<em>{EMS} &gt; \text{Sim}</em>{DMS}$</th>
<th>p-value</th>
<th>Cliff’s $\delta$</th>
<th>Overall $\text{Sim}<em>{EMS} &gt; \text{Sim}</em>{DMS}$</th>
<th>p-value</th>
<th>Cliff’s $\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>84.51%</td>
<td>$p &lt; 0.01$</td>
<td>0.629</td>
<td>87.40%</td>
<td>$p &lt; 0.01$</td>
<td>0.638</td>
<td>86.57%</td>
<td>$p &lt; 0.01$</td>
<td>0.637</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>83.70%</td>
<td>$p &lt; 0.01$</td>
<td>0.648</td>
<td>85.21%</td>
<td>$p &lt; 0.01$</td>
<td>0.620</td>
<td>84.95%</td>
<td>$p &lt; 0.01$</td>
<td>0.632</td>
</tr>
<tr>
<td>ALBERT</td>
<td>81.79%</td>
<td>$p &lt; 0.01$</td>
<td>0.610</td>
<td>86.80%</td>
<td>$p &lt; 0.01$</td>
<td>0.598</td>
<td>85.00%</td>
<td>$p &lt; 0.01$</td>
<td>0.605</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>77.99%</td>
<td>$p &lt; 0.01$</td>
<td>0.498</td>
<td>79.63%</td>
<td>$p &lt; 0.01$</td>
<td>0.493</td>
<td>79.11%</td>
<td>$p &lt; 0.01$</td>
<td>0.495</td>
</tr>
</tbody>
</table>

the two sets of sentence pairs. The large $|\delta|$ indicates that the similarity between the two groups (i.e., the sentence pairs in group $\text{Sim}_{EMS}$ compared to those in group $\text{Sim}_{DMS}$ is substantial. The cosine similarity values in Group $\text{Sim}_{EMS}$ are consistently higher than those in Group $\text{Sim}_{DMS}$, showing that the sentences in Group $\text{Sim}_{EMS}$ are more similar to each other compared to those in Group $\text{Sim}_{DMS}$. Together, these results suggest that the two groups of sentence pairs exhibit a notable and meaningful difference in their similarity scores, and this difference is not likely to be due to random chance.

However, the models still fail to recognize between 18.21% and 12.60% of figurative language instances. It is highly likely that if we can improve the models’ understanding of figurative language in such cases, they will function better in their use cases.

5.4.2 RQ2: Performance Improvement in Affective Analysis

*RQ2: Can the performance of software engineering-specific affective analysis be improved by a better insight into figurative language?*

Affect analysis involves identifying and evaluating human emotions, feelings, and sentiments expressed through written communication. Kovecses et al. noted that figurative expressions are vital in expressing emotions [288], while Mohammad et
al. [218] observed that metaphorical words tend to contain significantly more emotions than the literal sense of the same words. In software engineering, affect is often related to the software and its development process, including the emotional states of software developers, productivity, and burnout [230, 128, 202]. Thus, identifying and understanding affect is crucial for improving software quality and developer productivity. However, several studies have shown challenges in building reliable tools and datasets for mining emotions and opinions in the SE domain [184, 48]. We found in chapter 3 that using figurative language in SE-related text can hinder the accurate identification of emotions, partly motivating this RQ2 investigation.

The use of LLMs has become a widely adopted method for identifying and classifying affective expressions in written text [155, 79, 69]. LLMs are usually fine-tuned to address specific affect analysis tasks, such as recognizing sentiment or emotions. Recently, one of the most effective ways to fine-tune an LLM is by applying a contrastive learning approach. This approach uses sets of similar and dissimilar instances to train the model to understand the similar instances and differentiate them from the dissimilar ones [50]. To answer this RQ, we leverage contrastive learning as the means for LLMs to better capture the meaning and nuances in figurative language present in GitHub comments.

5.4.2.1 Compared models

Similar to RQ1, we assess the ability of the same four LLMs — BERT [170], RoBERTa [145], ALBERT [142], CodeBERT [113] — with the same model versions as RQ1 from Hugging Face. Previous research shows that BERT, RoBERTa and ALBERT work well in SE affect analysis [133, 65, 23].
5.4.2.2 Contrastive learning

Contrastive learning is a recently proposed machine learning technique that involves training a model to distinguish between two or more distinct data points by contrasting their differences \[121\] \[108\]. The steps for applying this approach to fine-tune LLMs for understanding figurative language elements in the text can be outlined as follows:

1. The LLM is presented with a triplet of anchor, positive and negative samples, which are representative of the figurative language elements to be learned.

2. The LLM processes the samples and generates output embeddings for the data triplet.

3. A loss function encourages the anchor and positive samples to be closer together and the anchor and negative samples to be further apart in the embedding space.

4. The process is repeated until the LLM has learned a satisfactory representation.

To apply contrastive learning, we use the original sentences and EMSs as anchor and positive classes and DMSs as negative classes. In other words, we created \(<\text{original sentence}, \text{EMS}, \text{DMS}>\) and \(<\text{EMS, original sentence, DMS}>\) as a pair of triplets, where the first element in each pair is anchor, the second element is positive, and the third element is negative. There is a total of 3322 such triplets of sentences in our dataset.

We use InfoNCE Loss as our loss function \[121\]. Given the embeddings of an anchor, a positive, and a negative sample denoted as \(a\), \(p\), and \(n\) respectively, the InfoNCE loss is computed as follows:

\[
\text{InfoNCE Loss}(a, p, n) = - \log \left( \frac{e^{\text{sim}(a,p)}}{e^{\text{sim}(a,p)} + e^{\text{sim}(a,n)}} \right)
\]
where \( \text{sim}(a, p) \) represents the cosine similarity between the embeddings of the anchor and positive samples, and \( \text{sim}(a, n) \) represents the cosine similarity between the embeddings of the anchor and negative samples. The InfoNCE loss maximizes the log-likelihood of anchor-positive similarity and minimizes anchor-negative similarity. We use the Adam optimizer.

Using contrastive learning, the LLM learns to create embeddings that capture the semantic similarity between the original and EMS while recognizing the semantic differences between the original and DMS. This allows the LLM to learn a representation that separates the positive and negative samples as much as possible. In this case, the LLM learns to recognize the figurative language elements.

After fine-tuning the models with contrastive learning, we assess their performance in two tasks: emotion recognition and incivility detection. We compare the performance of these fine-tuned models against baseline models that are not fine-tuned with figurative language.

### 5.4.2.3 Datasets

We apply the LLMs to two SE-affect datasets.

**Emotion Dataset.** We discussed details about curating this dataset in the chapter.

**Incivility Dataset.** Ferreira et al. [35] curated a dataset from GitHub’s heated issues for incivility detection. The dataset has three parts: comment level, issue level, and sentence level. We consider the comment-level dataset in this experiment, which has three classes: Civil, Uncivil, and Technical. We consider only Civil and Uncivil comments as we are interested in affective analysis for this RQ. The filtered dataset contains 718 comments, of which 232 (32.3%) are Civil comments, and 486 (67.7%) Uncivil comments.
5.4.2.4 Procedure and Metrics.

Using random stratified sampling for each class, we divide all two datasets into train (80%) and test (20%) sets \[194\]. For each task (i.e., incivility detection and emotion detection), we train (or fine-tune) both the LLMs’ contrastive learning and baseline versions. In other words, the contrastive learning models are fine-tuned twice, first with contrastive learning and figurative language and second with a task-specific dataset. The baselines are only fine-tuned with the task-specific dataset.

We use F1-score and micro-averaged F1-score as metrics like previous chapters.

5.4.2.5 Results and Discussion

**Emotion Classification:** Table \[10\] shows the results of the emotion classification task on GitHub emotion dataset, using BERT, RoBERTa, ALBERT, CodeBERT and their fine-tuned with figurative language counterparts (BERT-FL, RoBERTa-FL, ALBERT-FL, CodeBERT-FL). The table presents the F1-score for each emotion class, the micro-averaged F1-score, and the improvement in the F1-score achieved by the figurative language versions of the models. The results show that the use of contrastive learning with figurative language improves performance on the emotion classification task for most emotion types. For the micro-averaged F1-score, across all emotions, the figurative language versions of the models achieve an improvement of 6.60%, 6.75%, 3.63%, and 3.90% for BERT, RoBERTa, ALBERT, and CodeBERT respectively. This implies that adding figurative language to these models improves their capability to comprehend and interpret the subtleties that developers use in their communication.

In all four models, we see an increase in True Positives and a decrease in both False Negatives and False Positives. For instance, for the BERT model, across 6
Table 10.: Evaluation of LLMs finetuned with figurative language on the GitHub Emotion Dataset (F1-score).

<table>
<thead>
<tr>
<th>Model</th>
<th>Anger</th>
<th>Love</th>
<th>Fear</th>
<th>Joy</th>
<th>Sadness</th>
<th>Surprise</th>
<th>Micro Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>0.506</td>
<td>0.712</td>
<td>0.536</td>
<td>0.579</td>
<td>0.636</td>
<td>0.594</td>
<td>0.588</td>
</tr>
<tr>
<td>BERT-FL</td>
<td>0.547</td>
<td>0.709</td>
<td>0.562</td>
<td>0.608</td>
<td>0.661</td>
<td>0.632</td>
<td>0.627</td>
</tr>
<tr>
<td>+/-</td>
<td>+8.10%</td>
<td>-0.42%</td>
<td>+4.85%</td>
<td>+5.01%</td>
<td>+3.93%</td>
<td>+6.40%</td>
<td>+6.60%</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.525</td>
<td>0.683</td>
<td>0.492</td>
<td>0.500</td>
<td>0.613</td>
<td>0.673</td>
<td>0.592</td>
</tr>
<tr>
<td>RoBERTa-FL</td>
<td>0.551</td>
<td>0.733</td>
<td>0.545</td>
<td>0.667</td>
<td>0.667</td>
<td>0.617</td>
<td>0.632</td>
</tr>
<tr>
<td>+/-</td>
<td>+4.95%</td>
<td>+10.77%</td>
<td>+8.26%</td>
<td>+33.40%</td>
<td>+8.81%</td>
<td>-8.32%</td>
<td>+6.75%</td>
</tr>
<tr>
<td>ALBERT</td>
<td>0.462</td>
<td>0.658</td>
<td>0.430</td>
<td>0.487</td>
<td>0.628</td>
<td>0.564</td>
<td>0.531</td>
</tr>
<tr>
<td>ALBERT-FL</td>
<td>0.443</td>
<td>0.682</td>
<td>0.435</td>
<td>0.540</td>
<td>0.624</td>
<td>0.592</td>
<td>0.550</td>
</tr>
<tr>
<td>+/-</td>
<td>-4.11%</td>
<td>+3.52%</td>
<td>+1.15%</td>
<td>+9.81%</td>
<td>-0.64%</td>
<td>+4.73%</td>
<td>+3.63%</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>0.484</td>
<td>0.711</td>
<td>0.507</td>
<td>0.558</td>
<td>0.575</td>
<td>0.576</td>
<td>0.561</td>
</tr>
<tr>
<td>CodeBERT-FL</td>
<td>0.497</td>
<td>0.723</td>
<td>0.444</td>
<td>0.605</td>
<td>0.645</td>
<td>0.617</td>
<td>0.583</td>
</tr>
<tr>
<td>+/-</td>
<td>+2.79%</td>
<td>+1.70%</td>
<td>-14.08%</td>
<td>+7.75%</td>
<td>+10.92%</td>
<td>+6.61%</td>
<td>+3.90%</td>
</tr>
<tr>
<td>Avg. +/-</td>
<td>+2.93%</td>
<td>+3.89%</td>
<td>+0.05%</td>
<td>+13.99%</td>
<td>+5.76%</td>
<td>+2.36%</td>
<td>+5.22%</td>
</tr>
</tbody>
</table>

