

January 2024

# Identification And Mitigation Of Toxic Communications Among Open Source Software Developers

Jaydeb Sarker  
*Wayne State University*

Follow this and additional works at: [https://digitalcommons.wayne.edu/oa\\_dissertations](https://digitalcommons.wayne.edu/oa_dissertations)

---

## Recommended Citation

Sarker, Jaydeb, "Identification And Mitigation Of Toxic Communications Among Open Source Software Developers" (2024). *Wayne State University Dissertations*. 4061.  
[https://digitalcommons.wayne.edu/oa\\_dissertations/4061](https://digitalcommons.wayne.edu/oa_dissertations/4061)

This WSU Access is brought to you for free and open access by DigitalCommons@WayneState. It has been accepted for inclusion in Wayne State University Dissertations by an authorized administrator of DigitalCommons@WayneState.

**IDENTIFICATION AND MITIGATION OF TOXIC COMMUNICATIONS AMONG  
OPEN SOURCE SOFTWARE DEVELOPERS**

by

**JAYDEB SARKER**

**DISSERTATION**

Submitted to the Graduate School,

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

**DOCTOR OF PHILOSOPHY**

2024

MAJOR: COMPUTER SCIENCE

Approved By:

---

Advisor	Date
---------	------

---

---

---

## **DEDICATION**

Dedicated to my mother Minati Rani Sarker, and father Jagadish Chandra Sarker,  
my siblings, and my wife.

## ACKNOWLEDGEMENTS

I sincerely acknowledge the support and continuous guidelines from my supervisor, Dr. Amiangshu Bosu. My doctoral journey would not have been smooth without his continuous support and guidelines.

I am also grateful to Professor Dr. Ming Dong, for his support in deep learning concepts during my research and as a dissertation committee member. I sincerely thank my other dissertation committee members, Dr. Suzan Arslanturk and Dr. Gias Uddin, who provided valuable feedback. Dr. Steven R. Wilson supported me by providing valuable information while working on Natural Language Processing in my Software Engineering research.

Moreover, I am thankful to my lab colleague Asif Kamal Turzo, who joined the lab simultaneously and worked on several projects together. Dr. Rajshakhar Paul supported me a lot during the initial years of my Ph.D. Sayma Sultana is my colleague, and we have worked together on several projects. I want to remember the difficult days during my Ph.D. when COVID-19 started in Michigan in 2020. However, with the engagement of my roommates and other Wayne State University (WSU) friends, I recovered from the trauma of those lockdown days for COVID-19. I took several graduate-level courses at WSU that broadened my knowledge, and I am grateful to those faculty members. My parents, elder brother, and siblings inspired me to start my Ph.D. journey. I can not express how helpful they are from my childhood. My beloved wife, Tamalika Saha, came to the USA during the third year of my Ph.D. She supported me a lot by managing many household works. Finally, I would like to thank the Department of Computer Science and Graduate School at Wayne State University for its financial support.

## TABLE OF CONTENTS

Dedication . . . . .	ii
Acknowledgements . . . . .	iii
List of Tables . . . . .	x
List of Figures . . . . .	xii
Chapter 1    Introduction . . . . .	1
1.1   Research Plan . . . . .	1
1.2   Contributions . . . . .	5
1.3   Outline of the Dissertation . . . . .	6
Chapter 2    Background and Related Works . . . . .	8
2.1   What constitutes a toxic communication? . . . . .	8
2.2   Toxic communications in FOSS communities . . . . .	9
2.3   State of the art toxicity detectors . . . . .	11
2.4   Toxic span detection . . . . .	12
2.5   Contexts and consequences of anti-social behaviors in FOSS . . . . .	14
Chapter 3    Automated Identification of Toxic Code Reviews Using ToxiCR . . . . .	16
3.1   Introduction . . . . .	16
3.2   Research Methodology . . . . .	19
3.2.1   Supervised machine learning algorithms . . . . .	19
3.2.1.1   Classical ML algorithms: . . . . .	20
3.2.1.2   Ensemble methods: . . . . .	20
3.2.1.3   Deep neural networks: . . . . .	21
3.2.1.4   Transformer model: . . . . .	23
3.2.2   Word vectorization . . . . .	23
3.2.2.1   Tf-Idf: . . . . .	24

3.2.2.2	Word2vec: . . . . .	25
3.2.2.3	GloVe: . . . . .	25
3.2.2.4	fastText: . . . . .	25
3.2.2.5	BERT: . . . . .	26
3.3	Tool Design . . . . .	26
3.3.1	Conceptualization of Toxicity . . . . .	26
3.3.2	Training Dataset Creation . . . . .	27
3.3.2.1	Data Mining . . . . .	28
3.3.2.2	Stratified sampling of code review comments . . . . .	28
3.3.2.3	Manual Labeling . . . . .	29
3.3.2.4	Dataset aggregation . . . . .	31
3.3.3	Data preprocessing . . . . .	32
3.3.3.1	Mandatory preprocessing . . . . .	32
3.3.3.2	Optional preprocessing . . . . .	34
3.3.4	Word Vectorizers . . . . .	35
3.3.5	Architecture of the ML Models . . . . .	36
3.3.5.1	Classical and ensemble methods . . . . .	36
3.3.5.2	Deep Neural Networks Model . . . . .	37
3.3.5.3	Transformer models . . . . .	40
3.3.6	Model Training and Validation . . . . .	41
3.3.7	Classical and ensembles . . . . .	41
3.3.8	DNN and Transformers . . . . .	41
3.3.9	Tool interface . . . . .	42
3.4	Results . . . . .	45
3.4.1	Experimental Configuration . . . . .	45
3.4.2	Baseline Algorithms . . . . .	46
3.4.3	How do the algorithms perform without optional preprocessing? . . .	49
3.4.3.1	Classical and Ensemble (CLE) algorithms . . . . .	51
3.4.3.2	Deep Neural Networks (DNN) . . . . .	51
3.4.3.3	Transformer . . . . .	52
3.4.4	Do optional preprocessing steps improve performance? . . . . .	52

3.4.5	How do the models perform on another dataset? . . . . .	56
3.4.6	What are the distributions of misclassifications from the best performing model? . . . . .	57
3.4.7	General errors (GE) . . . . .	58
3.4.7.1	SE domain specific words (SE): . . . . .	59
3.4.7.2	Self deprecation (SD): . . . . .	59
3.4.7.3	Bad acronym (BA) . . . . .	60
3.4.7.4	Confounding contexts (CC) . . . . .	60
3.5	Implications . . . . .	61
3.6	Threats to Validity . . . . .	64
3.6.1	Internal validity . . . . .	64
3.6.2	Construct validity . . . . .	65
3.6.3	External validity . . . . .	66
3.6.4	Conclusion validity . . . . .	67
3.7	Conclusion and Future Directions . . . . .	67
Chapter 4	ToxiSpanSE: An Explainable Toxicity Detection in Code Review Comments . . . . .	68
4.1	Introduction . . . . .	68
4.2	Research Method . . . . .	71
4.2.1	Dataset . . . . .	71
4.2.1.1	Dataset Selection . . . . .	71
4.2.1.2	Dataset Annotation . . . . .	72
4.2.1.3	Inter-annotator Agreement . . . . .	73
4.2.1.4	Conflict Resolution and Ground Truth . . . . .	75
4.2.2	Tool Design . . . . .	75
4.2.2.1	Preprocessing . . . . .	75
4.2.2.2	IO Encoding . . . . .	77
4.2.2.3	Lexicon Based Model . . . . .	78
4.2.2.4	Transformer based model . . . . .	79
4.2.2.5	Post Processing . . . . .	81

4.3	Evaluation . . . . .	82
4.3.1	Evaluation Metrics . . . . .	82
4.3.2	Experimental Setup . . . . .	85
4.3.2.1	Hyperparameters . . . . .	85
4.3.2.2	Threshold Selection . . . . .	86
4.3.3	Results with optimal threshold . . . . .	88
4.3.4	Error Analysis from the best model . . . . .	89
4.3.4.1	Partial Disagreement . . . . .	89
4.3.4.2	False Negatives . . . . .	90
4.4	Discussion . . . . .	91
4.5	Threats to Validity . . . . .	92
4.6	Conclusion and Future Work . . . . .	94
Chapter 5	The Landscape of Toxicity: An Empirical Investigation of Antisocial Behaviors on GitHub . . . . .	96
5.1	Introduction . . . . .	96
5.2	Research Method . . . . .	101
5.2.1	Project Selection . . . . .	101
5.2.2	Dataset preparation . . . . .	102
5.2.3	Toxicity classification scheme . . . . .	103
5.2.4	Automated identification of toxic comments . . . . .	104
5.2.5	Manual categorization of toxic comments . . . . .	105
5.2.6	Attribute selection . . . . .	106
5.2.7	Attribute calculation . . . . .	108
5.2.8	Regression Model . . . . .	111
5.2.8.1	Multinomial Logistic Regression for RQ2 . . . . .	111
5.2.8.2	Bootstrapped Logistic Regression for RQ3 and RQ4 . . . . .	113



5.2.8.3	Correlation and redundancy analysis . . . . .	114
5.2.8.4	Model analysis . . . . .	114
5.3	Results . . . . .	115
5.3.1	RQ1: Nature of toxicity in PR Review Comments . . . . .	115
5.3.2	RQ2: Project characteristicss . . . . .	117
5.3.3	RQ3: Pull request context . . . . .	119
5.3.4	RQ4: Participants . . . . .	120
5.4	Discussion . . . . .	122
5.4.1	Comparison with prior SE studies . . . . .	123
5.4.2	Recommendations . . . . .	124
5.5	Threats to Validity . . . . .	126
5.5.1	Internal Validity . . . . .	126
5.5.2	Construct Validity . . . . .	127
5.5.3	External Validity . . . . .	128
5.5.4	Conclusion Validity . . . . .	128
5.6	Conclusion . . . . .	128
Chapter 6	Overall Conclusion and Future Direction . . . . .	130
6.1	Key Outcomes . . . . .	130
6.1.1	A customized toxicity detector for SE domain . . . . .	130
6.1.2	An explainable toxicity detector for Code Review comments . . . . .	131
6.1.3	A large-scale empirical investigation of toxicity on GitHub Pull Re- quests (PR) . . . . .	131
6.2	Directions for Future Work . . . . .	132
6.2.1	Notion of Toxicity according to diverse demographic factors . . . . .	132

6.2.2	Detoxification among the developer’s textual communication . . . . .	133
6.2.3	Promoting politeness among developers’ interaction . . . . .	133
Chapter 7	Appendix . . . . .	134
7.1	Journal Articles . . . . .	134
7.2	Refereed Conference Papers . . . . .	134
7.3	Short Papers . . . . .	135
References	. . . . .	137
Abstract	. . . . .	162
Autobiographical Statement	. . . . .	164

## LIST OF TABLES

Table 1	Anti-social constructs investigated in prior SE studies . . . . .	9
Table 2	An overview of the three SE domain specific toxicity datasets used in this study . . . . .	29
Table 3	Rubric to label the SE text as toxic or non-toxic, adjusted from [153] .	30
Table 4	Examples of text preprocessing steps implemented in ToxiCR . . . . .	32
Table 5	An overview of the hyper parameters for our deep neural networks and transformers . . . . .	41
Table 6	Performances of the four contemporary toxic detectors to establish a baseline performance. For our classifications, we consider toxic texts as the ‘class 1’ and non-toxic texts as the ‘class 0’. . . . .	48
Table 7	Mean performances of the ten selected algorithms based on 10-fold cross validations. For each group, a shaded background indicates significant improvements over the others from the same group . . . . .	50
Table 8	Best performing configurations of each model with optional preprocessing steps. A shaded background indicates significant improvements over its base configuration (i.e., no optional preprocessing). For each column, bold font indicates the highest value for that measure. † – indicates an optional SE domain-specific pre-processing step. . . . .	54
Table 9	Performance of ToxiCR on Gitter dataset . . . . .	57
Table 10	Confusion Matrix for our best performing model (i.e., BERT) for the combined code review dataset . . . . .	57
Table 11	Raw dataset with character spans. <b>Red</b> marked represents selected toxic words . . . . .	73
Table 12	Example of Inter-Rater Agreement and Conflict Resolution . . . . .	73
Table 13	Example of model predictions. <b>red</b> represents the toxic tokens . . . . .	84
Table 14	Experimental Results with the optimal threshold. The runtime of each model and performances during each fold is included in the replication package [152] . . . . .	88
Table 15	Example of some errors . . . . .	90

Table 16	The list of attributes selected to investigate their association with project characteristics (RQ2). . . . .	107
Table 17	The list of attributes selected to investigate their association with Pull request context (RQ3). . . . .	109
Table 18	The list of attributes selected to investigate their association with participants' characteristics (RQ4). . . . .	110
Table 19	Model fit measured using Psuedo $R^2$ and model significance evaluated using the log-likelihood ( $\chi^2$ ) test for the bootstrapped logistic regression models. A 95% confidence interval is also reported for each measure inside the brackets. . . . .	112
Table 20	The most common forms of toxicities within our sample of manually labeled 532 PR comments . . . . .	116
Table 21	Results of our MLR model to identify associations of project characteristics with toxicity. We set the 'Low toxic' group as the reference to compute odds ratios. Hence, $OR > 1$ indicates a higher likelihood of a project transitioning to the 'Medium toxic' or 'High toxic' group with a unit increment of that factor and vice versa. . . . .	118
Table 22	Associations between pull request contexts and toxicity. Values represent the median odds ratio for each factor with 95% confidence intervals inside brackets. . . . .	120
Table 23	Associations between characteristics of authors and targets and toxicity. Values represent the median odds ratio for each factor with 95% confidence intervals inside brackets. . . . .	121

## LIST OF FIGURES

Figure 1	Overview of Dissertation . . . . .	2
Figure 2	A simplified overview of ToxiCR showing key pipeline . . . . .	26
Figure 3	The command line interface of ToxiCR showing various customization options . . . . .	43
Figure 4	Distribution of the misclassifications from the BERT model . . . . .	58
Figure 5	Manual Labeling using Label Studio, toxic span is highlighted . . . . .	72
Figure 6	Model Architecture of ToxiSpanSE. Optimal threshold for each model was empirically selected (detailed in 4.3.2.2), Blue arrow → shows an example . . . . .	76
Figure 7	Threshold variation for RoBERTa model (Using Validation Set) . . . . .	87
Figure 8	An overview of our research method . . . . .	101

## CHAPTER 1 INTRODUCTION

Prior research found evidence of toxic communications among various Free and Open Source Software (FOSS) communities [61, 85, 126, 161]. Although toxic communications are less frequent among FOSS communities than in online platforms such as social media and opinion forums, they may seriously affect the productivity or survival of a FOSS project. For example, being demotivated and frustrated with toxic communications, such as insults, threats, and sexual attacks from their peers, developers may leave a FOSS project [50, 11]. Moreover, such communications often disproportionately impact newcomers, women, and other marginalized groups [165, 89, 77]. Therefore, toxicity is also a barrier to promoting diversity, equity, and inclusion. Although proactive identification and mitigation of toxic communications is crucial, it is challenging for large-scale FOSS communities with thousands of members (e.g., Mozilla, Debian, and OpenStack) to manually check all communications for toxicity. Therefore, an automated identification and mitigation mechanism can significantly help these communities combat toxicity. In this context, my proposed dissertation aims to: *“identify and mitigate toxic communications among open-source software developers.”*

### 1.1 Research Plan

This dissertation aims to achieve the above goal based on three studies. Figure 1 shows an overview of the three studies conducted for this dissertation, which are as follows.

**(Study 1) Development of a customized toxicity detector for the Software Engineering (SE) domain.**

Motivation: Despite the existence of many off-the-shelf toxicity detectors, those perform

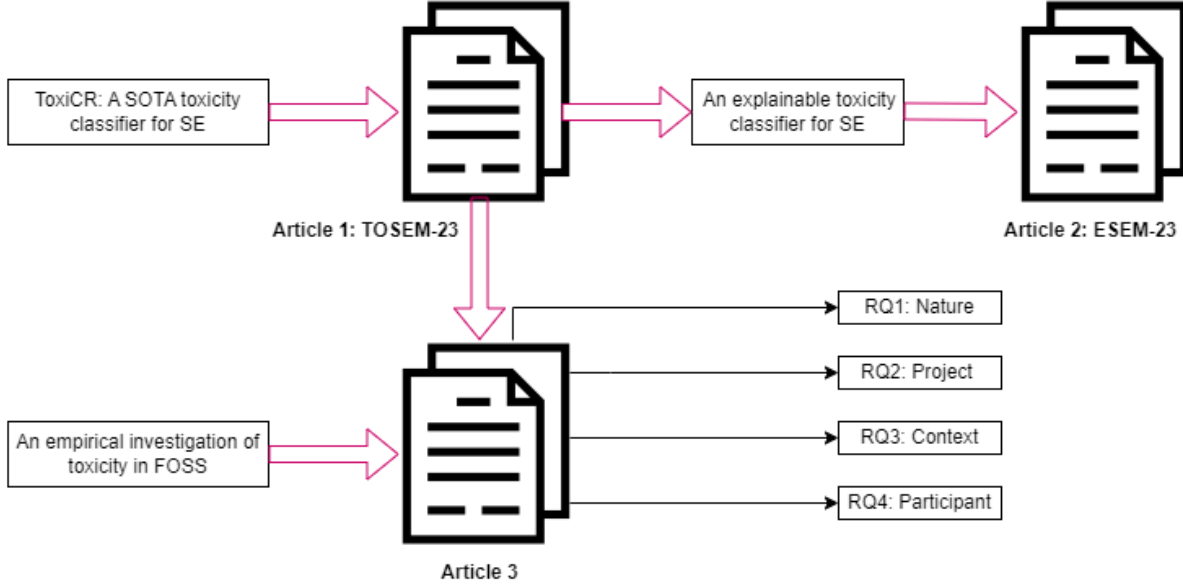


Figure 1: Overview of Dissertation

poorly in SE texts [153]. However, such performance degradation is hardly surprising since prior research on SE domain-specific sentiment analysis tools [91, 122, 4] established needs for SE domain-specific natural language processing (NLP) tools. Although Raman *et al.* [141] developed the first SE domain-specific toxicity detector, it performed poorly in later studies [112, 137, 153]. Therefore, a reliable toxicity detector for the SE domain is needed to better understand toxicity and its impacts.

Objective: To build an SE specific customized state-of-the-art toxicity detector. We anticipate that this tool will help to combat toxicity among FOSS communities.

Approach: First, we have developed a rubric to label an SE text as toxic or non-toxic. Using that rubric, we have manually labeled 19,651 Code Review (CR) texts. The off-the-shelf toxicity detector performed poorly on our labeled SE texts. Further, we have developed *ToxiCR*, a supervised learning-based toxicity detector for CR comments. *ToxiCR* is trained with 19,651 labeled CR comments, where we used ten different machine learn-

ing and deep learning models.

Key findings: This study found that designing a reliable toxicity detector for the SE domain is feasible. Preprocessing steps improve the performance of the models, but they are algorithm-dependent. Although ToxiCR provided reliable performance in our dataset, detecting the self-deprecating texts remains a challenge.

## **(Study 2) Development of an explainable toxicity detection tool for the SE developer’s conversations**

Motivation: A binary toxicity detector in the SE domain (i.e., *ToxiCR*) may help the FOSS community remove a specific text from the conversation. However, it does not help human moderators understand the specific reason(s) for toxicity. It is time-consuming for a moderator to identify the offending excerpt(s) from a large paragraph. Due to the lack of cultural differences, a moderator may fail to recognize the offending sentences from a paragraph classified as toxic by these binary toxicity identification tools. Being motivated by recent advances in explainable machine learning (ML) models, this study aims to create a new SE domain-specific toxicity detector that overcomes this particular shortcoming.

Objective: *To develop an explainable toxicity detector for the Software Engineering domain, which can precisely identify toxic excerpts from a text to assist FOSS moderators.*

Approach: We manually labeled 3,757 CR comments to develop ground-truth annotations for the toxic spans within the toxic samples. Further, we have developed *ToxiSpanSE* that is trained with labeled toxic span dataset. To develop this model, we trained and evaluated five sequence-to-sequence transformer models. During our 10-fold cross-validation-based evaluations, we found a model based on a fine-tuned RoBERTa, achieving the best *F1* score of 0.88.



Key findings: During the manual labeling of toxic spans, we found that toxic span selection is a highly subjective task for annotators. We recommend building a rubric before labeling the text’s toxic span. During the model evaluation of ToxiSpanSE, the lexicon-based approach performed well but lacked generalizability. Our findings suggested that Transformer-based models are reliable and explainable for the FOSS community. Moreover, this study may help SE researchers develop proactive toxic prevention tools.

### **(Study 3) An Empirical Investigation of Toxic Communication During Pull Request Reviews**

Motivation: Although we have developed two state-of-the-art toxicity detectors to automatically detect toxicity and toxic spans, there is a lack of empirical evidence how toxicity harms and which demographics are influenced in FOSS projects. Also, it is unknown how toxicity is associated with other measurable factors on a project. We aim to conduct a large scale empirical analysis of toxicity in FOSS projects that may help the project management i) to take the required steps to mitigate the toxic interactions and ii) to improve the developers’ relations. Moreover, this analysis may help the developers to get an overall idea of the negative impacts of toxicity and motivate them not to use toxic conversations with their peers.

Objective: To investigate the nature of toxicity, project characteristics, contextual factors, and participants with a large-scale empirical analysis.

Approach: We have conducted a large-scale mixed-method empirical study of 2,828 GitHub-based FOSS projects randomly selected based on a stratified sampling strategy. We wrote Python scripts to mine all those projects’ public issue /pull request comments. Our

sample includes 16 million pull requests (PRs), 69.5 million issue comments, and 32 million PR comments. Using *ToxicR*, a state-of-the-art SE domain-specific toxicity detector, we automatically classified each comment as toxic or non-toxic. We also manually investigated a randomly selected sample of 600 comments to characterize toxicity and validate our dataset's performance. To boost a reliable performance on our dataset, we trained several multivariate regression models to investigate the association between toxicity and various factors related to projects, contexts, and participants.

Key findings: Profanity seems to be the dominant form of toxicity during GitHub PRs. Insults, trolling, self-deprecations, and object-directed anger are other common forms in GitHub. Gaming projects are highly associated with toxic comments, and corporate projects are less likely to contain toxicity. While accepted PRs have a negative correlation with the occurrence of toxicity, fixing bugs, code churn, long review intervals, change entropy in the files, and iterative reviews show the opposite correlation. For participants, women and newcomers are less likely to be authors or targets of toxic comments in GitHub PR comments. While Long-term project contributors are less likely to author toxic comments, they are more likely to be targets of toxicity. The developers who use more toxic comments are likely to receive toxic replies.

## 1.2 Contributions

This dissertation provides key insights into the automatic identification and empirical investigation of toxicity in the FOSS domain. The key contributions of this dissertation include:

- An empirical understanding of toxic communications from an SE perspective.

- A rubric to identify an SE text as toxic or nontoxic.
- A manually labeled a large-scale toxicity dataset of 19,651 code review comments.
- ToxiCR, an SE domain-specific toxicity detector. ToxiCR is currently the state-of-the-art toxicity detector for the SE domain. ToxiCR and data are publicly available on GitHub at: <https://github.com/WSU-SEAL/ToxiCR>.
- ToxiSpanSE, the first explainable toxicity detector for the SE domain. The model and data are publicly available: <https://github.com/WSU-SEAL/ToxiSpanSE>.
- An expert-annotated, span-level toxicity labels for 3,757 toxic code review comments.
- An empirical investigation of various categories of toxic communication among GitHub Pull requests.
- A large-scale empirical investigation of factors associated with toxicity on GitHub.
- A set of actionable recommendations to mitigate toxicity from FOSS projects.

### 1.3 Outline of the Dissertation

This dissertation comprises seven chapters. The remainder of this dissertation is organized as follows.

**Chapter 2** discusses the prior works with toxicity in FOSS communities, state-of-the-art toxicity detectors, toxic span detections, and impacts of toxicity in Software Engineering practice.

**Chapter 3** presents the automated identification of toxic code reviews. This chapter is based on publication at the ACM Transactions on Software Engineering and Methodology (TOSEM) 2023 [155]. Apart from that, our works related to this study have been published at the Asia-Pacific Software Engineering Conference (APSEC) 2020 [153] and a student research competition track paper at the Automated Software Engineering Conference (ASE) 2022 [150].

**Chapter 4** presents an explainable toxicity detection for the SE domain. This work is based on the publication at Empirical Software Engineering and Measurement (ESEM) 2023 [151].

We have done a large-scale empirical investigation of toxicity on GitHub Pull Requests Comments, which are discussed in **Chapter 5**.

**Chapter 6** summarizes the dissertation's findings and provides some potential future research directions.

**Chapter 7** summarizes the academic manuscripts up to the dissertation.

## CHAPTER 2 BACKGROUND AND RELATED WORKS

This section defines toxic communications, provides a brief overview of prior works on toxicity in FOSS communities, describes state-of-the-art toxicity detectors, toxic span detection, and some empirical evidence of the impacts of toxicity in open source.

### 2.1 What constitutes a toxic communication?

The term ‘toxicity’ represents the negative or antisocial interactions in online conversations [10]. A report from [10] showed that 47% Americans experienced harassment and abuse during online communication, and toxicity deters users from online engagement. Toxicity is a complex phenomenon to construct as it is more subjective than other text classification problems (i.g., online abuse, spam) [99], and often subject to the opinions of beholders [104]. A broader view of toxicity is that of an umbrella of various antisocial behaviors such as hate speech, cyberbullying, trolling, and flaming [112]. Whether a communication should be considered as ‘toxic’ also depends on a multitude of factors, such as communication medium, location, culture, and relationship between the participants. In this dissertation, we focus especially on written online communications. According to the Google Jigsaw AI team, a text from an online communication can be marked as toxic if it contains disrespectful or rude comments that make a participant leave the discussion forum [5]. On the other hand, the Pew Research Center marks a text as toxic if it contains a threat, offensive call, or sexually expletive words [55]. Anderson *et al.*’s definition of toxic communication also includes insulting language or mockery [9]. Adinolf and Turkay studied toxic communication in online communities and their views of toxic communications include harassment, bullying, grieving (i.e., constantly making other players annoyed), and

SE study		Construct	Definition
Sarker al. [153]	et	Toxicity	<i>“includes any of the following: i) offensive name calling, ii) insults, iii) threats, iv) personal attacks, v) flirtations, vi) reference to sexual activities, and vii) swearing or cursing.”</i>
Miller al. [112]	et	Toxicity	<i>“an umbrella term for various antisocial behaviors including trolling, flaming, hate speech, harassment, arrogance, entitlement, and cyberbullying”.</i>
Ferreira al. [63]	et	Incivility	<i>“features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics”</i>
Gunawardena et al. [77]		Destructive criticism	<i>negative feedback which is nonspecific and is delivered in a harsh or sarcastic tone, includes threats, or attributes poor task performance to flaws of the individual.</i>
Egelman al. [57]	et	Pushback	<i>“the perception of unnecessary interpersonal conflict in code review while a reviewer is blocking a change request”</i>

Table 1: Anti-social constructs investigated in prior SE studies

trolling [3]. To understand, how persons from various demographics perceive toxicity, Kumar *et al.* conducted a survey with 17,280 participants inside the USA. To their surprise, their results indicate that the notion of toxicity cannot be attributed to any single demographic factor [99]. According to Miller *et al.*, various antisocial behaviors fit inside the Toxicity umbrella such as hate speech, trolling, flaming, and cyberbullying [112]. While some of the SE studies have investigated antisocial behaviors among SE communities using the ‘toxicity’ construct [153, 141, 112], other studies have used various other lenses such as incivility [63], pushback [57], and destructive criticism [77]. Table 1 provides a brief overview of the studied constructs and their definitions.

## 2.2 Toxic communications in FOSS communities

FOSS communities have reported toxic content in developers’ communications in blog posts [13, 53], podcasts [54], and talks [142]. Large open-source foundations face an increase in toxicity in their groups. For example, the Linux Community experiences toxic-

ity [53], and the founder apologized for using toxicity in Linux Kernel Mailing lists [183]. By conducting a survey, the Perl Foundation found that several members stepped down due to receiving abusive messages [145]. Several other studies also have identified toxic communications in FOSS communities [161, 141, 153, 33, 126]. Squire and Gazda found occurrences of expletives and insults in publicly available IRC and mailing-list archives of top FOSS communities, such as Apache, Debian, Django, Fedora, KDE, and Joomla [161]. More alarmingly, they identified sexist ‘maternal insults’ being used by many developers. Recent studies have also reported toxic communications among issue discussions on Github [141] and during code reviews [153, 126, 77, 63]. Miller *et al.* conducted a qualitative study to better understand toxicity in the context of FOSS development [112]. They created 100 Github issues representing various types of toxic interactions, such as insults, arrogance, trolling, entitlement, and unprofessional behavior. Their analyses also suggest toxicity in FOSS communities differs from those observed on online platforms such as Reddit or Wikipedia [112].

Ferreira *et al.* [63] investigated incivility during code review discussions based on a qualitative analysis of 1,545 emails from Linux Kernel Mailing Lists and found that the most common forms of incivility among those forums are frustration, name-calling, and importance. Egelman *et al.* studied the negative experiences during code review, which they referred to as “pushback”, a scenario when a reviewer blocks a change request due to unnecessary conflict [57]. Qiu *et al.* further investigated such “pushback” phenomena to automatically identify interpersonal conflicts [137]. Gunawardena *et al.* investigated negative code review feedback based on a survey of 93 software developers, and they found that destructive criticism can be a threat to gender diversity in the software industry as

women are less motivated to continue when they receive negative comments or destructive criticisms [77].

### 2.3 State of the art toxicity detectors

To combat abusive online content, Google’s Jigsaw AI team developed the Perspective API (PPA), which is publicly available [5]. PPA is one of the general purpose state-of-the-art toxicity detectors. For a given text, PPA generates the probability (0 to 1) of that text being toxic. As researchers are working to identify adversarial examples to deceive the PPA [83], the Jigsaw team periodically updates it to eliminate identified limitations. The Jigsaw team also published a guideline [6] to manually identify toxic contents and used that guideline to curate a crowd-sourced labeled dataset of toxic online contents [92]. This dataset has been used to train several deep neural network based toxicity detectors [68, 35, 59, 162, 76, 180]. Recently, Bhat *et al.* proposed *ToxiScope*, a supervised learning-based classifier to identify toxic in workplace communications [20]. However, *ToxiScope*’s best model achieved a low F1-Score (i.e., =0.77) during their evaluation

One of the major challenges in developing toxicity detectors is character-level obfuscations, where one or more characters of a toxic word are intentionally misplaced (e.g., fcuk), repeated (e.g., shiiit), or replaced (e.g., s\*ck), to avoid detection. To address this challenge, researchers have used character-level encoders instead of word-level encoders to train neural networks [113, 120, 100]. Although character-level encoding-based models can handle such character-level obfuscations, they come with significant increments of computation times [100]. Several studies have also found racial and gender bias among contemporary toxicity detectors, as some trigger words (i.e., ‘gay’, ‘black’) are more likely



to be associated with false positives (i.e, a nontoxic text marked as toxic) [179, 187, 148].

