M.Sc. Engg. (CSE) Thesis

Promoting Inclusive Developer Communication through Real-Time Toxicity Filtering

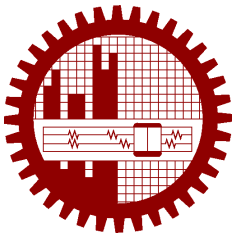
Submitted by

Md Awsaf Alam Anindya

0421052071

Supervised by

Dr. Anindya Iqbal



Submitted to

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

February 2026

Candidate's Declaration

I, do, hereby, certify that the work presented in this thesis, titled, "Promoting Inclusive Developer Communication through Real-Time Toxicity Filtering", is the outcome of the investigation and research carried out by me under the supervision of Dr. Anindya Iqbal, Professor, Department of CSE, BUET.

I also declare that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Md Awsaf Alam Anindya

0421052071

The thesis titled "**Promoting Inclusive Developer Communication through Real-Time Toxicity Filtering**", submitted by Md Awsaf Alam Anindya, Student ID 0421052071, Session April 2021, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents on February 22, 2026.

Board of Examiners

1. _____

   Dr. Anindya Iqbal — Chairman
   Professor — (Supervisor)
   Department of CSE, BUET, Dhaka

2. _____

   Professor Tanzima Hashem — Member
   Professor and Head — (Ex-Officio)
   Department of CSE, BUET, Dhaka

3. _____

   Chowdhury Md. Rakin Haider — Member
   Assistant Professor
   Department of CSE, BUET, Dhaka

4. _____

   Dr. Mohammad Saifur Rahman — Member
   Professor
   Department of CSE, BUET, Dhaka

5. _____

   Prof. Kazi Muheymin-Us-Sakib — Member
   Professor — (External)
   Institute of Information Technology
   University of Dhaka
   Dhaka

Acknowledgement

I am deeply grateful to my supervisor, Dr. Anindya Iqbal, for their invaluable guidance, support, and encouragement throughout this research journey. Their insightful feedback and dedication to excellence have been instrumental in shaping this thesis.

I extend my sincere appreciation to my thesis committee members for their time, constructive suggestions, and scholarly input that significantly improved the quality of this work.

I would also like to acknowledge the participants who generously contributed their time and insights during the user studies, without whom this research would not have been possible.

Special thanks to the open source community and the developers whose code review comments formed the foundation of our dataset, contributing to the advancement of research in this field.

Finally, I am profoundly grateful to my family for their unwavering support, understanding, and encouragement during this challenging academic journey. Their love and patience have been my greatest source of strength.

Dhaka
February 22, 2026

Md Awsaf Alam Anindya
0421052071

Contents

Candidate's Declaration i

Board of Examiners ii

Acknowledgement iii

List of Figures xi

List of Tables xvi

List of Algorithms xvii

Abstract xviii

1 Introduction 1

1.1 Background and Motivation 1
1.2 Problem Statement 1
1.3 Research Objectives 2
1.4 Research Questions 3
1.5 Approach and Methodology 3
1.6 Key Contributions 3
1.7 Thesis Scope and Limitations 4
1.8 Thesis Organization 4

2 Literature Review and Background 6

2.1 Toxicity in Software Engineering 6
2.2 Automated Toxicity Detection 6
2.3 Text Style Transfer and Detoxification 7
2.4 Repercussions of Toxicity on OSS Projects 7
2.5 Comparison with Existing Tools 8
2.6 Research Gaps 8
2.7 Summary 8

3 Problem Statement and Research Objectives 10

3.1 The Problem: Reactive Approaches to Toxicity . . . 10
3.2 The Need for Proactive Intervention . . . 10
3.3 Research Objectives . . . 11
3.4 Research Questions . . . 11
3.5 Summary . . . 12

4 System Design and Architecture **13**
4.1 System Overview . . . 13
4.1.1 Design Principles . . . 13
4.1.2 High-Level Architecture . . . 13
4.2 Module 1: Toxicity Filter . . . 15
4.2.1 Design Rationale . . . 15
4.2.2 Architecture Components . . . 15
4.2.3 Performance Requirements . . . 17
4.3 Module 2: Communication Coach . . . 17
4.3.1 Educational Framework . . . 17
4.3.2 Multi-Class Classification System . . . 17
4.3.3 Prompt Design Framework . . . 18
4.4 Module 3: The Reframer . . . 18
4.4.1 Text Style Transfer Framework . . . 19
4.4.2 Teacher-Student Architecture . . . 19
4.4.3 Evaluation Metrics Framework . . . 20
4.5 System Integration Architecture . . . 21
4.5.1 Component Interaction Design . . . 21
4.5.2 Deployment Architecture . . . 21
4.6 Design Trade-offs and Decisions . . . 22
4.6.1 Design Trade-offs . . . 22
4.7 Summary . . . 22