emotions, micro-averaged recall increases by 5.58%, and micro-averaged precision increases by 7.59%. This indicates improved precision in predictions after applying contrastive learning.

When considering the average improvement in individual emotions across all models in Table 10, we observe that ‘Joy’ has most improvement (13.99%), followed by ‘Sadness’ (5.76%) and the least improved category is ‘Fear’ (0.05%). This correlates with the frequency of occurrence of these emotions in GitHub comments, e.g., Joy is much more commonly found than Fear. As our figurative language dataset is randomly sampled, it is likely to contain figurative expressions closely related to the emotions that are more commonly observed in GitHub. This result suggests that curating a larger and more diverse set of comments that include figurative language could lead to a stronger and more balanced performance improvement.

Error Analysis of BERT-FL vs. BERT. To gain deeper insight into figurative language-based models’ predictive accuracy relative to baseline models, we perform qualitative analysis. Our focus is solely on BERT and BERT-FL models’ predictions. We ex-
amine two specific areas: 1) True Positives where BERT-FL is correct while baseline BERT is not, and 2) True Positives where baseline BERT is correct while BERT-FL is not.

Among the positive instances, BERT-FL correctly predicts 39 utterances that the baseline BERT model does not. Consider the following sentence: “Bah. Wasn’t supposed to add anything – it was a debugging leftover...”. In this case, BERT-FL correctly predicts ‘Anger’, whereas BERT misclassifies it. Here, the word ‘leftover’ is used metaphorically. Normally, ‘leftover’ refers to ‘something that remains unused or unconsumed’, particularly in the context of food. However, in the given sentence, the word is used to imply that some code or modifications were unintentionally left behind or overlooked during the debugging process. The BERT-FL model likely captures the context more effectively. Another example, “Oh nice!! I’ve seen that syntax floating around, wanting to try it for a while ‘raising-hands’” - BERT-FL correctly classifies as ‘Joy’ which the BERT baseline model misclassifies. Here, the BERT-FL is likely able to capture that ‘floating around’ is an idiom and interpret the meaning. In some instances, BERT-FL makes correct predictions by adopting a more conservative classification approach. For instance, BERT classifies the following sentence as ‘Anger’: “Please put this below line 5 (together with the other non-app imports) :pray;”. However, BERT-FL accurately predicts that it is not ‘Anger’.

On the other hand, in 27 cases, BERT-FL makes wrong predictions where BERT does not. Consider this utterance: “I have currently no clue, but I’ll have a look”, this sentence contains the idioms ‘have a clue’ and ‘have a look’. The author
Table 11.: Evaluation of LLMs finetuned with figurative language on the Incivility Dataset (F1-score).

<table>
<thead>
<tr>
<th>Model</th>
<th>Civil</th>
<th>Uncivil</th>
<th>Micro Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>0.537</td>
<td>0.814</td>
<td>0.734</td>
</tr>
<tr>
<td>BERT-FL</td>
<td><strong>0.587</strong></td>
<td><strong>0.853</strong></td>
<td><strong>0.783</strong></td>
</tr>
<tr>
<td>+/-</td>
<td>+8.54%</td>
<td>+4.84%</td>
<td>+6.67%</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.424</td>
<td>0.827</td>
<td>0.734</td>
</tr>
<tr>
<td>RoBERTa-FL</td>
<td><strong>0.535</strong></td>
<td><strong>0.847</strong></td>
<td><strong>0.769</strong></td>
</tr>
<tr>
<td>+/-</td>
<td>+20.73%</td>
<td>+2.33%</td>
<td>+4.76%</td>
</tr>
<tr>
<td>ALBERT</td>
<td>0.151</td>
<td>0.807</td>
<td>0.685</td>
</tr>
<tr>
<td>ALBERT-FL</td>
<td><strong>0.423</strong></td>
<td><strong>0.809</strong></td>
<td><strong>0.713</strong></td>
</tr>
<tr>
<td>+/-</td>
<td>+64.28%</td>
<td>+0.30%</td>
<td>+4.08%</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>0.185</td>
<td>0.810</td>
<td>0.692</td>
</tr>
<tr>
<td>CodeBERT-FL</td>
<td><strong>0.431</strong></td>
<td><strong>0.833</strong></td>
<td><strong>0.741</strong></td>
</tr>
<tr>
<td>+/-</td>
<td>+57.01%</td>
<td>+2.74%</td>
<td>+7.07%</td>
</tr>
<tr>
<td>Avg. +/-</td>
<td>+37.64%</td>
<td>+2.55%</td>
<td>+5.65%</td>
</tr>
</tbody>
</table>

of the comment likely was puzzled about some functions or errors. BERT identifies correctly as ‘Surprise’ but BERT-FL does not. Possibly, BERT-FL interpreted these idiomatic expressions more of a literal interpretation of the words. In some cases, BERT-FL just misclassifies without any involvement of any figurative expressions. For example, “I guess my concern is that it sets a precedent where somebody could see it and think that it would be fine to use in ‘core’.” This expression express concern which is annotated as ‘Fear’. This expression conveys concern, annotated as ‘Fear’. BERT identifies it correctly, but BERT-FL does not. It is possible that during the contrastive learning process, BERT-FL may lose some of the baseline BERT model’s ability to capture nuanced emotional indicators in certain sentences accurately. This suggests that while this approach improves the overall model performance but may introduce limitations or biases in some cases.

**Incivility Classification**: Table 11 presents the results of the incivility classification task on Ferreira et al.’s incivility dataset [35], using the same four large language models (BERT, RoBERTa, ALBERT, and CodeBERT) with and without the con-
contrastive learning approach. The micro-averaged F1-scores indicate that the models perform better when applying the contrastive learning approach. Overall, the BERT, RoBERTa, ALBERT, and CodeBERT models have an average improvement of 6.67%, 4.76%, 4.08%, and 7.07% respectively, when the contrastive learning approach is applied. Since the incivility dataset is small and imbalanced, the baseline models often struggle to classify the minor ‘Civil’ class, except for BERT.

We also observed a significant average improvement of 37.64% across all models in the ‘Civil’ class, compared to a modest 2.55% improvement in the ‘Uncivil’ class. This discrepancy likely arises because the figurative language dataset used for contrastive learning primarily consists of ‘Civil’ comments, which are much more common on GitHub than ‘Uncivil’ comments. Incorporating more figurative expressions from ‘Uncivil’ comments into the dataset could potentially enhance performance in this category as well.

It is important to note that the substantial improvements in identifying ‘Civil’ comments are largely attributable to ALBERT and CodeBERT, which showed improvements of 180% and 133%, respectively. These models started from a lower performance baseline, making such large gains more achievable compared to other models. However, BERT and RoBERTa also demonstrated stronger performance improvements in the ‘Civil’ class.

5.4.3 RQ3: Software Engineering Task Automation Where Affect Plays a Role

RQ3: Can a better understanding of figurative language enhance software engineering automation where affect plays a role?

To answer this RQ, we focus on a specific use case: automatic bug report priority detection, a major research area in software engineering [254, 23, 235, 156], where
previous research has highlighted the role of affect [189].

**Dataset.** Bugzilla bug reports are widely used for priority detection [254, 23, 189]. The bug priority reports in Bugzilla are divided into 5 classes (i.e., P1 to P5, where P1 represents the highest priority while P5 represents the lowest priority). Wang et al. collected 220k bug reports from Bugzilla [23]. We sample 25% of this dataset using stratified sampling across the 5 classes. We sample separately from the training and testing splits provided by the authors, which yielded a total of 49.6k bug reports. The distributions provided by the authors are: 1) training: P1 - 19.56%, P2 - 18.45%, P3 - 58.12%, P4 - 1.66%, and P5 - 2.21%; and 2) testing: P1 - 19.21%, P2 - 17.66%, P3 - 59.5%, P4 - 1.48%, and P5 - 2.15%.

**Procedure and Metrics.** We use the same four LLMs (BERT, RoBERTa, ALBERT, and CodeBERT) as baselines and follow the same approach for training and testing described in RQ2. We use F1-score as evaluation metric.

**Results and Discussion.** Table 12 shows the results of bug report priority prediction on the Bugzilla dataset. All four models made small improvements (1.96%, 2.40%, 3.71%, and 1.61% respectively) when fine-tuned with figurative languages. On the other hand, the improvements across classes (P1-P5) varied. The change in the P5 class was minimal (0.27%), and none of the models succeeded in recognizing any of the P4 instances. This is likely due to the fact that these two classes have the smallest amounts of data, comprising only 1.66% for P4 and 2.21% for P5 of the training data, respectively. Such findings suggest that fine-tuning with figurative language is not beneficial in cases of extreme data imbalance. For the average performance improvement across all models in the other three bug priority classes, we observe that P3 improved least (1.43%) while P1 and P2 make more substantial gains of 4.37% and 8.23%. Umer et al. [189] noted that a substantial number of instances in the
Table 12.: Evaluation of LLMs finetuned with figurative language on the Bug Report Priority dataset (F1-score).