According to our best information, to combat toxicity in open source, Raman *et al.* are the first ones to develop a toxicity classifier trained with a small-scale GitHub issue discussion dataset [141] by leveraging the PPA tool and a customized version of Stanford’s Politeness Detector [49]. However, several studies showed that their tool [141] performed poorly on the large-scale SE texts [153, 137, 112, 150]. In 2022, Qiu *et al.* developed a classifier for identifying interpersonal conflicts during code reviews [137]. Cheriyan *et al.* developed a classifier to detect swearing and profanity [38] for four different SE platforms. However, off-the-shelf toxicity detectors suffer significant performance degradation on SE datasets [153]. Such degradation is not surprising since prior studies found off-the-shelf natural language processing (NLP) tools also performing poorly on SE datasets [91, 4, 122, 105]. We conducted a benchmark study and investigated the performance of the STRUDEL tool and four other off-the-shelf toxicity detectors on two SE datasets [153]. In our benchmark study [153], none of the tools achieved reliable performance to justify practical applications on SE datasets. However, we also achieved encouraging performance boosts when we retrained two of the tools (i.e., DPCNN [195] and BERT with FastAI [100]) using their SE datasets.

## 2.4 Toxic span detection

Although the detection of toxicity [129, 20, 68], hate speech [30, 31, 69], and offensive language [37, 86] are common in online platforms, the idea of span detection of toxicity has only more recently gained attention with the SemEval-2021 toxic span detection task [127]. Toxic spans represent a part of the text that is responsible for the

toxicity of the posts [130]. This direction is inspired by prior NLP studies on aspect-based sentiment analysis [188, 135], which aims to detect the sentiment of a text and find the specific region of a text that expresses the sentiment using attention-based deep neural network models [188]. While earlier studies focused on the attention mechanism championed by Vaswani *et al.* [182], Sen *et al.* found that machine attention does not reliably overlap with human attention maps [160]. To improve explainability using attention-based mechanisms, recent works have proposed transformer-based sequence-to-sequence models [34, 136].

In the SemEval-2021 task, Pavlopoulos *et al.* provided a labeled dataset [127] of toxic spans with 10,000 samples [130] curated from the Civil Comments dataset. The raters marked the span that corresponds to the toxicity of a text. The task is a binary classification because it contains toxic and non-toxic tokens. Moreover, they fixed the ground truth of the dataset if the majority of the raters labeled the span as toxic. Ninety-one teams made submissions with different methods in the SemEval-2021 competition to detect toxic spans [127]. One of the teams proposed a BERT-based ensemble method toxic span detection approach where they achieved 70.83% *F1* score and secured first place in SemEval-2021 task [197]. A RoBERTa-based method performed only slightly worse, with a 70.77% *F1* score, and other approaches based on fine-tuning of pre-trained transformer models ([172, 39]) also performed well for that toxic span detection tasks. Since several studies worked on toxic span detection for online civil comments, Pavlopoulos *et al.* annotated a new dataset for toxic to civil transfer [128]. Although several studies have proposed toxic span detectors for online comments, no such tool exists for the SE domain. Since NLP tools may not work reliably on a cross-domain dataset [91], developing and

evaluating an SE domain-specific toxic span detector is essential. Such a tool will not only enable a finer-grained analysis of toxicity but also enable proactive notification to authors.

## 2.5 Contexts and consequences of anti-social behaviors in FOSS

Prior SE studies investigated contexts and consequences of anti-social behaviors using surveys and qualitative analyses. These studies suggest toxic interactions among FOSS developers as a ‘poison’ that not only degrades their mental health but [33] also can cause stress and burnouts [141]. The threat of a FOSS community disintegrating rises with the levels of toxicity due to developers’ turnover [33]. Miller *et al.*’s investigation of 100 locked issues on GitHub found toxicity originated from newcomers and experienced contributors due to technological disagreements, frustrations with a system, and past interactions with the target [112]. Project sponsorship and domain may influence toxicity as corporate association decreases toxicity [141], but belonging to the gaming domain increases [112]. A project’s toxicity may also decrease with age [141]. While uncivil discussions may arise in various locked issue contexts, they are more common among versioning and licensing discussions [62]. On the Linux kernel mailing list, inappropriate feedback from maintainers and violation of community conventions are the top causes of incivility [63]. On the other hand, among industrial developers, excessive workloads and poor-quality code are top factors [140]. Two lenses of antisocial behaviors, destructive criticism and pushback, are specific to code reviews. They occur due to unnecessary harsh critiques of code and interpersonal conflicts caused by disagreements over development directions [57, 115, 77]. Both pushback and destructive criticisms not only decrease productivity and degrade interpersonal relationships [115, 57], they disproportionately harm underrepresented mi-

norities and barriers to promoting DEI [77]. Besides these academic works, several gray literature have also documented burnouts and turnover of long-term FOSS contributors due to toxicity [13, 53, 183, 103, 145, 11, 186].

## CHAPTER 3 AUTOMATED IDENTIFICATION OF TOXIC CODE REVIEWS USING TOXICR

### 3.1 Introduction

Communications among the members of many Free and Open Source Software(FOSS) communities include manifestations of toxic behaviours [161, 61, 13, 117, 85, 126]. These toxic communications may have decreased the productivity of those communities by wasting valuable work hours [23, 141]. FOSS developers being frustrated over peers with ‘prickly’ personalities [25, 64] may contemplate leaving a community for good [50, 11]. Moreover, as most FOSS communities rely on contributions from volunteers, attracting and retaining prospective joiners is crucial for the growth and survival of FOSS projects [139]. However, toxic interactions with existing members may pose barriers against the successful onboarding of newcomers [89, 165]. Therefore, it is crucial for FOSS communities to proactively identify and regulate toxic communications.

Large-scale FOSS communities, such as Mozilla, OpenStack, Debian, and GNU, manage hundreds of projects and generate large volumes of text-based communications among their contributors. Therefore, it is highly time-consuming and infeasible for the project administrators to identify and intervene in ongoing toxic communications in a timely manner. Although many FOSS communities have codes of conduct, those are rarely enforced due to time constraints [11]. As a result, toxic interactions can be easily found within the communication archives of many well-known FOSS projects. As an anonymous FOSS developer wrote after leaving a toxic community, “*..it’s time to do a deep dive into the mailing list archives or chat logs. ... Searching for terms that degrade women (chick, babe, girl, bitch, cunt), homophobic slurs used as negative feedback (“that’s so gay”), and ableist terms (dumb,*

*retarded, lame*), may allow you to get a sense of how aware (or not aware) the community is about the impact of their language choice on minorities.” [11]. Therefore, it is crucial to develop an automated tool to identify toxic communication in FOSS communities.

Toxic text classification is a Natural Language Processing (NLP) task to automatically classify a text as ‘toxic’ or ‘non-toxic’. There are several state-of-the-art tools to identify toxic content in blogs and tweets [5, 100, 76, 8]. However, off-the-shelf toxicity detectors do not work well on Software Engineering (SE) communications [153], since several characteristics of such communications (e.g., code reviews and bug interactions) are different from those of blogs and tweets. For example, compared to code review comments, tweets are shorter and are limited to a maximum length. Tweets rarely include SE domain-specific technical jargon, URLs, or code snippets [4, 153]. Moreover, due to different meanings of some words (e.g., ‘kill’, ‘dead’, and ‘dumb’) in the SE context, SE communications with such words are often incorrectly classified as ‘toxic’ by off-the-shelf toxicity detectors [141, 153].

To encounter this challenge, Raman *et al.* developed a toxicity detector tool (referred as the ‘STRUDEL tool’ hereinafter) for the SE domain [141]. However, as the STRUDEL tool was trained and evaluated with only 611 SE texts. Recent studies have found that it performed poorly on new samples [137, 112]. To further investigate these concerns, Sarker *et al.* conducted a benchmark study of the STRUDEL tool and four other off-the-shelf toxicity detectors using two large scale SE datasets [153]. To develop their datasets, they empirically developed a rubric to determine which SE texts should be placed in the ‘toxic’ group during their manual labeling. Using that rubric, they manually labeled a dataset of 6,533 code review comments and 4,140 Gitter messages [153]. The results of their analyses suggest that none of the existing tools are reliable in identifying toxic texts

from SE communications, since all the five tools’ performances significantly degraded on their SE datasets. However, they also found noticeable performance boosts (i.e., accuracy improved from 83% to 92% and F-score improved from 40% to 87%) after retraining two of the existing off-the-shelf models (i.e., DPCNN [195] and BERT with FastAI [100]) using their datasets. Being motivated by these results, we hypothesize that a SE domain specific toxicity detector can boost even better performances, since off-the-shelf toxicity detectors do not use SE domain specific preprocessing steps, such as preprocessing of code snippets included within texts. On this hypothesis, this paper presents ToxiCR, a SE domain specific toxicity detector. ToxiCR is trained and evaluated using a manually labeled dataset of 19,651 code review comments selected from four popular FOSS communities (i.e., Android, Chromium OS, OpenStack and LibreOffice). We selected code review comments since a code review usually represents a direct interaction between two persons (i.e., the author and a reviewer). Therefore, a toxic code review comment has the potential to be taken as a personal attack and may hinder future collaboration between the participants. ToxiCR is written in Python using the Scikit-learn [131] and TensorFlow [1]. It provides an option to train models using one of the ten supervised machine learning algorithms including five classical and ensemble-based, four deep neural network-based, and a Bidirectional Encoder Representations from Transformer (BERT) based ones. It also includes eight preprocessing steps with two being SE domain specific and an option to choose from five different text vectorization techniques.

We empirically evaluated various optional preprocessing combinations for each of the ten algorithms to identify the best performing combination. During our 10-fold cross-validations evaluations, the best performing model of ToxiCR significantly outperforms

existing toxicity detectors on the code review dataset with an accuracy of 95.8% and F-score of 88.9%.

The primary contributions of this paper are:

- ToxiCR: An SE domain specific toxicity detector. ToxiCR is publicly available on GitHub at: <https://github.com/WSU-SEAL/ToxiCR>.
- An empirical evaluation of ten machine learning algorithms to identify toxic SE communications.
- Implementations of eight preprocessing steps including two SE domain specific ones that can be added to model training pipelines.
- An empirical evaluation of three optional preprocessing steps in improving the performances of toxicity classification models.
- Empirical identification of the best possible combinations for all the ten algorithms.

## 3.2 Research Methodology

To better understand our tool design, this section provides a brief overview of the machine learning (ML) algorithms integrated with ToxiCR and five word vectorization techniques for NLP tasks.

### 3.2.1 Supervised machine learning algorithms

For ToxiCR we selected ten supervised ML algorithms from the ones that have been commonly used for text classification tasks. Our selection includes three classical, two ensemble methods based, four deep neural networks (DNN) based, and a Bidirectional



Encoder Representations from Transformer (BERT) based algorithms. The following subsections provide a brief overview of the selected algorithms.

**3.2.1.1 Classical ML algorithms:** We have selected the following three classical algorithms, which have been previously used for the classification of SE texts [4, 32, 101, 174, 174].

1. Decision Tree (DT) : In this algorithm, the dataset is continuously split according to a certain parameter. DT has two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split into two or more sub-nodes [138].
2. Logistic Regression (LR): LR creates a mathematical model to predict the probability for one of the two possible outcomes and is commonly used for binary classification tasks [18].
3. Support-Vector Machine (SVM): After mapping the input vectors into a high dimensional non-linear feature space, SVM tries to identify the best hyperplane to partition the data into n-classes, where n is the number of possible outcomes [44].

**3.2.1.2 Ensemble methods:** Ensemble methods create multiple models and then combine them to produce improved results. We have selected the following two ensemble methods based algorithms, based on prior SE studies [4, 101, 174].

1. Random Forest (RF): RF is an ensemble based method that combines the results produced by multiple decision trees [80]. RF creates independent decision trees and combines them in parallel using on the ‘bagging’ approach [28].

2. Gradient-Boosted Decision Trees (GBT): Similar to RF, GBT is also an ensemble based method using decision trees [67]. However, GBT creates decision trees sequentially, so that each new tree can correct the errors of the previous one and combine the results using the ‘boosting’ approach [158].

**3.2.1.3 Deep neural networks:** In recent years, DNN based models have shown significant performance gains over both classical and ensemble based models in text classification tasks [195, 96]. In this research, we have selected four state-of-the-art DNN based algorithms.

1. Long Short Term Memory (LSTM): A Recurrent Neural Network (RNN) processes inputs sequentially, remembers the past, and makes decisions based on what it has learned from the past [144]. However, traditional RNNs may perform poorly on long-sequence inputs, such as those seen in text classification tasks due to ‘the vanishing gradient problem’. This problem occurs when an RNN’s weights are not updated effectively due to exponentially decreasing gradients. To overcome this limitation, Hochreiter and Schmidhuber proposed LSTM, a new type of RNN architecture, that overcomes the challenges posed by long-term dependencies using a gradient-based learning algorithm [81]. LSTM consists of four units: i) input gate, which decides what information to add from the current step, ii) forget gate, which decides what is to keep from prior steps, iii) output gate, which determines the next hidden state, and iv) memory cell, stores information from previous steps.
2. Bidirectional LSTM (BiLSTM): A BiLSTM is composed of a forward LSTM and a backward LSTM to model the input sequences more accurately than a unidirectional

LSTM [74, 43]. In this architecture, the forward LSTM takes input sequences in the forward direction to model information from the past, while the backward LSTM takes input sequences in the reverse direction to model information from the future [84]. BiLSTM has been shown to be performing better than the unidirectional LSTM in several text classification tasks, as it can identify language contexts better than LSTM [74].

3. Gated Recurrent Unit (GRU): Similar to LSTM, GRU belongs to the RNN family of algorithms. However, GRU aims to handle ‘the vanishing gradient problem’ using a different approach than LSTM. GRU has a much simpler architecture with only two units, an update gate and a reset gate. The reset gate decides what information should be forgotten for the next pass and the update gate determines which information should pass to the next step. Unlike LSTM, GRU does not require any memory cell and therefore needs shorter training time than LSTM [60].
4. Deep Pyramid CNN (DPCNN): Convolutional neural networks (CNN) are a specialized type of neural network that utilizes a mathematical operation called convolution in at least one of their layers. CNNs are most commonly used for image classification tasks. Johnson and Zhang proposed a special type of CNN architecture, named deep pyramid convolutional neural network (DPCNN) for text classification tasks [90]. Although DPCNN achieves faster training time by utilizing word-level CNNs to represent input texts, it does not sacrifice accuracy over character-level CNNs due to its carefully designed deep but low-complexity network architecture.

**3.2.1.4 Transformer model:** In recent years, Transformer based models have been used for sequence-to-sequence modeling such as neural machine translations. For a sequence to sequence modeling, a Transformer architecture includes two primary parts: i) *the encoder*, which takes the input and generates the higher dimensional vector representation, and ii) *the decoder*, which generates the output sequence from the abstract vector from the encoder. For classification tasks, the output of encoders is used for training. Transformers solve the ‘vanishing gradient problem’ on long text inputs using the ‘self attention mechanism’, a technique to identify the important features from different positions of an input sequence [182].

In this study, we select Bidirectional Encoder Representations from Transformers, which is commonly known as BERT [52]. BERT based models have achieved remarkable performances in various NLP tasks, such as question answering, sentiment classification, and text summarization [2, 52].

BERT’s transformer layers use multi-headed attention instead of recurrent units (e.g., LSTM, GRU) to model the contextualized representation of each word in an input.

### 3.2.2 Word vectorization

To train an NLP model, input texts need to be converted into a vector of features that machine learning models can work on. *Bag-of-Words (BOW)* is one of the most basic representation techniques, that turns an arbitrary text into a fixed-length vector by counting how many times each word appears. As BOW representations do not account for grammar and word order, ML models trained using BOW representations fail to identify relationships between words. On the other hand, *word embedding* techniques convert words to

n-dimensional vector forms in such a way that words having similar meanings have vectors close to each other in the n-dimensional space. Word embedding techniques can be further divided into two categories: i) context-free embedding, which creates the same representation of a word regardless of the context where it occurs; ii) contextualized word embeddings aim at capturing word semantics in different contexts to address the issue of polysemous (i.e., words with multiple meanings) and the context-dependent nature of words. For this research, we have experimented with five word vectorization techniques: one BOW based, three context-free, and one contextualized. The following subsections provide a brief overview of those techniques.

**3.2.2.1 Tf-Idf:** TF-IDF is a BOW based vectorization technique that evaluates how relevant a word is to a document in a collection of documents. TF-IDF score for a word is computed by multiplying two metrics: how many times a word appears in a document ( $Tf$ ), and the inverse document frequency of the word across a set of documents ( $Idf$ ). The following equations show the computation steps for Tf-Idf scores.

$$Tf(w, d) = f(w, d) / \sum_{t \in d} f(t, d) \quad (3.1)$$

Where,  $f(t, d)$  is the frequency of the word ( $w$ ) in the document ( $d$ ), and  $\sum_{t \in d} f(t, d)$  represents the total number of words in  $d$ . Inverse document frequency ( $Idf$ ) measures the importance of a term across all documents.

$$Idf(w) = \log_e(N/w_N) \quad (3.2)$$

Here,  $N$  is the total number of documents and  $w_N$  represents the number of documents having  $w$ . Finally, we computed the Tfidf score of a word as:

$$TfIdf(w, d) = Tf(w, d) * Idf(w) \quad (3.3)$$

**3.2.2.2 Word2vec:** In 2013, Mikolaev *et al.* [111] proposed Word2vec, a context free word embedding technique. It is based on two neural network models named Continuous Bag-of-Words (CBOW) and Skip-gram. CBOW predicts a target word based on its context, while skip-gram uses the current word to predict its surrounding context. During the training, word2vec takes a large corpus of text as input and generates a vector space, where each word in the corpus is assigned a unique vector, and words with similar meaning are located close to one another.

**3.2.2.3 GloVe:** Proposed by Pennington *et al.* [132], Global Vectors for Word Representation (GloVe) is an unsupervised algorithm to create context-free word embedding. Unlike word2vec, GloVe generates vector space from the global co-occurrence of words.

**3.2.2.4 fastText:** Developed by the Facebook AI team, *fastText* is a simple and efficient method to generate context-free word embeddings [21]. While Word2vec and GloVe cannot provide embedding for out-of-vocabulary words, fastText overcomes this limitation by taking into account the morphological characteristics of individual words. A word's vector in fastText based embedding is built from vectors of substrings of characters contained in it. Therefore, fasttext performs better than Word2vec or GloVe in NLP tasks, if a corpus contains unknown or rare words [21].

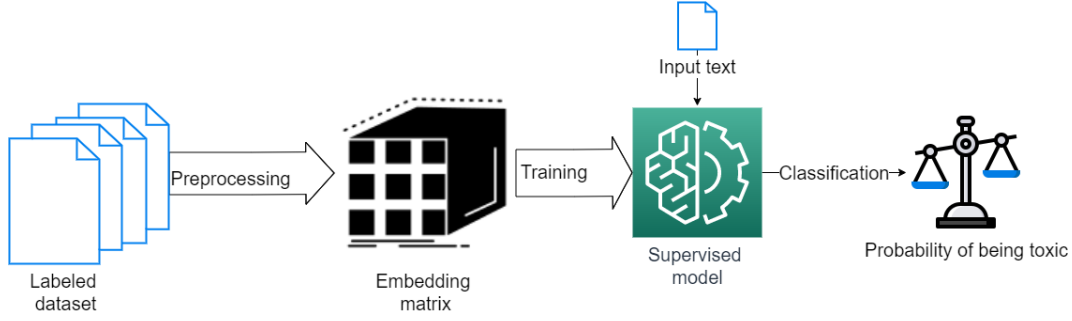


Figure 2: A simplified overview of ToxiCR showing key pipeline

**3.2.2.5 BERT:** Unlike context-free embeddings (e.g., word2vec, GloVe, and fastText), where each word has a fixed representation regardless of the context within which the word appears, a contextualized embedding produces word representations that are dynamically informed by the words around them. In this study, we use BERT [52]. Similar to fastText, BERT can also handle out of vocabulary words.

### 3.3 Tool Design

Figure 2 shows the architecture of ToxiCR. It takes a text ( i.e., code review comment) as input and applies a series of mandatory preprocessing steps. Then, it applies a series of optional preprocessing based on selected configurations. Preprocessed texts are then fed into one of the selected vectorizers to extract features. Finally, output vectors are used to train and validate our supervised learning-based models. The following subsections detail the research steps to design ToxiCR.

#### 3.3.1 Conceptualization of Toxicity

As we have mentioned in Section 2.1, what constitutes a ‘toxic communication’ depends on various contextual factors. In this study, we specifically focus on popular FOSS projects such as Android, Chromium OS, LibreOffice, and OpenStack, where participants represent

diverse cultures, education, ethnicity, age, religion, gender, and political views. As participants are expected and even recommended to maintain a high level of professionalism during their interactions with other members of those communities [123, 134, 66], we adopt the following expansive definition of toxic contents for this context.<sup>1</sup>:

*“An SE conversation will be considered toxic, if it includes any of the following: i) offensive name calling, ii) insults, iii) threats, iv) personal attacks, v) flirtations, vi) reference to sexual activities, and vii) swearing or cursing.”*

Our conceptualization of toxicity closely aligns with another recent work by Bhat *et al.* that focuses on professional workplace communication [20]. According to their definition, toxic behavior includes any of the following: sarcasm, stereotyping, rude statements, mocking conversations, profanity, bullying, harassment, discrimination, and violence.

### 3.3.2 Training Dataset Creation

As of May 2021, there are three publicly available labeled datasets of toxic communications from the SE domain. Raman *et al.*’s dataset created for the STRUDEL tool [141] includes only 611 texts. In our recent benchmark study (referred to as ‘the benchmark study’ hereinafter), we created two datasets, i) a dataset of 6,533 code review comments selected from three popular FOSS projects (referred as ‘code review dataset 1’ hereinafter), i.e., Android, Chromium OS, and LibreOffice; ii) a dataset of 4,140 Gitter messages selected from the Gitter Ethereum channel (referred as ‘gitter dataset’ hereinafter) [153]. We followed the exact same process used in the benchmark study to select and label an additional 13,038 code review comments selected from the OpenStack projects. In the following, we

---

<sup>1</sup>We introduced this definition in our prior study [153]. We are repeating this definition to assist better comprehension of this paper’s context



briefly describe our four-step process, which is detailed in our prior publication [153].

**3.3.2.1 Data Mining** In the benchmark, we wrote a Python script to mine the Gerrit [114] managed code review repositories of three popular FOSS projects, i.e., Android, Chromium OS, and LibreOffice. Our script leverages Gerrit’s REST API to mine and store all publicly available code reviews in a MySQL dataset. We use the same script to mine  $\approx 2.1$  million code review comments belonging to 670,996 code reviews from the OpenStack projects’ code review repository hosted at <https://review.opendev.org/>. We followed an approach similar to Paul *et al.* [126] to identify potential bot accounts based on keywords (e.g., ‘bot’, ‘auto’, ‘build’, ‘auto’, ‘travis’, ‘CI’, ‘jenkins’, and ‘clang’). If our manual validations of comments authored by a potential bot account confirmed it to be a bot, we excluded all comments posted by that account.

**3.3.2.2 Stratified sampling of code review comments** Since toxic communications are rare [153] during code reviews, a randomly selected dataset of code review comments will be highly imbalanced with less than 1% toxic instances. To overcome this challenge, we adopted a stratified sampling strategy as suggested by Särndal *et al.* [157]. We used Google’s Perspective API (PPA) [5] to compute the toxicity score for each review comment. If the PPA score is more than 0.5, then the review comment is more likely to be toxic. Among the 2.1 million code review comments, we found 4,038 comments with PPA scores greater than 0.5. In addition to those 4,038 review comments, we selected 9,000 code review comments with PPA scores less than 0.5. We selected code review comments with PPA scores less than 0.5 in a well-distributed manner. We split the texts into five categories

Table 2: An overview of the three SE domain specific toxicity datasets used in this study

<b>Dataset</b>	<b># total texts</b>	<b># toxic</b>	<b># non-toxic</b>
Code review 1	6,533	1,310	5,223
Code review 2	13,038	2,447	10,591
Gitter dataset	4,140	1,468	2,672
Code review (combined)	19,571	3,757	15,819

(i.e, score: 0-0.1, 0.11-0.2, and so on) and took the same amount (1,800 texts) from each category. For example, we took 1,800 samples with a score between 0.3 and 0.4.

**3.3.2.3 Manual Labeling** During the benchmark study [153], we developed a manual labeling rubric fitting our definition and the study context. Our initial rubric was based on the guidelines the Conversation AI Team (CAT) published [6]. With these guidelines as our starting point, two authors independently went through 1,000 texts to adopt the rules to fit our context better. Then, we had a discussion session to merge and create a unified set of rules. Table 3 represents our final rubric used for manual labeling during both the benchmark study and this study.

Although we have used the guideline from the CAT as our starting point, our final rubric differs from the CAT rubric in two key aspects to better fit our target SE context. First, our rubric is targeted toward professional communities in contrast to the CAT rubric, which is targeted toward general online communications. Therefore, profanities and swearing to express a positive attitude may not be considered as toxic by the CAT rubric. For example, “That’s fucking amazing! thanks for sharing.” is given as an example of ‘Not Toxic, or Hard to say’ by the CAT rubric. On the contrary, any sentence with profanities or swearing is considered ‘toxic’ according to our rubric since such a sentence does not constitute a healthy interaction. Our characterization of profanities also aligns with the recent SE studies on

Table 3: Rubric to label the SE text as toxic or non-toxic, adjusted from [153]

#	Rule	Rationale	Example*
Rule 1:	If a text includes profane or curse words it would be marked as 'toxic'.	Profanities are the most common sources of online toxicities.	<i>"fuck! Consider it done!"</i>
Rule 2:	If a text includes an acronym, that generally refers to expletive or swearing, it would be marked as 'toxic'.	Sometimes people use acronyms of profanities, which are equally toxic as their expanded form.	<i>"WTF are you doing!"</i>
Rule 3:	Insulting remarks regarding another person or entities would be marked as 'toxic'.	Insulting another developer may create a toxic environment and should not be encouraged.	<i>"...shut up, smartypants."</i>
Rule 4:	Attacking a person's identity (e.g., race, religion, nationality, gender or sexual orientation) would be marked as 'toxic'.	Identity attacks are considered toxic among all categories of online conversations.	<i>"Stupid fucking superstitious Christians."</i>
Rule 5:	Aggressive behavior or threatening another person or a community would be marked as 'toxic'.	Aggregations or threats may stir hostility between two developers and force the recipients to leave the community.	<i>"yeah, but I'd really give a lot for an opportunity to punch them in the face."</i>
Rule 6:	Both implicit or explicit References to sexual activities would be marked as 'toxic'.	Implicit or explicit references to sexual activities may make some developers, particularly females, uncomfortable and make them leave a conversation.	<i>"This code makes me so horny. It's beautiful."</i>
Rule 7:	Flirtations would be marked as 'toxic'.	Flirtations may also make a developer uncomfortable and make a recipient avoid the other person during future collaborations	<i>"I really miss you my girl".</i>
Rule 8:	If a demeaning word (e.g., 'dumb', 'stupid', 'idiot', 'ignorant') refers to either the writer him/herself or his/her work, the sentence would not be marked as 'toxic', if it does not fit any of the first seven rules.	It is common in the SE community to use those words to express their own mistakes. In those cases, the use of those toxic words to themselves or their does not make toxic meaning. While such texts are unprofessional [112], those do not degrade future communication or collaboration.	<i>"I'm a fool and didn't get the point of the deincrement. It makes sense now."</i>
Rule 9:	A sentence, that does not fit rules 1 through 8, would be marked as 'non-toxic'.	General non-toxic comments.	<i>"I think Resource-WithProps should be there instead of GenericResource."</i>

\* Examples are provided verbatim from the datasets, to accurately represent the context. We did not censor any text, except omitting the reference to a person's name.

toxicity [112] and incivility [63]. Second, the CAT rubric is for labeling on a four-point scale (i.e., ‘Very Toxic’, ‘Toxic’, ‘Slightly Toxic or Hard to Say’, and ‘Non toxic’) [6]. On the contrary, our labeling rubric is much simpler on a binary scale (‘Toxic’ and ‘Non-toxic’), since the development of a four-point rubric as well as classifier is significantly more challenging. We consider the development of a four-point rubric as a potential future direction.

Using this rubric, two authors independently labeled the 13,038 texts as either ‘toxic’ or ‘non-toxic’. After the independent manual labeling, we compared the labels from the two raters to identify conflicts. The two raters had agreements on 12,528 (96.1%) texts during this process and achieved a Cohen’s Kappa ( $\kappa$ ) score of 0.92 (i.e., an almost perfect agreement)<sup>2</sup>. We had meetings to discuss the conflicting labels and assign agreed-upon labels for those cases. At the end of conflict resolution, we found 2,447 (18.76%) texts labeled as ‘toxic’ among the 13,038 texts. We refer to this dataset as ‘code review dataset 2’ hereinafter. Table 2 provides an overview of the three datasets used in this study.

**3.3.2.4 Dataset aggregation** Since the reliability of a supervised learning-based model increases with the size of its training dataset, we decided to merge the two code review datasets into a single dataset (referred as ‘combined code review dataset’ hereinafter). We believe such merging is not problematic due to the following reasons.

1. Both datasets are labeled using the same rubrics and following the same protocol.
2. We used the same set of raters for manual labeling.
3. Both of the datasets are picked from the same type of repository (i.e., Gerrit based

---

<sup>2</sup>Kappa ( $\kappa$ ) values are commonly interpreted as follows: values  $\leq 0$  as indicating ‘no agreement’ and 0.01 – 0.20 as ‘none to slight’, 0.21 – 0.40 as ‘fair’, 0.41 – 0.60 as ‘moderate’, 0.61–0.80 as ‘substantial’, and 0.81–1.00 as ‘almost perfect agreement’.

Table 4: Examples of text preprocessing steps implemented in ToxiCR

Step	Original	Post Preprocessing
URL-rem	ah crap. Not sure how I missed that. <a href="http://goo.gl/5NFKcD">http://goo.gl/5NFKcD</a>	ah crap. Not sure how I missed that.
Cntr-exp	this line shouldn't end with a period	this line <i>should not</i> end with a period
Sym-rem	Missing: Partial-Bug: #1541928	Missing Partial Bug 1541928
Rep-elm	haha... looooooooooser!	haha.. loser!
Adv-ptn	oh right, <i>sh*t</i>	oh right, shit
Kwrđ-rem†	These <i>static</i> values should be put at the top	These values should be put at the top
Id-split†	idp = self._create_dummy_idp (add_clean_up = False)	idp = self.     create dummy idp(add clean up= False)

† – an optional SE domain specific pre-processing step

code reviews).

The merged code review dataset includes 19,651 code review comments, where 3,757 comments (19.2%) are labeled as ‘toxic.’

### 3.3.3 Data preprocessing

Code review comments are different from news, articles, books, or even spoken language. For example, review comments often contain word contractions, URLs, and code snippets. Therefore, we implemented eight data preprocessing steps. Five steps are mandatory since they aim to remove unnecessary or redundant features. The remaining three steps are optional and their impacts on toxic code review detection are empirically evaluated in our experiments. Two out of the three optional pre-processing steps are SE domain specific. Table 4 shows examples of texts before and after preprocessing.

**3.3.3.1 Mandatory preprocessing** ToxiCR implements the following five mandatory pre-processing steps.

- *URL removal (URL-rem)*: A code review comment may include a URL (e.g., a refer-

ence to documentation or a StackOverflow post). Although URLs are irrelevant for a toxicity classifier, they can increase the number of features for supervised classifiers. We used a regular expression matcher to identify and remove all URLs from our datasets.