5 Implementation **23**
5.1 Module 1: Toxicity Filter . . . 23
5.1.1 Dataset Preparation . . . 23
5.1.2 Model Training . . . 24
5.1.3 Model Evaluation . . . 24
5.2 Module 2: Communication Coach . . . 25
5.2.1 Dataset for Toxicity Subcategories . . . 25
5.2.2 Prompt Engineering Strategy . . . 26
5.2.3 Iterative Prompt Refinement: Detailed Sub-Iteration Analysis . . . 26
5.2.4 Model Evaluation . . . 28

5.2.5 Per-Category Classification Performance . . . 28
5.3 Module 3: The Reframer . . . 28
5.3.1 Teacher-Student Framework . . . 30
5.3.2 Student Model Training . . . 31
5.3.3 Evaluation . . . 31
5.4 Summary . . . 31

6 Browser Extension and GitHub Integration **33**
6.1 Browser Extension Overview . . . 33
6.2 Extension Architecture . . . 35
6.3 GitHub Interface Integration . . . 35
6.4 Real-time Analysis Integration . . . 36
6.5 Technical Implementation Details . . . 36
6.5.1 ONNX Export and Model Quantization . . . 36
6.5.2 Two-Stage Inference Pipeline . . . 36
6.5.3 Privacy-Preserving On-Device Architecture . . . 37
6.6 Participant Recruitment and Survey . . . 38
6.7 Qualitative Evaluation Results . . . 38
6.8 Summary . . . 39

7 Evaluation and Results **40**
7.1 Module 1: Toxicity Filter Evaluation . . . 40
7.1.1 10-Fold Cross-Validation Stability Analysis . . . 42
7.2 Module 2: Communication Coach Evaluation . . . 44
7.2.1 Confusion Matrix Analysis . . . 44
7.2.2 Manual Evaluation of Communication Coach . . . 45
7.2.3 Per-Category Performance Analysis . . . 46
7.3 Module 3: The Reframer Evaluation . . . 48
7.4 User Evaluation with Integrated Browser Extension . . . 50
7.4.1 Participant Profile . . . 50
7.4.2 Technology Acceptance Model Results . . . 50
7.4.3 Participant Background and Code Review Experience . . . 51
7.4.4 Open-Ended Response Analysis . . . 53
7.5 Summary . . . 54

8 Discussion and Implications **55**
8.1 Interpretation of Key Findings . . . 55
8.1.1 Hypothesis Validation and Surprising Results . . . 55
8.1.2 Performance Analysis Deep Dive . . . 56

8.2 Effectiveness of LLMs in Code Review Detoxification . . . 56
8.3 Open-Source LLMs vs Proprietary Models . . . 57
8.4 Socio-Technical Implications . . . 57
8.5 Fundamental Insights and Theoretical Contributions . . . 58
8.5.1 The Linguistic Explicitness Spectrum . . . 58
8.5.2 Knowledge Distillation Efficiency for Domain-Specific Style Transfer . . . 58
8.5.3 The Fundamental Detection-Preservation Tradeoff . . . 59
8.5.4 The Real-Time Constraint as Architectural Driver . . . 59
8.6 Implications for Software Engineering . . . 60
8.6.1 Impact on Developer Communication Practices . . . 60
8.6.2 Community Health and Inclusion . . . 61
8.7 Technical Contributions and Innovations . . . 61
8.7.1 Methodological Advances . . . 61
8.7.2 Prompt Engineering Contributions . . . 61
8.8 Practical Implications and Deployment Considerations . . . 62
8.8.1 Adoption Strategies for Organizations . . . 62
8.8.2 Scalability and Infrastructure Requirements . . . 63
8.9 Limitations and Challenges . . . 63
8.9.1 Technical Limitations . . . 63
8.9.2 Ethical Considerations . . . 64
8.9.3 Generalizability Concerns . . . 64
8.10 Lessons Learned . . . 65
8.10.1 Research Methodology Insights . . . 65
8.10.2 Development Process Insights . . . 66
8.10.3 User Study Revelations . . . 66
8.11 Broader Impact and Societal Implications . . . 67
8.11.1 Contribution to Inclusive Software Development . . . 67
8.11.2 AI Ethics and Responsible Development . . . 68
8.11.3 Future of AI-Assisted Communication . . . 69
8.12 Threats to Validity . . . 69
8.12.1 Internal Validity . . . 69
8.12.2 External Validity . . . 70
8.12.3 Construct Validity . . . 70
8.12.4 Ecological Validity . . . 70
8.13 Recommendations for Future Deployment . . . 71
8.13.1 Technical Recommendations . . . 71
8.13.2 Organizational Recommendations . . . 71
8.14 Summary . . . 72