<table>
<thead>
<tr>
<th>Model</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>Micro Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>0.606</td>
<td>0.329</td>
<td>0.833</td>
<td>0.0</td>
<td>0.663</td>
<td>0.716</td>
</tr>
<tr>
<td>BERT-FL</td>
<td>0.632</td>
<td>0.359</td>
<td>0.842</td>
<td>0.0</td>
<td>0.667</td>
<td>0.730</td>
</tr>
<tr>
<td>+/-</td>
<td>+4.31%</td>
<td>+9.14%</td>
<td>+1.10%</td>
<td>-</td>
<td>+0.52%</td>
<td>+1.96%</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.61</td>
<td>0.293</td>
<td>0.827</td>
<td>0.0</td>
<td>0.677</td>
<td>0.707</td>
</tr>
<tr>
<td>RoBERTa-FL</td>
<td>0.624</td>
<td>0.343</td>
<td>0.839</td>
<td>0.0</td>
<td>0.674</td>
<td>0.724</td>
</tr>
<tr>
<td>+/-</td>
<td>+1.91%</td>
<td>17.24%</td>
<td>+1.39%</td>
<td>-</td>
<td>-0.51%</td>
<td>+2.40%</td>
</tr>
<tr>
<td>ALBERT</td>
<td>0.564</td>
<td>0.288</td>
<td>0.810</td>
<td>0.0</td>
<td>0.670</td>
<td>0.683</td>
</tr>
<tr>
<td>ALBERT-FL</td>
<td>0.602</td>
<td>0.299</td>
<td>0.827</td>
<td>0.0</td>
<td>0.674</td>
<td>0.709</td>
</tr>
<tr>
<td>+/-</td>
<td>+6.71%</td>
<td>+3.88%</td>
<td>+2.14%</td>
<td>-</td>
<td>+0.53%</td>
<td>+3.71%</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>0.608</td>
<td>0.363</td>
<td>0.830</td>
<td>0.0</td>
<td>0.667</td>
<td>0.714</td>
</tr>
<tr>
<td>CodeBERT-FL</td>
<td>0.636</td>
<td>0.373</td>
<td>0.839</td>
<td>0.0</td>
<td>0.670</td>
<td>0.726</td>
</tr>
<tr>
<td>+/-</td>
<td>+4.55%</td>
<td>2.64%</td>
<td>+1.08%</td>
<td>-</td>
<td>0.52%</td>
<td>+1.61%</td>
</tr>
<tr>
<td>Avg. +/-</td>
<td>+4.37%</td>
<td>+8.23%</td>
<td>+1.43%</td>
<td>-</td>
<td>+0.27%</td>
<td>+2.42%</td>
</tr>
</tbody>
</table>

Bugzilla dataset are ‘Neutral’, indicating that including figurative expressions from ‘Neutral’ utterances — which our dataset predominantly omits — could potentially yield additional benefits.

Error Analysis of BERT-FL vs. BERT. To get an understanding of where fine-tuned models are getting results correctly compared to baseline models, we look into 51 instances where BERT-FL makes the right predictions but BERT does not. We find that, indeed, some of these bug reports include metaphors and idioms. For example, consider the following bug report description, which is at the P2 priority level: “Deadlock when adding JSF framework I have experienced a deadlock while I was adding JSF framework to regular web project. [...]” Here ‘Deadlock’ is a SE-specific figurative expression. The baseline model predicted P3, but BERT-FL made the correct prediction. Another example “Toot your own horn, put your name in the credits window The credits window is empty [...]”, annotated as P3. Here, ‘toot your own horn’ is an idiom. BERT-FL correctly predicted but the baseline model did not.

However, there are also cases with figurative language where the fine-tuned model
predicted incorrectly, while the baseline model was right. For example, consider the following bug report “offline task data is not retrieved on query [...] (i.e., fetch all things before hitting the road). [...]” Here, ‘hitting the road’ is an idiom. The BERT-FL model predicts P1 when the original label is P3. It is possible that BERT-FL recognizes the idiom, prioritizes its figurative meaning, and predicts a higher class than the original label.

5.4.4 Implications

There are a number of actionable implications to our study. Creating a glossary of common figurative language for a software project can be an invaluable tool for efficiently onboarding new developers. It would help newcomers understand project-specific or domain-specific terms, which are essential for their quick integration. Minimizing the use of obscure jargon that may cause misunderstandings can enhance mutual understanding and collaboration among project participants. Lastly, it’s important to consider cultural differences that may influence the interpretation of figurative language, as these nuances can significantly affect comprehension and communication within a diverse team.

Our study paves the way for several promising research directions in the realm of figurative language comprehension within software engineering: 1) Integrating figurative language into cutting-edge software engineering tools, such as CleBPI, could be achieved through innovative approaches like contrastive learning, self-supervised learning, or adversarial training; 2) Investigating the role of figurative language in specific scenarios, including toxic or uncivil comments, bug reports, and documentation, may yield insights into its effects on software development workflows; 3) Ex-

33\[33\url{https://www.thefreedictionary.com/hitting+the+road}\]
ploring the use of figurative language as a means for data augmentation presents an intriguing opportunity, building on established data augmentation techniques; 4) Broadening the scope of analysis to encompass various forms of figurative language, such as similes, hyperbole, and personification, could enhance the depth of model training; 5) Extending our analysis to software engineering communication platforms beyond GitHub, including Stack Overflow, Gitter, JIRA, and app reviews, would offer a more holistic view of figurative language usage across different settings. Adapting Large Language Models (LLMs) for domain-specific figurative language has recently garnered interest in the NLP community [8, 39, 52, 25]. Our work compliments this by adapting LLMs to the figurative language in software engineering.

5.5 Threats To Validity

Several limitations may impact the interpretation of our findings. We categorize and list each of them below.

Construct validity. Construct validity refers to the degree to which the study measures the concepts and constructs it claims to measure. A threat may arise from the manual annotations for the dataset, specifically in creating semantically similar EMS and DMS sentences. To mitigate this, we provided clear instructions and examples to the annotators. Additionally, we only examined metaphors and idioms; including other figurative language may alter results. To investigate this, our annotation approach can be expanded to analyze other forms. Another potential threat is that our figurative language dataset was sourced from developer communication in 9 GitHub repositories, which may not be representative of the figurative language present on GitHub.

Internal validity. Internal validity concerns the extent to which the study’s findings can be attributed to the manipulation of the independent variable. A threat is that the improved affect analysis performance with figurative language fine-tuning may not be
solely due to the figurative language. However, we see consistent improvements across all models and datasets, indicating it is a key factor. Not doing cross-validation on the smaller datasets can be another threat. To mitigate this, we use stratified sampling for representativeness and a standard 80-20% train-test split.

External validity. External validity pertains to the generalization of the findings of our study to other settings and contexts. Our results may not generalize beyond the specific studied models, datasets, and any other domain than GitHub. However, we use diverse pre-trained LLMs and a Bugzilla dataset, showing some cross-domain applicability. Further investigation is needed to validate our results beyond the tools, data, and platforms used in our study.

5.6 Chapter Contributions and Summary

This chapter examined the relevance and impact of figurative language in software engineering communication. To investigate this, we annotated metaphors and idioms in a set of 2000 sentences collected from GitHub issues and PRs which resulted in 1661 sentences with figurative expressions, conducted a comprehensive analysis of the prevalence of figurative language in messages posted on PRs and issues in top 100 GitHub repositories, fine-tuned several state-of-the-art pre-trained LLMs with the annotated dataset, and evaluated the performance of these fine-tuned models on three publicly available SE-specific datasets. Our results indicated that figurative language is prevalent in software engineering communication, and fine-tuning LLMs with figurative language leads to improved performance on affect analysis tasks (on the best model, 6.75% improvement on a GitHub emotion dataset, 7.07% improvement on a GitHub incivility dataset, and 3.71% improvement on a bug report prioritization dataset). Overall, our findings provide evidence for the relevance and impact of figurative language in software engineering communication and the potential benefits
of fine-tuning LLMs with figurative language in the context of software engineering. However, there is room for further investigation.

Beyond the future work directions discussed in Section 5.4.4, our error analysis shows that fine-tuned models may sometimes overemphasize figurative language, motivating the need for a different fine-tuning approach. Addressing this issue while preserving interpretive abilities presents an area for future research. Experimenting with generative language models like ChatGPT and LLaMa to assess their potential in enhancing the automatic interpretation of complex figurative expressions could significantly benefit communication and understanding in software development contexts. Overall, this chapter provides a starting point for further empirical research on figurative language’s impact on software engineering communications in different application domains.
6.1 Background

Going beyond detecting the occurrence of different emotions in developer communication channels, identifying the causes of those emotions is key for many uses. Merely recognizing the presence of emotions often fails to provide a comprehensive understanding of its target or the appropriate response required [201]. However, when both the emotion cause and type are known, it becomes feasible to reliably assess potential implications. This knowledge facilitates comprehension of developer sentiment towards different aspects of a project, such as technical debt [75] or code reviews [71].

For instance, a developer made the following comment on an open issue in the flutter/flutter GitHub project, “this is a really severe issue, the ux is pretty awful when you have a splash and then a landing page to simulate splash because it is very obvious that is a different view than the splash” [1], which expresses Frustration (a sub-emotion of Anger). The cause of this emotion is the text span, “the ux is pretty awful”. Extracting emotion causes automatically is challenging because of the distinct nature of software engineering communication (e.g., it includes domain-specific idioms like “spaghetti code.”), the variety of different channels (e.g., chats vs. issue comments), and the informal nature of developer communication. It is likely that emotion-cause extraction requires a large amount of software engineering-specific training data that

[1] https://github.com/flutter/flutter/issues/63156
can capture this variability, in both emotion and language 163 162.

Large Language Models (LLMs) have recently emerged as a new powerful type of deep learning technique. These models are built by unsupervised pre-training on a very large dataset, followed by supervised fine-tuning on a smaller dataset. During pre-training, the model learns to predict the next word, given a sequence of words. During fine-tuning, the model is provided with labeled data relevant to a specific task. Some of the largest and most powerful LLMs, such as ChatGPT 16 and GPT-4 17, are now widely available but do not disclose details about their dataset, training process, or model weights. Consequently, fine-tuning them for a specific task or dataset, such as detecting emotion causes in software engineering text, is not possible. However, these LLMs can still be used as “zero-shot” models, where no task-specific fine-tuning is performed. Since constructing a large training dataset for emotion-cause extraction task in software engineering communication is expensive, using a zero-shot setup is an attractive option.