- *Contraction expansion (Cntr-exp)*: Contractions, which are shortened forms of one or two words, are common among code review texts. For example, some common words are: doesn't → does not, we're → we are. By creating two different lexicons of the same term, contractions increase the number of unique lexicons and add redundant features. We replaced the commonly used 153 contractions, each with its expanded version.
- *Symbol removal (Sym-rem)*: Since special symbols (e.g., &, #, and ^ ) are irrelevant for toxicity classification tasks, we use a regular expression matcher to identify and remove special symbols.
- *Repetition elimination (Rep-elm)*: A person may repeat some of the characters to misspell a toxic word to evade detection from a dictionary-based toxicity detectors. For example, in the sentence "You're duumbbbb!", 'dumb' is misspelled through character repetitions. We have created a pattern based matcher to identify such misspelled cases and replace each with its correctly spelled form.
- *Adversarial pattern identification (Adv-ptn)*: A person may misspell profane words by replacing some characters with a symbol (e.g., 'f\*ck' and 'b!tch') or use an acronym for a slang (e.g., 'stfu'). To identify such cases, we have developed a profanity prepro-

cessor, which includes pattern matchers to identify various forms of the 85 commonly used profane words. Our preprocessor replaces each identified case with its correctly spelled form.

**3.3.3.2 Optional preprocessing** ToxiCR includes options to apply the following three optional preprocessing steps.

- *Identifier splitting (Id-split)*: In this preprocessing, we use a regular expression matcher to split identifiers written in camelCase and under\_score forms. For example, this step will replace 'isCrap' with 'is Crap' and replace 'is\_shitty' with 'is shitty.' This preprocessing may help to identify example code segments with profane words.
- *Programming Keywords Removal (Kwr-drem)*: Code review texts often include programming language-specific keywords (e.g., 'while', 'case', 'if', 'catch', and 'except'). These keywords are SE domain specific jargon and are not useful for toxicity prediction. We have created a list of 90 programming keywords used in popular programming languages (e.g., C++, Java, Python, C#, PHP, JavaScript, and Go). This step searches and removes occurrences of those programming keywords from a text.
- *Count profane words (profane-count)*: Since the occurrence of profane words is suggestive of a toxic text, we think the number of profane words in a text may be an excellent feature for a toxicity classifier. We have created a list of 85 profane words, and this step counts the occurrences of these words in a text. While the remaining seven pre-processing steps modify an input text pre-vectorization, this step adds dimension to the post-vectorized output of a text.

### 3.3.4 Word Vectorizers

Toxicr includes the option to use five different word vectorizers. However, due to the limitations of the algorithms, each of the vectorizers can work with only one group of algorithms. In our implementation, Tfidf works only with the classical and ensemble (CLE) methods, Word2vec, GloVe, and fastText work with the deep neural network based algorithms, and the BERT model includes its pre-trained vectorizer. For vectorizers, we chose the following implementations.

1. *Tfidf*: We select the TfidfVectorizer from the scikit-learn library. We discard words not belonging to at least 20 documents in the corpus to prevent overfitting.
2. *Word2vec*: We select the pre-trained word2vec model available at: <https://code.google.com/archive/p/word2vec/>. This model was trained with a Google News dataset of 100 billion words and contains 300-dimensional vectors for 3 million words and phrases.
3. *GloVe*: Among the publicly available, pretrained GloVe models (<https://github.com/stanfordnlp/GloVe>), we select the common crawl model. This model was trained using web crawl data of 820 billion tokens and contains 300 dimensional vectors for 2.2 million words and phrases.
4. *fastText*: From the pretrained fastText models <sup>3</sup>, we select the common crawl model. This model was trained using the same dataset as our selected GloVe model and contains 300 dimensional vectors for 2 million words.

---

<sup>3</sup><https://fasttext.cc/docs/en/english-vectors.html>



5. *BERT*: We select a variant of BERT model published as ‘BERT\_en\_uncased’. This model was pre-trained on a dataset of 2.5 billion words from Wikipedia and 800 million words from Bookcorpus [198].

### 3.3.5 Architecture of the ML Models

This section discusses the architecture of the ML models implemented in ToxiCR.

**3.3.5.1 Classical and ensemble methods** We have used the scikit-learn [131] implementations of the CLE classifiers.

- Decision Tree (DT): We have used the `DecisionTreeClassifier` class with default parameters.
- Logistic Regression (LR): We have used the `LogisticRegression` class with default parameters.
- Support-Vector Machine (SVM): Among the various SVM implementations offered by scikit-learn, we have selected the `LinearSVC` class with default parameters.
- Random Forest (RF): We have used the `RandomForestClassifier` class from scikit-learn ensembles. To prevent overfitting, we set the minimum number of samples to split at to 5. For the other parameters, we accepted the default values.
- Gradient-Boosted Decision Trees (GBT): We have used the `GradientBoostingClassifier` class from the scikit-learn library. We set `n_iter_no_change = 5`, which stops the training early if the last 5 iterations did not achieve any improvement in accuracy. We accepted the default values for the other parameters.

**3.3.5.2 Deep Neural Networks Model** We used version 2.5.0 of the TensorFlow library [1] for training the four deep neural network models (i.e., LSTM, BiLSTM, GRU, and DPCNN).

Common parameters of the four models are:

- We set `max_features = 5000` (i.e., the maximum number of features to use) to reduce the memory overhead as well as to prevent model overfitting.
- Maximum length of input is set to 500, which means our models can take texts with at most 500 words as inputs. Any input over this length would be truncated to 500 words.
- As all three pre-trained word embedding models use 300 dimensional vectors to represent words and phrases, we have set embedding size to 300.
- The embedding layer takes input embedding matrix as inputs. Each word ( $w_i$ ) from a text is mapped (embedded) to a vector ( $v_i$ ) using one of the three context-free vectorizers (i.e., fastText, GloVe, and word2vec). For a text  $T$ , its embedding matrix will have a dimension of  $(300 \times n)$ , where  $n$  is the total number of words in that text.
- Since we are developing binary classifiers, we have selected `binary_crossentropy` loss function for model training.
- We have selected the Adam optimizer (Adaptive Moment Estimation) [94] to update the weights of the network during the training time. The initial `learning_rate` is set to 0.001.

- During the training, we set *accuracy* ( $A$ ) as the evaluation metric.

The four deep neural models of ToxiCR are primarily based on three layers, as described briefly in the following. Architecture diagrams of the models are included in our replication package [152].

- **Input Embedding Layer:** After preprocessing of code review texts, those are converted to input matrix. Embedded layer maps input matrix to a fixed dimension input embedding matrix. We used three pre-trained embeddings, which help the model to capture the low-level semantics using position based texts.
- **Hidden State Layer:** This layer takes the position wise embedding matrix and helps to capture the high level semantics of words in code review texts. The configuration of this layer depends on the choice of the algorithm. ToxiCR includes one CNN (i.e., DPCNN) and three RNN (i.e., LSTM, BiLSTM, GRU) based hidden layers. In the following, we describe the key properties of these four types of layers.
  - **DPCNN blocks:** Following the implementation of DPCNN [90], we set 7 convolution blocks with Conv1D layer after the input embedding layer. We also set the other parameters of DPCNN model following [90]. Outputs from each of the CNN blocks is passed to a GlobalMaxPooling1D layer to capture the most important features from the inputs. A dense layer is set with 256 units which is activated with a linear activation function.
  - **LSTM blocks:** From the Keras library, we use *LSTM* unit to capture the hidden sequence from the input embedding vector. LSTM unit generates the high di-

mensional semantic representation vector. To reshape the output dimension, we use flatten and dense layer after the LSTM unit.

- BiLSTM blocks: For text classification tasks, BiLSTM works better than LSTM for capturing the semantics of long sequences of text. Our model uses 50 units of Bidirectional LSTM units from the Keras library to generate the hidden sequence of the input embedding matrix. To downsample the high dimension hidden vector from BiLSTM units, we set a GlobalMaxPool1D layer. This layer downsamples the hidden vector from BiLSTM layer by taking the maximum value of each dimension and thus captures the most important features for each vector.
- GRU blocks: We use bidirectional GRUs with 80 units to generate the hidden sequence of input embedding vector. To keep the most important features from GRU units, we set a concatenation of GlobalAveragePooling1D and GlobalMaxPooling1D layers. GlobalAveragePooling1D calculates the average of entire sequence of each vector and GlobalMaxPooling1D finds the maximum value of entire sequence.
- Classifier Layer: The output vector of the hidden state layer projects to the output layer with a dense layer and a sigmoid activation function. This layer generates the probability of the input vector from the range 0 to 1. We chose a sigmoid activation function because it provides the probability of a vector within the 0 to 1 range.

**3.3.5.3 Transformer models** Among the several pre-trained BERT models<sup>4</sup>, we have used `bert_en_uncased`, which is also known as the *BERT\_base* model. We downloaded the models from the `tensorflow_hub`, which consists of trained machine learning models ready for fine-tuning.

Our BERT model architecture is as follows:

- Input layer: takes the preprocessed input text from our SE dataset. To fit into BERT pretrained encoder, we preprocess each text using a matching preprocessing model (i.e., `bert_en_uncased_preprocess`<sup>5</sup>).
- BERT encoder: From each preprocessed text, this layer produces BERT embedding vectors with higher level semantic representations.
- Dropout Layer: To prevent overfitting as well as eliminate unnecessary features, outputs from the BERT encoder layer are passed to a dropout layer with a probability of 0.1 to drop an input.
- Classifier Layer: Outputs from the dropout layer is passed to a two-unit dense layer, which transforms the outputs into two-dimensional vectors. From these vectors, a one-unit dense layer with a linear activation function generates the probabilities of each text being toxic. Unlike deep neural network's output layer, we have found that the linear activation function provides better accuracy than non-linear ones (e.g, *relu*, *sigmoid*) for the BERT-based models.

---

<sup>4</sup><https://github.com/google-research/bert>

<sup>5</sup>[https://tfhub.dev/tensorflow/bert\\_en\\_uncased\\_preprocess/3](https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3)

Table 5: An overview of the hyper parameters for our deep neural networks and transformers

Hyper-Parameters	Deep neural networks (i.e., DPCNN, LSTM, BiLSTM, and GRU)	Transformer (BERT)
Activation	sigmoid	linear
Loss function	binary crossentropy	binary crossentropy
Optimizer	adam	Adamw
Learning rate	0.001	$3e-5$
Early stopping monitor	val_loss	val_loss
Epochs	40	15
Batch size	128	256

- **Parameters:** Similar to the deep neural network models, we use `binary_crossentropy` as the loss function and *Binary Accuracy* as the evaluation metric during training.
- **Optimizer:** We set the optimizer as Adamw [109], which improved the generalization performance of ‘adam’ optimizer. Adamw minimizes the prediction loss and does regularization by decaying weight. Following the recommendation of Devlin *et al.* [52], we set the initial learning rate to  $3e - 5$ .

### 3.3.6 Model Training and Validation

The following subsections detail our model training and validation approaches.

### 3.3.7 Classical and ensembles

We evaluated all the models using 10-fold cross validations, where the dataset was randomly split into 10 groups and each of the ten groups was used as a test dataset once, while the remaining nine groups were used to train the model. We used stratified split to ensure similar ratios of the classes between the test and training sets.

### 3.3.8 DNN and Transformers

We have customized several hyper-parameters of the DNN models to train our models. Table 5 provides an overview of those customized hyper-parameters. A DNN model

can be overfitted due to over-training. To encounter that, we have configured our training parameters to find the best-fit model that is not overfitted. We split our dataset into three sets during training according to 8:1:1 ratio. These three sets are used for training, validation, and testing during our 10-fold cross validations to evaluate our DNN and transformer models. For training, we have set a maximum of 40 epochs<sup>6</sup> for the DNN models and a maximum of 15 epochs for the BERT model. During each epoch, a model is trained using 80% samples, is validated using 10% samples, and the remaining 10% is used to measure the performance of the trained model. To prevent overfitting, we have used an *EarlyStopping* function from the Keras library, which monitors *minimum val loss*. If the performance of a model on the validation dataset starts to degrade (e.g., loss begins to increase or accuracy begins to drop), then the training process is stopped.

### 3.3.9 Tool interface

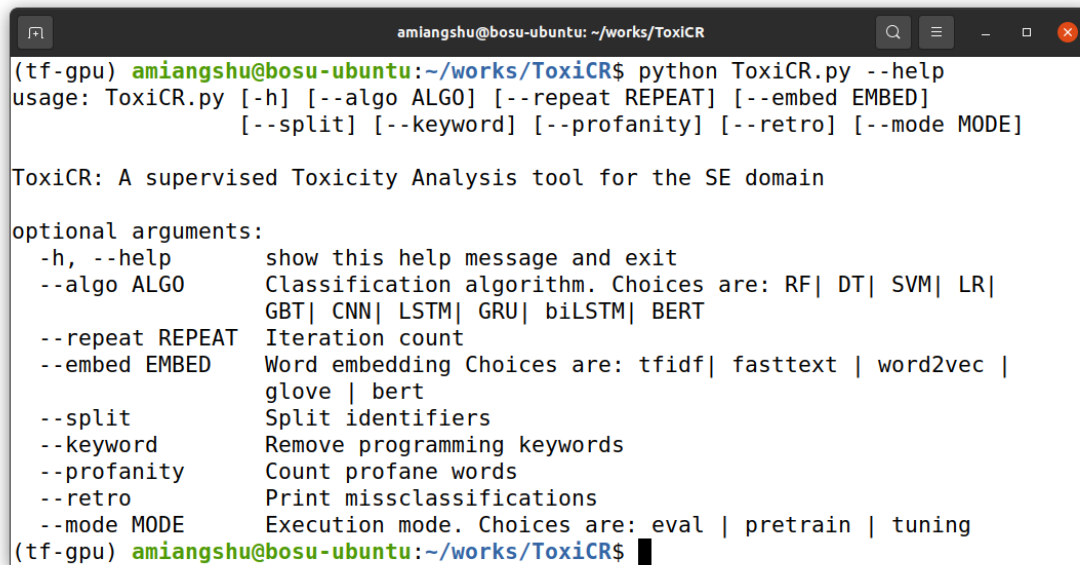
We have designed ToxiCR to support standalone evaluation and be used as a library for toxic text identification. We have also included pre-trained models to save model training time. Listing 1 shows a sample code to predict the toxicity of texts using our pretrained BERT model.

We have also included a command line-based interface for model evaluation, retraining, and fine tuning hyperparameters. Figure 3 shows the usage help message of ToxiCR. Users can customize execution with eight optional parameters, which are as follows:

```
1 from ToxiCR import ToxiCR
2
3 clf=ToxiCR(ALGO="BERT", count_profanity=False, remove_keywords=True,
```

---

<sup>6</sup>the number of times that a learning algorithm will work through the entire training dataset



```

(tf-gpu) amiangshu@bosu-ubuntu:~/works/ToxiCR$ python ToxiCR.py --help
usage: ToxiCR.py [-h] [--algo ALGO] [--repeat REPEAT] [--embed EMBED]
                [--split] [--keyword] [--profanity] [--retro] [--mode MODE]

ToxiCR: A supervised Toxicity Analysis tool for the SE domain

optional arguments:
  -h, --help            show this help message and exit
  --algo ALGO           Classification algorithm. Choices are: RF| DT| SVM| LR|
                        GBT| CNN| LSTM| GRU| biLSTM| BERT
  --repeat REPEAT       Iteration count
  --embed EMBED         Word embedding Choices are: tfidf| fasttext | word2vec |
                        glove | bert
  --split               Split identifiers
  --keyword             Remove programming keywords
  --profanity           Count profane words
  --retro               Print missclassifications
  --mode MODE           Execution mode. Choices are: eval | pretrain | tuning
(tf-gpu) amiangshu@bosu-ubuntu:~/works/ToxiCR$

```

Figure 3: The command line interface of ToxiCR showing various customization options

```

4         split_identifier=False,
5         embedding="bert", load_pretrained=True)
6
7 clf.init_predictor()
8 sentences=["this is crap", "thank you for the information",
9           "shi*tty code" ]
10
11 results=clf.get_toxicity_class(sentences)

```

Listing 1: Example usage of ToxiCR to classify toxic texts

- Algorithm Selection: Users can select one of the ten included algorithms by using the `-algo ALGO` option.
- Number of Repetitions: Users can specify the number of times to repeat the 10-fold cross-validations in evaluation mode using `-repeat n` option. Default value is 5.



- **Embedding:** ToxiCR includes five different vectorization techniques: `tfidf`, `word2vec`, `glove`, `fasttext`, and `bert`. `tfidf` is configured to be used only with the CLE models. `word2vec`, `glove`, and `fasttext` can be used only with the DNN models. Finally, `bert` can be used only with the transformer model. Users can customize this selection using the `-embed EMBED` option.
- **Identifier splitting:** Using the `-split` option, users can select to apply the optional preprocessing step to split identifiers written in camelCases or `under_scores`.
- **Programming keywords:** Using the `-keyword` option, users can select to apply the optional preprocessing step to remove programming keywords.
- **Profanity:** The `-profanity` optional preprocessing step allows to add of the number of profane words in a text as an additional feature.
- **Missclassification diagnosis:** The `-retro` option is useful for error diagnosis. If this option is selected, ToxiCR will write all misclassified texts in a spreadsheet to enable manual analyses.
- **Execution mode:** ToxiCR can be executed in three different modes. The `eval` mode will run 10-fold cross validations to evaluate the performance of an algorithm with the selected options. In the `eval` mode, ToxiCR writes the results of each run and model training time in a spreadsheet. The `retrain` mode will train a classifier with the full dataset. This option is useful for saving models in a file to be used in the future. Finally, the `tuning` mode allows exploring various algorithm hyperparameters to identify the optimum set.

## 3.4 Results

We empirically evaluated the ten algorithms included in ToxiCR to identify the best possible configuration to identify toxic texts from our datasets. The following subsections detail our experimental configurations and the results of our evaluations.

### 3.4.1 Experimental Configuration

To evaluate the performance of our models, we use precision, recall, f-score, and accuracy for both toxic (class 1) and non-toxic (class 0) classes. We computed the following evaluation metrics.

- *Precision (P)*: For a class, precision is the percentage of identified cases that truly belong to that class.
- *Recall (R)*: For a class, recall is the ratio of correctly predicted cases and total number of cases.
- *F1-score (F1)*: F1-score is the harmonic mean of precision and recall.
- *Accuracy (A)*: Accuracy is the percentage of cases that a model predicted correctly.

In our evaluations, we consider the F1-score for the toxic class (i.e.,  $F1_1$ ) as the most important metric to evaluate these models, since i) identification of toxic texts is our primary objective, and ii) our datasets are imbalanced with more than 80% non-toxic texts.

To estimate the performance of the models more accurately, we repeated 10-fold cross validations five times and computed the means of all metrics over those  $5 * 10 = 50$  runs. We use Python's Random module, which is a pseudo-random number generator, to create

stratified 10-fold partitions, preserving the ratio between the two classes across all partitions. If initialized with the same seed number, `Random` would generate the exact same sequence of pseudo-random numbers. At the start of each algorithm’s evaluation, we initialized the `Random` generator using the same seed to ensure the exact same sequence of training/testing partitions for all algorithms. As the model performances are normally distributed, we use paired sample t-tests to check if observed performance differences between two algorithms are statistically significant ( $p < 0.05$ ). We use the ‘paired sample t-test’, since our experimental setup guarantees cross-validation runs of two different algorithms would get the same sequences of train/test partitions. We have included the results of the statistical tests in the replication package [152].

We conducted all evaluations on an Ubuntu 20.04 LTS workstation with an Intel i7-9700 CPU, 32GB RAM, and an NVIDIA Titan RTX GPU with 24 GB memory. For Python configuration, we created an Anaconda environment with Python 3.8.0, and `tensorflow / tensorflow-gpu 2.5.0`.

### 3.4.2 Baseline Algorithms

To establish baseline performances, we computed the performances of four existing toxicity detectors (Table 6) on our dataset. We briefly describe the four tools in the following.

1. Perspective API [5] (off-the-shelf): To prevent the online community from abusive content, Jigsaw and Google’s Counter Abuse Technology team developed Perspective API [5]. Algorithms and datasets to train these models are not publicly available. Perspective API can generate the probability score of a text being toxic, `severe_toxic`,

insult, profanity, threat, identity\_attack, and sexually explicit. The score for each category is from 0 to 1, where the probability of a text belonging to that category increases with the score. For our two class classifications, we considered a text as toxic if its Perspective API score for the toxicity category is higher than 0.5.

2. STRUDEL tool [141] (off-the-shelf): The STRUDEL tool is an ensemble based on two existing tools: Perspective API and Stanford politeness detector and BoW vector obtained from preprocessed text. Its classification pipeline obtains the toxicity score of a text using the Perspective API, computes the politeness score using the Stanford politeness detector tool [49], and computes BoW vector using TfIdf. For SE specificity, its TfIdf vectorizer excludes words that occur more frequently in the SE domain than in a non-SE domain. Although STRUDEL tool also computes several other features such as sentiment score, subjectivity score, polarity score, number of LIWC anger words, and the number of emoticons in a text, none of these features contributed to improved performances during its evaluation [141]. Hence, the best performing ensemble from STRUDEL uses only the Perspective API score, Stanford politeness score, and TfIdf vector. The off-the-shelf version is trained on a manually labeled dataset of 654 Github issues.
3. STRUDEL tool [154] (retrain): Due to several technical challenges, we were unable to retrain the STRUDEL tool using the source code provided in its repository [141]. Therefore, we wrote a simplified re-implementation based on the description included in the paper and our understanding of the current source code. Upon contacting, the primary author of the tool acknowledged our implementation as correct.

Table 6: Performances of the four contemporary toxic detectors to establish a baseline performance. For our classifications, we consider toxic texts as the ‘class 1’ and non-toxic texts as the ‘class 0’.

Models	Non-toxic			Toxic			Accuracy
	$P_0$	$R_0$	$F1_0$	$P_1$	$R_1$	$F1_1$	
Perspective API [5] (off-the-shelf)	0.92	0.79	0.85	0.45	0.70	0.55	0.78
Strudel Tool (off-the-shelf) [141]	0.93	0.76	0.83	0.43	0.77	0.55	0.76
Strudel (retrain) [154]	<b>0.97</b>	<b>0.96</b>	<b>0.97</b>	<b>0.85</b>	<b>0.86</b>	<b>0.85</b>	<b>0.94</b>
DPCNN (retrain) [153]	0.94	0.95	0.94	0.81	0.76	0.78	0.91

Our implementation is publicly available inside the WSU-SEAL directory of the publicly available repository: <https://github.com/WSU-SEAL/toxicity-detector>. Our pull request with this implementation has also been merged into the original repository. For computing baseline performance, we conducted a stratified 10-fold cross validation using our code review dataset.

4. DPCNN [153] (retrain): We cross-validated a DPCNN model [90], using our code review dataset. We include this model in our baseline since it provided the best retrained performance during our benchmark study [153].

Table 6 shows the performances of the four baseline models. Unsurprisingly, the two retrained models provide better performances than the off-the-shelf ones. Overall, the retrained Strudel tool provides the best scores among the four tools on all seven metrics. So, we’re thinking about this model as the key baseline to improve on. The best toxicity detector among the ones participating in the 2020 SemEval challenge achieved 0.92  $F_1$  score on the Jigsaw dataset [192]. As the baseline models listed in Table 6 are evaluated on a different dataset, it may not be fair to compare these models against the ones trained on the Jigsaw dataset. However, the best baseline model’s  $F_1$  score is 7 (i.e., 0.92 -0.85 )

points lower than the ones from a non-SE domain. This result suggests that with existing technology, it may be possible to train SE domain specific toxicity detectors with better performances than the best baseline listed in Table 6.

**Finding 1:** *Retrained models provide considerably better performances than the off-the-shelf ones, with the retrained STRUDEL tool providing the best performances. Still, the  $F1_1$  score from the best baseline model lags 7 points behind the  $F1_1$  score of state-of-the-art models trained and evaluated on the Jigsaw dataset during 2020 SemEval challenge [192].*

### 3.4.3 How do the algorithms perform without optional preprocessing?

The following subsections detail the performances of the three groups of algorithms described in Section 3.3.5.

Table 7: Mean performances of the ten selected algorithms based on 10-fold cross validations. For each group, a shaded background indicates significant improvements over the others from the same group

Group	Models	Vectorizer	Non-toxic			Toxic			Accuracy ( <i>A</i> )
			$P_0$	$R_0$	$F1_0$	$P_1$	$R_1$	$F1_1$	
CLE	DT	tfidf	0.954	0.963	0.959	0.841	0.806	0.823	0.933
	GBT	tfidf	0.926	0.985	0.955	0.916	0.672	0.775	0.925
	LR	tfidf	0.918	0.983	0.949	0.900	0.633	0.743	0.916
	RF	tfidf	0.956	0.982	0.969	0.916	0.810	0.859	0.949
	SVM	tfidf	0.929	0.979	0.954	0.888	0.688	0.775	0.923
DNN1	DPCNN	word2vec	0.962	0.966	0.964	0.870	0.841	0.849	0.942
	DPCNN	GloVe	0.963	0.966	0.964	0.871	0.842	0.851	0.943
	DPCNN	fasttext	0.964	0.967	0.965	0.870	0.845	0.852	0.944
DNN2	LSTM	word2vec	0.929	0.978	0.953	0.866	0.698	0.778	0.922
	LSTM	GloVe	0.944	0.971	0.957	0.864	0.758	0.806	0.930
	LSTM	fasttext	0.936	0.974	0.954	0.853	0.718	0.778	0.925
DNN3	BiLSTM	word2vec	0.965	0.974	0.969	0.887	0.851	0.868	0.950
	BiLSTM	GloVe	0.965	0.976	0.968	0.895	0.828	0.859	0.948
	BiLSTM	fasttext	0.966	0.974	0.970	0.888	0.854	0.871	0.951
DNN4	GRU	word2vec	0.964	0.976	0.970	0.894	0.847	0.870	0.951
	GRU	GloVe	0.965	0.977	0.971	0.901	0.851	0.875	0.953
	GRU	fasttext	0.965	0.974	0.969	0.888	0.852	0.869	0.951
Transformer	BERT	en uncased	0.971	0.976	0.973	0.901	0.876	0.887	0.957

**3.4.3.1 Classical and Ensemble (CLE) algorithms** The top five rows of Table 7 (i.e., CLE group) show the performances of the five CLE models. Among, those five algorithms, RF achieves significantly higher  $P_0$  (0.956),  $F1_0$  (0.969),  $R_1$  (0.81),  $F1_1$  (0.859) and accuracy (0.949) than the four other algorithms from this group. The RF model also significantly outperforms (One-sample t-test) the key baseline (i.e., retrained STRUDEL) in terms of the two key metrics, accuracy ( $A$ ) and  $F1_1$ . Although, STRUDEL retrain achieves better recall ( $R_1$ ), our RF based model achieves better precision ( $P_1$ ).

**3.4.3.2 Deep Neural Networks (DNN)** We evaluated each of the four DNN algorithms using three different pre-trained word embedding techniques (i.e., word2vec, GloVe, and fastText) to identify the best performing embedding combinations. Rows 6 to 17 (i.e., groups: DNN1, DNN2, DNN3, and DNN4) of the Table 7 show the performances of the four DNN algorithms using three different embeddings. For each group, statistically significant improvements (paired-sample t-tests) over the other two configurations are highlighted using a shaded background. Our results suggest that the choice of embedding does influence the performances of the DNN algorithms. However, such variations are minor.

For DPCNN, only  $R_1$  score is significantly better with fastText than with GloVe or word2vec. The other scores do not vary significantly among the three embeddings. Based on these results, we recommend fastText for DPCNN in ToxiCR. For LSTM and GRU, GloVe boosts significantly better  $F1_1$  scores than those based on fastText or word2vec. Since  $F1_1$  is one of the key measures to evaluate our models, we recommend the GloVe for both LSTM and GRU in ToxiCR. Glove also boosts the highest accuracy for both LSTM (although not statistically significant) and GRU. For BiLSTM, since fastText provides significantly higher



$P_0$ ,  $R_1$ , and  $F1_1$  scores than those based on GloVe or word2vec. We recommend fastText for BiLSTM in ToxiCR. These results also suggest that three out of the four selected DNN algorithms (i.e., except LSTM) significantly outperform (one-sample t-test) the key baseline (i.e., retrained STRUDEL) in terms of both accuracy and  $F1_1$ -score.

**3.4.3.3 Transformer** The bottom row of Table 7 shows the performance of our BERT based model. This model achieves the highest mean accuracy (0.957) and  $F1_1$  (0.887) among all the 18 models listed in Table 7. This model also outperforms the baseline STRUDEL retrain on all seven metrics.

**Finding 2:** *From the CLE group, RF provides the best performances. From the DNN group, GRU with glove provides the best performances. Among the 18 models from the six groups, BERT achieves the best performance. Overall, ten out of the 18 models also outperform the baseline STRUDEL retrain model.*

#### 3.4.4 Do optional preprocessing steps improve performance?

For each of the ten selected algorithms, we evaluated whether the optional preprocessing steps (especially SE domain specific ones) improve performances. Since ToxiCR includes three optional preprocessing (i.e., identifier splitting (*id-split*), keyword removal (*kwr-d-remove*), and counting profane words (*profane-count*), we ran each algorithm with  $2^3 = 8$  different combinations. For the DNN models, we did not evaluate all three embeddings in this step, as that would require evaluating  $3*8 = 24$  possible combinations for each one. Rather, we used only the best performing embedding identified in the previous step (i.e., Section 3.4.3.2).

To select the best optional preprocessing configuration from the eight possible configurations, we use mean accuracy and mean  $F1_1$  scores based on five time 10-fold cross validations. We also used pair sampled t-tests to check whether any improvement over its base configurations, as listed in the Table 7 (i.e., no optional preprocessing selected), is statistically significant (paired sample t-test,  $p < 0.05$ ). Table 8 shows the best performing configurations for all algorithms and the mean scores for those configurations. Checkmarks (✓) in the preprocessing columns for an algorithm indicate that the best configuration for that algorithm does use that pre-processing. To save space, we report the performances of only the best combination for each algorithm. Detailed results are available in our replication package [152]. These results suggest that optional pre-processing steps improve the models' performances. Notably, CLE models gained higher improvements than the other two groups. RF's accuracy improved from 0.949 to 0.955 and  $F1_1$  improved from 0.859 to 0.879 with the *profane-count* preprocessing.

Table 8: Best performing configurations of each model with optional preprocessing steps. A shaded background indicates significant improvements over its base configuration (i.e., no optional preprocessing). For each column, bold font indicates the highest value for that measure. † – indicates an optional SE domain-specific pre-processing step.