8.14.1 Key Insights . . . 72
8.14.2 Research Contributions . . . 73

9 Conclusion and Future Work **74**
9.1 Research Summary . . . 74
9.1.1 Problem Addressed . . . 74
9.1.2 Solution Approach . . . 74
9.2 Research Questions Revisited . . . 75
9.2.1 RQ1: Optimal Toxicity Detection . . . 75
9.2.2 RQ2: Fine-grained Classification with Explanations . . . 75
9.2.3 RQ3: Effective Text Detoxification . . . 76
9.2.4 RQ4: Real-time Workflow Integration . . . 76
9.2.5 RQ5: Developer Acceptance and Adoption . . . 76
9.3 Key Contributions . . . 76
9.3.1 Technical Contributions . . . 77
9.3.2 Empirical Contributions . . . 77
9.3.3 Practical Contributions . . . 78
9.4 Broader Impact and Significance . . . 78
9.4.1 Academic Impact . . . 78
9.4.2 Industry Relevance . . . 79
9.4.3 Societal Implications . . . 80
9.5 Limitations and Constraints . . . 80
9.5.1 Technical Limitations . . . 80
9.5.2 Evaluation Limitations . . . 81
9.6 Future Work . . . 82
9.6.1 Short-term Improvements . . . 82
9.6.2 Medium-term Research Directions . . . 82
9.6.3 Long-term Vision and Research Directions . . . 83
9.6.4 Research Methodology Advances . . . 84
9.7 Broader Research Implications . . . 85
9.7.1 AI-Assisted Human Communication . . . 85
9.7.2 Human-Computer Interaction Evolution . . . 86
9.8 Call for Further Research . . . 87
9.8.1 Critical Research Needs . . . 87
9.8.2 Collaborative Research Opportunities . . . 88
9.9 Final Reflections . . . 88
9.9.1 Lessons from the Research Journey . . . 88
9.9.2 Vision for the Future . . . 89
9.9.3 Closing Thoughts . . . 90

9.10 Summary . . . 90
9.10.1 Novelty . . . 91

BIBLIOGRAPHY **92**

A Algorithms **100**
A.1 Dataset Curation Process . . . 100
A.2 Toxicity Filter Processing Pipeline . . . 100
A.3 Communication Coach Classification . . . 101
A.4 The Reframer Training Process . . . 101
A.5 The Reframer Detoxification . . . 101

B Codes **103**
B.1 Model Training Implementation . . . 103
B.1.1 BERT-based Toxicity Filter Training . . . 103
B.1.2 Communication Coach Prompt Engineering . . . 103
B.1.3 The Reframer Teacher-Student Training . . . 104
B.2 Browser Extension Implementation . . . 104
B.2.1 Architecture Overview . . . 104
B.2.2 Key Implementation Features . . . 105
B.3 Replication Package . . . 105

C Additional Supporting Materials **106**
C.1 User Study Materials . . . 106
C.1.1 Demographic Survey Questions . . . 106
C.1.2 Technology Acceptance Model Questionnaire . . . 106
C.2 System Configuration Details . . . 108
C.2.1 Model Hyperparameters . . . 108
C.2.2 Data Processing Pipeline Configuration . . . 108
C.3 Detailed Experimental Results . . . 109
C.3.1 Cross-Validation Results . . . 109
C.3.2 Error Analysis Examples . . . 109
C.4 Prompt Templates . . . 110
C.4.1 Communication Coach Classification Prompt . . . 110
C.4.2 Detoxification Prompt Template . . . 111
C.5 System Performance Benchmarks . . . 112
C.5.1 Latency Measurements by Component . . . 112
C.5.2 Deployment Optimization . . . 112
C.6 Data Collection Details . . . 112
C.6.1 Repository Selection Criteria . . . 112