In this work, we apply zero-shot LLMs, ChatGPT 16, GPT-4 17, and one that is open-source, flan-alpaca 10, to the problem of emotion-cause extraction in software engineering. We first examine the ability of such models to detect emotions in software engineering text, relative to state-of-the-art techniques and to LLMs fine-tuned for detecting emotions. Next, we examine the effectiveness of the zero-shot LLMs for the emotion-cause extraction task. Finally, we perform a case study on the causes of Frustration, an undesirable emotion within a large open-source software project, to further highlight the utility of emotion-cause extraction for software engineering.

Next we discuss two key aspects of this chapter: prompt engineering for zero-shot LLMs, and automated emotion-cause extraction in NLP.
6.1.1 Prompt Engineering for Zero-Shot LLMs

Zero-shot learning, a task where a model is trained to recognize and classify unseen classes without any explicit training data for those classes, has been a recent focus among researchers and practitioners for a variety of tasks, including image and text classification, question answering, language generation, and data augmentation [83, 46, 64, 12]. Recently, researchers have focused on leveraging LLMs for zero-shot learning [191, 95, 98, 212].

In the context of zero-shot learning, prompt engineering with LLMs has emerged as an area of interest in recent years [98, 95, 44, 49]. One approach that has been explored is the use of task-specific prompts, which are designed to elicit the desired response from the model. These prompts can be constructed manually or generated automatically and can be tailored to the specific task at hand [44]. For example, Brown et al. used an LLM to perform zero-shot text classification using task-specific prompts [106]. Another approach is the use of general-purpose prompts, which are designed to be broadly applicable across a range of tasks [98, 17].

The recent advancements in language models such as ChatGPT [16], GPT-4 [17], BARD [5], LLaMA [22], and Alpaca [20] have made the general-purpose prompt approach increasingly popular. These models have achieved impressive performance across a range of tasks and continue to push the boundaries of what is possible with LLMs.

6.1.2 Automated Emotion-Cause Extraction in NLP

Automated emotion-cause extraction in NLP is an area of research that has gained attention in recent years [93, 84, 88, 61, 96, 160]. Emotion-cause extraction is challenging, as both emotions and their causes can be expressed in various ways,
including but not limited to explicit statements, implicit suggestions, and contextual cues.

Several techniques have been proposed to address this challenge, including rule-based approaches [269], machine learning-based approaches [174], deep learning-based approaches [203], and LLM approaches [88]. In recent years, the focus has been on LLM approaches [93, 84, 88]. Researchers have explored this area with prompting as well [63]. Wang et al. noted that ChatGPT achieves comparable performance on the emotion-cause extraction task in news articles [24]. In this study, we apply prompt-based emotion-cause extraction for three state-of-the-art LLMs, namely ChatGPT, GPT-4, and flan-alpaca [16, 17, 33]. We also perform a case study to demonstrate how these models can be applied in real-world scenarios.

6.2 Preliminary Study: Detecting Emotion Types

Detecting the causes of emotions in text requires a reliable model that can accurately identify the type of emotion expressed. Therefore, before proceeding, we conduct a preliminary investigation to determine if zero-shot LLMs can accurately detect emotions in software engineering texts. We compare the performance of these models with 1) existing state-of-the-art emotion classification models in software engineering, and 2) fine-tuned LLMs. The models are evaluated on three different types of datasets: a) our GitHub comments dataset from previous chapter 3, b) a Stack Overflow comments dataset [183], and c) a JIRA comments dataset [233].

6.2.1 Datasets

GitHub Dataset. We discussed details about curating this dataset in the previous chapter 3.

Stack Overflow Dataset. Novielli et al. annotated a rich multi-label dataset
comprising 4800 Stack Overflow questions, answers, and comments. Within this dataset, 18.1% of the samples are labeled with Anger, 25.4% with Love, 2.2% with Fear, 10.2% with Joy, 4.8% with Sadness, and 0.9% with Surprise. The remaining contents of the dataset are neutral.

**JIRA Dataset.** Ortu et al. annotated a comprehensive collection of 4000 comments extracted from JIRA, classifying them into four distinct emotional categories: Love, Joy, Anger, and Sadness (1000 comments each). Within each category, Love, Joy, Sadness, and Anger account for 16.6%, 12.4%, 32.4%, and 30.2% respectively, while the remaining comments are neutral.

For training and testing with each dataset, we employ an 80%-20% stratified sampling approach.

### 6.2.2 Emotion Model

All these three datasets rely on the well-known Shaver’s tree-structured emotion model. In Shaver’s model, for each of the six basic emotions, there are secondary and tertiary-level emotions, which refine the granularity of the previous level. GoEmotions is an alternative emotion model used in the literature that was proposed by researchers at Google focusing on emotions that can be observed in written text.

In the previous chapter, we extended Shaver’s model by incorporating a few emotions from GoEmotions’s taxonomy in order to study emotions present in GitHub communications. Out of 27 emotions in GoEmotions’ list, 20 of them are in Shaver’s taxonomy (Table 1) and we have mapped 6 of them in the Table 2. The only emotion that we did not map, is Gratitude.

In order to study both GoEmotions and Shaver’s models, in this chapter, we map the remaining emotion - Gratitude - within Shaver’s tree-structured emotion model. We look into the definitions - how the authors defined Gratitude in GoEmotions.
and if any emotion is defined similar way in Shaver et al. [303]'s definition. GoEmotions defined *Gratitude* as “a feeling of thankfulness and appreciation.”, while Shaver et al. defined *Love* “involving the appreciation of someone.” Therefore, we mapped *Gratitude* as a secondary emotion to the basic emotion *Love* in this study.

The extended model is shown in Table 6.2.3 with blue-colored emotions also appearing in the GoEmotions listing.

### 6.2.3 Compared Models

We compared the Zero-shot LLMs with fine-tuned LLMs.

**Fine-tuned LLMs.** We fine-tune two popular LLMs – BERT and RoBERTa – that have been widely used as emotion and sentiment analysis, including in software engineering [65, 104, 42, 144]. Like previous chapters, we leverage the pre-trained model weights from HuggingFace [7].

**Zero-shot LLMs.** We use three (two commercial and one open-source) recent pre-trained and instruction-tuned models in a zero-shot setting, i.e., the models are not tuned for the task of emotion (cause) detection in software engineering.

- **ChatGPT (GPT-3.5-turbo):** We use the gpt-3.5-turbo API by OpenAI [16]. GPT-3.5-based models are pre-trained on a massive corpus of text data from diverse sources, including books, articles, websites, and other publicly available online content. The model was then instruction-tuned (from a large dataset of instructions with desired output) using Reinforcement Learning from Human Feedback (RLHF) [53].

- **GPT-4:** We use the gpt-4 API by OpenAI. GPT-4 is a transformer-style model pre-trained using both publicly available data and data licensed from third-party providers; details of the training data are not released at the time of
Table 13: Extended Shaver’s tree-structured taxonomy.

<table>
<thead>
<tr>
<th>Basic Emotion</th>
<th>Secondary Emotion → Tertiary Emotion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Anger</strong></td>
<td>Irritation → Annoyance, Aggravation, Grumpiness, Grouchiness</td>
</tr>
<tr>
<td></td>
<td>Exasperation → Frustration</td>
</tr>
<tr>
<td></td>
<td>Rage → Anger, Fury, Hate, Dislike, Resentment, Outrage, Wrath, Hostility, Bitterness, Ferocity, Loathing, Scorn, Spite, Vengefulness</td>
</tr>
<tr>
<td></td>
<td>Envy → Jealousy</td>
</tr>
<tr>
<td></td>
<td><strong>Disgust</strong> → Revulsion, Contempt, Loathing</td>
</tr>
<tr>
<td></td>
<td>Torment</td>
</tr>
<tr>
<td></td>
<td><strong>Disapproval</strong> †</td>
</tr>
<tr>
<td><strong>Love</strong></td>
<td>Affection → Liking, Caring, Compassion, Fondness, Affection, Love, Attraction, Tenderness, Sentimentality, Adoration</td>
</tr>
<tr>
<td></td>
<td>Lust → Desire, Passion, Infatuation</td>
</tr>
<tr>
<td></td>
<td>Longing</td>
</tr>
<tr>
<td></td>
<td><strong>Gratitude</strong> ‡</td>
</tr>
<tr>
<td><strong>Fear</strong></td>
<td>Horror → Alarm, Fright, Panic, Terror, Fear, Hysteria, Shock, Mortification</td>
</tr>
<tr>
<td></td>
<td>Nervousness → Anxiety, Distress, Worry, Uneasiness, Tenseness, Apprehension, Dread</td>
</tr>
<tr>
<td><strong>Joy</strong></td>
<td>Cheerfulness → Happiness, Amusement, Satisfaction, Bliss, Gaiety, Glee, Jolliness, Joviality, Joy, Delight, Enjoyment, Gladness, Jubilation, Elation, Ecstasy, Euphoria</td>
</tr>
<tr>
<td></td>
<td>Zest → Enthusiasm, Excitement, Thrill, Zeal, Exhilaration</td>
</tr>
<tr>
<td></td>
<td>Contentment → Pleasure</td>
</tr>
<tr>
<td></td>
<td><strong>Optimism</strong> → Eagerness, Hope</td>
</tr>
<tr>
<td></td>
<td>Pride → Triumph</td>
</tr>
<tr>
<td></td>
<td>Enthrallement → Enthrallement, Rapture</td>
</tr>
<tr>
<td></td>
<td>Relief</td>
</tr>
<tr>
<td></td>
<td><strong>Approval</strong> †</td>
</tr>
<tr>
<td></td>
<td><strong>Admiration</strong> †</td>
</tr>
<tr>
<td><strong>Sadness</strong></td>
<td>Suffering → Hurt, Anguish, Agony</td>
</tr>
<tr>
<td></td>
<td>Sadness → Depression, Sorrow, Despair, Gloom, Hopelessness, Glumness, Unhappiness, Grief, Woe, Misery, Melancholy</td>
</tr>
<tr>
<td></td>
<td><strong>Disappointment</strong> → Displeasure, Dismay</td>
</tr>
<tr>
<td></td>
<td>Shame → Guilt, Regret, Remorse</td>
</tr>
<tr>
<td></td>
<td>Neglect → Embarrassment, Insecurity, Insult, Rejection, Alienation, Isolation, Loneliness, Homesickness, Defeat, Dejection, Humiliation</td>
</tr>
</tbody>
</table>

102
### Notes:
Emotions in blue appear in the list of emotions proposed by GoEmotions. Emotions added in the previous chapter from GoEmotions’ list onto Shaver’s taxonomy are denoted with †. A single emotion – *Gratitude* – is added in this chapter, denoted by ‡.