Group	Algo	Vectorizer	Preprocessing			Non-toxic			Toxic			A
			profane-count	kwrd-remove	id-split†	P <sub>0</sub>	R <sub>0</sub>	F1 <sub>0</sub>	P <sub>1</sub>	R <sub>1</sub>	F1 <sub>1</sub>	
CLE	DT	tfidf	✓	✓	-	0.960	0.968	0.964	0.862	0.830	0.845	0.942
	GBT	tfidf	✓	✓	-	0.938	0.981	0.959	0.901	0.729	0.806	0.932
	LR	tfidf	✓	✓	-	0.932	0.981	0.956	0.898	0.698	0.785	0.927
	RF	tfidf	✓	-	-	0.964	<b>0.981</b>	0.972	<b>0.917</b>	0.845	0.879	0.955
	SVM	tfidf	✓	✓	-	0.939	0.977	0.958	0.886	0.736	0.804	0.931
DNN	DPCNN	fasttext	✓	-	-	0.964	0.973	0.968	0.889	0.846	0.863	0.948
	LSTM	glove	✓	✓	✓	0.944	0.974	0.959	0.878	0.756	0.810	0.932
	BiLSTM	fasttext	✓	-	✓	0.966	0.975	0.971	0.892	0.858	0.875	0.953
	BiGRU	glove	✓	-	✓	0.966	0.976	0.971	0.897	0.856	0.876	0.954
Transormer	BERT	bert	-	✓	-	<b>0.970</b>	0.978	<b>0.974</b>	0.907	<b>0.874</b>	<b>0.889</b>	<b>0.958</b>

During these evaluations, other CLE models also achieved between 0.02 to 0.04 performance boosts in our key measures (i.e.,  $A$  and  $F1_1$ ). Improvements from optional preprocessing also depend on algorithm choices. While the *profane-count* preprocessing improved performances of all the CLE models, *kwrd-remove* improved all except RF. On the other hand, *id-split* improved none of the CLE models.

All the DNN models also improved performances with the *profane-count* preprocessing. Contrasting the CLE models, *id-split* was useful for three of the four DNNs. *kwrd-remove* preprocessing improved only LSTM models. Noticeably, gains from optional preprocessing for the DNN models were less than 0.01 over the base configurations' and statistically insignificant (paired-sample t-test,  $p > 0.05$ ) for most cases. Finally, although we noticed slight performance improvement (i.e., in  $A$  and  $F1_1$ ) of the BERT model with *kwrd-remove*, the differences are not statistically significant. **Overall, at the end of our extensive evaluation, we found the best performing combination was a BERT model with *kwrd-remove* optional preprocessing. The best combination provides 0.889  $F1_1$  score and 0.958 accuracy.** The best performing model also significantly outperforms (one sample t-test,  $p < 0.05$ ) the baseline model (i.e., STRUDEL retrain in Table 6) in all the seven performance measures.

**Finding 3:** *Eight out of the ten models (i.e., except SVM and DPCNN) achieved significant performance gains through SE domain preprocessing, such as programming keyword removal and identifier splitting. Although keyword removal may be useful for all four classes of algorithms, identifier splitting is useful only for three DNN models. Our best model is based on BERT, which significantly outperforms the STRUDEL retrain model on all seven measures.*

### 3.4.5 How do the models perform on another dataset?

To evaluate the generality of our models, we have used the Gitter dataset of 4,140 messages from our benchmark study [153]. In this step, we conducted two types of evaluations. First, we ran 10-fold cross validations of the top CLE model (i.e., RF) and the BERT model using the Gitter dataset. Second, we evaluated cross dataset prediction performance (i.e., off-the-shelf) by using the code review dataset for training and the Gitter dataset for testing.

The top two rows of Table 9 show the results of 10-fold cross-validations for the two models. We found that the BERT model provides the best accuracy (0.898) and the best  $F1_1$  (0.856). All seven performance measures achieved by the BERT model on the Gitter dataset are lower than those on the code review dataset. This may be due to the smaller size of the Gitter dataset (4,140 texts) than the code review dataset (19,571 texts). The bottom two rows of the Table 9 shows the results of our cross-predictions (i.e., off-the-shelf). Our BERT model achieved similar performances in terms of  $A$  and  $F1_1$  in both modes. However, the RF model performed better on the Gitter dataset in cross-prediction mode (i.e., off-the-shelf) than in cross-validation mode. This result further supports our

Table 9: Performance of ToxiCR on Gitter dataset

Mode	Models	Vectorizer	Non-toxic			Toxic			Accuracy
			$P$	$R$	$F1$	$P$	$R$	$F1$	
Cross-validation (retrain)	RF	Tfidf	0.851	0.945	0.897	0.879	0.699	0.779	0.859
	BERT	BERT-en-uncased	0.931	0.909	0.919	0.843	0.877	0.856	0.898
Cross-prediction (off-the-shelf)	RF	Tfidf	0.857	0.977	0.914	0.945	0.704	0.807	0.881
	BERT	BERT-en-uncased	0.897	0.949	0.923	0.897	0.802	0.847	0.897

Table 10: Confusion Matrix for our best performing model (i.e., BERT) for the combined code review dataset

		Predicted	
		Toxic	Non-toxic
Actual	Toxic	3259	483
	Non-toxic	373	15,446

hypothesis that the performance drops of our models on the Gitter dataset may be due to smaller-sized training data.

**Finding 4:** *Although our best performing model provides higher precision off-the-shelf on the Gitter dataset than that from the retrained model, the later achieves better recall. Regardless, our BERT model achieves similar accuracy and  $F1_1$  during both off-the-shelf usage and retraining.*

### 3.4.6 What are the distributions of misclassifications from the best performing model?

The best-performing model (i.e., BERT) misclassified only 856 texts out of the 19,571 texts from our dataset. There are 373 false positives and 483 false negatives. Table 10 shows the confusion matrix of the BERT model. To understand the reasons behind misclassifications, we adopted an open coding approach where two of the authors independently inspected each misclassified text to identify general scenarios. Next, they had a discussion session, where they developed an agreed upon higher level categorization scheme of five groups. With this scheme, those two authors independently labeled each misclassified text

into one of those five groups. Finally, they compared their labels and resolved conflicts through mutual discussions.

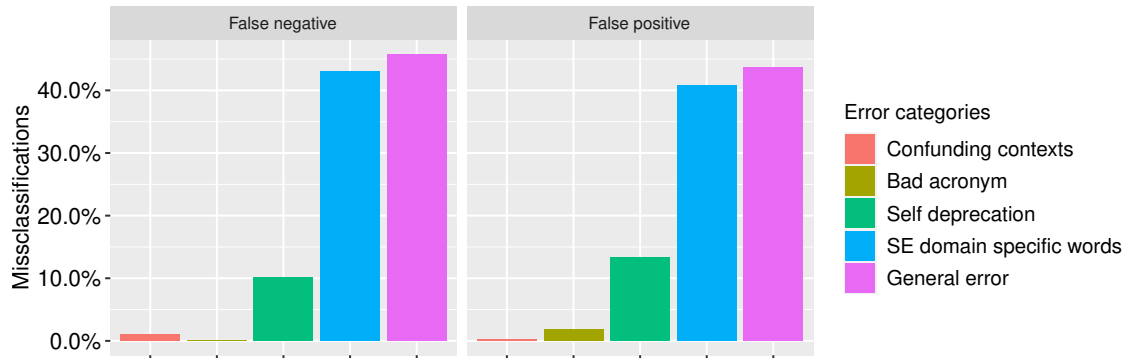


Figure 4: Distribution of the misclassifications from the BERT model

Figure 4 shows distributions of the five categories of misclassifications from ToxiCR grouped by False Positives (FP) and False Negatives (FN). The following subsections detail those error categories.

### 3.4.7 General errors (GE)

General errors are due to failures of the classifier to identify the pragmatic meaning of various texts. These errors represent 45% of the false positives and 46% of the false negatives. Many GE false positives are due to words or phrases that more frequently occur in toxic contexts and vice versa. For example, “*If we do, should we just get rid of the HBoundType?*” and “*Done. I think they came from a messed up rebase.*” are two false positive cases due to the phrases ‘get rid of’ and ‘messed up’ that have occurred more frequently in toxic contexts.

GE errors also occurred due to infrequent words. For example, “*“Oh, look. The stupidity that makes me rant so has already taken root. I suspect it’s not too late to fix this, and fixing this rates as a mitzvah in my book.”* – is incorrectly predicted as non-toxic as very

few texts in our dataset include the word ‘stupidity’. Another such instance was *“this is another instance of uneducated programmers calling any kind of polymorphism overloading, please translate it to override.”*, due to the word ‘uneducated’. As we did not have many instances of identified attacks in our dataset, most of those were also incorrectly classified. For example, *“most australian dummy var name ever!”* was predicted as non-toxic by our classifier.

**3.4.7.1 SE domain specific words (SE):** Words that have different meanings in the SE domain than its’ meaning in the general domain (die, dead, kill, junk, and bug) [153] were responsible for 40% false positives and 43% false negatives. For example, the text *“you probably wanted ‘die’ here. error is not fatal.”* , is incorrectly predicted as toxic due to the presence of the words ‘die’ and ‘fatal’. On the other hand, although the word ‘junk’ is used to harshly criticize a code in the sentence *“I don’t actually need all this junk...”*, this sentence was predicted as non-toxic as most of the code review comments from our dataset do not use ‘junk’ in such a way.

**3.4.7.2 Self deprecation (SD):** Usage of self-deprecating texts to express humility is common during code reviews [153, 112]. We found that 13% of 373 false positives and 11% of 493 false negatives were due to the presence of self deprecating phrases. For example, *“Missing entry in kerneldoc above... (stupid me)”* is labeled as ‘non-toxic’ in our dataset but is predicted as ‘toxic’ by our model. Although, our model did classify many of the SD texts expressing humbleness correctly, those texts also led to some false negatives. For example, although *“Huh? Am I stupid? How’s that equivalent?”* was misclassified as



non-toxic, it fits ‘toxic’ according to our rubric due to its aggressive tone.

**3.4.7.3 Bad acronym (BA)** In few cases, developers have used acronyms with with alternate toxic expansion. For example, the webkit framework used the acronym ‘WTF’ -‘Web Template Framework’<sup>7</sup>, for a namespace. Around 2% of our false positive cases were comments referring to the ‘WTF’ namespace from Webkit.

**3.4.7.4 Confounding contexts (CC)** Some of the texts in our dataset represent confounding contexts and were challenging even for the human raters to make a decision. Such cases represent 0.26% false positives and 1.04% false negatives. For example, *“This is a bit ugly, but this is what was asked so I added a null ptr check for |inspector\_agent\_|. Let me know what you think.”* is a false positive case from our dataset. We had labeled it as non-toxic since the word ‘ugly’ is applied to critique code written by the author of this text. On the other hand, *“I just know the network stack is full of \_bh poop. Do you ever get called from irq context? Sorry, I didn’t mean to make you thrash.”* is labeled as toxic due to thrashing another person’s code with the word ‘poop’. However, the reviewer also said sorry in the next sentence. During labeling, we considered it as toxic, since the reviewer could have critiqued the code in a nicer way. Probably due to the presence of mixed contexts, our classifier incorrectly predicted it as ‘non-toxic’.

**Finding 5:** *Almost 85% of the misclassifications are due to either our model’s failure to accurately comprehend the pragmatic meaning of a text (i.e., GE) or words having SE domain specific synonyms.*

<sup>7</sup><https://stackoverflow.com/questions/834179/wtf-does-wtf-represent-in-the-webkit-code-base>

### 3.5 Implications

Based on our design and evaluation of ToxiCR, we have identified the following lessons.

**Lesson 1: Development of a reliable toxicity detector for the SE domain is feasible.**

Despite creating an ensemble of multiple NLP models (i.e., Perspective API, Sentiment score, Politeness score, Subjectivity, and Polarity) and various categories of features (i.e., BoW, number of anger words, and emoticons), the STRUDEL tool achieved only 0.57 F-score during their evaluation. Moreover, a recent study by Miller *et al.* found false positive rates as high as 98% [112]. On the other hand, the best model from the ‘2020 Semeval Multilingual Offensive Language Identification in Social Media task’ achieved a  $F1_1$  score of 92.04% [193]. Therefore, the question remains, whether we can build a SE domain specific toxicity detector that achieves similar performances (i.e.,  $F1_1 = 0.92$ ) as the ones from non-SE domains.

In designing ToxiCR, we adopted a different approach, i.e., focusing on text preprocessing and leveraging state-of-the-art NLP algorithms rather than creating ensembles to improve performances. Our extensive evaluation with a large scale SE dataset has identified a model that has 95.8% accuracy and boosts 88.9%  $F1_1$  score in identifying toxic texts. This model’s performances are within 3% of the best one from a non-SE domain. This result also suggests that with a carefully labeled large scale dataset, we can train an SE domain specific toxicity detector that achieves performances that are close to those of toxicity detectors from non-SE domains.

**Lesson 2: Performance from Random Forest’s optimum configuration may be adequate if GPU is not available.**

While a deep learning-based model (i.e., BERT) achieved the best performances during our evaluations, that model is computationally expensive. Even with a high-end GPU such as Titan RTX, our BERT model required on average 1,614 seconds for training. We found that RandomForest based models trained on a Core-i7 CPU took only 64 seconds on average.

During a classification task, RF generates the decision using majority voting from all sub-trees. RF is suitable for high dimensional noisy data like the ones found in text classification tasks [87]. With carefully selected preprocessing steps to better understand contexts (e.g., profanity count) RF may perform well for binary toxicity classification tasks. In our model, after adding profane count features, RF achieved an average accuracy of 95.5% and  $F1_1$ -score of 87.9%, which are within 1% of those achieved by BERT. Therefore, if computation cost is an issue, a RandomForest based model may be adequate for many practical applications. However, as our RF model uses a context-free vectorizer, it may perform poorly on texts, where prediction depends on surrounding contexts. Therefore, for a practical application, a user must take that limitation into account.

### **Lesson 3: Preprocessing steps do improve performances.**

We have implemented five mandatory and three optional preprocessing steps in ToxiCR. The mandatory preprocessing steps do improve performances of our models. For example, a DPCNN model without these preprocessing achieved 91% accuracy and 78%  $F1_1$  (Table 6). On the other hand, a model based on the same algorithm achieved 94.4% accuracy and 84.5%  $F1_1$  with these preprocessing steps. Therefore, we recommend using both domain specific and general preprocessing steps.

Two of our pre-processing steps are SE domain specific (i.e., Identifier Splitting, Programming Keywords removal). Our empirical evaluation of those steps (Section 3.4.4) suggest that eight out of the ten models (i.e., except SVM and DPCNN) achieved significant performance improvements through these steps. Although, none of the models showed significant degradation through these steps, significant gains were dependent on algorithm selection, with CLE algorithms gaining only from keyword removal and identifier splitting improving only the DNN ones.

**Lesson 4: Performance boosts from the optional preprocessing steps are algorithm dependent.**

The three optional preprocessing steps also improved the performances of the classifiers. However, performance gains through these steps were algorithm-dependent. The profane-count preprocessing had the highest influence as nine out of the ten models gained performance with this step. On the other `id-split` was the least useful one with only three DNN models gaining minor gains with this step. CLE algorithms gained the most with  $\approx 1\%$  boost in terms of accuracies and 1 -3% in terms of  $F1_1$  scores. On the other hand, DNN algorithms had relatively minor gains (i.e., less than 1%) in both accuracies and  $F1_1$  scores. Since DNN models utilize embedding vectors to identify the semantic representation of texts, those are less dependent on these optional preprocessing steps.

**Lesson 5: Accurate identification of self-deprecating texts remains a challenge.**

Almost 11% (out of 856 misclassified texts) of the errors from our best performing model was due to self-deprecating texts. Challenges in identifying such texts have also been acknowledged by prior toxicity detectors [78, 184, 194]. Due to the abundance of

self-deprecating texts among code review interactions [153, 112], this can be an area to improve on for future SE domain specific toxicity detectors.

**Lesson 6: Achieving even higher performance is feasible.** Since 85% of errors are due to failures of our models to accurately comprehend the contexts of words, we believe achieving further improved performance is feasible. Since supervised models learn better from larger training datasets, a larger dataset (e.g., Jigsaw dataset includes 160K samples), may enable even higher performances. Second, NLP is a rapidly progressing area with state-of-the-art techniques changing almost every year. Although, we have not evaluated the most recent generation of models, such as GPT-3 [29] and XLNet [191] in this study, those may help achieve better performances, as they are better at identifying contexts.

## 3.6 Threats to Validity

In the following, we discuss the four common types of threats to the validity of this study.

### 3.6.1 Internal validity

The first threat to validity for this study is our selection of data sources which come from four FOSS projects. While these projects represent four different domains, many domains are not represented in our dataset. Moreover, our projects represent some of the top FOSS projects with organized governance. Therefore, several categories of highly offensive texts may be underrepresented in our datasets.

The notion of toxicity also depends on many factors such as culture, ethnicity, country of origin, language, and relationship between the participants. We did not account for any such factors during our dataset labeling.

### 3.6.2 Construct validity

Our stratified sampling strategy was based on toxicity scores obtained from the perspective API. Although we manually verified all the texts classified as ‘toxic’ by the PPA, we randomly selected only  $(5,510^8 + 9,000 = 14,510)$  texts that had PPA scores of less than 0.5. Among those 14,510 texts, we identified only 638 toxic ones (4.4%). If both the PPA and our random selections missed some categories of toxic comments, instances of such texts may be missing in our datasets. Since our dataset is relatively large (i.e., 19,571 ), we believe this threat is negligible.

According to our definition, Toxicity is a large umbrella that includes various anti-social behaviors such as offensive names, profanity, insults, threats, personal attacks, flirtations, and sexual references. Though our rubric is based on the Conversational AI team, we have modified it to fit a diverse and multicultural professional workplace such as a FOSS project. As the notion of toxicity is a context-dependent complex phenomenon, our definition may not fit many organizations, especially the homogeneous ones.

Researcher bias during our manual labeling process could also cause mislabeled instances. To eliminate such biases, we focused on developing a rubric first. With the agreed upon rubric, two of the authors independently labeled each text and achieved ‘almost perfect’ ( $\kappa = 0.92$ ) inter-rater agreement. Therefore, we do not anticipate any significant threat arising from our manual labeling.

We did not change most of the hyperparameters for the CLE algorithms and accepted the default parameters. Therefore, some of the CLE models may have achieved better performances on our datasets through parameter tuning. To address this threat, we used

---

<sup>8</sup>Code review 1 dataset

the `GridSearchCV` function from the `scikit-learn` library with the top two CLE models (i.e., `RandomForest` and `DecisionTree`) to identify the best parameter combinations. Our implementation explored six parameters with total 5,040 combinations for `RandomForest` and five parameters with 360 combinations for `DecisionTree`. Our results suggest that most of the default values are identical to those from the best performing combinations identified through `GridSearchCV`. We also reevaluated RF and DT with the `GridSearchCV` suggested values, but did not find any statistically significant (paired sample t-tests,  $p > 0.05$ ) improvements over our already trained models.

For the DNN algorithms, we did not conduct extensive hyperparameter search due to computational costs. However, parameter values were selected based on the best practices reported in the deep learning literature. Moreover, to identify the best DNN models, we used validation sets and used `EarlyStopping`. Still, we may not have been able to achieve the best possible performances from the DNN models during our evaluations.

### 3.6.3 External validity

Although we have not used any project or code review specific pre-processing, our dataset may not adequately represent texts from other projects or other software development interactions such as issue discussions, commit messages, or question /answers on StackExchange. Therefore, our pretrained models may have degraded performances in other contexts. However, our models can be easily retrained using a different labeled datasets from other projects or other types of interactions. To facilitate such retraining, we have made both the source code and instructions to retrain the models publicly available [152].

### 3.6.4 Conclusion validity

To evaluate the performances our models, we have standard metrics such as accuracy, precision, recall, and F-scores. For the algorithm implementations, we have extensively used state-of-the-art libraries such as scikit-learn [131] and TensorFlow [1]. We also used 10-fold cross-validations to evaluate the performances of each model. Therefore, we do not anticipate any threats to validity arising from the set of metrics, supporting library selection, and evaluation of the algorithms.

## 3.7 Conclusion and Future Directions

This paper presents the design and evaluation of ToxiCR, a supervised learning-based classifier to identify toxic code review comments. ToxiCR includes a choice to select one of the ten supervised learning algorithms, an option to select text vectorization techniques, five mandatory and three optional processing steps, and a large-scale labeled dataset of 19,571 code review comments. With our rigorous evaluation of the models with various combinations of preprocessing steps and vectorization techniques, we have identified the best combination that boosts 95.8% accuracy and 88.9%  $F1_1$  score. We have released our dataset, pretrained models, and source code publicly available on Github [152]. We anticipate this tool being helpful in combating toxicity among FOSS communities. As a future direction, we aim to conduct empirical studies to investigate how toxic interactions impact code review processes and their outcomes among various FOSS projects.



## CHAPTER 4 TOXISPANSE: AN EXPLAINABLE TOXICITY DETECTION IN CODE REVIEW COMMENTS

### 4.1 Introduction

Toxicity, which is a large umbrella term comprising various antisocial behaviors such as offensive language, cyberbullying, hate speech, and sexually explicit content [12], is pervasive among various online platforms [112, 9]. As most of the Free and Open Source Software (FOSS) communities operate online, they are not immune from such toxic interactions [141, 112, 155]. As software development requires close collaboration and rapport among participants, toxicity can have severe repercussions for a FOSS community, which include decreased productivity, wastage of valuable time [141], negative feelings among the participants [57], barriers to newcomers' onboarding [165, 89], hostile environments towards minorities [77]. As proactive identification and mitigation of toxic interactions among FOSS developers are crucial, automated approaches can help FOSS moderators.

Prior studies [155, 153] found that off-the-shelf toxicity detectors do not perform well in the SE texts because some words ('die', 'kill', 'dead') in the SE context have a different meaning. Due to the unreliability of off-the-shelf natural language processing (NLP) tools on Software Engineering (SE) datasets [153, 91], recent works have proposed customized toxicity detectors trained on SE communications [141, 155]. While these tools boost reliable performances on SE datasets, we have identified a shortcoming of these two solutions. First, existing tools classify an entire paragraph on a binary scale, including hundreds of sentences. Even if only one of those sentences is toxic, it classifies the whole paragraph as toxic. A binary, paragraph-level classification of toxic texts may help the FOSS community to decide to remove a particular paragraph or establish a code of conduct for toxic

comments. However, it becomes time-consuming for a moderator to identify the offending excerpt(s) from a large paragraph. Second, due to the lack of cultural differences, a moderator may fail to identify the offending sentences from a paragraph classified as toxic by these tools. Being motivated by recent advances in explainable machine learning (ML) models, this study aims to create a new SE domain-specific toxicity detector that overcomes this particular shortcoming. We aim to *develop an explainable toxicity detector for the Software Engineering domain, which can precisely identify toxic excerpts from a text to assist FOSS moderators*. ‘Explainable’ in the context of this study indicates the ability of the classifier to pinpoint the words/phrases responsible for a text’s toxic classification [143].

Our solution aims to pave a path for automated text moderation to foster healthy and inclusive communication by reducing manual efforts to locate the toxic contents in FOSS developers’ communication and helping project maintainers quickly identify the negative parts of the comment to decide whether the text should be approved or rejected. Moreover, this technique will also enable finer-grained toxicity analyses from the patterns of toxic excerpts to determine possible remedies. Finally, our work can be a building block to develop solutions to proactively prevent toxic communications, similar to grammatical mistakes/typos detection tools.

A *toxic span* is defined as the fragment of a sentence or text that potentially causes the meaning of the text to be toxic [130]. A toxic span may contain a single word or a sequence of words. For example, “**Yuck**, this code is a **crap**” where the toxic spans are highlighted with **red** color. The SemEval-2021’s Task 5 organizers provided 10K toxic posts from the Civil Comments dataset [22] with labeled span character offsets. An ensemble solution using BERT [52] achieved the best performance among the teams participating in

this challenge. As prior research shows the necessity of SE domain-specific customization for NLP tools [91, 153], these toxic span detectors may not perform well on SE texts. Hence, we aim to build a customized solution.

On this goal, we select SE domain-specific toxicity dataset from Sarker *et al.* [155], which consists of a total of 19,651 Code Review (CR) comments with 3,757 ( $\sim 19\%$ ) toxic samples. We manually labeled this dataset using two independent raters to develop ground-truth annotations for the toxic spans within the toxic samples. We measured inter-annotator agreement using Krippendorff’s  $\alpha$  [97], which was 0.81 (almost perfect agreement). Using this dataset to establish a baseline model, we first developed a lexicon-based classifier. We trained and evaluated five sequence-to-sequence transformer models. During our 10-fold cross-validation-based evaluations, we found a model based on a fine-tuned RoBERTa [106] achieving the best an  $F1$ –score of 0.88. Primary contributions of this work include:

- *ToxiSpanSE*: The first explainable toxicity detector for the SE domain.
- An expert-annotated, span-level toxicity labels for 3,757 toxic code review comments.
- An overview of metrics to develop explainable NLP tools for the SE domain.
- An empirical evaluation of five transformer-based models with 19,651 code review texts.
- We make our model and dataset available for further analysis and use in the software engineering community. Available at: <https://github.com/WSU-SEAL/ToxiSpanSE>

## 4.2 Research Method

After selecting a dataset from a prior work [155], two authors independently annotated toxic spans in each text. Using this annotated dataset, we train and evaluate sequence-to-sequence transformer models that output the probability of each word belonging to a toxic span in the current text context. Finally, we use postprocessing steps to identify toxic spans from the output probabilities based on empirically determined thresholds. The following subsections detail our research methodology.

### 4.2.1 Dataset

The following subsections detail the dataset.

**4.2.1.1 Dataset Selection** The number of datasets for toxicity detection in Software Engineering communication is small [141, 153]. We explored previous studies on toxicity and antisocial behaviors in open-source interactions and found four studies that provided manually labeled datasets for toxicity [141, 153, 155] and incivility [63] detection. Raman *et al.* labeled only 611 texts from GitHub issue discussions as toxic or non-toxic [141]. In 2020, Sarker *et al.* provided a dataset of 6,533 CR comments and 4,140 Gitter messages labeled as toxic or non-toxic [153]. They also provided a rubric to identify a text as toxic or non-toxic. In a subsequent study of building a toxicity detection tool, they annotated 19,651 CR comments with binary, comment-level toxicity scores [155]. Moreover, Ferreira *et al.* provided an annotated 1,545 emails from the Linux Kernel Mailing List where they labeled each message as civil or uncivil [63]. Given that Sarker *et al.*'s dataset [155] is the largest one in the Software Engineering domain for toxicity detection, we selected their dataset for our study. Moreover, their detailed rubric also guides our annotators on how to

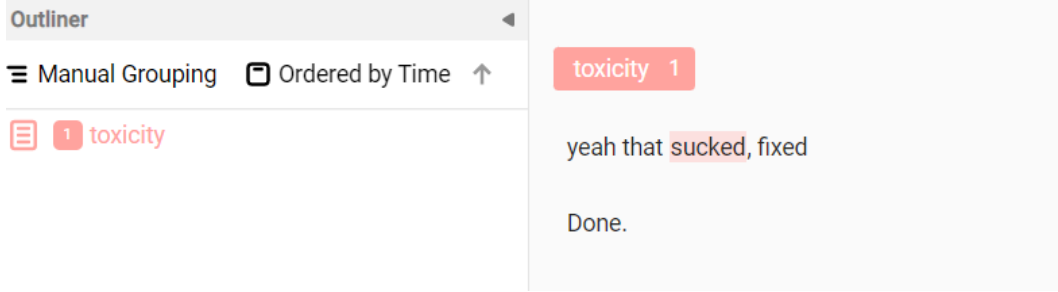


Figure 5: Manual Labeling using Label Studio, toxic span is highlighted

label toxic spans.

**4.2.1.2 Dataset Annotation** We got each of our toxic samples manually annotated by two independent annotators. To diversify the annotators, we chose one woman and one man for the annotation task. As manual labeling toxic text is a subjective task, we sought to reduce subjectivity bias during manual annotation by asking our annotators to carefully read and follow the rubric for toxicity developed by [155]. Although Sarker *et al.*'s dataset [155] includes 19,651 CR comments, only 3,757 are labeled as toxic. Therefore, our annotators only labeled the 3,757 toxic ones, assuming that the non-toxic samples do not include any toxic spans (empty span offsets).

For annotation, we use the Label Studio platform [176]. Figure 5 shows an example of our annotation interface. We exported the labeled data from Label Studio, which returns the code review text and corresponding character span offsets of the toxicity annotations for each sample. Table 11 shows two example annotations. The first example shows a toxic sample where the word 'sucked' makes text toxic, and this span occurs in character offsets 10-15. The third example is non-toxic and, therefore, has no span selected, resulting in an empty list ([]) of character offsets.

Table 11: Raw dataset with character spans. Red marked represents selected toxic words

Character Span Offsets	CR Text
[10, 11, 12, 13, 14, 15]	Yeah that <b>sucked</b> , fixed done.
[39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 74, 75, 76, 77]	I think the formatting may have gotten <b>screwed up</b> (or Gerrit made it look <b>ugly</b> )
[ ]	below assignments also should be removed

Table 12: Example of Inter-Rater Agreement and Conflict Resolution

Rater	Text	Token Array	Character Spans
Rater1	if you think it <b>sucks horri- bly</b> , that's fine as long as we can fix it	[0,0,0,0,1,1,0,0,0,0,0,0,0,0]	[16-29]
Rater2	if you think it <b>sucks</b> horri- bly, that's fine as long as we can fix it	[0,0,0,0,1,0,0,0,0,0,0,0,0,0]	[16-20]
Final Label	if you think it <b>sucks horri- bly</b> , that's fine as long as we can fix it	[0,0,0,0,1,1,0,0,0,0,0,0,0,0]	[16-29]

**4.2.1.3 Inter-annotator Agreement** We wrote a Python script to compare the spans produced by the two annotators. Unsurprisingly, we have found conflicts between the labeling samples. Previous studies have suggested several *chance-corrected* agreement measures to compute the inter-annotator agreement (IAA) [27]. *Chance-corrected* measures such as Cohen's  $\kappa$  [41], Fleiss'  $\kappa$  [65], and Scott's  $\pi$  [159] distinguish the observed disagreements ( $D_o$ ) from expected disagreements ( $D_e$ ). Therefore, these IAA measures are unsuitable for sequential tagging with potential partial overlaps [110].

Hence, similar to prior studies developing sequence tagging datasets [110, 124, 51, 27], we chose Krippendorff's  $\alpha$  [97] as the IAA measure. Krippendorff's  $\alpha$  is more robust as it can handle multiple annotators and missing values and considers partial agreement/dis-

agreements among the labelers. Krippendorff’s  $\alpha$  allows the distance-based formulation designed for context-specific tasks. The formula of Krippendorff’s is:  $\alpha = 1 - \frac{\hat{D}_o}{\hat{D}_e}$ , for a given distance function of  $D(a, b)$  where  $\hat{D}_o$  represents the observed average distance and  $\hat{D}_e$  is expected average distance [27]. Since Krippendorff’s  $\alpha$  calculates several distance functions such as nominal, interval, and ordinal [97], we chose the nominal distance function for our measurement. To calculate Krippendorff’s  $\alpha$  score, we wrote our script using the existing implementation [75].

Our labeled dataset has 3757 toxic code review samples labeled by two raters for toxic spans. For calculating Krippendorff’s  $\alpha$  with nominal distance, created two arrays of labels. We split each sample ( $s$ ) to a set of tokens  $s = t_0, t_1, \dots, t_j$  where  $t_j$  is a token inside the sample  $s$ . As our primary dataset contains the character level span offsets, we preprocessed it for token-level offsets. Table 12 shows an example of defining the token array for a sample. There is a total of 15 tokens after excluding the comma (,) from the input text. Therefore, we generate an array of 15 elements (same as the length of tokens) in which each position corresponds to a token from the CR text. We have a same-length array for Rater1 and Rater2 where we set 1 if the token is inside the span selection; otherwise 0. Following this process, we generated 3,757 arrays for all toxic samples of Rater1 and Rater2. For computing agreement, we merge all the token-level annotations for each rater into a single array where each array contains a total of 84,951 ratings. We calculated Krippendorff’s  $\alpha$  using the nominal distance between these two arrays and found the  $\alpha$  value as 0.81 (almost perfect agreement). This agreement score is significantly higher than a prior work [110] where the agreement score  $\alpha$  is 0.46.