C.6.2 Data Annotation Guidelines 113
C.7 Browser Extension Implementation Details 113
C.7.1 Manifest V3 Configuration 113
C.7.2 On-Device Inference Architecture 115
C.8 Summary 115

List of Figures

1.1 Motivational workflow of ToxiShield. A code reviewer writing a GitHub pull request comment triggers the three-module pipeline: Module 1 (Toxicity Filter) identifies whether the text is toxic; if so, Module 2 (Communication Coach) classifies the toxicity subcategory with explanations; and Module 3 (The Reframer) generates a constructive, professional alternative. The developer can then accept or revise before submitting. 4

4.1 High-level conceptual overview of ToxiShield's intervention paradigm. The system positions itself between the moment a developer composes a code review comment and the moment it is submitted—a design choice that enables proactive prevention rather than reactive moderation. The three-module pipeline analyses and optionally transforms the draft comment before it enters the review thread. . 14

4.2 Motivational Workflow of ToxiShield (Figure 1 from FSE 2026 paper). The workflow shows how ToxiShield proactively identifies and mitigates toxicity in three stages: (1) binary toxicity detection, (2) classification into toxicity subcategories with explanations, and (3) text detoxification generating constructive alternatives. 14

4.3 Detailed system architecture diagram of ToxiShield showing the component relationships between the browser extension frontend, the ONNX-quantized BERT binary classifier (Module 1), the LLM-based Communication Coach (Module 2), and the Llama 3.2 3B detoxification engine (Module 3). The cascade architecture ensures non-toxic comments bypass the computationally expensive LLM stages. 15

4.4 ToxiShield system flow diagram illustrating the decision logic and data flow between the three modules. A submitted comment enters Module 1 for binary screening; if flagged as toxic, it proceeds to Module 2 for subcategory classification and explanation generation, then to Module 3 for detoxified alternative generation. At each stage the developer is shown intermediate results and retains control over the final submission. 16

4.5 Workflow of the Toxicity Filter module (Figure 2 from FSE 2026 paper). A Masked Language Model (BERT-base-uncased) is trained to classify comments as toxic or non-toxic, with the dataset comprising 38,761 code review samples (10,120 toxic and 28,641 non-toxic). 16
4.6 Communication Coach: Five-stage iterative prompt evolution for multi-class toxicity classification (Figure 3 from FSE 2026 paper). The prompt is progressively refined from a baseline zero-shot approach through operationalized definitions, linguistic nuance handling, lexical constraints, and rare category refinement. Stage 4 achieves the best Macro MCC and serves as the final prompt configuration used to evaluate all LLM families. 17
4.7 The Reframer workflow (Figure 4 from FSE 2026 paper). A teacher model is used to create a parallel dataset from 10,120 toxic comments, which is then used to train a student detoxification model with iterative prompting. The workflow includes style transfer metrics for evaluation. 19

5.1 Iterative prompt evolution for the Communication Coach module, showing Macro MCC scores across 5 stages with GPT-4o. Stage 4 (lexical rules and constraints) achieves the best MCC of 0.37, which was used as the final prompt configuration for evaluating all LLM families. 27
5.2 Iterative prompt engineering process for the Communication Coach module. The diagram illustrates how each iteration cycle improves the Macro MCC score through targeted refinements: adding behavioral cue definitions, incorporating few-shot examples, embedding lexical resources (profanity lists, anger indicators), and introducing logic constraints for mutual-exclusivity. 27
5.3 Multi-class prompt engineering performance chart across the five reported prompt evolution stages for the Communication Coach. Each bar represents the Macro MCC achieved by GPT-4o on the validation set. Stage 4 (Lexical Rules and Constraints) achieves the highest Macro MCC of 0.37, which was adopted as the final prompt configuration for cross-model evaluation. 28
5.4 Chain-of-Thought (CoT) prompting strategy used during the Reframer's teacher-model data generation phase. The teacher LLM is guided to first analyse the toxic elements in the comment step by step, then reconstruct the sentence with the toxic signal removed while preserving the technical core. This structured reasoning produces higher-quality parallel training data compared to direct rewriting without explicit reasoning steps. 29
5.5 Teacher model comparison for parallel dataset generation. GPT-4o (2024-05-13) achieves the highest J-Score (88.14%), balancing strong style accuracy (96.17%), fluency (99.01%), and content preservation (73.86%). 30