*GPT-4* introduced a rule-based reward model (RBRM) approach on top of RLHF.

- **flan-alpaca** [10]: This is a variation of the Alpaca [20] fine-tuned model. Alpaca was developed by Stanford, based on Meta’s LLaMA [22] model using 52K instruction-based data instances. Due to licensing issues, the original Alpaca model is not accessible at the time of our experiment. Instead, using the Alpaca instructions dataset, Chia et al. [10] fine-tuned Google’s instruction-tuned Flan-T5 [33] model and released the weights on Huggingface. We use the flan-alpaca-xl version.

#### 6.2.4 Basic Emotion Prompting

The zero-shot LLMs we are considering are all instruction- (or prompt-) tuned. This recent category of LLMs use a fine-tuning process with instructional data, which helps the LLMs to better comprehend and respond to user-composed prompts. To our knowledge, there is no prior work on how to formulate prompts for emotion recognition in software engineering text using these LLMs.

---

A recent study by Kocon et al. [9] evaluated the performance of ChatGPT on various natural language processing tasks by designing over 38k prompts that covered 25 different tasks, including emotion classification using the GoEmotions dataset [110]. Inspired by this study, we designed a prompt for emotion classification that we used on all three datasets. More specifically, we asked the models to act as a user in a specific platform, i.e., GitHub, Stack Overflow, and JIRA, and provided the utterances and a list of the basic (top-level) emotions: Anger, Fear, Love, Joy, Sadness, and, Surprise. The prompt is the following:

```
You are a [GitHub/Stack Overflow/JIRA] user. You are reading comments from [GitHub/Stack Overflow/JIRA]. Your task is to detect whether there is one of the following emotions, arouses in you while reading the utterance.


Utterance: <insert utterance>.

If there is no emotion in the text, write Neutral. Otherwise write exactly one word, the exact emotion from the emotions list.
```

Since the JIRA dataset does not contain Fear and Surprise, we do not list these two emotions in the prompt when evaluating with this dataset.

**Results and Discussion.** Table 14 shows the results for the three emotion classification datasets and for all the models. It is clearly noticeable from the results that the zero-shot LLMs performed poorly across all datasets, lagging behind the SE-specific models and the fine-tuned LLM models. The fine-tuned LLMs performed best, e.g., RoBERTa achieved the best micro-averaged F1-score overall by averaging 0.592, 0.735, and 0.818 respectively for GitHub, Stack Overflow, and JIRA datasets.

In order to understand where the zero-shot LLMs are making mistakes, next, we
Table 14.: Micro-averaged F1-score of emotion classification models for three different datasets.

<table>
<thead>
<tr>
<th></th>
<th>GitHub</th>
<th>SO [183]</th>
<th>JIRA [233]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fine-tuned LLMs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT</td>
<td>0.588</td>
<td>0.716</td>
<td>0.817</td>
</tr>
<tr>
<td>RoBERTa</td>
<td><strong>0.592</strong></td>
<td><strong>0.735</strong></td>
<td><strong>0.818</strong></td>
</tr>
<tr>
<td><strong>Zero-shot LLMs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChatGPT</td>
<td>0.234</td>
<td>0.339</td>
<td>0.276</td>
</tr>
<tr>
<td>flan-alpaca</td>
<td>0.424</td>
<td>0.293</td>
<td>0.432</td>
</tr>
<tr>
<td>GPT-4</td>
<td>0.355</td>
<td>0.444</td>
<td>0.256</td>
</tr>
</tbody>
</table>

conduct an error analysis.

**Error analysis.** One of the most common errors we observed is that zero-shot LLMs are misclassifying *Love* utterances as *Joy* for all datasets. For example, on the Stack Overflow dataset, the F1-score for *Love* is 0.0, 0.116, and 0.078 for flan-alpaca, ChatGPT, and GPT-4 respectively. Compared to this, BERT and RoBERTa obtained an F1-score of 0.840 and 0.861 respectively. This is also evident in the number of false positive (FP) utterances in the *Joy* category, i.e., for the Stack Overflow dataset, the number of FPs for BERT and RoBERTa are 34 and 21 respectively, whereas, for flan-alpaca, ChatGPT, and GPT-4, the FPs are 259, 72, and 91.

Another common type of error was that the models predicted *Neutral* often. In many cases a secondary or tertiary emotion for Shaver’s categorization most closely describes the annotated utterances. However, those emotions were not provided to the model. For example, consider the following sentence from the GitHub dataset: “*Any updates on this? I’m implementing a flutter application with barcode scanners, the soft keyboard on screen is really annoying.*”, annotated as *Anger* and, on a more granular level, as *Annoyance*. All zero-shot LLMs models predicted it as *Neutral*. As another example, the following sentence is annotated as *Worry*, which is a tertiary-level emotion of *Fear*: “*My concern is that more new attributes may appear [...] it*
may break their behavior.”, while flan-alpaca and ChatGPT classified it as \textit{Neutral}.

We also observed a number of hallucinations in the zero-shot LLMs output \cite{2}, where the models generated responses that were outside of what was asked. This led to situations where the models outputted emotions such as \textit{Apology} and \textit{Appreciation}, despite them not being in the prompted emotions list. For example, GPT-4 predicted the following sentence as \textit{Apology}: “Doh. Sorry for wasting your time.” even though the set of basic emotions provided in the prompt does not contain this emotion.

In order to address these issues, we experiment with constructing prompts with a more granular level of emotions, i.e., by considering the second and tertiary-level emotions in Shaver’s extended taxonomy. This is also motivated by the study of Kocon et al. \cite{9}, who used all of GoEmotions’ 27 emotions in their prompting experiments with ChatGPT.

\subsection*{6.2.5 Granular-level Emotion Prompting}

In order to experiment with more granular emotions, we require a labeled dataset that includes these emotions. Therefore, we specifically conducted these experiments with our curated dataset in the previous chapter, as it provides a secondary and tertiary-level emotion annotation while the other datasets do not. First, we conducted prompt experiments using a part of the training set (note that the zero-shot LLMs are not using the training data) varying the information used in the prompts for each instruct-tuned language model. More specifically, we randomly selected 400 comments from the training dataset using stratified sampling and tested with granular-level prompting using the following strategies: 1) all emotions (basic, secondary and tertiary) from the extended Shaver’s categories – a total of 141 emotions; 2) only the basic and secondary emotions from the extended Shaver’s categories – a total of 36 emotions; 3) GoEmotions’ list of 27 emotions.
We mapped the output emotion from the secondary and tertiary emotions to corresponding basic emotions as shown in Table 6.2.3 and compared the results of the models at this level (as the SE-specific models can only produce results at the basic emotion level). We also found during the granular-level prompting that the models sometimes produced minor wording variations of the provided emotions, such as *Confused* instead of *Confusion*, *Excited* instead of *Excitement*. While mapping the outputs of the zero-shot LLMs to the basic emotions, we made adjustments as not to punish the models for these minor differences.

During our experimentation with the full set of emotions using the three zero-shot LLMs, ChatGPT, GPT-4, and flan-alpaca, we observed that all of them tend to suffer more strongly from the issue of hallucination when more granular-level emotions are provided [2]. In particular, the models tended to generate extrinsic hallucinations [40], i.e., information beyond what is asked in the given prompt. This led to situations where the models generated emotions such as *Concern*, *Apology*, and *Appreciation*, despite them not being in the prompted emotions. This suggests that providing a very large list of emotions may not be optimal.

Out of the strategies we attempted, providing GoEmotions’ 27 emotions list produced the best performance. For example, on the sample of the training dataset, ChatGPT achieved an F1-score of 0.201 when all emotions from Table 6.2.3 were provided, 0.341 when basic and secondary emotions are provided, and 0.419 when GoEmotions’ emotions are provided. As noted earlier that the GoEmotions’ taxonomy is developed specifically for text-based emotion recognition [110]. This can explain why it performed better than emotions selected directly from Shaver’s taxonomy, which was developed based on psychological evidence and not specifically for text [303].

Therefore, we opt to use GoEmotions’ list of emotions for prompting for emotion
classification using the zero-shot LLMs. Next, we report the results on our held-out test dataset.

**Results and Discussion.** Table 15 shows the results for emotion classification on the GitHub dataset for all the models. Overall, BERT and RoBERTa still achieve the best results with an average F1-score of 0.588 and 0.592 respectively. From the table, it is clear that all three zero-shot LLMs improve in most categories of emotions and overall micro-averaged F1-score. It is also noticeable that they improved in distinguishing *Love* and *Joy* utterances. However, the zero-shot LLMs still perform badly for *Fear*. Overall, surprisingly, the open-source model flan-alpaca achieved the best performance with an average F1-score of 0.507 – an improvement of 19.58% from the basic emotion-level prompting, while the proprietary model GPT-4 achieved 0.481 – an improvement of 35.49%. Both of these are improvement over the three SE-specific models and the proprietary ChatGPT (*gpt-3.5-turbo*) model.

The results again point out that despite there having been major advancements in instruction-tuned LLMs, the fine-tuned deep learning models still perform better for specific, well-defined tasks that require domain-specific knowledge. To understand more where zero-shot LLMs are still making errors in detecting emotions, we conduct an error analysis on the errors in granular level prompting.

**Error Analysis.** From the 356 non-*Neutral* instances in the test set, all three models correctly predicted 75 instances, two models made the right prediction for 67 instances, only one of the models made the right prediction for 74 instances, and no zero-shot LLMs made the right prediction on 140 instances. A Venn diagram of the classification error of each of the three zero-shot LLMs is shown in Figure 11. We examine more closely the 140 utterances where all three zero-shot LLMs models made wrong predictions.