**4.2.1.4 Conflict Resolution and Ground Truth** We found that two labelers have at least partial disagreement in 928 samples. Two of our raters (Rater1 and Rater2) discussed resolving the conflicts and assigned the final labels. Table 12 shows an example conflict with token arrays and corresponding character spans to illustrate our resolution process. At the end of this step, our final dataset includes CR comments and the corresponding character spans.

## 4.2.2 Tool Design

We compared two different approaches to design *ToxiSpanSE*. First, we used a lexicon-based naive approach, where words belonging to a predefined list are marked as toxic spans. Second, we used a supervised learning-based approach with five different transformer-based encoders. Figure 6 depicts our model architecture for the transformer-based models with an example prediction.

*ToxiSpanSE* takes input texts and associated labeled spans as input. After preprocessing, inputs are passed to the transformer models. The output of those models are arrays of floating point numbers ranging from 0 to 1, which indicate the probability of each token belonging to a toxic span. The following subsections detail lexicon-based and transformer-based approaches.

**4.2.2.1 Preprocessing** The model takes the CR text and the target spans (labeled spans) with character offsets as input. Further, we split each text into sentences using `en_core_web_sm` from the spacy library [82] and keep corresponding character span offsets for each sentence. We have 39,438 sentences after splitting 19,651 CR texts; among those, 5,465 sentences have toxic spans. Therefore, around 13.85% samples in our dataset have at



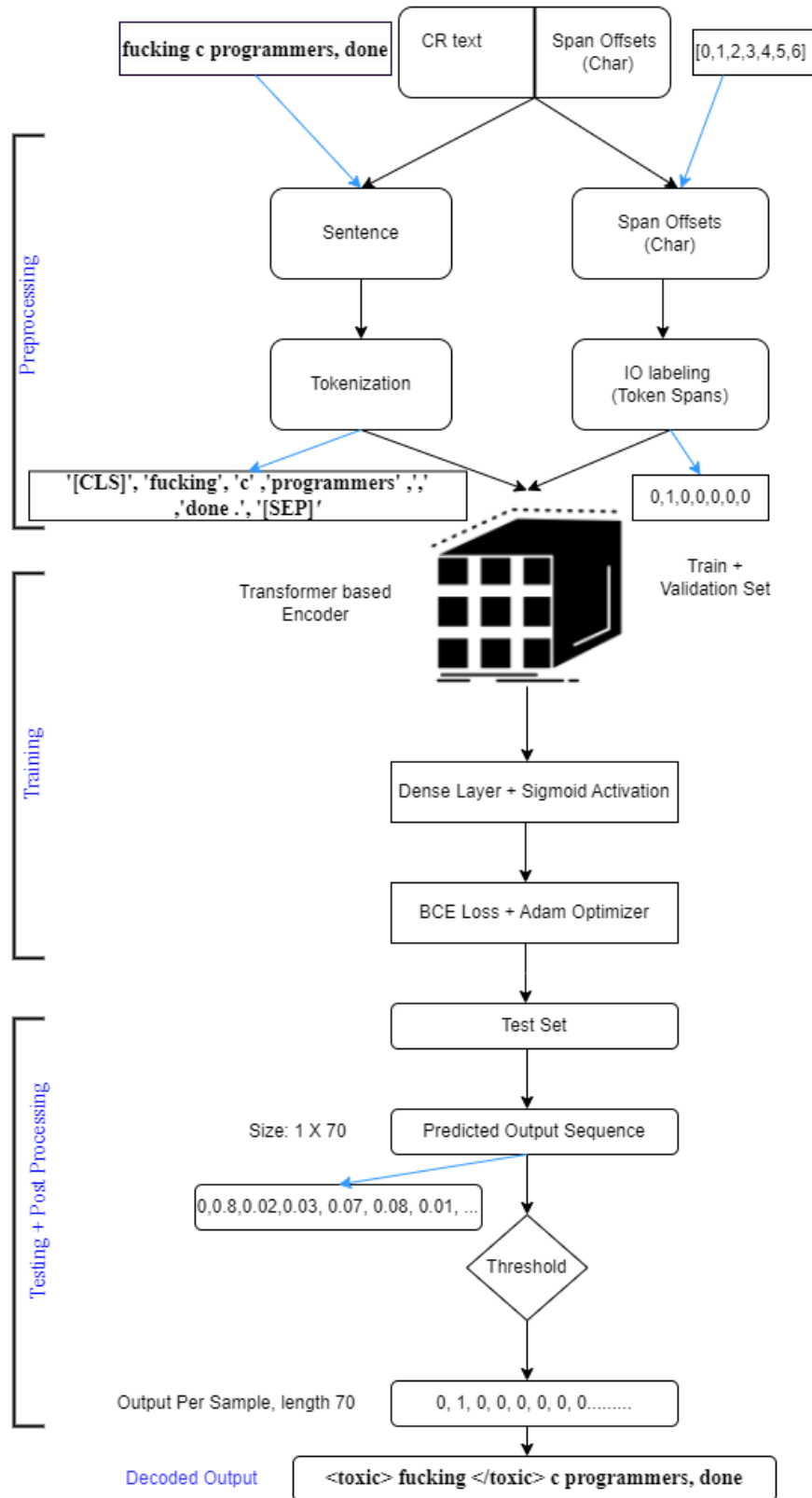


Figure 6: Model Architecture of ToxiSpanSE. Optimal threshold for each model was empirically selected (detailed in 4.3.2.2), Blue arrow → shows an example

least one toxic span. We chose sentence-level evaluation for two reasons: i) a sentence may itself have toxic spans, ii) prior work also did sentence splitting for toxic span detection [172]. Further, we apply a tokenizer to convert each sentence to corresponding tokens. In this study, we use tokenizers that are appropriate for each model. For the lexicon-based model, we chose *NLTK word\_tokenize* [108] from Python. On the other hand, we use transformer-based encoder models' corresponding tokenizer from hugging-face [185]. We use *AutoTokenizer* function and select: i) *bert-base-uncased*, ii) *roberta-base*, iii) *distilbert-baseuncased*, iv) *albert-base-v2*, and v) *xlnet-base-cased* tokenizers for their corresponding encoder model. Moreover, we set the maximum length to 70 during the tokenization of each sentence, as using a Python script we empirically found that 98.5% of our sentence samples have less than 70 tokens. This pruning was essential as taking a large token length significantly increases both required memory and training time. Each transformer-based pre-trained tokenizer splits the sentences into sub-word token strings and adds an unknown token to its dictionary if it finds them. For each sentence, each transformer-based tokenizer generates a special token at the start of the sentence at the end of the sentence (i.e., the bert-base tokenizer puts the [CLS] token at first and the [SEP] token at the end of each text). To pass the tokens of each sentence into the encoder model, we take three inputs for each sample from the tokenizers: `input_ids`, `token_type_ids`, and `attention_mask`. We can decode the vector to the original string using `input_ids`.

**4.2.2.2 IO Encoding** Prior NLP works with sequence labeling datasets followed BIO [172] or IO [36] tagging to encode the spans. *BIO* stands for Beginning, Inside, and Outside, where *B*-indicates the beginning token of a toxic span, *I*- indicates that the token is inside

the toxic span, and *O* indicates a token outside the toxic span. BIO is suitable for NLP tasks to divide a span of text into multiple chunks. As we aim to identify which text spans contain something toxic, a simpler one, i.e., the IO-encoding, is sufficient for our goal. Moreover, IO simplifies our processing steps. In our IO encoding, every *I* tag corresponds to a token inside a toxic span, and *O* indicates outside.

To get the target span, we use their *offset\_mapping* to determine whether that token is inside the selected toxic span. *Offset\_mapping* provides each token’s starting and ending character. Next, we generate a sequence of 0s (non-toxic token) and 1s (toxic token) for each sentence. Hence, our ground truth target is a sequence of 1’s and 0’s of maximum length (70). We consider the first and last token value as 0 for each sample because each tokenizer of pre-trained transformers generates a special token at the start and another at the end. Finally, for each sentence, we have a vector (length = 70) containing a sequence of 0s and 1s, which is the ground truth target vector.

**4.2.2.3 Lexicon Based Model** We also designed a naive model referred to as the ‘lexicon-based’ model for detecting toxic spans from our dataset. In general, toxic spans contain many common words, including profanity, sexually explicit, and swear words. The purpose of developing this model is to evaluate whether a simple lexicon search-based approach compares against state-of-the-art transformer-based models. Our lexicon-based model matches each token in a text against a list of common toxic tokens with our ground truth tokens. We curated a list of toxic tokens ( $\Sigma TOK = tok_0, tok_1, \dots, tok_i$ ) from two prior studies, which include 85 profane words from Sarker *et al.* [155] and the top 100 toxic tokens from Kurita *et al.* [100]. In total, our lexicon list contains 167 tokens since there

are overlapping tokens between those two lists.

*Ground Truth For Lexicon Based Classification:* To tokenize each sentence, we use *NLTK word\_tokenize* from Python. Further, we use *textspan* library [173] to get the exact location of selected tokens from the human labeling spans. We use a similar IO encoding approach for this model (70-length vector for each input), where if that is inside the labeled span, we put the token position as 1, otherwise 0.

*Lexicon Based Classifier Output:* We generate an output vector( $vec$ ) for each input sentence with a length of 70. We set the  $vec_i = 1$  if that token of the input matches with one of the tokens from the  $TOK$ , and  $vec_i = 0$ , otherwise.

**4.2.2.4 Transformer based model** The Transformer deep learning architecture that emerged in 2017 [182] is based on multi-head self-attention and has shown to perform significantly better than Recurrent Neural Network (RNN)-based models for sequence-to-sequence tasks. Transformers use a self-attention mechanism for computing the internal representation of input and outputs. Moreover, the transformer-based model does not require any pre-computed context-free embedding vectors. Instead, it can generate context-based embeddings by pre-training the entire model as an encoder for sequence-to-sequence tasks.

From the preprocessing steps, we have inputs ( $input\_ids$ ,  $token\_type\_ids$ ,  $attention\_mask$ ) for each sentence, and we have generated ground truth (target labels) using IO encoding. Inputs and targeted IO encoding are passed to the encoder layer to generate context-based embeddings. Since there are several Transformer based encoders available for sequence classification tasks, we consider the following transformers, which performed well in a

prior token-level classification task [127]. In this work, we used Transformer based encoders from the HuggingFace library [185], selecting the following pre-trained encoders:

- BERT: Devlin *et al.* proposed the pretraining of the Deep Bidirectional Transformers for Language Understanding (BERT) model in 2018 that was trained with masked language modeling (MLM) and next sentence prediction (NSP) [52]. We use the BERT-base model, which has 12 transformer layers with 768 hidden states and 12 attention heads with 110 M parameters. BERT can be fine-tuned with domain-specific datasets for sentence and sequence classification tasks.
- DistilBERT: Sanh *et al.* proposed a lighter and faster version of the bert-base model, using modeling distillation, referred to as “DistilBERT” [146]. It has around 66 M parameters (40%
- RoBERTa: An optimized version of BERT is RoBERTa, which achieved better performance than BERT-base in some NLP tasks by pretraining the model for a longer time and on more data than the original BERT [106]. The base model has the same architecture as the BERT-base model.
- ALBERT: ALBERT has a similar architecture to the BERT-base but has only 128 hidden embedding layers that reduced the total parameters to 12 M [102]. We chose to use the ALBERT-base model for this study.
- XLNet: Unlike the autoencoder (AE) language models (i.e., BERT), Yang *et al.* proposed XLNet, which is based on autoregressive language modeling [191]. XLNet sought to overcome the limitations of the BERT model by maximizing the expected

likelihood over all permutations of the factorization order. Moreover, its performance does not rely on data corruption. We use *xlnet-base-cased* model from the transformer library, which has a similar size as *BERT-base* model.

In this experiment, we select those pre-trained encoders from the HuggingFace library [185]. After the embeddings with size (1 X 70), we set a Dense layer to set the required final output size. Moreover, since we are doing a binary sequence classification task, the ‘sigmoid’ activation function is added to this Dense layer to generate the final output’s probability. Therefore, our final output vector is a sequence of floating point values (from 0 to 1 due to the sigmoid function) with a length of 70.

**4.2.2.5 Post Processing** After fine-tuning the model (details in next section), we predict the probability score with the test samples. The model provides a probability score from 0 to 1 for each token (70 per sample ( $s$ )). Using an empirically determined threshold (Section 4.3.2.2) parameter, we decide whether a token is in the toxic class (1) or non-toxic (0). Further, from the prediction vector, we generate a set ( $Pred_s$ ) of indexes for the output tokens in the toxic class. Our ground truth has already been preprocessed as toxic and non-toxic tokens. We also generate a set of the indexes of toxic tokens from the ground truth ( $G_s$ ) of the test set. Finally, we wrote a Python script to decode each token from the sample and show the output like figure 6. We have input “*fucking c programmers, done*”, and the model provides the output “*<toxic>fucking </toxic>c programmers, done*”. To make the tool user-friendly, we use a tag (*<toxic>*) at the start and (*</toxic>*) at the end for predicted tokens inside toxic spans.

### 4.3 Evaluation

The following subsections describe the evaluation.

#### 4.3.1 Evaluation Metrics

Since our task is based on a sequence tagging approach for toxic spans, we adjusted our evaluation metric from Martino *et al.* [46] that is based on Potthast *et al.*'s plagiarism detection work [133]. Recently, Pavlopoulos *et al.* also used the same metric for toxic span detection in online discussions [128]. We decided to use this metric because it provides partial credit for matching the toxic spans inside a sequence. Unlike prior studies [46, 133, 128], we have chosen token-level comparison instead of character level for measuring the precision, recall, and  $F_1$  score. The token-level comparison is taken because token-to-token comparison provides more explainability (comparing the ground truth toxic word to predicted toxic word) than the character label comparison. For example, a token(s) can represent the toxicity of the whole text, whether a character inside a token does not represent that meaning.

Let a code review sample( $s$ ) represent a sequence of tokens  $tok_0, \dots, tok_j \subseteq s$ . After IO encoding, the ground truth vector is a sequence of 1's and 0's with 70 values. We calculate the ground truth token offset as  $G_s = pos_{tok_m}, \dots, pos_{tok_n}$ . So, for each sample( $s$ ),  $G_s$  contains the position of all toxic tokens ( $pos_{tok_m}$ ). When no toxic token exists in the sample, the  $G_s = empty$ . Similarly, a predictor model predicts the tokens with a floating value. Further, we use a threshold (our experimental evaluation to identify optimal thresholds for each setup is detailed in Section 4.3.2.2) to decide whether that token is toxic (1) or non-toxic (0). We generate the predicted token offsets  $Pred_s$  for each sample from

that vector. For better understanding, we put five examples in table 13 with ground truth (GT) and predicted (Pred) token offsets. Since we processed our text into tokens in the preprocessing steps, the first token offset ( $tok_0$ ) is for the special token (such as [CLS] for bert tokenizer). Therefore, our first token (i.e., ‘it’) position count starts from 1.

In the first example of table 13, we observe that [7, 8, 9] token offsets are marked as toxic, whereas [7,8,11] offsets are predicted. So, there are two exact matches (7, 8), one position is not predicted (9), and one position is falsely predicted (10) as toxic. Hence, we used precision (P), recall (R), and F1 for each sample  $s$  are calculated as follows:

$$P^s(Pred_s, G_s) = \frac{|Pred_s \cap G_s|}{|Pred_s|} \quad (4.1)$$

$$R^s(Pred_s, G_s) = \frac{|Pred_s \cap G_s|}{|G_s|} \quad (4.2)$$

$$F1^s(Pred_s, G_s) = \frac{2 * P^s(Pred_s, G_s) * R^s(Pred_s, G_s)}{P^s(Pred_s, G_s) + R^s(Pred_s, G_s)} \quad (4.3)$$

In the equation 4.1, we define the precision  $P^s$  for each sample. We define the numerator as the length of the intersection of the set of predicted offsets ( $Pred_s$ ) and ground truth token offsets ( $G_s$ ). The denominator is the length of predicted offsets ( $Pred_s$ ). Similarly, we calculate the recall ( $R^s$ ) by using equation 4.2. Finally, we combined equation 4.1 and 4.2 to calculate the  $F1$  in equation 4.3.

However, these equations can fail due to 0 in denominators. For example, if a model predicts none of the tokens from a sentence belonging to toxic spans, precision is undefined



Table 13: Example of model predictions. **red** represents the toxic tokens

Ground Truth Text	Predicted Text	GT Off-set	Pred Offset	P	R
it is not clear in code <b>what the hell</b> rest means	it is not clear in code <b>what the hell</b> rest <b>means</b>	[7,8,9]	[7,8,11]	0.67	0.67
This will become a trash quick with such a generic name.	This will become a <b>trash</b> quick with such a generic name.	[]	[5]	0	0
Your indentation is <b>messed up</b> again	Your indentation is messed up again	[4,5]	[]	0	0
I do the same as you're suggesting in other code	I do the same as you're suggesting in other code	[]	[]	1	1
<b>Oh, shit</b> , you're right	Oh, <b>shit</b> , you're right	[1,2,3]	[3]	1	0.33

for that sentence. Similarly, for a correctly marked non-toxic instance, recall is undefined.

We used the same approach as both Pavlopoulos *et al.* [128] and the SemEval-2021 Task 5 [130] to measure a variation of precision, recall, and F-score for span detection tasks.

In this variation, if the number of predicted toxic tokens is 0 (i.e.,  $|Pred_s| = 0$ ), we check the number of toxic tokens in the ground truth set ( $|G_s|$ ). If both sets are empty, the prediction is correct, and we assign this prediction a  $precision = 1$ ; otherwise, we assign  $precision = 0$ . On the other hand, if the ground truth set is empty (i.e.,  $|G_s| = 0$ ), we assign  $recall = 1$  only if the predicted set is also empty (i.e.,  $|Pred_s| = 0$ ), and  $recall = 0$  otherwise. We would also like to let you know that these custom precision/recall measures do not follow traditional precision/recall curve characteristics due to this variation.

We compute and report mean precision, recall, and F-score for the toxic and non-toxic instances separately since our dataset is highly imbalanced. In our results,  $P_0$ , and  $P_1$  denote precision for the non-toxic and toxic instances, respectively. We consider  $F1_1$  as our main measure for the experiments because it shows the measurement of the model for the minority (toxic) class tokens.

To clarify the metric measurement, we show the calculation from the examples of Ta-

ble 13. Here, for the first sample, the numerator for equation 4.1 and 4.2 is 2 (i.e., two offsets are intersected). The denominator for equation 4.1 and 4.2 is 3. So, precision for toxic class ( $P_1$ ) is:  $\frac{2}{3} = 0.67$ , recall for toxic class ( $R_1$ ) is:  $\frac{2}{3} = 0.67$ . We calculated the  $F_1$  as 0.67. While considering the second sample, the length of ground truth offset  $|G_i = 0|$ , but its' predicted offset length  $|Pred_i = 1|$ . Since its ground truth is empty, its' metric aligns with the non-toxic class. Hence, equation 4.2 (recall) would be  $\frac{0}{0}$  that would be undefined. For that reason, we put  $P_0 = 0$  and  $R_0 = 0$  in the second case. Similarly, the third example belongs to the toxic class metric where the length of  $|Pred_i = 0|$ . In this case, the equation 4.1 (precision value) will be  $\frac{0}{0}$ . For that reason, we set  $P_1 = 0$  and  $R_1 = 0$  in this case. For the fourth example, both ground truth and prediction are empty. In those cases, we consider both  $P_0 = 1$  and  $R_0 = 1$  because we provide full credit for this. The last example shows the measurement where precision and recall are not the same.

### 4.3.2 Experimental Setup

We have done an extensive analysis of each model in our experiment. For accurate estimation of the model performance, we have done 10-fold cross-validation. Using Python's `random.seed()`, we create stratified 10-folds, which keep a similar ratio between toxic and non-toxic classes for all splits. Further, in each fold, we keep 80% for the train set, 10% for the validation set, and the rest 10% for the test set. We used an NVIDIA Titan RTX GPU with 24 GB memory in Ubuntu 20.04 LTS workstation to conduct the evaluation.

**4.3.2.1 Hyperparameters** We set the following hyperparameters during the training of our model:

- *Loss Function:* We chose a variant of a binary cross-entropy loss function for our task.

Since we have multiple tokens (length = 70) with a range of fractional values from 0 to 1, we have added a too-small value (epsilon from Keras) with each prediction. This procedure will help our model to be more stable and prevent the prediction from 0 that can cause ( $\log 0 = \text{undefined}$ ) problems. Moreover, we added a clipping between 1 and 0 inside the binary cross-entropy loss to avoid exploding gradients.

- *Optimizer and Learning Rate:* We use *Adam* optimizer with a learning rate  $1e - 5$ .
- *Number of Epochs:* We set the number of epochs as 30 in each fold.
- *Early Stopping Monitor:* To prevent the model from overfitting during the training, we set the *EarlyStopping* function from the Keras library [40] where with the monitor with 'val loss'. During training for each epoch, the model is trained with the training dataset and tested with the validation dataset. While the validation loss does not decrease for four consecutive epochs, the model stops training and saves the best model. We also empirically monitored that the optimal 'val\_loss' provided the best  $F1_1$  score for the validation set.

**4.3.2.2 Threshold Selection** Since the models from each fold generate a sequence of 70 length vectors, with a floating value for each token, we need to set a threshold to convert to binary (0 or 1) values. In text classification tasks, a threshold of 0.5 is a common choice [155, 128]. We evaluated our validation set to identify optimum thresholds by varying the threshold from 0.01 to 0.99 with a 0.01 increment. We aim to find the threshold resulting in the best  $F1_1$  score for the validation set. We empirically evaluated each model with this threshold variation for the validation set in 10-fold. With the mean from

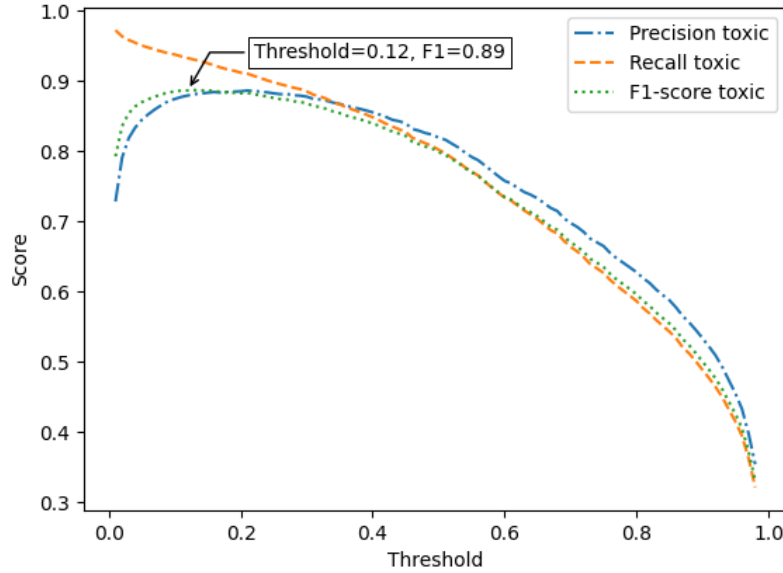


Figure 7: Threshold variation for RoBERTa model (Using Validation Set)

10-folds, we found the optimal threshold value for each model to maximize  $F1_1$  score. For example, the RoBERTa model achieved the best  $F1_1$  of 0.89 with a threshold of 0.12 with validation data set. Figure 7 shows performance (precision, recall, and F1) variations for the RoBERTa model against threshold variations using the validation set. We also noticed that the  $F1 - score$  for the toxic class remains the same from threshold 0.08 to 0.18 for the RoBERTa model. As we take a *variation* of precision and recall measurement, the plot does not behave like the general characteristics of precision and recall. Figure 7 also depicts that increasing the threshold value decreased both precision and recall. After calculating the optimal threshold from each model using the validation set, we use that optimal threshold for the corresponding model to predict the test set. During the test set prediction, we also did a similar 10-fold cross-validation and got the mean of each metric. Finally, we report the results of each model's performance with the optimal threshold in Table 14.

Table 14: Experimental Results with the optimal threshold. The runtime of each model and performances during each fold is included in the replication package [152]

Models	Optimal Threshold	Non-toxic words			Toxic words		
		$P_0$	$R_0$	$F1_0$	$P_1$	$R_1$	$F1_1$
Lexicon-based	NA	<b>0.95</b>	<b>0.95</b>	<b>0.95</b>	0.75	0.67	0.69
BERT-base	0.15	<b>0.95</b>	<b>0.95</b>	<b>0.95</b>	<b>0.87</b>	0.89	0.86
RoBERTa	0.12	0.92	0.92	0.92	<b>0.87</b>	<b>0.93</b>	<b>0.88</b>
DistilBERT	0.17	0.94	0.94	0.94	0.85	0.89	0.85
ALBERT	0.11	0.92	0.92	0.92	0.85	0.89	0.85
XLNet	0.10	0.90	0.90	0.90	0.79	0.88	0.81

#### 4.3.3 Results with optimal threshold

We present the results with the optimal threshold for each model in table 14. In the first row, we put the lexicon-based models’ performance. Many of the spans in our ground truth contain some specific toxic words. Therefore, the lexicon-based model performed quite well in our study and achieved 0.69  $F1_1$  score. This lexicon-based matching approach also performed better than other transformer models (except the BERT-base model) for non-toxic classes. However, there is a generalizability issue with using the matching approach for detecting toxic spans.

Since our dataset is highly imbalanced, having a large number of empty spans, all of the five transformer models achieved similar scores for  $P_0$ ,  $R_0$ , and  $F1_0$  in the range of  $0.90 \sim 0.95$ . For toxic tokens, RoBERTa outperformed the other four models and achieved 0.87 precision, 0.93 recall, and 0.88  $F1_1$  score. BERT-base model achieved the second best performance with  $F1_1 = 0.86$ . DistilBERT and ALBERT models achieved similar performance with 0.85  $F1_1$  score. However, DistilBERT has fewer parameters than other transformer models in our study, and this model is faster during fine-tuning than others. On the other hand, XLNet lacks the performance for both toxic and non-toxic classes than other

transformer-based models.

**Finding 1:** *While all five transformer-based models achieved better performance than the lexicon-based approach for toxic class, the RoBERTa model outperformed other models with 0.88  $F1_1$  score.*

#### 4.3.4 Error Analysis from the best model

To provide more clarity on our model performance, we have manually analyzed the misclassification with our best-performing model. Therefore, we ran our best-performing RoBERTa model with a threshold of 0.12 to print misclassification instances. In our final preprocessed dataset, we have a total of 39,438 sentences. During misclassified instance printing, we have done 10 folds. For that reason, we can cover all the samples from our dataset. We have found a total of 3406 ( $\sim 8.63\%$ ) sentences where misclassification occurred. However, we categorized the errors into three different types because we are doing a sequence classification problem. Table 15 depicts some examples of errors from our model where the first column shows the error types, the second column is for ground truth (GT) token span offsets, and the third column is for predicted token span offsets.

**4.3.4.1 Partial Disagreement** Since we are giving partial credit for the sequence classification metric, we decided to formulate a new error category as Partial Disagreement (PD). We consider an error as PD where the ground truth span offset has some values and the predicted span has some value with some disagreements. We found a total of 945 sentences (2.4% of the total sample and 27.75% of the error sample) in this category. The first three examples of Table 15 represent the PD category. In the first example, we can see

Table 15: Example of some errors

Error Types	GT Span	Predicted Span	Actual Text	Predicted Text
PD	[14,15,16]	[15,16]	rest seems like too generic name and it's not clear in code what the hell rest means.	rest seems like too generic name and it's not clear in code what the hell rest means.
PD	[1]	[1,2]	Damn grammar :-P	Damn grammar :-P
PD	[1,2]	[1,2,3,4,5]	O crap, hate me: do we still need this one?.	O crap, hate me: do we still need this one?.
FP	[]	[1]	FC related code should be removed.	FC related code should be removed.
FP	[]	[1]	stupid design on my part.	stupid design on my part.
FP	[]	[7]	As far as I understood, WTF::HashMap doesn't support it.	As far as I understood, WTF::HashMap doesn't support it.
FN	[1]	[]	Evil spaces must die.	Evil spaces must die.
FN	[2,3,4]	[]	Your brain is deficient, please fix, also done.	Your brain is deficient, please fix, also done.

that our rater labeled the ‘what the hell’ phrase inside the toxic span, whereas the model predicted ‘the hell’ as toxic.

**4.3.4.2 False Negatives** We consider the occurrence of False Negatives (FN) where the sentence has single/multiple toxic span offsets but the model predicts no span. The high number of FNs would cause a serious problem for the user of this model because it will miss many toxic instances. Our model has a low amount of FNs where it can not predict toxic span for 235 sentences ( $< 1\%$  of our total samples, and  $6.90\%$  of the error sample). The last two examples on the table 15 are FNs that contain some rare toxic phrases (i.e., ‘Evil’, ‘brain is deficient’). For that reason, our classifier could not predict them as toxic.

**Finding 2:** *False Positives instances dominated the list of misclassifications. Our models’ reliable performances can be attributed to lower instances of ‘Partial Disagreements’ and ‘False Negatives.’*

## 4.4 Discussion

**Lesson #1: Toxic span selection is a highly subjective task for annotators:** After the initial labeling of the toxic spans, we found that two of our raters showed at least partial disagreement for 928 samples. Human raters do not agree with all samples in selecting the toxic spans. In Some cases, both annotators select the profane words, but one may miss the associated words. For example, “*doesn’t this just mean we fucked up the mips syscall.S in bionic?*” text where first labeler marked “*fucked up*” as toxic span and second annotators marked only “*fucked*” as toxic span. In some of the other cases, self-directed anger words such as ‘argh’ or ‘damn’ were mislabeled. Therefore, for similar labeling tasks, we would recommend spending time building a rubric and agreed-upon understandings among the annotators to achieve high IAA.

**Lesson #2: Lexicon-based approach performs well but does not provide generalizability:** We found that our lexicon-based matching approach achieved 0.69  $F1_1$  score. Moreover, it performed better for non-toxic classes than the transformer-based supervised training approaches because the lexicon-based approach has less probability of flagging a non-toxic token as toxic (less FPs). Though it performed well in our dataset, using this model for a new software engineering dataset may cause serious threats. This approach is just token-matching and will miss the associated toxic tokens. Moreover, some tokens do not always represent toxicity. For example, in *I will kill you*, where *kill* is toxic. But in *Make sure you kill the process first*, here *kill* is not toxic. For that reason, the lexicon-based approach may generate a large number of FPs.

**Lesson #3: Transformer-based models are reliable and explainable for the FOSS com-**



**munity:** In our extensive evaluation, we found that the RoBERTa model outperformed others by achieving 0.88  $F1_1$  score while the other three transformer models also performed well for the toxic class. Since the sequence tagging approach is a challenging task for a new domain, our model can be used by the project maintainers to flag the toxic portion of a text. Moreover, since our best model has fewer false negative cases, FOSS maintainers can use this tool to detect the actual toxic segment from a toxic comment. Apart from that, we have used friendly post-processing, which provides an output with tagging: “*you’re not talking about neutron, (<toxic>) shut up (</ toxic>)*”.