6.1 End-to-end system design of ToxiShield integrated with GitHub pull request reviews. The diagram illustrates how the three-module pipeline operates within the browser extension: Module 1 (Toxicity Filter) screens comments on the fly, Module 2 (Communication Coach) categorises the toxicity subcategory and generates an explanation, and Module 3 (The Reframer) produces a constructive alternative—all on-device without external API calls. 34
6.2 ToxiShield browser extension in action during a GitHub pull request code review. When the developer types a potentially toxic comment, the extension's real-time inference pipeline detects the toxicity, presents the flagged subcategory with an explanation, and offers a professionally reframed alternative—all within the GitHub interface without leaving the page. 34
6.3 End-to-end view of ToxiShield operating within the GitHub pull request workflow. The annotated screenshot shows the three-stage feedback panel that appears when toxic content is detected: the toxicity label (Module 2 output), the natural-language explanation, and the suggested reframed alternative (Module 3 output). The developer retains full agency to accept, modify, or dismiss the suggestion. 35
6.4 ToxiShield system architecture showing the on-device inference stack. The browser extension embeds the quantized ONNX BERT model for Stage 1 binary detection (sub-500 ms), while Stage 2 delegates to a locally-hosted Llama 3.2 3B model served by Ollama for fine-grained classification and detoxification. No comment content leaves the developer's machine. 37

7.1 Performance comparison of binary toxicity detection models across accuracy and toxic-class F1-score. Fine-tuned BERT-base-uncased achieves the best performance (Accuracy=0.98, F1=0.97), significantly outperforming zero-shot LLMs and the off-the-shelf ToxiCR baseline. 41
7.2 Detailed architecture of Module 1 (Toxicity Filter), showing the end-to-end pipeline from the 15 million raw GitHub PR comments through the retrained ToxiCR filtering step, stratified sampling, manual annotation, and the final 38,761-sample dataset used to fine-tune BERT-base-uncased. The retrained model achieves 98% accuracy and 97% toxic-class F1 on the final test split. . . 41
7.3 10-fold cross-validation misclassification analysis for the Toxicity Filter (BERT-base-uncased). Left: False Positive (FP) and False Negative (FN) counts per fold. Right: Total misclassifications per fold. The consistent error rates (ranging 1.52%–2.06%) across folds confirm the model's training stability and domain generalisability. 42

7.4 Stacked error composition across 10 folds: False Negatives (FN, toxic comments missed) consistently constitute 55%–72% of total misclassifications across all folds. This FN-dominant pattern reflects the dataset's class imbalance and is the operationally preferred error mode for a proactive writing assistant, where false alarms are more disruptive to workflow than occasional misses. 43
7.5 Prompt design structure for the Communication Coach multi-class toxicity classifier. The five-component prompt schema (Role Clarification, Task Assignment, Category Definitions with Few-Shot Examples, Classification Guidelines, and XML Output Specification) is shown with the iterative refinement annotations indicating which components were introduced or modified at each of the five major prompt evolution stages. 43
7.6 Multi-class toxicity classification performance comparison across LLM families. Claude 3.5 Sonnet achieves the best Macro F1-score (0.42) and Macro MCC (0.39) across 12 categories. 44
7.7 Confusion matrix heatmap for Claude 3.5 Sonnet's multiclass toxicity classification. Explicit categories (Non-Toxic, Profanity) show high accuracy, while rare categories (Arrogance, Entitlement) struggle due to data sparsity. . . 45
7.8 Per-category MCC scores for Claude 3.5 Sonnet with Stage 4 prompt across 12 toxicity categories. Categories with explicit linguistic markers (Profanity, Non-Toxic) achieve strong performance, while behaviorally-defined categories (Arrogance, Entitlement) present significant challenges due to definitional ambiguity and class imbalance. 46
7.9 Radar chart of per-category MCC scores for the Communication Coach (Claude 3.5 Sonnet, Stage 4 prompt). The asymmetric profile reveals strong detection for explicit toxicity categories (Profanity, Non-Toxic) and near-chance performance for implicit behavioral categories (Arrogance, Entitlement, Others-Toxic). . . . 47
7.10 Performance comparison of detoxification models across style accuracy, fluency, content preservation, and J-Score. The fine-tuned Llama 3.2 3B model achieves the highest J-Score (84.00%), outperforming all other evaluated models. 49
7.11 Demographic profile of the 10 user study participants: age distribution, gender identity, and years of professional experience. Participants span early-to-mid career professional software developers from the United States and Bangladesh. 51
7.12 ToxiShield Technology Acceptance Model (TAM) survey results from 10 professional software developers. The tool received particularly high ratings for ease of use (PEOU: 4.30 for browser extension usability) and overall satisfaction (7.83/10). 52