As also noticeable in Table 15, *Fear* is the most often misclassified category with
Table 15.: Micro averaged F1-score of emotion classification for different models on GitHub dataset. The zero-shot LLMs use the GoEmotions list of 27 emotions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fine-tuned LLMs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT</td>
<td>0.506</td>
<td>0.712</td>
<td>0.536</td>
<td>0.579</td>
<td>0.636</td>
<td>0.594</td>
<td>0.588</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.525</td>
<td>0.683</td>
<td>0.492</td>
<td>0.500</td>
<td>0.613</td>
<td>0.673</td>
<td>0.592</td>
</tr>
<tr>
<td><strong>Zero-shot LLMs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChatGPT</td>
<td>0.337</td>
<td>0.49</td>
<td>0.182</td>
<td>0.458</td>
<td>0.412</td>
<td>0.511</td>
<td>0.423</td>
</tr>
<tr>
<td>flan-alpaca</td>
<td>0.447</td>
<td>0.543</td>
<td>0.140</td>
<td>0.446</td>
<td>0.451</td>
<td>0.740</td>
<td>0.507</td>
</tr>
<tr>
<td>GPT-4</td>
<td>0.437</td>
<td>0.698</td>
<td>0.0</td>
<td>0.446</td>
<td>0.487</td>
<td>0.517</td>
<td>0.481</td>
</tr>
</tbody>
</table>

35 instances. The errors in this category are especially discernible with GPT-4. With basic emotions only, GPT-4 achieved an F1-score of 0.353 in the Fear category while at the granular level, the F1-score went down to 0.0. The primary reason for it is that GPT-4 generated hallucinated output with labels such as Worry, Anxiety, etc., which are missing in the GoEmotions list. However, these emotions are present in Shaver’s extended list and in our GitHub Emotion dataset annotation. In the annotated data, most of the Fear utterances are due to the tertiary-level emotion Worry. For example, the utterance “Isn’t this a breaking change? Can we get away with it?” is annotated as Worry (3rd level of Fear) in the ground truth. Another example is the utterance: “I guess my concern is that it sets a precedent where somebody could see it and think that it would be fine to use in ‘core’.”

The second most misclassified emotion category is Joy with 33 instances. Many of these errors are because the models are predicting conservatively, i.e., predicting Neutral instead of a specific emotion. For example, “Anyway, the syntax change is fine.” – this utterance is annotated as Approval (2nd level of Joy). Another example, “[USER] can you assign this ticket to me, I can help in this.” – this utterance is annotated as Enthusiasm (2nd level of Joy). Also, notable here is that Enthusiasm is
The third most misclassified category is Sadness with 31 instances. We observed that these utterances are often misclassified as Anger, Surprise, or Neutral. For example, flan-alpaca and GPT-4 predicted Surprise for this utterance: “Ah sorry I thought ‘ScaleUpdateDetails’ was constructed in ‘_update’ nvm.”

The next misclassified category is Anger. Similar to Joy, the errors in the Anger category often result from the models predicting conservatively or misclassifying as Surprise. For example, “Does this file really belong to this PR? It seems unrelated 😞” – is annotated as Annoyance, however, all zero-shot LLMs models have predicted it as Surprise. Among the rest of the emotion categories, most of the Love errors are due to misclassifying them as Joy. For example, “PS: I am fan of yours, I love your
content out there! :smiley;” – this utterance is predicted as Joy by all models.

We observed hallucinated emotions as well, especially Concern, Worry, and Anxiety significant among Fear utterances; Apology among Sadness utterances; and Appreciation among Love utterances.

Overall, the error analysis points out the need for having a more specialized emotion taxonomy for text-based emotion detection, in particular for software-engineering-related text. As noted earlier, Shaver’s taxonomy, developed in Psychology, includes many additional emotions that do not appear in the text and confuse the zero-shot LLMs. Meantime, while the GoEmotions list focuses on text-based emotions, they are still missing some commonly observed emotions in software engineering such as Worry and Frustration.

6.3 Emotion-Cause Extraction

The results of the preliminary study indicate that zero-shot LLMs are relatively good at detecting emotion categories, on par with the best models built for this purpose. In this section, we examine their feasibility for the more challenging task of emotion-cause extraction.

The use of LLMs for emotion-cause extraction has experienced a significant uptick in interest in recent years [88, 93]. Emotion-cause extraction seeks to identify the cause or event that instigates a specific emotion in a given text, providing essential insights into human behavior and deepening our comprehension of the underlying emotions behind text-based communication. Researchers have explored the potential of LLMs in detecting emotion causes across multiple domains, such as social media and news articles [93, 84, 88].

Despite the growing interest in emotion-cause extraction in different domains, there is a lack of research on this problem in software engineering communication.
text. This research gap inspires our study, which aims to investigate the effectiveness of zero-shot LLMs in detecting emotion causes in GitHub comments.

To this end, we first manually annotate emotion causes in a subset of our data, identifying the text span that represents the cause of emotion in the comment. We then use zero-shot LLMs to extract emotion causes and compare their performance against the annotated emotion causes using the BLEU score [289], a standard metric in machine translation to evaluate text sequence similarity. Below, we present a detailed description of our annotation process, zero-shot LLMs, and the comparison of BLEU scores across different models and configurations.

6.3.1 Annotation

To create a dataset for the emotion-cause extraction task, we begin by selecting 75 utterances for each of the 6 basic emotion categories (Anger, Love, Fear, Joy, Sadness, Surprise) from GitHub Emotion dataset, totaling 450 utterances. Two senior undergraduate students (with 3+ years of experience in programming) are then tasked with annotating the dataset by identifying emotion causes, if any, based on the previously annotated basic, secondary, and tertiary emotions by us (Chapter 3). We provide them with the following instructions:

For each instance containing an emotion (Anger, Love, Fear, Joy, Sadness, Surprise), find the span of text (if any) that contributes to the annotated emotion. Each instance then should be annotated with its corresponding causes if existing. Emotion can sometimes be associated with more than one cause, in such a case, both causes should be marked. Since in some cases, more than one emotion can be present in an instance, the causes for emotion should be mapped as <emotion, cause span>.
The above instructions are adapted from Chen et al.’s seminal work on detecting emotion causes [265]. We also provide the annotators with definitions and examples of different types of emotion causes. After the annotation task is completed, one of the authors of the paper manually reviewed both sets of annotations and noted disagreements in 44 of the 450 instances. To resolve these discrepancies, the judges are asked to meet on Zoom and discuss and resolve their differences. This process ensures the annotated dataset’s reliability and consistency.

6.3.2 Model Selection

For the automated emotion-cause extraction task, we evaluate the same three instruction-tuned models (ChatGPT, GPT-4, and flan-alpaca) that we used for emotion detection in Section 6.2 i.e., the preliminary study. We do not use BERT or RoBERTa as those models require a large amount of domain-specific training data [12], which we lack, i.e., they would perform poorly as zero-shot LLMs.

6.3.3 Prompt Design

The structure of our emotion-cause extraction prompt is intended to mimic a real-world scenario where a GitHub user is going through issues and pull requests, experiencing various emotions, and trying to pinpoint the cause of a specific emotion in a given utterance. We use a two-step prompt that asks the model to first detect the emotion in the utterance using the procedure outlined in Section 6.2. Then, we prompt the model to identify the cause of this emotion, as shown in the framed box structure.
6.3.4 Results

To ensure consistency in our evaluation, we preprocess all comments, annotated causes, and model-extracted causes by removing punctuation, lemmatizing, and stemming. After preprocessing, the average length of the 450 utterances is 28.08 words, while the average length of the manually annotated emotion cause spans is 7.43 words. We find that the emotion cause spans extracted by GPT-4, ChatGPT, and flan-alpaca have average lengths of 8.85, 8.64, and 13.12 words, respectively.

6.3.4.1 BLEU score

The BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate the quality of machine-generated text by comparing it to human-generated reference text [289]. The BLEU score measures the similarity between the machine-generated text and the reference text based on the n-gram overlap between them. The higher the BLEU score, the closer the machine-generated text is to the reference text. The formula for the BLEU score is:

\[
BLEU = BP \cdot \exp \left( \sum_{n=1}^{N} w_n \cdot \log(p_n) \right)
\]

where,

- \( BP \) is the brevity penalty, which is 1 if the machine-generated text is longer
than the reference texts and less than or equal to them otherwise.

- \( N \) is the maximum n-gram order.
- \( p_n \) is the precision score for n-grams.
- \( w_n \) is the weight for n-grams, which is usually set to \( \frac{1}{N} \) for uniform weighting of all n-gram orders.

### 6.3.4.2 BLEU Score Interpretation

The interpretation of BLEU scores can vary depending on the specific domain and language being evaluated. In the software engineering domain, a BLEU score is commonly used to evaluate the quality of generated bug reports, code comments, and code summarization. Denkowski and Lavie [266] suggest that BLEU scores above 0.30 generally indicate that the generated text is understandable, while scores above 0.50 are indicative of good and fluent results. Previous research [256, 266], including studies in software engineering [114], has used this scale to interpret the results of BLEU scores. It is important to note that the choice of n-gram order used to calculate the BLEU score can impact the final score; typically, 4-gram is used for BLEU score calculation [266, 289]. In the case of our study, however, the emotion cause spans are often short, making the bigram a more suitable choice for BLEU score calculation, i.e., BLEU-2.

### 6.3.4.3 Discussion

The BLEU scores for the three models using unigram, bigram, trigram, and four-gram are shown in Table 16. The score ranges between 0.450 to 0.637, which indicates that all models are generally able to extract the right emotion causes to some extent, especially GPT-4 and flan-alpaca as both models’ BLEU scores are always above 0.5.
Table 16: BLEU scores of different zero-shot LLMs.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatGPT</td>
<td>0.522</td>
<td>0.489</td>
<td>0.467</td>
<td>0.450</td>
</tr>
<tr>
<td>GPT-4</td>
<td>0.637</td>
<td>0.598</td>
<td>0.571</td>
<td>0.554</td>
</tr>
<tr>
<td>flan-alpaca</td>
<td>0.571</td>
<td>0.543</td>
<td>0.525</td>
<td>0.508</td>
</tr>
</tbody>
</table>

When considering BLEU-2, GPT-4 obtains the highest score of 0.598, followed by flan-alpaca with 0.543 and ChatGPT with 0.489. Out of 450 utterances, 107 cases are identified where all three models’ BLEU-2 scores are higher than 0.5. We observe that these 107 utterances are relatively short, with an average length of 15.26 words, while the annotated cause spans have an average length of 7.02 words. The three models, GPT-4, ChatGPT, and flan-alpaca, extract similar length spans on average, which are 7.89, 7.79, and 7.95 words, respectively. For example, in the following utterance, “I’m not sure how to fix this, nor if this is acceptable in this test case. Namespaces in TS are magic to me 😊”, the annotated cause of Amusement (3rd level Joy) is “Namespaces in TS are magic to me”. GPT-4 also extracted the same span as the cause. However, it is not always the case that the annotated cause span completely overlaps with the spans extracted by the models. For example, in this utterance, “Oh, you didn’t add composes and values. Well, I like it even more. Those features are hard to maintain.”, the annotated cause span is “I like it even more”, and the extracted cause span by GPT-4 is “Well, I like it even more.”