**Lesson #4: Proactive toxic prevention tool development:** Since *ToxiSpanSE* is highly precise in identifying toxic excerpts, it is possible to leverage this model to proactively discourage toxic texts. For example, a Gerrit code review plug-in can be developed that highlights toxic excerpts similar to grammatical mistakes or typos while a review is being written. Such highlights will make an author aware of potential toxic interpretations and may initiate a self-reflection.

Although the project maintainers would decide on content moderation, they can use our work to develop a tool to rephrase the toxic content to civil comments. Prior work introduced this concept for online communication text [128]. The research community from the SE domain and FOSS maintainers may think of this step to reduce the toxic comments from developers’ communication.

## 4.5 Threats to Validity

*A. Internal Validity:* Our selection of code review dataset from a prior work [155] remains a threat to validity. Biases in the curation of this dataset propagate to our study as well.

However, Sarker *et al.* [155]’s dataset remains the largest labeled toxicity dataset for the SE domain, and it was curated using stratified sampling criteria to span various toxic instances. Since this selected dataset contains only code review comments, it may not adequately represent various other categories of developer communications such as issue discussions or technical question answering. However, that threat may be minimal as we focus on toxic phrases separate from a text’s technical contents.

*B. Construct Validity:* Annotator bias during manual labeling is a potential threat to validity. To mitigate this threat, we reused an already established rubric [155], used a gender-diverse group of annotators, including one woman and one man, and arranged a discussion with the annotators to build a shared understanding of the rubric before starting the annotation process. Moreover, we followed recommended practices of independent labeling and conflict resolution through discussions. A high value of Krippendorff’s  $\alpha$  (i.e.,  $-0.81$ , ‘almost perfect agreement’) indicates the reliability of our labeling process.

We followed the definition and rubric of toxicity established by Sarker *et al.* [155]. While Sarker *et al.*’s conceptualization of toxicity is similar to the ones proposed by Raman *et al.* [141] and Miller *et al.* [112], there are subtle differences between their rubrics and ours. Therefore, models trained using our dataset may have degraded performance on datasets released by other studies. Similarly, our models may encounter degraded performance on SE datasets of other anti-social communication, such as incivility [63] and destructive criticism [77]. However, this limitation does not apply to our tool pipeline, and it can be retrained to fit other conceptualizations.

*C. External Validity:* Our dataset includes code review comments from four FOSS projects

using Gerrit. While we do not have any evidence suggesting the code review interactions on Gerrit are different from other review platforms, such as GitHub pull requests, Phabricator, CodeFlow, and Critique. Our dataset may not adequately represent communication on those platforms. Similarly, as ToxiSpanSE is trained on code review comments, it may have degraded performance on other SE datasets, such as issue discussion, app reviews, and technical question answering. However, this limit does not apply to our approach, and using a dataset curated from other sources, ToxiSpanSE can be retrained to develop context-specific detectors.

*D. Conclusion Validity:* Using the position-based metric threatens conclusion validity. To mitigate this threat, we adopted our metrics from prior studies with span detection [128, 46]. Moreover, since most of our labeled instances are non-toxic, we separately report the performance measures (i.e.,  $P$ ,  $R$ , and  $F1$ ) for both toxic and non-toxic classes.

## 4.6 Conclusion and Future Work

In this work, we introduced *ToxiSpanSE*, a SE domain-specific explainable toxicity detector that, in addition to identifying toxic texts, precisely marks the phrases responsible for this prediction. We trained and evaluated *ToxiSpanSE* using 19,651 Code review comments that were manually annotated to mark toxic phrases. We have fine-tuned five different transformers based on encoders that predict the probability of a word being toxic in a given context. We also empirically identified optimum probability thresholds for each of the five models. Our evaluation found a RoBERTa model achieving the best performance with 88%  $F1_1$  score. We have made our dataset, scripts, and evaluation results publicly available at <https://github.com/WSU-SEAL/ToxiSpanSE>. In addition to facilitating

finer-grained toxicity analysis among SE communication, we hope this tool will motivate explainable models for other SE domain-specific NLP classifiers, such as sentiment analysis and opinion mining.

## CHAPTER 5 THE LANDSCAPE OF TOXICITY: AN EMPIRICAL INVESTIGATION OF ANTISOCIAL BEHAVIORS ON GITHUB

### 5.1 Introduction

GitHub has become the most popular platform for hosting Free and Open Source Software (FOSS) projects. In 2023, GitHub hosted more than 284 million public repositories [48]. As FOSS communities continue to grow, so do the interactions among contributors during various software development activities, such as issue discussions, pull request reviews, and Gitter messages. While these interactions are crucial to facilitating collaborations among the contributors, sometimes they may cause harm due to anti-social behaviors. Recent studies have investigated such interactions among FOSS developers using various lenses, which include ‘toxicity,’ [112, 155, 153, 141, 151], ‘incivility’ [63], ‘destructive criticism’ [77], and ‘sexism and misogyny’ [169]. Although definitions of these lenses differ, they all share a common attribute: the potential to cause severe repercussions among the victims. Negative consequences of these antisocial behaviors include stress and burnout [141], negative feelings [57, 63, 77], pushbacks [115], turnovers of long-term contributors [53, 103, 186, 11], adding barriers to newcomers’ onboarding [141], and hurting diversity, equity, and inclusion (DEI) by disproportionately affecting women and other underrepresented minorities [77, 7, 116, 115]. Moreover, the prevalence of antisocial behaviors present substantial challenges to the growth and sustainability of a FOSS project.

Recent Software Engineering (SE) research has focused on characterizing antisocial behaviors and their consequences through surveys, interviews, and qualitative analyses [63, 112, 57, 77, 62]. In a sample of 100 GitHub issue comments, Miller *et al.* found en-

titlement, arrogance, insult, and trolling as the most common forms of toxicity [112]. Destructive criticism is another anti-social behavior found in code reviews [77]. Although destructive criticisms are rare, they may have severe repercussions, which include conflicts, demotivation, and even hindering the participation of minorities [77, 57]. Ferreira *et al.*'s qualitative study of rejected patches in Linux kernel mailing lists reported frustration, name-calling, and impatience as the most prevalent forms of incivility [63]. Another recent workplace investigation reported inappropriate communication style as the primary cause of incivility [140]. However, many of these studies suffer from limitations such as small sample sizes or narrow focuses on specific projects [62, 112], organizations [137, 57], or a small developer group [140], which raises questions regarding the external validity of these findings at different contexts. We are also missing a quantitative empirical investigation of how various measurable characteristics of project contexts and participants are associated with the prevalence of various anti-social behaviors. Such an investigation is necessary to formulate mitigation strategies for the broader FOSS ecosystem.

In response to this need, we have conducted a large-scale empirical investigation of toxicity during Pull requests (PRs). We selected PR since it is a crucial mechanism to attract contributions from non-members and facilitate newcomers' onboarding among FOSS projects [70]. PRs allow contributors to propose changes, which are then reviewed by other community members. Due to the interpersonal nature of PR interactions and the potential for dissatisfaction due to unfavorable decisions, PR interactions may raise conflicts and anti-social behaviors. Among various anti-social lenses, we select the 'toxicity' since it has been most widely investigated [141, 112, 151, 153] and has a reliable automated identification tool [155], which is a prerequisite for a large-scale empirical investigation.

Following the toxicity investigation framework proposed by Miller *et al.* [112], we investigate four research questions to characterize i) the nature of toxicity, ii) projects that had higher toxic communication than others, iii) contextual factors that are more likely to be associated with toxicity, and iv) the participants, respectively. We briefly motivate each of the research questions as follows.

**RQ1: [Nature]** *What are the common forms of toxicity observed during GitHub Pull Requests?*

*Motivation:* Understanding the nature of toxicity may help project maintainers improve guidelines and interventions to foster respectful and constructive interactions. Although existing studies proposed various categorizations of SE domain-specific toxicity [112, 155] and incivility [63], due to inadequate samples and potential sampling biases (i.e., only locked issues), two key insights remain missing: i) whether these studies missed additional forms of toxicity and ii) how frequently various toxicity categories occur on GitHub. RQ1 aims to fill in these missing insights.

**RQ2: [Projects]** *What are the characteristics of the project that are more likely to encounter toxicity?*

*Motivation:* Does toxicity vary across project sponsorship, age, popularity, quality, domain, or community size? The identification of these factors will help project management undertake context-specific mitigation strategies. Determining which projects are more likely to suffer from toxicity may help a project's management allocate resources and define mitigation strategies. Moreover, this insight will help a prospective joiner select projects.

**RQ3: [PR Context]** *Which pull requests are more likely to be associated with toxicity on GitHub?*

*Motivation:* Does toxicity occur during poor-quality changes, unfavorable decisions, large changes, or delayed decisions? Understanding contextual factors is crucial to educating developers in avoiding specific scenarios.

**RQ4: [Participants]** *What are the characteristics of participants associated with toxic comments?*

*Motivation:* Prior studies [112, 63] suggested that some participants are likelier to author toxic comments due to their communication style or cultural background. On the other hand, another study suggests that participants representing underrepresented groups or newcomers are more likely to be targets [115]. RQ4 aims to identify personal characteristics associated with being authors or victims of toxicity. This insight will help project management prepare community guidelines to combat toxicity.

**Research method:** We conducted a large-scale mixed-method empirical study of 2,828 GitHub-based FOSS projects randomly selected based on a stratified sampling strategy. Our sample includes 16 million pull requests, 69.5 million issue comments, and 32 million PR comments. Using ToxiCR [155], a state-of-the-art SE domain-specific toxicity detector, we automatically classified each comment as toxic or non-toxic. Additionally, we manually analyzed a random sample of 600 comments to validate ToxiCR’s performance and gain insights into the nature of toxicity within our dataset. With ToxiCR demonstrating a reliable performance, we trained multivariate regression models to explore the associations between toxicity and various attributes of projects, PR contexts, and participants.



**Key findings:** We found 11 forms of toxic comments on GitHub PR review comments, with object-directed toxicity being a new form unreported in prior studies. The results of our study suggest that profanity is the most frequent toxicity on GitHub, followed by trolling and insults. While a project’s popularity is positively associated with the prevalence of toxicity, its issue resolution rate has the opposite association. Corporate-sponsored projects are less toxic, but gaming projects are seven times more likely than non-gaming ones to have a high volume of toxicities. FOSS developers who have authored toxic comments in the past are significantly more likely to repeat them and become toxicity targets.

**Novelty of this study:** This study differs from prior academic empirical investigations in three ways. First, prior studies suffer from sampling biases, such as locked issues [112, 141] or rejected patches [63]. Therefore, characteristics of toxicity outside these known negative contexts are missing. Second, these investigations are qualitative. While these investigations are crucial to forming hypotheses, whether these hypotheses apply to a broader spectrum of FOSS projects remains unanswered. Finally, these studies explored a limited set of factors, whether other plausible factors, such as community size, project popularity, code complexity, and unresolved defects associated with toxicity, remain unanswered.

**Contributions** The primary contributions of this study include:

- An empirical investigation of various categories of toxic communication among GitHub Pull requests.
- A large-scale empirical investigation of factors associated with toxicity on GitHub.
- Actionable recommendations to mitigate toxicity among FOSS projects.

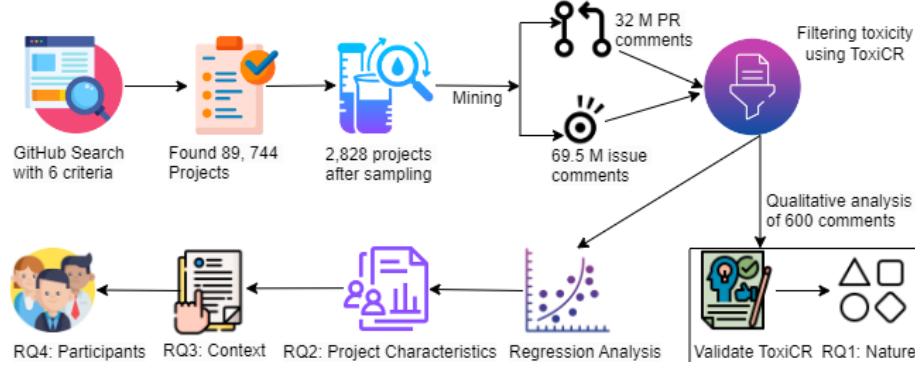


Figure 8: An overview of our research method

## 5.2 Research Method

Figure 8 provides an overview of our research methodology, detailed in the following subsections.

### 5.2.1 Project Selection

We leverage the GitHub search tool developed by Dabic *et al.* [47], which enables filtering based on various criteria such as the number of contributors, programming language, forks, commits, and stars to select candidate projects. Following recommendations by Kalliamvakou *et al.* [93], we searched for projects satisfying the following six criteria: i) uses one of the top ten programming languages on GitHub: Java, C, C++, Python, JavaScript, C#, Go, PHP, Typescript, and Ruby; ii) has at least 20 contributors, iii) is publicly available with an open source license; iv) at least two years old, v) has at least 20 PRs; vi) and has at least 10 stars. The first five criteria ensure the selection of FOSS projects with adequate analyzable interpersonal communication among the contributors. The remaining criterion reduces the search space; without this filter, the number of projects grows exponentially but adds only trivial FOSS projects.

Our search conducted in September 2022 found 89,744 projects. We exported the re-

sults as a CSV file and computed each project’s activity level as the PRs per month. We divided the projects into three groups based on Pull Request Frequency (PRF) on the GitHub – i) Low PRF (PRF-L): if it has less than 8 PRs per month (i.e., less than 2 PRs/week); ii) Medium PRF (PRF-M): if it has between 8 -31 PRs per month (i.e., 2-8 PRs/week); and iii) High PRF (PRF-H): if it has more than 32 PRs per month (i.e., >8 PRs/week). These two thresholds, 8 and 32, represent approximately 75 and 90 percentiles based on activity level, respectively. We randomly selected 800 projects from each of the three groups. We chose this sample size to satisfy a 3% margin of error with a 95% confidence interval [190]. Since prior studies have suggested higher occurrences of toxicity among gaming projects [112], we also added projects with the topic ‘game.’ We found 1,000 such projects since GitHub search API limits access to search results to the first 1,000. However, most projects did not satisfy criteria such as a minimum number of participants or PRs. Hence, the gaming group includes 439 projects, adequate for 95% confidence interval (CI) and a 5% Margin of Error [190]. However, some projects were no longer available for mining (e.g., deleted or moved). Therefore, our final dataset contains 2,828 projects.

### 5.2.2 Dataset preparation

We wrote a Python Script using the PyGithub library [88] to mine all the PR details, metadata, PR labels, user information, PR review comments, and issue comments from the selected projects and store them in a MySQL database. Our data mining started in October 2022 and completed in January 2023. Our dataset requires approximately 172 GB of storage. Our dataset includes approximately 16.1 million (M) PRs, 69.5 M issue comments, 32 M PR comments, and 1.3 M unique users. We also mined all publicly available user information, including profile photos, emails, full names, and user types. We exclude all

the interactions from Bot accounts (i.e., `userType='Bot'`). However, we also noticed many bot accounts using incorrect flags (i.e., `userType='User'`). Therefore, we filtered accounts with bot-specific keywords (bot, robot, auto, Jenkins, static, etc.) in user names and full names. We manually inspected the filtered accounts to make a final determination.

### 5.2.3 Toxicity classification scheme

Anti-social behaviors within FOSS communities, which have been investigated by multiple recent studies [155, 112, 140, 141, 63, 168, 77, 57], conclude that such behaviors in the FOSS context are different from the general domain such as social media. These studies have identified various categories of anti-social behaviors through qualitative analyses. To answer RQ1, we focused on aggregating various categories of anti-social behaviors to prepare our manual labeling scheme. We started with consolidating categories derived from ‘toxicity’ studies [141, 112, 155, 137]. During this process, we identified overlapping concepts based on definitions included in those papers and merged those into a single category. We noticed a conflict as ‘self-deprecation’ was marked as non-toxic by Sarker *et al.* [155], while Miller *et al.* [112] marks ‘self-directed pejorative’, a similar concept as toxic. We follow Sarker *et al.*’s (i.e., our first study at Chapter 3) definition, marking it as toxic only if it involved profanity since we use their dataset. Additionally, Ferreira *et al.*’s incivility lens, which encompasses a broader spectrum of anti-social behaviors, including toxicity, also includes frustration, impatience, irony, mocking, name-calling, threat, and vulgarity [63]. Based on their definitions, ‘name-calling, threat, mocking, and vulgarity’ overlap with existing categories identified by Miller *et al.* and Sarker *et al.* As such, these were considered toxic. Although ‘irony,’ ‘frustration,’ and ‘impatience’ are not a part of these toxicity schemes, they may fit existing categories, such as trolling, arrogance, and

insult, depending on context. It’s worth noting that existing SE studies have used different terminologies to study these subjective social constructs, and an agreed-upon scheme or definitions is currently missing. At the end of this step, we prepared our manual labeling scheme of 10 categories (Table 20) with a broader definition for each group based on current studies.

#### 5.2.4 Automated identification of toxic comments

We select ToxiCR [155] for automated classification since it is i) trained on large-scale training data, ii) developed as a reusable standalone tool in contrast to Jupiter notebook format used by some of the alternatives, iii) is publicly available on GitHub, iv) provides a well-defined interface to conduct a large-scale classification required by this study, v) trained on Code review data which is similar to pull requests that this study aims to analyze, and vi) reports the best performance according to its evaluation with 95.8% accuracy, 90.7% precision, 87.4% recall, and an 88.9% F1-score. ToxiCR provides the toxicity probability of a text from 0 to 1, and its authors recommend using a threshold  $\geq 0.5$  to consider a text as toxic.

*Evaluation of ToxiCR:* Prior research on SE domain-specific NLP tools [122, 121] recommends independent assessments before application on new settings. Therefore, we conducted an empirical evaluation to assess ToxiCR’s reliability on our dataset. To achieve this goal, we randomly selected 600 PR comments marked as toxic by ToxiCR. This sample adequately provides results within a 2.6% margin of error and 95% confidence interval [190]. We also use these same samples for our qualitative toxicity analysis in Section 5.3.1.

*Precision:* Two raters independently labeled those 600 samples as toxic or non-toxic and resolved the conflicts after a discussion. To mitigate the bias of the labeling process, two

labelers follow the toxicity rubric from [155]. The agreement between the two labelers is 95.8%, and Cohen’s kappa [41] value is  $\kappa = 0.80$ , which is ‘substantial’. After conflict resolution, 532 comments were labeled toxic, suggesting 88.8% precision. This result is within the margin of error (i.e., 2.6%) of ToxiCR’s claimed precision (90.7%) [155].

*Recall:* Evaluation of recall is also essential to ensure that ToxiCR does not miss many positive instances. On this goal, we focused on finding existing labeled toxicity datasets curated from GitHub issue requests. We did not use Raman *et al.*’s dataset since it has only 106 toxic instances [141]. We chose Ferreira *et al.*’s dataset of locked GitHub issues, which includes 896 uncivil sentences out of 1,364 [62]. According to Ferreira *et al.*, incivility is a superset of toxicity [63]. While all toxic comments are uncivil, some uncivil comments (e.g., irony and impatience) may not fit the toxicity lens. To encounter this challenge, two authors relabeled the 896 uncivil comments based on Sarker *et al.*’s toxicity labeling rubric. The raters achieved an inter-rater agreement of  $\kappa = 0.76$  and resolved the conflicts through a mutual discussion. On this dataset, ToxiCR achieved 87% recall, which is also within the sampling margin of error of ToxiCR’s reported recall (i.e., 87.4%).

*Automated classification* We are satisfied with ToxiCR’s performance in our context, so we classified all the PR and issue comments, totaling 101.5 million, using the best-performing configuration reported by its authors. ToxiCR found approximately 756K toxic comments (0.74%) from our dataset.

### 5.2.5 Manual categorization of toxic comments

Using the 10 class classification scheme described in Section 5.2.3, two of the authors independently placed the 532 toxic comments identified during ToxiCR’s evaluation (Section 5.2.4) into one or more groups. We also included the ‘Others’ category to label toxic

comments that do not fit the existing ten categories. We measured the inter-rater reliability of this multiclass labeling using Krippendorff’s alpha, which was 0.35, indicating a ‘Fair’ agreement. We noticed higher ratios of disagreements since, theoretically, the number of possible labeling for a single instance is  $2^{11}$ . Conflicting labels were resolved through mutual discussions. After conflict resolution, the raters reviewed the 34 instances from the ‘Others’ group to identify missing categories. The new category identified is ‘Object-Directed Toxicity,’ which includes anger, frustration, or profanity directed toward software, products, or artifacts. For example, “also the mask sprite is beyond horrid, I might have something that could do better.” represents this form. With this category, they went through the labeled instances again to identify other cases that may also fall under this category since a text can fall under multiple categories. They identified 49 instances belonging to this new category in total.

#### 5.2.6 Attribute selection

Table 16, Table 17, and Table 18 list attributes selected to answer RQ2, RQ3, RQ4 accordingly those are introduced in Section 5.1. In addition to each attribute’s definition, Table 16, table 17, and table 18 list why an attribute may be associated with toxicity. We selected this set of attributes since prior studies on code reviews and anti-social behaviors suggest their likelihood of association with toxicity contexts.

*RQ2: Project* We select eleven project characteristics attributes based on prior studies on toxicity and incivility [141, 112, 62, 63]. These 11 attributes in Table 16 characterize a project’s activity, popularity, domain, governance, and age.

*RQ3: PR Context* We select nine contextual attributes in Table 17 based on prior studies [112, 175, 141, 171, 57, 140]. These attributes characterize the type of change, out-

Table 16: The list of attributes selected to investigate their association with project characteristics (RQ2).

Variable Name	Definition	Rationale
PR/month	Average number of PRs per month.	Indicates the volume of development activity. Active projects may have a higher probability of toxic interactions [112].
issues/month	Average number of issues per month.	A higher number of bugs indicates the lack of quality, which may cause frustration among users and developers [112, 141].
commits/month	Average number of commits per month.	Commit is another indication of the volume of development activity. High activity may cause burnouts [141], and lack may cause frustration among users [112].
release/month	Average number of releases per month.	Frequent releases may satisfy the customers to decrease toxicity, and vice versa [45].
issue resolution rate	Percentage of issues resolved.	Users may become frustrated due to issues affecting them not being resolved [112].
isCorporate	Whether the project is sponsored by a corporation.	Corporate projects may have less toxicity than non-corporate ones due to the consequences of HR policy violations [141].
project age	Number of months since a project's creation.	Older projects showed more toxicity [141].
member count	Number of users with write access.	Toxicity increases with community size due to diverse views and higher potential conflicts [14].
isGame	Whether the project is gaming or not.	Prior studies have found prevalence of toxicity among gaming communities [112, 16, 125, 17].
stars	Number of stars on GitHub project.	Popularity shows users' interests. Scrutiny and expectations increase with popularity and therefore stress on developers [141].
forks	Number of forks on GitHub project.	Fork is another measure of project popularity [196].



come, complexity, required review /resolution efforts, completion time, and number of identified issues in a PR.

*RQ4: Participant* We select six participant attributes in Table 18 based on prior studies [112, 168, 42, 115, 140, 62]. These attributes represent a participant’s GitHub tenure, project experience, gender, and communication history. We compute each attribute for both the author and the target of a comment.

### 5.2.7 Attribute calculation

To investigate RQ3 and RQ4, it is necessary to compute attributes at pull request (PR) and comment levels, respectively. Given that our dataset comprises 16 M PRs and 101.5 M comments, calculating the PR and comment-level attributes listed in Table 16, Table 17, and Table 18 for the entire dataset would be exceedingly time-consuming and resource-intensive. Therefore, we reduced the sample size for RQ3 and RQ4 by randomly selecting 385 projects from the three project groups (i.e., ‘PRF-L’, ‘PRF-M’, and ‘PRF-H’). We choose this sample size to satisfy a 5% error margin and 95% confidence interval [190]. This sample of 1,155 projects includes 6.3 M PRs, 30 M comments, and 416 K users. We wrote Python scripts and MySQL queries to compute the 26 attributes listed in Table 16, Table 17, and Table 18 based on their definitions. While most attributes are straightforward to calculate, five require additional heuristics, as defined in the following.

**Gender:** We adopted a similar protocol to Sultana *et al.* [171] to automatically predict users’ genders. In this process, we have used genderComputer [181] and WikiGendersort [19] tools to resolve the gender from a user’s name, preferred pronoun, and location if available. We have also downloaded a user’s GitHub avatar and applied an automated human face detection model [72]. Further, we used a pre-trained photo to

Table 17: The list of attributes selected to investigate their association with Pull request context (RQ3).

Variable Name	Definition	Rationale
commit count	Number of commits in a PR.	Large number of commits increases review effort [175]. Increased efforts may frustrate reviewers, cause delays, and, therefore, also frustrate the author.
number of changed files	The number of files changes in each PR.	A higher number of file changes requires a longer review time [95] and comprehension difficulty.
code churn (log)	The total number of rewritten or deleted code.	There is a higher probability of defects in the code due to a high number of changed lines [118, 119], and it may be associated with a higher number of toxic comments.
isAccepted	Whether the code review is accepted or rejected.	Developers used more toxic comments in rejected codes/patches [63].
isBugFix	Whether the code review is for fixing a bug or not.	Issue discussions may instigate toxicity when the resolution is not liked by affected parties [62, 112].
change entropy (log)	A measure of change complexity, which estimates how much dispersed a changeset is among multiple files [175].	Complexity of code change affects review time and participation [175]. Moreover, unnecessary complexity may be a sign of a poor quality change, which may receive harsh critique [140].
review interval	Time difference from the start of the code review to the end.	Delayed code reviews are more likely to cause frustration for developers [57, 177].
number of iterations (num iter)	Total number of iterations (i.e., number of times changes requested) in a PR.	Higher number of iterations frustrates both developers and reviewers due to additional time [177]. Higher iteration also indicates the lack of common understandings [56] and potential disagreements [115].
review comments	The total number of review comments from reviewers in a PR.	Each review comment may indicate a change suggestion. A higher number of review comments indicate significant concerns from the reviewers over its quality, which often causes toxicity [140].

Table 18: The list of attributes selected to investigate their association with participants' characteristics (RQ4).

Variable Name	Definition	Rationale
isWoman	Whether the person is a woman	Prior studies have found women and marginalized minorities as frequent victims of toxicity [141, 77].
isMember	Whether the person is a project member or not.	Project members are the authors of many toxic comments in replying to the outside members' query [42, 112].
isNewComer	Whether the person is a newcomer to the current project.	Newcomers may get frustrated due to delays [166] and unfavorable decisions [63].
GitHub tenure	Age of GitHub account, in terms of the number of months, at the time of an event.	Miller <i>et al.</i> reported toxic comments from accounts with no prior activity on GitHub [112].
project tenure	Tenure with the current project, in terms of the number of months.	Although long-term members of a project are more committed to maintaining a professional environment in a community, Miller <i>et al.</i> found toxic comments from them [112]. Moreover, they may be targets if their decisions are not liked by issue reporters [62].
toxicity/month	The total number of toxic comments a user posts per month.	Miller <i>et al.</i> found many repeat offenders, as many FOSS developers have toxic communication styles [112, 11].

gender-resolution model [58] to predict the user’s gender. Conflicts between the two approaches were resolved by manually investigating users’ profiles. Finally, we successfully resolved 75.4% of the total users (92% with full names). We only include gender-resolved users for RQ4.

**Project Member:** Following the recommendation of Gousios *et al.* [71], we consider a user a project member if that user has write access (i.e., merged at least one PR or created an intra-branch PR) to the repository.

**GitHub Tenure and Project Tenure:** We compute a user’s GitHub tenure at an event as the months between their account creation and the event’s timestamp. Similarly, we calculate a developer’s project tenure during each project interaction (e.g., commit, pull request, or comment).

**Newcomer:** Following the definitions of prior studies [166, 167], we consider a user as a newcomer to a FOSS project until they have got their first PR accepted to this project.

### 5.2.8 Regression Model

We train four multivariate inferential regression models to analyze associations between the toxicity and the 26 attributes listed in Table 16, Table 17, and Table 18. The following subsections detail the three regression models to answer RQ2, RQ3, and RQ4.

**5.2.8.1 Multinomial Logistic Regression for RQ2** We found training a regression model for RQ2 challenging since computing various project characteristics variables at the creation timestamp of a comment requires the entire event log for a project (e.g., when a new star was added), which is resource-intensive to mine due to the enormous size of our dataset. While Google’s BigQuery hosts a dataset of GitHub events, it would be expen-

Table 19: Model fit measured using Psuedo  $R^2$  and model significance evaluated using the log-likelihood ( $\chi^2$ ) test for the bootstrapped logistic regression models. A 95% confidence interval is also reported for each measure inside the brackets.

Project Category	Research Question	Measure	95% Confidence Interval
PRF-L	RQ3	Pseudo $R^2$	0.16 [0.15, 0.17]
		$\chi^2$ (lrtest)	$1.7x10^3$ *** [ $1.5x10^3$ , $1.8x10^3$ ]
	RQ4 (author)	Pseudo $R^2$	0.01 [0.01, 0.01]
		$\chi^2$ (lrtest)	$1.4x10^2$ *** [ $1.2x10^2$ , $1.6x10^2$ ]
	RQ4 (target)	Pseudo $R^2$	0.01 [0.01, 0.01]
		$\chi^2$ (lrtest)	$1.1x10^2$ *** [ $0.9x10^2$ , $1.2x10^2$ ]
PRF-M	RQ3	Pseudo $R^2$	0.19 [0.19, 0.20]
		$\chi^2$ (lrtest)	$1.2x10^4$ *** [ $1.2x10^4$ , $1.3x10^4$ ]
	RQ4 (author)	Pseudo $R^2$	0.02 [ 0.02, 0.02]
		$\chi^2$ (lrtest)	$1.1x10^3$ *** [ $1.1x10^3$ , $1.2x10^3$ ]
	RQ4 (target)	Pseudo $R^2$	0.01 [ 0.01, 0.01]
		$\chi^2$ (lrtest)	$4.7x10^2$ *** [ $4.1x10^2$ , $5.2x10^2$ ]
PRF-H	RQ3	Pseudo $R^2$	0.18 [0.18, 0.19]
		$\chi^2$ (lrtest)	$1.0x10^5$ *** [ $1.0x10^5$ , $1.1x10^5$ ]
	RQ4 (author)	Pseudo $R^2$	0.09 [0.09, 0.09]
		$\chi^2$ (lrtest)	$7.8x10^4$ *** [ $7.8x10^4$ , $7.9x10^4$ ]
	RQ4 (target)	Pseudo $R^2$	0.11 [0.11, 0.11]
		$\chi^2$ (lrtest)	$7.3x10^4$ *** [ $7.2x10^4$ , $7.4x10^4$ ]

\*\*\*, \*\*, and \* represent statistical significance at  $p < 0.001$ ,  $p < 0.01$ , and  $p < 0.05$  respectively.

sive to query this service. Therefore, we used aggregated attributes over the lifetime of a project. We calculated toxicity per hundred comments (*percent\_toxic*) for each project over its lifetime and used it as the dependent variable for RQ2. However, if the dependent variable is a ratio, a model can identify spurious associations [98]. Following the recommendation of Long and Freese [107], we transform the *percent\_toxic* variable into a three-level categorical variable named *toxicity\_group*. We selected the number of categories and thresholds for this grouping based on the inflection points<sup>9</sup> in the cumulative distribution curve. The ‘Low toxic’ group includes 324 projects with  $\text{percent\_toxic} < 0.02\%$ . The 2,082 projects from the ‘Medium toxic’ group have  $0.02 \leq \text{percent\_toxic} < 1\%$ . The remaining 421 projects belong to the ‘High toxic’ group with  $\text{percent\_toxic} \geq 1\%$ . Since *toxicity\_group* has three levels, we use a Multinomial Logistic Regression (MLR) model, where *toxicity\_group* is the dependent variable and 11 project characteristics attributes are independent.