7.13 User study survey responses: (left) distribution of overall tool satisfaction scores (1–10 scale, mean=7.83); (right) participant-reported frequency of incivility exposure in their development workplace. Half the participants reported incivility approximately once per month, underscoring the relevance of ToxiShield's intervention in realistic developer workflows. 53

List of Tables

2.1 Feature comparison of toxicity detection tools. MC: Multi-Class classification, DT: Detoxification. Checkmarks (✓) indicate supported features; crosses (–) indicate unsupported features. 8

5.1 The 12 toxicity categories used in Module 2 (Communication Coach), including the Non-Toxic class. Definitions derived from Sarker et al. [39] and operationalised in the prompt schema. 25

7.1 Performance Metrics of Binary Toxicity Detection Models. For each model, bold indicates significant improvements over the others. 40
7.2 Comparison of LLMs for Multi-Class Classification – Best Model per Family. Avg: macro over instances; Macro: unweighted average over 12 categories. . . 44
7.3 Performance Comparison of Detoxification Models 48
7.4 Complete TAM Survey Results 51

C.1 Complete Hyperparameter Configuration for Toxicity Filter 108
C.2 Hyperparameter Configuration for The Reframer (Llama 3.2 3B) 108
C.3 5-Fold Cross-Validation Results for Toxicity Detection 109
C.4 Detailed Latency Breakdown by System Component 112

List of Algorithms

1 ToxiShield Processing Pipeline 21
2 Multi-Stage Dataset Curation Process 100
3 Toxicity Filter Workflow 101
4 Communication Coach Multi-Class Classification 101
5 Teacher-Student Model Training for Detoxification 102
6 Real-Time Text Detoxification Process 102

Abstract

Toxic interactions during code reviews can undermine teamwork and hinder productivity in software engineering (SE) teams. While prior studies explore toxicity detection and empirical investigation, they lack real-time detoxification tools to support the SE community. This thesis presents ToxiShield, a comprehensive framework for promoting inclusive developer communication through real-time toxicity filtering in GitHub pull request reviews.

ToxiShield is built using three modules: (i) **Module 1 – Toxicity Filter**, which identifies whether a text is toxic using fine-tuned BERT models; (ii) **Module 2 – Communication Coach**, which facilitates just-in-time fine-grained toxicity categorization with explanations powered by large language models; and (iii) **Module 3 – The Reframer**, which generates a revised, constructive alternative of a toxic text while preserving technical intent. The system is implemented as a browser extension that operates proactively, identifying and mitigating toxicity as developers write comments, rather than reactively flagging content after publication.

Our BERT-based binary detection model, trained on 38,761 code review samples, achieves 98% accuracy and an F1-score of 97%. The multiclass toxicity classification component provides detailed reasoning for toxicity categorization across 12 toxicity categories, with Claude 3.5 Sonnet achieving 42% macro F1-score and 39% macro MCC in multiclass toxicity classification with detailed reasoning. For detoxification, we fine-tune five large language models on 10,120 code review comments, with the fine-tuned Llama 3.2 model achieving 95.27% style accuracy, 97.03% fluency, 67.07% content preservation, and an 84% J-score.

The system's effectiveness is validated through comprehensive evaluation including automated metrics and human evaluation studies. A user study with 10 software developers using the Technology Acceptance Model confirms the tool's perceived usefulness and ease of adoption, with high satisfaction scores across all dimensions. ToxiShield represents the first real-time detoxification tool specifically designed for software engineering environments, setting a benchmark for advancing constructive communication and fostering inclusivity in open-source communities.