Out of 450 utterances, we observe that in 41 cases, all three models’ BLEU scores are less than 0.30. These comments are relatively longer, containing an average of 44.17 words, while the annotated cause spans contain on average 5.05 words. The extracted average lengths of spans for GPT-4, ChatGPT, and flan-alpaca are 10.10,
13.14, and 22.83 words, respectively.

6.3.4.4 Error Analysis

In order to gain insight into the models’ mistakes, we conduct an analysis of the 41 utterances where all three models had a BLEU score of less than 0.30. Our examination reveals that the errors can be classified into a few primary categories, which are elaborated below.

**Incorrect Emotion.** The main source of error for all three models is the misidentification of the emotion expressed in the utterance. This misidentification leads to the detection of an incorrect cause event. For instance, consider the utterance, “Oh right! 😞 This started as a Mac issue, I forgot to add the rest.” The annotated emotion for this utterance was “Neglect (2nd level Sadness)” and the annotated cause span is “I forgot to add the rest.” However, ChatGPT identifies the utterance as “Confusion (2nd level Surprise)” and extracts “🌅” as the cause event instead. GPT-4 detects Amusement in the utterance and extracts the cause span as ”Oh right! 😞.” Meanwhile, flan-alpaca identifies “Curiosity (2nd level Surprise)” and extracts the cause span as “Oh right!” This error category emphasizes the importance of accurately detecting the emotion expressed in the text before extracting emotion causes.

**Incorrect Cause.** This error occurs when the models correctly classify the emotion but detect a different cause than the ground truth. For example, in the following utterance “[USER] yep, it is bug, we will fix it, so we have it in ‘experiments‘ :+1:”, the annotated emotion is Approval, and the annotated cause span is “it is a bug”, while GPT-4 detected the cause span “we will fix it”. This error category highlights the difficulty in identifying the exact cause of events in conversational text, especially in longer, multi-part comments.

**Hallucinations.** In addition to the two error categories described above, we also
observe instances of hallucinations in the cause event extraction process. In some cases, the models’ outputs are “the entire sentence.”, “the span: <followed by the span>”, “span starting from word X to word Y”, and other nonsensical outputs. We observe ChatGPT produces more hallucinated data than the other two models, which is one reason why its BLEU score is lower. This highlights the need for continued research into developing more accurate and reliable models that can follow the prompt exactly.

6.4 Investigating the Causes of Frustration in the Tensorflow Repository: A Case Study

Frustration is a pervasive emotion in software development [228], and it is particularly relevant in the context of open-source projects [242]. Wrobel et al. noted that Frustration is the most commonly felt emotion during software development [255]. Collaborative work, lack of control over external contributors’ code, and the complexity of software development processes can all contribute to the Frustration of developers and end-users. In contrast to other emotions, such as Confusion or Excitement, Frustration is more strongly associated with obstacles, challenges, and difficulties. It is also often accompanied by other negative emotions, such as Anger, Disappointment, or Helplessness [267]. Given the complexity and collaborative nature of open-source software development, Frustration is undesirable but likely to be a common experience for many contributors and users. Therefore, understanding the causes of Frustration in open-source development can provide valuable insights for project maintainers into what are the key issues that impede collaboration and the productivity of project participants.
Tensorflow\textsuperscript{3} is a popular open-source platform for developing machine learning models and has a large number of developers and a huge user-base, which makes it an interesting case study for investigating the causes of \textit{Frustration} in open-source software development. For instance, monitoring of the causes of \textit{Frustration} in TensorFlow contributors can aid in the construction of project maintainer dashboards that help attract and retain open source contributors \cite{154,37}.

6.4.1 Data Collection and Cause Extraction

To conduct our analysis, we collect all publicly available issues and pull requests comments made on the Tensorflow repository, hosted on GitHub, between March 30, 2022, and March 30, 2023. We choose this time period to ensure that our analysis covers a recent and substantial range of comments. Most GitHub repositories, including Tensorflow, differentiate different types of comment authors based on their relationship to the project, such as Contributors, Collaborators, Members and None\textsuperscript{4}. A Collaborator is a GitHub user invited to work on the repository, a Contributor has committed code before, a Member belongs to the owning organization, and None has no affiliation with the repository. Collaborators, Contributors, and Members are active developers, while None comprises user commenters. To analyze software developer \textit{Frustration}, we exclude comments from the None category.

Following the emotion-cause extraction procedure described in Section \ref{sec:emotion-cause}, we extract the emotions and causes of each comment. We use the \textit{flan-alpaca} model for this purpose, as it performed reasonably well in both emotion detection and emotion-cause extraction tasks compared to the proprietary zero-shot LLMs. Another ad-

\textsuperscript{3}https://github.com/tensorflow/tensorflow

\textsuperscript{4}https://docs.github.com/en/graphql/reference/enums#commentauthorassociation
vantage of flan-alpaca is that it is open-source and its weights are publicly available. This ensures the reproducability of our results. In contrast, closed-source LLMs may become unavailable, e.g., OpenAI’s Codex LLM was deprecated in March, 2023.

We collect only the utterances that the model identified as expressing Frustration, resulting in a dataset of 1275 comments.

6.4.2 Clustering

To identify common themes among the causes of Frustration, we employ the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm \[298\]. It has been effectively used in previous software engineering studies involving clustering textual data \[224, 207\]. The main advantage of using the DBSCAN algorithm is that it does not require a pre-specified number of clusters, which can be difficult to estimate in advance. This is particularly useful in the context of identifying common themes among the causes of Frustration, as it is difficult to know beforehand what the common themes are. Another advantage of the DBSCAN algorithm is its ability to automatically handle noise and outliers, which is relevant as the extracted causes by flan-alpaca can contain errors, as discussed in the previous sections.

While DBSCAN does not require to specify the number of clusters, it requires two key parameters \[208\]: 1) \(\epsilon\) - a real positive value - the maximum allowed distance between two samples to be considered that they are part of the same dense region, and 2) \(MinPts\) - a small positive constant integer - the minimum number of samples required to consider a dense region as a cluster. We performed a manual parameter sweep, testing \(\epsilon\) values from 0.1 to 0.8 in increments of 0.05, and \(MinPts\) values from 2 to 6, following standard guidelines for parameter tuning in machine learning and data mining \[273\]. Based on the number of clusters, average number of elements per cluster, and cluster composition, we selected \(\epsilon = 0.3\) and \(MinPts = 4\), which yielded
23 clusters. Before applying the DBSCAN algorithm, we perform standard text pre-processing such as removing punctuation, URL removal, and lemmatizing on the list of causes. We use the scikit-learn library’s implementation of the DBSCAN algorithm with cosine similarity and sentence-level embeddings (all-mpnet-base-v2 model \cite{152}).

To focus our analysis on the most common causes of Frustration, we limit our discussion to the top 6 clusters in terms of the number of comments in each cluster. The clusters are presented in Table \ref{tab:clusters}, along with their description, size, and examples. We read the GitHub comments and the emotion causes to identify the underlying theme in each cluster that leads to Frustration.

### 6.4.3 Causes of Frustration

We utilized thematic analysis to identify the themes of the clusters \cite{204}. Specifically, one of the authors of this paper read each comment and coded the initial themes. Then another author reviewed the themes, then both authors discussed resolving discrepancies and finalizing the themes until the analysis reached saturation, with no new themes emerging \cite{188}. Each cluster theme is described below:

**TensorFlow Version and Dependency Issues:** This cluster primarily includes project participants struggling with incompatibility issues due to version mismatches between TensorFlow and its related dependencies. They express frustration over difficulties in configuring TensorFlow to operate correctly on their system. They also express frustration over transitioning from legacy versions to newer versions. One possible way to address these issues is to provide a more comprehensive documentation on version compatibility between TensorFlow and its dependencies.

**Pull Request (PR) Delays and Merge Conflicts:** This cluster is related to PR merging and associated communication, as well as merge conflicts. The project participants express Frustration when they have to wait a long time for a PR to be
reviewed or merged by the project maintainers. Merge conflict-induced *Frustration* is a well-known issue in open source software development [45]. Implementing automated review bots and streamlined conflict-resolution procedures can help mitigate this form of *Frustration*.

**Failing Tests:** The cluster highlights the *Frustration* felt due to test failing, possibly flaky tests [54]. The project participants report two main sources of *Frustration:* first, the inability to identify the root cause of test failures that seem unrelated to their code changes; second, unexpected test failures leading to their PRs being reverted.

**Too Fine-Grained Commits:** This cluster reflects developers’ *Frustration* on commits that capture incomplete changes or partial progress on a task, which need to be squashed. The comments demonstrate developer sensitivities around balancing incremental changes with maintaining a coherent commit history. Setting PR guidelines about git commit hygiene can help to mitigate this issue.

**CI Flakiness:** Like test flakiness, CI flakiness is another common source of developer *Frustration* [158, 54]. This cluster highlights the complexity of CI failures. The *Frustration* is evident as the developers grapple with failed CI tests, yet believe these problems are unrelated to their own contributions.

**CUDA/CuDNN Compatibility Issues:** The project participants express *Frustration* regarding GPU library compatibility. This reflects the challenge of managing interdependent, rapidly evolving software ecosystems [3]. TensorFlow relies on quickly changing GPU libraries like CUDA and CuDNN. Expanding CI testing across more diverse versions, detecting CUDA/CuDNN versions and alerting if incompatible, and explicitly documenting supported versions can help to reduce this pitfall.
Table 17.: Clusters of Frustration emotion causes in TensorFlow.