**5.2.8.2 Bootstrapped Logistic Regression for RQ3 and RQ4** For RQ3, the dependent variable is *HasToxicComment*, set to 1 if a PR has at least one toxic comment and 0 otherwise. For RQ4, we use participant attributes computed at the comment level as independents. We use *isToxic* as the dependent, 1 if the comment is toxic, and 0 otherwise. We train two models for RQ4, one with the author’s attributes as the independents and the other with the target’s attributes. Since the dependents are binary for RQ3 and RQ4, we use Logistic Regression models. As the dataset of RQ3 and RQ4 consist of a rare binary outcome variable (i.e., *HasToxicComment*, *isToxic*), we use a bootstrapped regression

---

<sup>9</sup>points of a curve at which a change in the direction of curvature occurs

modeling technique [189]. In this technique, we choose a desired ratio between the minority and the majority. We randomly downsample the majority until the desired ratio is reached. With this sample, we fit a logistic regression model, measure its fit, and compute regression coefficients. This process is repeated 100 times, and we record the results of each iteration in a dataset. We report median and 95% confidence interval for model fit and regression coefficients. We also explored various ratios between the minority and the majority and found that the model's goodness of fit reduces with the increment of the majority's share. We chose a ratio of 1:10 since increasing the majority's share beyond that sometimes produced unreliable models according to the log-likelihood test.

**5.2.8.3 Correlation and redundancy analysis** For an inferential regression model, multicollinearity, which occurs when two independent variables are highly correlated to each other, is a threat to validity. We used the variable clustering approach suggested by Sarle [156] to identify multicollinearity. With this approach, we create a hierarchical cluster representation of independents using Spearman's rank-order correlation test [163]. As recommended by Hinkle *et al.* [79], we set the cutoff value at  $|\rho| = 0.7$  for the correlation coefficient. Only the explanatory variable with the strongest correlation with the dependent was chosen from a cluster of variables with  $|\rho| \geq 0.7$ .

**5.2.8.4 Model analysis** We also use the Log-likelihood test (*lrtest*) to assess whether a model significantly differs (Chi-Square,  $p < 0.05$ ) from a null model and can be reliably used for inference. We evaluate each model's goodness-of-fit using Veall-Zimmermann Psuedo- $R^2$ , with a higher  $R^2$  value indicating a better fit. We use the Odds ratio (OR)

to quantify the association between the dependent and independents and estimate effect size. For a binary independent (e.g., *isGame*), OR indicates the odds of an outcome if the independent variable changes from 0 to 1, while all other factors remain constant. For a continuous variable (e.g., project age), OR indicates an increase or decrease in odds for the dependent with one unit change in the factor. In simple terms,  $OR > 1$  indicates a positive association and vice versa. We use the p-value of the regression coefficient to assess the significance of an association with  $p < 0.05$ , indicating a statistical significance. Table 19 shows goodness-of-fit measured with Pseudo- $R^2$  and results of Log-likelihood tests for the three regression models trained for RQ3 and RQ4. Since we bootstrapped each model 100 times, we report median and 95% confidence intervals for each model. The results of *lrtest* indicate that all models are significantly better than a null model and are reliable to infer insights to answer our RQs.

### 5.3 Results

The following subsections detail the results of the four research questions.

#### 5.3.1 RQ1: Nature of toxicity in PR Review Comments

Table 20 shows the distributions of 11 forms of toxicities among our manually labeled dataset of 532 PR review comments. Similar to Miller *et al.*'s investigation, we found profanity (i.e., severe language, swearing, cursing) as the most common form, with more than half of the sample ( $\approx 58\%$ ) belonging to it. Words such as shit, fuck, ass, crap, suck, and damn are the most common forms of profanities in our sample. We found trolling as the second most common form with 18%, followed by insult, self-deprecation, and object-directed toxicity. Identity attacks, insults, and threats, which are regarded as severe



Table 20: The most common forms of toxicities within our sample of manually labeled 532 PR comments

Type	Example	Count <sup>‡</sup>	Ratio
Profanity [153, 63, 112, 155]	“You know, at some point, github fucked me over. In Visual Studio, this was just fine, wtf....”	311	58.45%
Trolling [112, 63]	“@clusterfuck There’s a difference between being in cryogenics and not in cryogenics you big nerd”	96	18.04%
Insult [112, 155]	“Acknowledge that the vote wasn’t entirely singulo shitposters > ARE YOU SCHIZOPHRENIC?? Jesus dude why are you even here still”	92	17.3%
Self-deprecation [112]	“@ComicIronic Okay, I’ll fix it when I fix my shitty code, which will have to happen tomorrow”	67	12.6%
Object Directed Toxicity	“the PR has fallen into conflict hell, I’ll be closing this and re-opening some of its changes shornestly”	49	9.21%
Entitled [112]	“Again I didn’t break it Are you fucking stupid lol > merge your update into PR > buckling doesn’t work”	17	3.2%
Identity at-tack [57, 63, 155]	“Fuck those argentinians. Did you test it?”	17	3.2%
Threats [63, 112, 155]	“Done - I can always revoke your access if you mess things up ;”	12	2.25%
Obscenity	“dude you need to spend less time on programming and more time with women”	6	1.1%
Arrogance [112]	“Araneus is shit and generic as hell I think steely is an acceptable name”	5	< 1%
Flirtation [155]	“Frigging love you Niki. Seriously”	5	< 1%

<sup>‡</sup> -since a text can belong to multiple categories, the sum of the categories is greater than our sample size.

toxicities [73], were found among  $\approx 22\%$  of the samples. We also noticed over two-thirds of our samples  $\approx 72\%$  belonging to multiple forms. For example, “:laughing: Holy shit, you are fucking stupid. It is an extremely simple proc with a switch. Seriously, did you even look at the code? Next, you’ll tell me all switches are copypaste.” represents both profanity and insult. While toxicity on social media has higher occurrences of flirtation and obscenity [76, 73], we found  $\approx 2\%$  such cases in our sample.

**Key finding 1:** *Profanity seems to be the dominant form of toxicity in GitHub PRs. Severe toxicities, such as insults, identity attacks, and threats, represent  $\approx 22\%$  cases. Flirtation and obscenity were less common in our sample, unlike other online mediums.*

### 5.3.2 RQ2: Project characteristics

Two factors (i.e., forks and pulls per month) were dropped due to multicollinearity and were not included in our MLR. We estimated the fit of our MLR with Nagelkerke  $R^2 = 0.224$ . Our Log likelihood test results suggest that this model significantly differs ( $\chi^2 = 545.16, p < 0.001$ ) from a null model. In addition to modeling the probability of a specific result based on a group of independent variables, MLR also enables the assessment of the probability of transitioning to a different dependent category from the current one when a specific independent variable changes [15]. Hence, we set the ‘Low toxic’ projects as the reference group in MLR and compute the odds of a project moving to the ‘Medium toxic’ or ‘High toxic’ group if one of the independents changes by a unit. Table 21 shows the result of our MLR model, with OR values for each factor. Our results suggest that projects with corporate sponsorship (*isCorporate*) are significantly less likely to belong to the ‘Medium toxic’ or ‘High toxic’ groups than the ‘Low toxic’ group. HR rules, professional

Table 21: Results of our MLR model to identify associations of project characteristics with toxicity. We set the ‘Low toxic’ group as the reference to compute odds ratios. Hence,  $OR > 1$  indicates a higher likelihood of a project transitioning to the ‘Medium toxic’ or ‘High toxic’ group with a unit increment of that factor and vice versa.

Attribute	Medium toxic		High toxic	
	OR	$p$	OR	$p$
isCorporate	0.888	<b>0.000</b> ***	0.47	<b>0.000</b> ***
member count	0.999	0.821	1.0001	0.754
stars	1.001	<b>0.000</b> ***	1.001	<b>0.000</b> ***
issues/month	1.001	0.234	0.998	0.061
project age	1.004	<b>0.000</b> ***	1.009	<b>0.000</b> ***
commits/month	1.0001	0.316	1.0001	0.447
release/month	1.003	0.310	0.998	0.813
bug resolution	0.204	<b>0.000</b> ***	0.985	<b>0.000</b> ***
isGame	0.486	<b>0.000</b> ***	7.259	<b>0.000</b> ***

\*\*\*, \*\*, and \* represent statistical significance at  $p < 0.001$ ,  $p < 0.01$ , and  $p < 0.05$  respectively.

codes of conduct, and the potential for job loss may be the reasons. We also notice a significantly higher level of toxicity among the popular projects (i.e., stars). Popularity increases pressure on the contributors to deliver new features and maintain quality. However, a rapid development pace can cause stress, burnout, and toxicity. We also noticed that the prevalence of toxicity significantly increased with project age. Our manual investigation of sample projects suggests staleness (i.e., lack of response to issues or PRs) as a frequent reason. We found no significant association between toxicity and development activities (i.e., commits/month and releases/month) and project quality (i.e., issues/month). On the other hand, issue resolution rates (i.e., percentage of resolved issues) significantly reduce toxicity, as projects with higher rates are more likely to belong to the ‘Low toxic’ group than the ‘Medium toxic’ or ‘High toxic’ group. Supporting observations from prior studies [112], we also noticed the significantly higher prevalence of toxicity among gaming projects, as a gaming project is seven times more likely to belong to the ‘High toxic’ group than the ‘Low toxic’ or ‘Medium toxic’ group.

**Key finding 2:** *While popularity and staleness are positively associated with the prevalence of toxicity, issue resolution rate has the opposite association. While corporate-sponsored projects are 'low toxic,' gaming projects belong to the opposite spectrum.*

### 5.3.3 RQ3: Pull request context

One of the nine pull request context factors (i.e., the number of changed files) was dropped due to multicollinearity. Table 22 shows median odds ratios with 95% confidence intervals for the remaining eight factors based on our bootstrapped logistic regression models repeated over 100 times. All eight factors show significant associations for the PRF-H group (i.e.,  $\text{PR/month} > 32$ ). For this group, bug fix PRs, code churn, review interval, the number of review comments, the number of review iterations, and change entropy are positively associated with toxicity. On the other hand, the number of commits and acceptance decisions are negatively associated. Similarly, we noticed almost identical associations for the PRF-M group (i.e.,  $8 < \text{PR/month} < 32$ ), except *isBugFix* does not have a statistically significant association. For the PRF-L group (i.e.,  $\text{PR/month} < 8$ ), only three factors have statistically significant associations with toxicity, where the review interval and the number of review comments have positive ones. In contrast, *isAccepted* has a negative one. The directions of associations mostly follow our rationale included in Table 17 for these factors. However, contrary to our expectations, we noticed the number of commits in a PR negatively associated with toxicity among both PRF-M and PRF-H groups. We hypothesize that this negative association may be due to large code reviews (i.e., ones including several code commits) being less likely to have discussions [175], and the lack of discussions may be a reason.

Table 22: Associations between pull request contexts and toxicity. Values represent the median odds ratio for each factor with 95% confidence intervals inside brackets.

Attribute	PRF-L	PRF-M	PRF-H
isBugFix	1.01[0.99, 1.05]	1.02 [1.01, 1.03]	1.06*** [1.06, 1.07]
commit count	1 [1, 1.01]	0.99* [0.99, 0.99]	0.99*** [0.99, 1]
code churn (log)	1.11 [1.10, 1.12]	1.13*** [1.13, 1.14]	1.11*** [1.11, 1.11]
review interval	1.11*** [1.11, 1.12]	1.17*** [1.17, 1.17]	1.20*** [1.20, 1.20]
review comments	1.06*** [1.05, 1.07]	1.06*** [1.05, 1.06]	1.04*** [1.04, 1.04]
isAccepted	0.77*** [0.75, 0.80]	0.71*** [0.70, 0.73]	0.93*** [0.93, 0.94]
num iter	1.01 [0.99, 1.03]	1.01*** [1.01, 1.02]	1.01*** [1.01, 1.01]
change entropy (log)	1.57 [1.51, 1.67]	1.35*** [1.32, 1.37]	1.26*** [1.25, 1.27]

\*\*\*, \*\*, and \* represent statistical significance at  $p < 0.001$ ,  $p < 0.01$ , and  $p < 0.05$  respectively.

**Key finding 3:** Accepted PRs are less likely to have toxicity. On the contrary, code churn, review intervals, the number of review comments, change entropy, and the number of review iterations are positively associated with toxicity on GitHub.

#### 5.3.4 RQ4: Participants

We train two types of regression models, one to investigate the characteristics of persons authoring toxic comments and the other with the targets. Similar to the RQ3, we train bootstrapped logistic regression models repeated over 100 times. Table 23 shows the odds ratios for each factor with 95% confidence intervals for the three PRF-based project groups.

**Characteristics of authors of toxic comments:** Our results suggest significantly lower odds of women authoring toxic comments among PRF-L and PRF-H groups. Similarly, newcomers have lower authoring odds among PRF-M and PRF-H groups. Among all three groups, project members are significantly less likely to author toxic comments, and the

Table 23: Associations between characteristics of authors and targets and toxicity. Values represent the median odds ratio for each factor with 95% confidence intervals inside brackets.

Project Category	Attributes	RQ4: Author	RQ4: Target
PRF-L	isWoman	0.80** [ 0.71, 0.793]	0.48*** [ 0.46, 0.50]
	isNewComer	1.09 [ 1.05, 1.14]	0.91 [0.88, 0.95]
	isMember	0.89** [0.87, 0.92]	1.12* [1.09, 1.15]
	github tenure	0.99*** [ 0.99, 0.99]	1.01 [0.99, 1.01]
	project tenure	1.01** [1.01, 1.01]	0.99* [0.99, 0.99]
	toxicity/month	1.06*** [1.05, 1.07]	1.99*** [ 1.81, 2.19]
PRF-M	isWoman	1 [0.96, 1.01]	0.95 [ 0.93, 0.98]
	isNewComer	0.88*** [0.86, 0.89]	0.90***[0.89, 0.92]
	isMember	0.77*** [0.77, 0.88]	1.04 [1.02, 1.05]
	github tenure	0.99*** [0.99, 0.99]	1.01*** [1.01, 1.01]
	project tenure	1.01 [0.99, 1.01]	1.01*** [ 1.01, 1.01]
	toxicity/month	1.02*** [1.02, 1.02]	1.46*** [1.36, 1.52]
PRF-H	isWoman	0.90*** [0.86, 0.87]	0.86*** [ 0.85, 0.87]
	isNewComer	0.69*** [0.68, 0.69]	0.74*** [0.74, 0.75]
	isMember	0.64***[0.64, 0.64]	0.84*** [0.84, 0.84]
	github tenure	0.99*** [0.99, 0.99]	1.01 [1.01, 1.01]
	project tenure	1.01*** [1.01, 1.01]	1.01*** [1.01, 1.01]
	toxicity/month	1.01*** [1.01, 1.01]	1.26*** [1.25, 1.26]

\*\*\*, \*\*, and \* represent statistical significance at  $p < 0.001$ ,  $p < 0.01$ , and  $p < 0.05$  respectively.

likelihood of being an author significantly decreases with GitHub tenure. The likelihood of authoring toxic comments significantly increased with project tenure among PRF-L and PRF-H groups. Our results support Miller *et al.*'s observation that there are many repeat offenders [112] since the likelihood of authoring toxic comments significantly increases with the prior frequency of such occurrences.

**Characteristics of targets of toxic comments:** Contrary to our expectations, formed based on results [77, 141, 164], we did not find any significantly higher odds of women or newcomers being targets of toxicity on GitHub. We noticed the opposite among PRF-H and PRF-L. Similarly, newcomers have significantly lower odds of becoming targets among PRF-H and PRF-M. Being a project member significantly increases the odds of being a target among PRF-L and PRF-M but reduces among PRF-H. Project tenure increases the odds of being a target for PRF-M and PRF-H but reduces among PRF-L. The age of a GitHub account is positively associated with being a target only for PRF-M. Finally, these results suggest a 'quid pro quo,' i.e., prior frequent authoring of toxic comments significantly increases the odds of becoming a target.

**Key finding 4:** *Women and newcomers are less likely to be either authors or targets of toxic comments in GitHub PR comments. FOSS developers who have authored toxic comments frequently in the past are significantly more likely to repeat and significantly more likely to become toxicity targets.*

## 5.4 Discussion

The following subsections compare and contrast our findings against prior works and suggest recommendations.

### 5.4.1 Comparison with prior SE studies

We investigated 26 different attributes. Only a few of those were also subject to prior SE studies. Similar to Miller *et al.* [112], profanity is the prevalent toxicity in our randomly sampled dataset. They reported a high share (25%) of entitled issue comments, which are demands to project maintainers as if they had a contractual relationship or obligation [112]. However, we found only 3.2% such cases in our sample. Supporting their findings, our results indicate repeat offenders, toxicity increasing with project popularity, long-term project contributors being authors of toxicity, and gaming projects harboring more toxic cases [112]. They also reported toxic comments from new GitHub accounts [112]. Aligning with this finding, we noticed the likelihood of authoring toxic comments decreased with GitHub tenure. However, contrary to their findings, we notice a lower likelihood of project newcomers authoring toxic comments. Our results also concur with Raman *et al.*'s [141] since we also found a lower likelihood of toxicity among corporate-sponsored projects. During their manual investigation of the Linux kernel, Ferreira *et al.* found uncivil comments during reviews of rejected codes [63]. Aligning with their findings, we noticed lower odds of toxicity among accepted PRs. Ferreira *et al.* also reported incivility among project maintainers' feedback [63]. However, contrasting their findings, we noticed a lower likelihood of toxicity from project members. Egelman *et al.* reported a higher likelihood of pushback on large code changes [57]. Our result aligns with this finding, as we found that the odds of toxicity increase with code churn. Rahman *et al.* reported incivility due to poor quality code changes [140]. While we did not measure code quality directly, we may use the number of review comments as an indication of code



quality since each review comment indicates an issue identified by a reviewer. Aligning with their findings, our results indicate higher odds of toxicity with the number of review comments.

#### 5.4.2 Recommendations

While we cannot claim causal relationships for the associations identified in this study due to our study design, some of the following recommendations apply only if such relationships exist.

**I. Project Maintainers:** Our results suggest that delays in fixing bugs or answering user queries may create unhappy users and toxic comments targeted toward maintainers. As a project’s popularity grows, maintainers should focus on improving bug resolution since our results also show that a higher bug resolution rate is negatively associated with toxicity. Even if an issue is delayed, maintainers should respond politely and suggest workarounds, if possible, to avoid toxic interactions. We also find project tenure positively associated with toxicity. Therefore, building a positive culture needs to start with project maintainers since they are likelier to be project members with the longest tenures [183]. Supporting prior studies [112, 16, 125, 17], we also found a proliferation of toxicity among gaming projects. Therefore, we recommend that maintainers of gaming projects adopt a Code of Conduct and its enforcement mechanism to build a diverse community.

**II. Developers:** We recommend developers avoid creating pull request contexts that are positively associated with toxicity. For example, our results indicate that delayed pull requests are associated with toxicity. Therefore, reviewers should provide on-time reviews to avoid frustrating authors. Similarly, large code changes are not only bug-prone [24] and

difficult to review [175] but also likely to encounter toxicity. Therefore, when possible, creating pull requests with smaller changes is recommended. Pull requests with a large number of issues indicate poor quality codes and are more likely to receive harsh critiques. Therefore, developers should not create pull requests with changes that do not yet meet the quality standards for a project. A higher number of review iterations also frustrates authors and may cause toxicity. Hence, if possible, reviewers should request all required changes within a single cycle to avoid back and forth. Complex changes are hard to review and are more likely to receive toxicity. Hence, authors should annotate such changes and include helpful descriptions to avoid confusion [56] as well as toxicity. Even when frustrated or angry, developers should not use toxic languages since developers who use such languages are more likely to become victims. Finally, while contrary to prior evidence, we find that women and newcomers are less likely to be targets of toxicity, we still recommend long-term contributors avoid such language if such persons are present in a discussion since toxicity not only dissuades newcomers from becoming a part of the communities [164] but also disproportionately hurts minorities [77].

**III. Prospective joiners:** If a newcomer wants to avoid negative experiences associated with toxic cultures, we recommend they start with a corporate-sponsored FOSS project that matches their expertise and interests. We also recommend such contributors to avoiding gaming or old stale projects.

**IV. Researchers:** We found variations among terminologies used for almost identical concepts among SE studies investing in anti-social behaviors. We also noticed conflicting opinions about whether a particular category should be considered anti-social. Since ex-

isting schemes are primarily based on the decisions of the respective researchers, they may not reflect the broader FOSS community. Therefore, existing identification tools based on these schemes may not align with FOSS developers’ needs and would fail to achieve broader adoption. Moreover, recent research suggests whether a text should be considered toxic depends on various demographic characteristics [73]. Hence, understanding the opinions of the broader FOSS community and how their demographics influence perspectives of toxicity is essential to developing a custom mitigation strategy. We consider that as a pressing research need in this direction.

## **5.5 Threats to Validity**

The threats to validity have been described below.

### **5.5.1 Internal Validity**

Our selection of 2,828 GitHub projects based on our sampling method threatens internal validity. GitHub hosts over 284 million projects, and mining all of them is infeasible. We defined six filtering criteria to reduce this sample space to 89k without excluding projects with significant communication and collaboration. A lower threshold for the number of contributors or stars would increase the number of projects in this sample and may potentially change our results. We applied a stratified sampling strategy to categorize the projects according to PR activity to encounter this threat. Therefore, threats due to threshold selection are more likely to influence only the PRF(L) group since most of the projects with a lower number of contributors or stars would fall under this group. However, there is no evidence that changing these thresholds would significantly alter the results, even for the PRF(L). Our selection of the list of attributes represents another threat to internal

validity. Prior studies have found various factors such as politics or ideology triggering toxicity [112]. However, we could not investigate those factors due to the unavailability of automated mechanisms to identify such scenarios at a large scale. This study only investigates automatically measurable factors that may be associated with toxicity.

### 5.5.2 Construct Validity

Our *(first)* threat in this category is due to using ToxiCR [155] to identify toxic comments automatically. Our validation of ToxiCR found 88.88% precision, which is within the sampling error margin reported by ToxiCR’s authors. ToxiCR has false positives in approximately one out of 10 cases. Similarly, ToxiCR has a false negative rate of between 10-14%. Hence, these false positives and negatives may have influenced our results if ToxiCR is biased for/against any particular attributes (e.g., review interval or woman) included in our study. However, we do not have any evidence of such biases. *(Second)*, our manual labeling scheme to identify the nature of toxicities to answer RQ1 is a threat. Although multiple SE studies have studied antisocial behaviors, no agreed-upon scheme exists. Moreover, researchers from NLP and SE domains have used different terminologies to characterize similarly subjective concepts. To mitigate this threat, we have analyzed existing studies [155, 112, 63, 57, 62] and aggregated their categories to build our scheme. We acknowledge the subjectivity bias, where another set of researchers disagree with our scheme and definitions. *(Third)*, our manual labeling process may have subjectivity biases. We prepared a scheme with category definitions and examples to mitigate this threat. The labelers had a discussion session before starting to build an agreed-upon understanding. We also measured inter-rater reliability to assess your labeling process. *(Finally)*, automated gender resolution is another threat. We followed a procedure as the ones in mul-

multiple recent empirical studies [171, 147, 26]. We used multiple gender resolution tools, considered users' location and profile photos, and searched LinkedIn to improve resolution accuracy. This resolution process may be subject to misclassification. We did not attempt to identify non-binary genders since we are unaware of any automated resolution of those without users' inputs.

### 5.5.3 External Validity

The nature of toxicities in a FOSS project may depend on factors such as project domain, governance, the number of contributors, and project age. We used a stratified random sampling strategy to select 2,828 projects representing diverse demographics, including the top FOSS projects on GitHub, such as Kubernetes, Odoo, PyTorch, Rust, Ansible, pandas, rails, Django, numpy, angular, flutter, CPython, and node.js. Yet, our sample and its results may not adequately represent the entire FOSS spectrum.

### 5.5.4 Conclusion Validity

We used recommended practices and well-known libraries such as *rms*, *stats*, and *nnet* to build our regression models. We assess the reliability of our models using goodness-of-fit metrics and log-likelihood tests. Hence, we do not anticipate any threats from the results obtained from our models. Although our models account for various confounding variables, these models identified associations between dependents and predictors, and no causal relationships can be implied.

## 5.6 Conclusion

We conducted a large-scale mixed-method empirical study of 2,828 GitHub-based FOSS projects to understand the nature of toxicities on GitHub and how various measurable

characteristics of a project, a pull request's context, and participants associate with their prevalence. We found profanity as the dominant form of toxicity on GitHub, followed by trolling, and insults. While a project's popularity is positively associated with the prevalence of toxicity, its issue resolution rate has the opposite association. Corporate-sponsored projects are less toxic, but gaming projects are seven times more likely than non-gaming ones to have a high volume of toxicities. FOSS developers who have authored toxic comments in the past are significantly more likely to repeat them and become toxicity targets. Based on the results of this study, and our experience conducting it, we provide recommendations to FOSS contributors and researchers.

## CHAPTER 6 OVERALL CONCLUSION AND FUTURE DIRECTION

This doctoral dissertation focused on providing a way for the FOSS community to make healthy communication by *identification and mitigation of toxicity during their interactions*. To achieve this broader goal, we have conducted three studies. In our first study, we developed a state-of-the-art binary toxicity detector and built an explainable toxicity detector in the second. In our final study, we have done a large-scale empirical analysis to understand the measurable outcomes of toxicity in GitHub projects.

### 6.1 Key Outcomes

We have provided the details of the key outcomes of our three studies in the following subsections:

#### 6.1.1 A customized toxicity detector for SE domain

Perceiving the toxicity in the SE domain is different from real-life communication mediums. For that reason, we first developed a detailed rubric for better understanding toxicity in the SE domain, and it may help future researchers to label the toxic text from the SE perspective. We constructed a dataset of 19,651 code review comments and made them publicly available for further use. Using that dataset, we have developed a state-of-the-art toxicity detector *ToxiCR* for SE domain toxicity detection. After an empirical investigation with different vectorizers, models, and preprocessing, the best combination of *ToxiCR* achieved an accuracy of 95.8% and an F-score of 88.9%. This tool is user-friendly and has been getting public attention in recent days.

### 6.1.2 An explainable toxicity detector for Code Review comments

Though a binary toxicity classifier may help the FOSS community remove a particular paragraph for being toxic, it may not help them understand why a person perceives the text as toxic. On this goal, we have developed the first explainable toxicity detector *ToxiSpanSE* for the SE domain in our second study. To achieve this goal, we manually labeled 3,757 CR comments with span-level toxicity. Using five different transformer models, we evaluated the performance of *ToxiSpanSE* with a total of 19,651 CR comments dataset. The best combination achieved 88% F-score in our dataset. This tool can be used for finer-grained toxicity analysis in the SE domain.

### 6.1.3 A large-scale empirical investigation of toxicity on GitHub Pull Requests (PR)

In our third study, we conducted a large-scale empirical study on toxicity for GitHub Pull Request (PR) comments. We set four research questions to understand the nature of toxicity, association of toxicity and project characteristics, context, and participants. During the qualitative analysis of RQ1 (i.e., nature of toxicity), we found that profanity, insult, trolling, self-deprecation, and frustration are the most common toxic categories in GitHub. The RQ2, RQ3, and RQ4 findings are based on quantitative analysis. Regression models suggested that corporate projects are likely less toxic, while gaming projects are the opposite. We also found that poor code review has a positive correlation with toxicity. Though experienced developers are positively correlated with toxicity, women and newcomers are the opposite.

This dissertation developed two customized SE-specific tools to automatically identify toxicity in the FOSS domain. Moreover, it provided key insights into how toxicity is asso-



ciated with measurable outcomes on FOSS projects. Finally, it provided some actionable recommendations for project maintainers to combat toxicity in the FOSS community.

## 6.2 Directions for Future Work

This dissertation provided actionable insights and improvement opportunities to mitigate toxicity in the FOSS community. Some of the potential future directions of this dissertation are discussed in the following:

### 6.2.1 Notion of Toxicity according to diverse demographic factors

Unlike other text classification problems (i.g., spam, online abuse), toxicity is a highly subjective [99]. For example, females got more negative comments in the FOSS community than men during code reviews [126]. Moreover, newcomers may get more demotivated than experienced ones due to getting toxic comments from their peers. In the first study of this dissertation, we introduced a rubric of toxicity for the SE domain [153]. However, toxicity may differ based on many factors, such as culture, ethnicity, country of origin, language, and relationship between the participants. To understand how people from different demographics perceive toxicity online, Kumar *et al.* surveyed 17,280 participants across the United States [99]. Surprisingly, they found diverse labeling from the raters, and no single demographic factor could define the construction of toxicity. However, there are two limitations of that study: i) they conducted their survey only inside the United States, which does not cover the whole region of the world, and ii) they provided survey questions that were not related to the SE conversations. No such study has yet been done in the SE domain to understand toxicity across demographic factors and experience. A further study may help the developers i) determine whether the toxicity phenomenon

differs in different demographics and ii) develop finer-grained tools to detect toxicity.

### **6.2.2 Detoxification among the developer's textual communication**

Our developed tools, ToxiCR and ToxiSpanSE, help the FOSS community identify toxic comments from the SE domain. One potential future direction would be detoxifying toxic text from the FOSS community. Large Language Models can be used to detoxify text in the SE domain. The detoxification model may help the project members use detoxified comments during their conversations.

### **6.2.3 Promoting politeness among developers' interaction**

Fostering a politeness culture is essential to enhancing the FOSS community's healthy environment. A promising direction for future efforts may involve conducting human-centric studies to devise strategies to cultivate politeness among developers. The outcome of this study would promote diversity, equity, and inclusion among FOSS developers and make a healthy SE community.

## CHAPTER 7 APPENDIX

Up to this point, I have authored the following academic publications:

### 7.1 Journal Articles

- Published

1. **Jaydeb Sarker**, Asif Kamal Turzo, Ming Dong, and Amiangshu Bosu. Automated Identification of Toxic Code Reviews Using ToxiCR. ACM Transactions on Software Engineering and Methodology (TOSEM), 32(5), July 2023.

- In Review

1. Sayma Sultana, **Jaydeb Sarker**, Farjzana Israt, Paul Rajshakhar, and Amiangshu Bosu. Automated Identification of Sexual Orientation and Gender Identity Discriminatory Texts from Issue Comments. In Review at the ACM Transactions on Software Engineering and Methodology, Submission Date: 14 November, 2023 [170].

### 7.2 Refereed Conference Papers

- Published

1. **Jaydeb Sarker**, Sayma Sultana, Steven R Wilson, and Amiangshu Bosu. ToxiSpanSE: An Explainable Toxicity Detection in Code Review Comments. In the 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–12. IEEE, 2023.

2. Asif Kamal Turzo, Fahim Faysal, Ovi Poddar, **Jaydeb Sarker**, Anindya Iqbal, and Amiangshu Bosu. Towards Automated Classification of Code Review Feedback to Support Analytics. In the 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–12. IEEE, 2023 [178].
3. **Jaydeb Sarker**, Asif Kamal Turzo, and Amiangshu Bosu. A Benchmark Study of the Contemporary Toxicity Detectors on Software Engineering Interactions. In 2020 27th Asia Pacific Software Engineering Conference (APSEC), pages 218–227. IEEE, 2020.

- In Review

1. **Jaydeb Sarker**, Asif Kamal Turzo, and Amiangshu Bosu. The Landscape of Toxicity: An Empirical Investigation of Antisocial Behaviors on GitHub. TBD 2024.

### 7.3 Short Papers

- Published

1. **Jaydeb Sarker**. ‘Who built this crap?’ Developing a Software Engineering Domain Specific Toxicity Detector. Student Research Competition on the International Conference on Automated Software Engineering (ASE), Rochester, MI, USA, pages 1–3, 2022.
2. **Jaydeb Sarker**. Identification and Mitigation of Toxic Communications Among Open Source Software Developers. Doctoral Symposium on the International

Conference on Automated Software Engineering (ASE), Rochester, MI, USA, pages 1–5, 2022 [149].

3. Sayma Sultana, **Jaydeb Sarker**, and Amiangshu Bosu. A Rubric to Identify Misogynistic and Sexist Texts from Software Developer Communications. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–6, 2021.

## REFERENCES

- [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, Nov. 2016. USENIX Association.
- [2] A. Adhikari, A. Ram, R. Tang, and J. Lin. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*, 2019.
- [3] S. Adinolf and S. Turkay. Toxic behaviors in esports games: player perceptions and coping strategies. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, pages 365–372, 2018.
- [4] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi. Senticr: a customized sentiment analysis tool for code review interactions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 106–111. IEEE, 2017.
- [5] C. AI. What if technology could help improve conversations online? <https://www.perspectiveapi.com/>, lastchecked = May 25, 2020.
- [6] C. AI. Annotation instructions for toxicity with sub-attributes. [https://github.com/conversationai/conversationai.github.io/blob/master/crowdsourcing\\_annotation\\_schemes/toxicity\\_with\\_subattributes.md](https://github.com/conversationai/conversationai.github.io/blob/master/crowdsourcing_annotation_schemes/toxicity_with_subattributes.md), 2018.
- [7] K. Albusays, P. Bjorn, L. Dabbish, D. Ford, E. Murphy-Hill, A. Serebrenik, and M.-A. Storey. The diversity crisis in software development. *IEEE Software*, 38(2):19–25,

2021.

- [8] B. Alshemali and J. Kalita. Improving the reliability of deep neural networks in nlp: A review. *Knowledge-Based Systems*, 191:105210, 2020.
- [9] A. A. Anderson, S. K. Yeo, D. Brossard, D. A. Scheufele, and M. A. Xenos. Toxic talk: How online incivility can undermine perceptions of media. *International Journal of Public Opinion Research*, 30(1):156–168, 2018.
- [10] L. Aroyo, L. Dixon, N. Thain, O. Redfield, and R. Rosen. Crowdsourcing subjective tasks: the case study of understanding toxicity in online discussions. In *Companion proceedings of the 2019 world wide web conference*, pages 1100–1105, 2019.
- [11] A. Author. Leaving toxic open source communities. <https://modelviewculture.com/pieces/leaving-toxic-open-source-communities>, 2014.
- [12] B. Barbarestani, I. Maks, and P. Vossen. Annotating targets of toxic language at the span level. In *Proceedings of the Third Workshop on Threat, Aggression and Cyberbullying (TRAC 2022)*, pages 43–51, 2022.
- [13] H. Barnes. Toxicity in open source. <https://boxofcables.dev/toxicity-in-linux-and-open-source/>, 2020.
- [14] M. R. Basirati, M. Otasevic, K. Rajavi, M. Böhm, and H. Krcmar. Understanding the relationship of conflict and success in software development projects. *Information and Software Technology*, 126:106331, 2020.
- [15] A. Bayaga. Multinomial logistic regression: Usage and application in risk analysis. *Journal of applied quantitative methods*, 5(2), 2010.
- [16] M. Belskie, H. Zhang, and B. M. Hemminger. Measuring toxicity toward women in game-based communities. *Journal of Electronic Gaming and Esports*, 1(1), 2023.

- [17] N. A. Beres, J. Frommel, E. Reid, R. L. Mandryk, and M. Klarkowski. Don't you know that you're toxic: Normalization of toxicity in online gaming. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–15, 2021.
- [18] J. Berkson. Application of the logistic function to bio-assay. *Journal of the American statistical association*, 39(227):357–365, 1944.
- [19] N. Bérubé, G. Ghiasi, M. Sainte-Marie, et al. Wiki-gendersort: Automatic gender detection using first names in wikipedia. 2020.
- [20] M. M. Bhat, S. Hosseini, A. Hassan, P. Bennett, and W. Li. Say 'yes'to positivity: Detecting toxic language in workplace communications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2017–2029, 2021.
- [21] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.
- [22] D. Borkan, L. Dixon, J. Sorensen, N. Thain, and L. Vasserman. Nuanced metrics for measuring unintended bias with real data for text classification. In *Companion proceedings of the 2019 world wide web conference*, pages 491–500, 2019.
- [23] A. Bosu and J. C. Carver. Impact of peer code review on peer impression formation: A survey. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 133–142. IEEE, 2013.
- [24] A. Bosu, M. Greiler, and C. Bird. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156. IEEE, 2015.



- [25] A. Bosu, A. Iqbal, R. Shahriyar, and P. Chakroborty. Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering*, 24(4):2636–2673, 2019.
- [26] A. Bosu and K. Z. Sultana. Diversity and inclusion in open source software (oss) projects: Where do we stand? In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019.
- [27] A. Braylan, O. Alonso, and M. Lease. Measuring annotator agreement generally across complex structured, multi-object, and free-text annotation tasks. In *Proceedings of the ACM Web Conference 2022*, pages 1720–1730, 2022.
- [28] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [29] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [30] P. Burnap and M. L. Williams. Hate speech, machine classification and statistical modelling of information flows on twitter: Interpretation and communication for policy decision making. 2014.
- [31] P. Burnap and M. L. Williams. Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy & Internet*, 7(2):223–242, 2015.
- [32] F. Calefato, F. Lanubile, and N. Novielli. Emotxt: a toolkit for emotion recognition from text. In *2017 seventh international conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, pages 79–80. IEEE, 2017.

- [33] K. D. A. Carillo, J. Marsan, and B. Negoita. Towards developing a theory of toxicity in the context of free/open source software & peer production communities. *SIGOPEN 2016*, 2016.
- [34] H. Chefer, S. Gur, and L. Wolf. Transformer interpretability beyond attention visualization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 782–791, 2021.
- [35] H. Chen, S. McKeever, and S. J. Delany. The use of deep learning distributed representations in the identification of abusive text. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 13, pages 125–133, 2019.
- [36] R. Chen, J. Wang, and X. Zhang. Ynu-hpcc at semeval-2021 task 5: Using a transformer-based model with auxiliary information for toxic span detection. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 841–845, 2021.
- [37] Y. Chen, Y. Zhou, S. Zhu, and H. Xu. Detecting offensive language in social media to protect adolescent online safety. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*, pages 71–80. IEEE, 2012.
- [38] J. Cheriyan, B. T. R. Savarimuthu, and S. Cranefield. Towards offensive language detection and reduction in four software engineering communities. In *Evaluation and Assessment in Software Engineering*, pages 254–259. 2021.
- [39] G. Chhablani, A. Sharma, H. Pandey, Y. Bhartia, and S. Suthaharan. Nlrg at semeval-2021 task 5: Toxic spans detection leveraging bert-based token classification and

- span prediction techniques. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 233–242, 2021.
- [40] F. Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [41] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- [42] S. Cohen. Contextualizing toxicity in open source: a qualitative study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1669–1671, 2021.
- [43] S. Cornegruta, R. Bakewell, S. Withey, and G. Montana. Modelling radiological language with bidirectional long short-term memory networks. *EMNLP 2016*, page 17, 2016.
- [44] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [45] D. A. d. Costa, S. McIntosh, C. Treude, U. Kulesza, and A. E. Hassan. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering*, 23:835–904, 2018.
- [46] G. Da San Martino, S. Yu, A. Barrón-Cedeno, R. Petrov, and P. Nakov. Fine-grained analysis of propaganda in news article. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 5636–5646, 2019.
- [47] O. Dabic, E. Aghajani, and G. Bavota. Sampling projects in github for MSR studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, pages 560–564. IEEE, 2021.

- [48] K. Daigle. Octoverse: The state of open source and rise of ai in 2023. <https://github.blog/2023-11-08-the-state-of-open-source-and-ai/>, 2023.
- [49] C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts. A computational approach to politeness with application to social factors. *51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 250–259, 2013.
- [50] R. V. W. De Joode. Managing conflicts in open source communities. *Electronic Markets*, 14(2):104–113, 2004.
- [51] F. Del Vigna<sup>12</sup>, A. Cimino<sup>23</sup>, F. Dell’Orletta, M. Petrocchi, and M. Tesconi. Hate me, hate me not: Hate speech detection on facebook. In *Proceedings of the First Italian Conference on Cybersecurity (ITASEC17)*, pages 86–95, 2017.
- [52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. pages 4171–4186, June 2019.
- [53] A. Diggs. Windows is sh\*t:’ linux users and the technical superiority problem. <https://medium.com/linuxforeveryone/windows-is-sh-t-linux-users-and-the-technical-superiority-problem-196a597aa860/>, 2021.
- [54] A. Diggs. Linux + coffee #4: Tribalism and toxicity. <https://www.linux4everyone.com/linux-plus-coffee-3-community-tribalism-toxicity-gatekeeping>, October 22nd, 2020.
- [55] M. Duggan. Online harassment. <https://www.pewresearch.org/internet/2017/07/11/online-harassment-2017/>, 2017.

- [56] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*, pages 49–60. IEEE, 2019.
- [57] C. D. Egelman, E. Murphy-Hill, E. Kammer, M. M. Hodges, C. Green, C. Jaspan, and J. Lin. Predicting developers’ negative feelings about code review. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 174–185. IEEE, 2020.
- [58] E. Eiding, R. Enbar, and T. Hassner. Age and gender estimation of unfiltered faces. *IEEE Transactions on information forensics and security*, 9(12):2170–2179, 2014.
- [59] A. Elnaggar, B. Walzl, I. Glaser, J. Landthaler, E. Scepankova, and F. Matthes. Stop illegal comments: A multi-task deep learning approach. In *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference*, pages 41–47, 2018.
- [60] N. Elsayed, A. S. Maida, and M. Bayoumi. Deep gated recurrent and convolutional network hybrid model for univariate time series classification. *International Journal of Advanced Computer Science and Applications*, 10(5), 2019.
- [61] S. Faci. The toxicity of open source. <https://www.esamir.com/20/12/23/the-toxicity-of-open-source/>, 2020.
- [62] I. Ferreira, B. Adams, and J. Cheng. How heated is it? understanding github locked issues. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 309–320, 2022.
- [63] I. Ferreira, J. Cheng, and B. Adams. The" shut the f\*\* k up" phenomenon: Characterizing incivility in open source code review discussions. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–35, 2021.

- [64] A. Filippova and H. Cho. The effects and antecedents of conflict in free and open source software development. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, pages 705–716, 2016.
- [65] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.
- [66] L. T. D. Foundation. Code of conduct. <https://www.documentfoundation.org/foundation/code-of-conduct/>.
- [67] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [68] S. V. Georgakopoulos, S. K. Tasoulis, A. G. Vrahatis, and V. P. Plagianakos. Convolutional neural networks for toxic comment classification. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, pages 1–6, 2018.
- [69] N. D. Gitari, Z. Zuping, H. Damien, and J. Long. A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, 10(4):215–230, 2015.
- [70] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*, pages 345–355, 2014.
- [71] G. Gousios and D. Spinellis. Ghtorrent: Github’s data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 12–21. IEEE, 2012.
- [72] K. Goyal, K. Agarwal, and R. Kumar. Face detection and tracking: Using opencv. In *2017 International conference of Electronics, Communication and Aerospace Technol-*

- ogy (*ICECA*), volume 1, pages 474–478. IEEE, 2017.
- [73] N. Goyal, I. D. Kivlichan, R. Rosen, and L. Vasserman. Is your toxicity my toxicity? exploring the impact of rater identity on toxicity annotation. *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW2):1–28, 2022.
- [74] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.
- [75] T. Grill and S. Castro. Python implementation of krippendorff’s alpha – inter-rater reliability. Github. <https://github.com/grrrr/krippendorff-alpha>, 2017.
- [76] I. Gunasekara and I. Nejadgholi. A review of standard text classification practices for multi-label toxicity identification of online content. In *Proceedings of the 2nd workshop on abusive language online (ALW2)*, pages 21–25, 2018.
- [77] S. D. Gunawardena, P. Devine, I. Beaumont, L. Garden, E. R. Murphy-Hill, and K. Blincoe. Destructive criticism in software code review impacts inclusion. 2022.
- [78] L. Hanu and Unitary team. Detoxify. Github. <https://github.com/unitaryai/detoxify>, 2020.
- [79] D. Hinkle, H. Jurs, and W. Wiersma. Applied statistics for the behavioral sciences, 1998.
- [80] T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.
- [81] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- [82] M. Honnibal and I. Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.
- [83] H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran. Deceiving google’s perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*, 2017.
- [84] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.
- [85] N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai, and E. Murphy-Hill. Investigating the effects of gender bias on github. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 700–711. IEEE, 2019.
- [86] V. Isaksen and B. Gambäck. Using transfer-based language models to detect hateful and offensive language online. In *Proceedings of the Fourth Workshop on Online Abuse and Harms*, pages 16–27, 2020.
- [87] M. Z. Islam, J. Liu, J. Li, L. Liu, and W. Kang. A semantics aware random forest for text classification. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1061–1070, 2019.
- [88] V. Jacques. Pygithub. <https://pygithub.readthedocs.io/>.
- [89] C. Jensen, S. King, and V. Kuechler. Joining free/open source software communities: An analysis of newbies’ first interactions on project mailing lists. In *2011 44th Hawaii international conference on system sciences*, pages 1–10. IEEE, 2011.
- [90] R. Johnson and T. Zhang. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 562–570, 2017.



- [91] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, 2017.
- [92] Kaggle. Toxic comment classification challenge. <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>, Mar 2018.
- [93] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21:2035–2071, 2016.
- [94] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2015.
- [95] O. Kononenko, O. Baysal, L. Guerrouj, and Y. Cao. Investigating code review quality: Do people and participation matter? pages 111–120, 09 2015.
- [96] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017.
- [97] K. Krippendorff. Reliability in content analysis: Some common misconceptions and recommendations. *Human communication research*, 30(3):411–433, 2004.
- [98] R. A. Kronmal. Spurious correlation and the fallacy of the ratio standard revisited. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 156(3):379–392, 1993.
- [99] D. Kumar, P. G. Kelley, S. Consolvo, J. Mason, E. Bursztein, Z. Durumeric, K. Thomas, and M. Bailey. Designing toxic content classification for a diversity

- of perspectives. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 299–318, 2021.
- [100] K. Kurita, A. Belova, and A. Anastasopoulos. Towards robust toxic content classification. *arXiv preprint arXiv:1912.06872*, 2019.
- [101] Z. Kurtanović and W. Maalej. On user rationale in software engineering. *Requirements Engineering*, 23(3):357–379, 2018.
- [102] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. *International Conference on Learning Representations*, 2020.
- [103] N. Lawson. What it feels like to be an open-source maintainer. <https://nolanlawson.com/2017/03/05/>, 2017.
- [104] A. Lenhart, M. Ybarra, K. Zickuhr, and M. Price-Feeney. Online harassment, digital abuse, and cyberstalking in america. data & society research institute. *Center for Innovative Public Health Research*. Retrieved from: [https://datasociety.net/pubs/oh/Online\\_Harassment\\_2016.pdf](https://datasociety.net/pubs/oh/Online_Harassment_2016.pdf), 2016.
- [105] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th international conference on software engineering*, pages 94–104, 2018.
- [106] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [107] J. S. Long and J. Freese. *Regression models for categorical dependent variables using Stata*, volume 7. Stata press, 2006.

- [108] E. Loper and S. Bird. Nltk: the natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70, 2002.
- [109] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2018.
- [110] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee. Hatexplain: A benchmark dataset for explainable hate speech detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14867–14875, 2021.
- [111] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [112] C. Miller, S. Cohen, D. Klug, B. Vasilescu, and C. Kästner. “did you miss my comment or what?” understanding toxicity in open source discussions. In *International Conference on Software Engineering, ICSE*. ACM, 2022.
- [113] P. Mishra, H. Yannakoudakis, and E. Shutova. Neural character-based composition models for abuse detection. *EMNLP 2018*, page 1, 2018.
- [114] M. Mukadam, C. Bird, and P. C. Rigby. Gerrit software code review data from android. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 45–48. IEEE, 2013.
- [115] E. Murphy-Hill, C. Jaspan, C. Egelman, and L. Cheng. The pushback effects of race, ethnicity, gender, and age in code review. *Communications of the ACM*, 65(3):52–57, 2022.

- [116] D. Nafus. ‘patches don’t have gender’: What is not open in open source software. *New Media & Society*, 14(4):669–683, 2012.
- [117] D. Nafus, J. Leach, and B. Krieger. Gender: Integrated report of findings. *FLOSSPOLs, Deliverable D*, 16, 2006.
- [118] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 284–292, 2005.
- [119] N. Nagappan and T. Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 364–373. IEEE, 2007.
- [120] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153, 2016.
- [121] N. Novielli, F. Calefato, F. Lanubile, and A. Serebrenik. Assessment of off-the-shelf se-specific sentiment analysis tools: An extended replication study. *Empirical Software Engineering*, 26(4):77, 2021.
- [122] N. Novielli, D. Girardi, and F. Lanubile. A benchmark study on sentiment analysis for software engineering research. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 364–375. IEEE, 2018.
- [123] OpenStack. Openstack code of conduct. <https://wiki.openstack.org/wiki/Conduct>.
- [124] N. Ousidhoum, Z. Lin, H. Zhang, Y. Song, and D.-Y. Yeung. Multilingual and multi-aspect hate speech analysis. In *Proceedings of the 2019 Conference on Empirical*

- Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4675–4684, 2019.
- [125] C. A. Paul. *The toxic meritocracy of video games: Why gaming culture is the worst*. U of Minnesota Press, 2018.
- [126] R. Paul, A. Bosu, and K. Z. Sultana. Expressions of sentiments during code reviews: Male vs. female. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering*, SANER ‘19. IEEE, 2019.
- [127] J. Pavlopoulos. Semeval 2021 task 5: Toxic spans detection. <https://competitions.codalab.org/competitions/25623>, 2020.
- [128] J. Pavlopoulos, L. Laugier, A. Xenos, J. Sorensen, and I. Androutsopoulos. From the detection of toxic spans in online discussions to the analysis of toxic-to-civil transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3721–3734, 2022.
- [129] J. Pavlopoulos, P. Malakasiotis, and I. Androutsopoulos. Deeper attention to abusive user content moderation. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1125–1135, 2017.
- [130] J. Pavlopoulos, J. Sorensen, L. Laugier, and I. Androutsopoulos. Semeval-2021 task 5: Toxic spans detection. In *Proceedings of the 15th international workshop on semantic evaluation (SemEval-2021)*, pages 59–69, 2021.
- [131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

- [132] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [133] M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso. An evaluation framework for plagiarism detection. In *Coling 2010: Posters*, pages 997–1005, 2010.
- [134] A. O. S. Project. Code of conduct. <https://source.android.com/setup/cofc>.
- [135] Y. Qiang, X. Li, and D. Zhu. Toward tag-free aspect based sentiment analysis: A multiple attention network approach. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.
- [136] Y. Qiang, D. Pan, C. Li, X. Li, R. Jang, and D. Zhu. Attcat: Explaining transformers via attentive class activation tokens. In *Advances in Neural Information Processing Systems*, 2022.
- [137] H. S. Qiu, B. Vasilescu, C. Kästner, C. Egelman, C. Jaspan, and E. Murphy-Hill. Detecting interpersonal conflict in issues and code review: Cross pollinating open- and closed-source approaches. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 41–55. IEEE, 2022.
- [138] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [139] I. Qureshi and Y. Fang. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, 14(1):208–238, 2011.
- [140] M. S. RAHMAN, Z. CODABUX, and C. K. ROY. Do words have power? understanding and fostering civility in code review discussion. 2024.

- [141] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 57–60, 2020.
- [142] L. Reinhard. This is bigger than us: Building a future for open source. <https://2014.jsconf.eu/speakers/lena-reinhard-this-is-bigger-than-us-building-a-future-for-open-source.html>, 2014.
- [143] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
- [144] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [145] J. SALTER. The perl foundation is fragmenting over code of conduct enforcement. <https://arstechnica.com/gadgets/2021/08/the-perl-foundation-is-fragmenting-over-code-of-conduct-enforcement/>, 2021.
- [146] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [147] L. Santamaría and H. Mihaljević. Comparison and benchmark of name-to-gender inference services. *PeerJ Computer Science*, 4:e156, 2018.
- [148] M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith. The risk of racial bias in hate speech detection. In *Proceedings of the 57th Annual Meeting of the Association for*

*Computational Linguistics*, pages 1668–1678, 2019.

- [149] J. Sarker. Identification and mitigation of toxic communications among open source software developers. In *37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.
- [150] J. Sarker. ‘who built this crap?’ developing a software engineering domain specific toxicity detector. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE ’22*, New York, NY, USA, 2023. Association for Computing Machinery.
- [151] J. Sarker, S. Sultana, S. Wilson, and A. Bosu. Toxispanse: An explainable toxicity detection in code review comments. In *Proceedings of the 17th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2023.
- [152] J. Sarker, A. Turzo, M. Dong, and A. Bosu. Toxicr: Replication package. Github. <https://github.com/WSU-SEAL/ToxiCR>, 2022.
- [153] J. Sarker, A. K. Turzo, and A. Bosu. A benchmark study of the contemporary toxicity detectors on software engineering interactions. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 218–227. IEEE, 2020.
- [154] J. Sarker, A. K. Turzo, M. Dong, and A. Bosu. Wsu seal implementation of the strudel toxicity detector. [https://github.com/WSU-SEAL/toxicity-detector/tree/master/WSU\\_SEAL](https://github.com/WSU-SEAL/toxicity-detector/tree/master/WSU_SEAL), 2022.
- [155] J. Sarker, A. K. Turzo, M. Dong, and A. Bosu. Automated identification of toxic code reviews using toxicr. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–32, 2023.



- [156] W. Sarle. Sas/stat user's guide: The varclus procedure. sas institute. *Inc., Cary, NC, USA*, 1990.
- [157] C.-E. Särndal, B. Swensson, and J. Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003.
- [158] R. E. Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pages 149–171, 2003.
- [159] W. A. Scott. Reliability of content analysis: The case of nominal scale coding. *Public opinion quarterly*, pages 321–325, 1955.
- [160] C. Sen, T. Hartvigsen, B. Yin, X. Kong, and E. Rundensteiner. Human attention maps for text classification: Do humans and neural networks focus on the same words? In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4596–4608, 2020.
- [161] M. Squire and R. Gazda. Floss as a source for profanity and insults: Collecting the data. In *2015 48th Hawaii International Conference on System Sciences*, pages 5290–5298. IEEE, 2015.
- [162] S. Srivastava, P. Khurana, and V. Tewari. Identifying aggression and toxicity in comments using capsule network. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 98–105, 2018.
- [163] L. Statistics. Spearman's rank-order correlation. *Laerd Statistics*, 2013.
- [164] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1379–1392, 2015.

- [165] I. Steinmacher and M. A. Gerosa. How to support newcomers onboarding to open source software projects. In *IFIP International Conference on Open Source Systems*, pages 199–201. Springer, 2014.
- [166] I. Steinmacher, I. Wiese, A. P. Chaves, and M. A. Gerosa. Why do newcomers abandon open source software projects? In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 25–32. IEEE, 2013.
- [167] V. N. Subramanian, I. Rehman, M. Nagappan, and R. G. Kula. Analyzing first contributions on github: What do newcomers do? *IEEE Software*, 39(1):93–101, 2020.
- [168] S. Sultana. Identification and mitigation of gender biases to promote diversity and inclusion among open source communities. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.
- [169] S. Sultana, J. Sarker, and A. Bosu. A rubric to identify misogynistic and sexist texts from software developer communications. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2021.
- [170] S. Sultana, J. Sarker, F. Israt, R. Paul, and A. Bosu. Automated identification of sexual orientation and gender identity discriminatory texts from issue comments. *arXiv preprint arXiv:2311.08485*, 2023.
- [171] S. Sultana, A. K. Turzo, and A. Bosu. Code reviews in open source projects: how do gender biases affect participation and outcomes? *Empirical Software Engineering*, 28(4):92, 2023.

- [172] T. A. Suman and A. Jain. Astartwice at semeval-2021 task 5: Toxic span detection using roberta-crf, domain specific pre-training and self-training. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 875–880, 2021.
- [173] Y. Tamura. Text span utilities for rust and python. Github. <https://github.com/tamuhey/textspan>, 2020.
- [174] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto. Bug or not? bug report classification using n-gram idf. In *2017 IEEE international conference on software maintenance and evolution (ICSME)*, pages 534–538. IEEE, 2017.
- [175] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. Review participation in modern code review: An empirical study of the android, qt, and openstack projects. *Empirical Software Engineering*, 22:768–817, 2017.
- [176] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2022. Open source software available from <https://github.com/heartexlabs/label-studio>.
- [177] A. K. Turzo and A. Bosu. What makes a code review useful to opendev developers? an empirical investigation. *Empirical Software Engineering*, 29(1):6, 2024.
- [178] A. K. Turzo, F. Faysal, O. Poddar, J. Sarker, A. Iqbal, and A. Bosu. Towards automated classification of code review feedback to support analytics. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2023.
- [179] A. Vaidya, F. Mai, and Y. Ning. Empirical analysis of multi-task learning for reducing identity bias in toxic comment detection. *Proceedings of the International AAAI*

- Conference on Web and Social Media*, 14:683–693, 2020.
- [180] B. van Aken, J. Risch, R. Krestel, and A. Löser. Challenges for toxic comment classification: An in-depth error analysis. *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 33–42, 2018.
  - [181] B. Vasilescu, A. Capiluppi, and A. Serebrenik. Gender, representation and online participation: A quantitative study. *Interacting with Computers*, 26(5):488–511, 2014.
  - [182] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
  - [183] S. Vaughan-Nichols. Linus torvalds takes a break from linux. <https://www.zdnet.com/article/linus-torvalds-takes-a-break-from-linux/>, 2018.
  - [184] S. Wang and Z. Marinho. Nova-wang at semeval-2020 task 12: Offenseemblert: An ensemble of offensive language classifiers. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 1587–1597, 2020.
  - [185] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.
  - [186] S. X. I am stepping down from psc and core, effective immediately. <https://perl.topicbox.com/groups/perl-core/T7a4f1bf9e069641f>, 2021.

- [187] M. Xia, A. Field, and Y. Tsvetkov. Demoting racial bias in hate speech detection. *Proceedings of the Eighth International Workshop on Natural Language Processing for Social Media*, pages 7–14, 2020.
- [188] L. Xu, L. Bing, W. Lu, and F. Huang. Aspect sentiment classification with aspect-specific opinion spans. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3561–3567, 2020.
- [189] L. Xu, C. Gotwalt, Y. Hong, C. B. King, and W. Q. Meeker. Applications of the fractional-random-weight bootstrap. *The American Statistician*, 74(4):345–358, 2020.
- [190] T. Yamane. Statistics: an introductory analysis-3. 1973.
- [191] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.
- [192] S. Zaheri, J. Leath, and D. Stroud. Toxic comment classification. *SMU Data Science Review*, 3(1):13, 2020.
- [193] M. Zampieri, P. Nakov, S. Rosenthal, P. Atanasova, G. Karadzhov, H. Mubarak, L. Derczynski, Z. Pitenis, and Ç. Çöltekin. Semeval-2020 task 12: Multilingual offensive language identification in social media (offenseval 2020). In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 1425–1447, 2020.
- [194] J. Zhang, J. P. Chang, C. Danescu-Niculescu-Mizil, L. Dixon, Y. Hua, N. Tahin, and D. Taraborelli. Conversations gone awry: Detecting early signs of conversational failure. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics.*, volume 1, 2018.

- [195] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.
- [196] S. Zhou, B. Vasilescu, and C. Kästner. How has forking changed in the last 20 years? a study of hard forks on github. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 445–456, 2020.
- [197] Q. Zhu, Z. Lin, Y. Zhang, J. Sun, X. Li, Q. Lin, Y. Dang, and R. Xu. Hitsz-hlt at semeval-2021 task 5: Ensemble sequence labeling and span boundary detection for toxic span detection. In *Proceedings of the 15th international workshop on semantic evaluation (SemEval-2021)*, pages 521–526, 2021.
- [198] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

**ABSTRACT****IDENTIFICATION AND MITIGATION OF TOXIC COMMUNICATIONS AMONG  
OPEN SOURCE SOFTWARE DEVELOPERS**

by

**JAYDEB SARKER****AUGUST 2024****Advisor:** DR. AMIANGSHU BOSU**Major:** COMPUTER SCIENCE**Degree:** DOCTOR OF PHILOSOPHY

Toxicity occurs during the developers' interaction in Free and Open Source Software (FOSS) projects. Prior studies in the Software Engineering domain reported toxic and unhealthy conversations during the developer's communication. These unhealthy behaviors may reduce the professional harmony and productivity of FOSS projects. For example, toxic code review comments may raise pushback from an author to complete suggested changes. Toxic communication with another person may hamper future communication and collaboration. Research also suggests that toxicity disproportionately impacts newcomers, women, and other participants from marginalized groups in open source communities. Therefore, toxicity is a barrier to promoting diversity, equity, and inclusion in the FOSS community. Since toxic communications are not uncommon among FOSS communities and such communications may have serious repercussions, the primary objective of this dissertation is *to automatically identify and mitigate toxicity during developers' textual interactions*. On this goal, this dissertation completed three studies, which include i) building an automated toxicity detector for the Software Engineering (SE) domain, ii)

developing an explainable toxicity detection tool for SE conversations, iii) an empirical investigation of the impacts of toxicity on the outcomes of FOSS projects.



## **AUTOBIOGRAPHICAL STATEMENT**

Jaydeb Sarker received his bachelor's degree in Computer Science and Engineering from the Rajshahi University of Engineering and Technology, Bangladesh, in 2016. He joined as a lecturer in the Department of Computer Science and Engineering at the University of Information Technology and Sciences, Dhaka, Bangladesh in 2017. Further, he worked as a research intern at Otto von Guericke University Magdeburg, Germany, with a Deutscher Akademischer Austauschdienst (DAAD) scholarship. He started his Ph.D. in Computer and Information Sciences at Wayne State University in 2019. Jaydeb Sarker earned his Masters in Computer Science from Wayne State University in 2022. His research interest lies in Human Aspects of Software Engineering, Empirical Software Engineering, and Natural Language Processing. He has been awarded as Thomas C. Rumble Graduate fellow from the Wayne State University graduate school for the 2022-2023 academic year. Also, Jaydeb Sarker received an Outstanding Graduate Teaching Assistant Award from the Department of Computer Science at Wayne State University in 2024. His research works have been published in top ACM and IEEE venues in Software Engineering, including ACM Transactions on Software Engineering and Methodology (TOSEM), the International Conference on Automated Software Engineering (ASE), the International Symposium on Empirical Software Engineering and Measurement (ESEM), and The Asia-Pacific Software Engineering Conference (APSEC).