<table>
<thead>
<tr>
<th>Cluster Description</th>
<th>Count</th>
<th>Example GitHub Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow Version and Dependency Issues: This cluster focuses on build and compatibility problems across various TensorFlow versions, challenges in reproducing issues in specific TensorFlow versions, and complications with related libraries and plugins such as TensorRT and Keras.</td>
<td>58</td>
<td>(1) [USER] Your original issue looks like you have a bad version of tensorflow_io_gcs_filesystem installed. [...] (2) It’s probably not a bug in Tensorflow but Apple’s tensorflow metal plugin. See for example the following discussion [...]</td>
</tr>
<tr>
<td>Pull Request Delays and Merge Conflicts: The cluster comprises developer frustration from unresolved merge conflicts and from delays in merging pull requests.</td>
<td>26</td>
<td>(1) [...] But there are a bunch of merge conflicts. Since Random seeds are such a common topic in software [...] (2) It might have been a wrong-way merge or something like that. At this point it’s usually easier to just close it [...]</td>
</tr>
<tr>
<td>Failing Tests: This type of Frustration arises from the ambiguity and complexity of test failures, which make it challenging for project participants to determine whether the issues are linked to their code changes or are caused by unrelated factors.</td>
<td>15</td>
<td>(1) [USER]: It is just a first draft. The test doesn’t even work. In the meantime, [...] (2) [...] Yes, I’ll work on this. It’s weird that these tests are failing because I thought I ran them successfully for PR [...]</td>
</tr>
<tr>
<td>Too Fine-Grained Commits: The cluster reflects developer Frustration caused by too granular commits in the repository. Some developers request a commit history devoid of incremental commits that represent only partial progress on a change task.</td>
<td>9</td>
<td>(1) Can you squash these commits please? It doesn’t make sense to have 5 commits for a line change and one extra empty line. (2) 3 commits for a single line change? Can you please merge the commits in just one? [...]</td>
</tr>
<tr>
<td>CI Flakiness: This type of Frustration is caused by Continuous Integration (CI) failures that seem unrelated, inconsistent, or uninformative to developers.</td>
<td>8</td>
<td>(1) [USER] there was failed ci. Is there anything to do? (2) CI failure does not look related to these changes, seeing the same failure on #56345 [...] so I assume this is noise. [...]</td>
</tr>
</tbody>
</table>
CUDA/CuDNN Compatibility Issues: This cluster reflects the Frustration experienced when dealing with compatibility issues related to CUDA and CuDNN.

Unfortunately this change needs to be rolled back, it seems it breaks JAX build under CUDA 11.4 and CuDNN 8.2

(1) - Did you downgrade the CUDA to 11.2? Looking at Nvidia docs it looks like the display driver and cuda driver do not match [...]
study. In our examination of an open-source project, Frustration causes represent an independent variable. However, there are potential threats to internal validity, such as unaccounted factors like prior experience with the project or technical expertise that could contribute to software developers’ Frustration. Moreover, the use of flan-alpaca for extracting frustration causes could result in the misclassification of some utterances, leading to the potential omission of certain clusters that could provide alternative explanations for Frustration or identification of some clusters that do not in fact represent this emotion. Nevertheless, the use of DBSCAN reduces the effect of random noise, and the list of Frustration causes provided in Table 6.5 follow the software engineering literature on common problems developers face during open-source software development [80, 82, 215].

**External validity.** External validity pertains to the generalization of our study’s findings to other settings and contexts. For emotion detection, we used the categories from extended Shaver’s taxonomy as well as GoEmotions’ taxonomy [110, 303]. However, our findings may not necessarily be transferable to other emotion categories. Another potential threat to external validity is the specific nature of the open-source project we studied, i.e., TensorFlow. The project’s characteristics, such as its size, development stage, and community culture, may not be representative of other open-source projects. Additionally, the programming language and technology stack used in the project may have influenced the types of causes of Frustration observed. Therefore, it is important to interpret our findings in the context of the specific project we studied and exercise caution when generalizing them to other open-source projects. Further investigation is needed to generalize these results beyond the three specific models and the data and projects we have used in our study.
6.6 Chapter Contributions and Summary

In this paper, we presented an approach for automated emotion-cause extraction in software developer communication using three zero-shot LLMs, namely ChatGPT, GPT-4, and (the open-source) flan-alpaca, through a prompting approach. We first conducted a preliminary study to evaluate the models’ performance in emotion classification tasks on an existing recent dataset, and we found that they perform well compared to state-of-the-art models. We then showed the feasibility of using these models for emotion-cause extraction on a subset of 450 utterances from the same dataset by manually annotating the emotion causes of these utterances and automatically extracting the causes using prompts. We compared the BLEU score performances of the models and found that GPT-4 achieved the highest BLEU-2 score of 0.598, followed by flan-alpaca with 0.543, and ChatGPT with 0.489. To demonstrate the possible real-world applications of emotion-cause extraction, we conducted a case study on the causes of *Frustration* in a large GitHub open-source project – Tensorflow.

There are several avenues for future work. First, our case study only focused on one emotion and one open-source project. Future studies that use emotion-cause extraction should investigate other emotions and a broader range of projects to generalize our findings. Second, further work is needed to improve the accuracy of emotion-cause extraction from text in software engineering communication. This could involve few-shot prompting, fine-tuning language models, or developing domain-specific models tailored for software engineering communication. Overall, our study provides a starting point for future research to explore the potential of emotion-cause extraction in software engineering communication.
CHAPTER 7

CONCLUDING REMARKS

7.1 Conclusions

In this dissertation, we have made several notable contributions toward advancing emotion detection and understanding in open-source software development communications.

Firstly, we provide a comprehensive evaluation of existing emotion classification tools tailored for software engineering text. Through error analysis on a newly annotated dataset of GitHub comments, we find the key limitations including struggles with implicit emotions and figurative language usage. However, our results show these tools can be enhanced through data augmentation techniques that expand available training data by introducing relevant lexical and semantic variations.

Secondly, we explored the use of LLMs, including BERT, RoBERTa, ALBERT, DeBERTa, CodeBERT, and GraphCodeBERT, for emotion classification in software engineering texts. Our results showed that these models generally outperform traditional tools, especially when polarity features are integrated during training, improving the contextual understanding of emotions.

Thirdly, we identified and analyzed the prevalence of figurative language in software engineering communication, such as metaphors and idioms. By fine-tuning LLMs with figurative language, we improved emotion classification accuracy, particularly in detecting nuanced emotional expressions in developer communications.

Fourthly, we demonstrate the feasibility of employing large language models in a zero-shot setting to automatically extract emotion causes from developer communi-
cations. Despite no explicit training, models like ChatGPT, GPT-4, and flan-alpaca showed reasonably good performance on this novel task when guided by suitable prompting. A case study that we conduct on uncovering causes of *Frustration* in the TensorFlow GitHub repository provided actionable insights.

Additionally, our research lead to a proposed extension of the standard emotion taxonomy to better accommodate the range of emotions expressed in software engineering text.

However, challenges remain in handling the subjectivity and implicit nature of emotions, as well as in generalizing across diverse software artifacts. An important consideration that emerged during our work is the presence of bias in large language models. LLMs, trained on vast amounts of text data, can inadvertently learn and propagate biases present in the training data. This bias can affect emotion detection accuracy, leading to skewed or unfair interpretations of developer communications. We have identified several promising directions for future research to address these limitations.

### 7.2 Future Work

There are a number of interesting directions for future work:

- **Emotion Classification**: Continue refining emotion classification tools by integrating advanced LLMs and experimenting with few-shot, multi-turn, chain-of-thought, and least-to-most prompting.

- **Predictive Models for Emotion Forecasting**: Building on our emotion extraction findings, we aim to develop predictive models that forecast emotions within software development teams. By analyzing historical communication patterns and social dynamics, these models can proactively suggest interven-
tions to improve team outcomes and prevent potential conflicts.

- **Time Series Analysis of Emotion Triggers**: Explore time series analysis techniques to study how emotions and their triggers evolve over the course of software projects. Understanding the temporal dynamics of emotions can provide valuable insights into project health and team dynamics.

- **Correlation with Productivity Metrics**: Quantify the impact of emotions on productivity by correlating extracted emotions and their causes with productivity metrics, such as code commit frequency, bug-fixing time, and team satisfaction. This will provide a deeper understanding of how emotions affect software development processes.

- **Real-Time Emotion Analysis**: Integrate real-time emotion analysis into software development tools to provide immediate feedback and support for developers. Research could focus on developing and testing systems that detect and respond to emotional cues in real-time, enhancing overall communication and collaboration within development teams.

- **Industrial Case Studies on Emotion-Aware Practices**: Conduct in-depth industrial case studies to explore the real-world adoption of emotion-aware practices in software development teams. By collaborating with industry partners, we can assess the feasibility and effectiveness of implementing emotion-aware approaches in practical settings.

- **Ethical Considerations and Bias Mitigation**: Address ethical considerations and mitigate biases in emotion detection models. This includes ensuring that models are fair, transparent, and do not inadvertently reinforce stereotypes or discrimination.
## Appendix A

### ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCU</td>
<td>Virginia Commonwealth University</td>
</tr>
<tr>
<td>RVA</td>
<td>Richmond Virginia</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>LLM</td>
<td>Large Language Model</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>OSS</td>
<td>Open-Source Software</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>DA</td>
<td>Data Augmentation</td>
</tr>
<tr>
<td>FN</td>
<td>False Negative</td>
</tr>
<tr>
<td>FP</td>
<td>False Positive</td>
</tr>
</tbody>
</table>
REFERENCES


[41] Ivaldir de Farias Junior et al. “C2M: a maturity model for the evaluation of communication in distributed software development”. In: *Empirical Software Engineering* 27.7 (2022), p. 188.


[92] Wei Song et al. “Verb metaphor detection via contextual relation learning”. In: Proceedings of the 59th Annual Meeting of the Association for Computational


[261] Mihalis A Nicolaou, Hatice Gunes, and Maja Pantic. “Continuous prediction of spontaneous affect from multiple cues and modalities in valence-arousal


VITA

Mia Mohammad Imran received his Bachelor of Science in Computer Science and Science in 2016 the University of Dhaka in Dhaka, Bangladesh. As a full-time graduate student in the Ph.D. program at Virginia Commonwealth University, his research is focused on human-centric software engineering. During his Ph.D., he has internship experience at Google (2022). Before starting Ph.D., he worked as a Software Engineer for three and half years.

Publications: Publications are available at https://scholar.google.com/citations?user=uVCaRjAAAAAJ