Chapter 1

Introduction

1.1 Background and Motivation

Successful Open Source Software (OSS) projects depend on robust coordination and teamwork among contributors [41, 44]. Developers primarily collaborate through textual communication channels, including mailing lists, code reviews (CR), version control systems, issue discussions, and bug trackers. These communication platforms serve as the backbone of distributed software development, enabling knowledge sharing, technical discussions, and collaborative problem-solving.

However, research shows that communication in OSS communities can breach professional boundaries and devolve into toxic interactions [8, 23, 32, 39]. Such negativity has far-reaching consequences that extend beyond individual interactions, creating systemic barriers that affect the entire development ecosystem. Toxicity disproportionately creates barriers for newcomers [32] and causes existing contributors to leave projects [38], leading to talent drain and reduced project sustainability.

The impact of toxic behavior is particularly severe for underrepresented groups in software engineering. Toxicity disproportionately harms minorities and junior developers [14], leading to emotional distress and burnout that significantly hinder productivity [10, 32]. Over time, these interpersonal issues degrade the health of the community by undermining diversity [14, 23] and fostering conflicts [6, 10], which can ultimately compromise the final quality of the software products [23].

1.2 Problem Statement

While many OSS projects have adopted Codes of Conduct (CoC) to foster respectful environments, their practical impact has been limited [2, 26]. The primary obstacle is enforcement, which typically relies on manual validation—a process that is both labor-intensive

and unsustainable for large communities [23]. The inadequacy of this manual approach has spurred the development of automated systems designed to identify and mitigate harmful discourse. Consequently, a growing body of Software Engineering (SE) research focuses on creating tools to detect not only overt toxicity [32, 38, 40] but also nuanced antisocial behaviors like incivility [10, 11], destructive criticism [14], sexism [4, 42], and pushback [6].

However, existing approaches for detecting toxic interactions in OSS communities are primarily *reactive*, operating on a post hoc basis where classifiers flag problematic texts for later review and removal [40]. This delay means that by the time an intervention occurs, significant harm may have already been done to contributor relationships [10, 49]. To foster healthier communication, a shift from reactive flagging to proactive prevention is necessary.

The core research problem addressed in this thesis is the lack of *real-time, proactive tools* for toxicity detection and mitigation in software engineering environments. Current solutions fail to provide:

- **Real-time detection:** Immediate identification of toxic content as it is being written
- **Educational feedback:** Explanations of why content is considered toxic and how to improve it
- **Constructive alternatives:** Suggestions for rephrasing toxic content while preserving technical intent
- **Seamless integration:** Non-intrusive incorporation into existing developer workflows

1.3 Research Objectives

This thesis aims to address the identified gaps by developing ToxiShield, a comprehensive framework for real-time toxicity detection and mitigation in code review environments. The primary research objectives are:

1. **Develop a high-accuracy toxicity detection system** that can reliably identify toxic comments in code review contexts with minimal false positives and negatives.
2. **Create a fine-grained toxicity classification system** that can categorize toxic comments into specific subcategories and provide explanations for the classifications.
3. **Design and implement a text detoxification module** that can automatically rewrite toxic comments into constructive alternatives while preserving technical content and intent.
4. **Build a real-time integration system** that seamlessly incorporates toxicity detection and mitigation into existing developer workflows, specifically GitHub pull request reviews.

5. **Evaluate the system's effectiveness** through comprehensive metrics including automated performance evaluation and user acceptance studies.

1.4 Research Questions

To achieve these objectives, this thesis addresses five research questions detailed in Chapter 3, focusing on optimal detection models, fine-grained classification with explanations, effective text detoxification, real-time integration requirements, and developer acceptance.

1.5 Approach and Methodology

This thesis presents ToxiShield, a proactive tool for detecting and mitigating toxicity in code review comments. ToxiShield employs a three-module architecture (detailed in Chapter 4), comprising:

- **Module 1 – Toxicity Filter:** Acting as the first line of defense, this module performs binary classification to distinguish toxic from non-toxic code review comments. When a comment is flagged as toxic, it triggers the downstream modules for further analysis and remediation.
- **Module 2 – Communication Coach:** This module classifies detected toxic content into specific subcategories (e.g., Insult, Threat, Obscenity) and generates a targeted explanation detailing precisely why the comment is problematic, transforming moderation into a constructive learning opportunity.
- **Module 3 – The Reframer:** This generative module analyzes the author's core intent and produces a revised, constructive alternative that preserves the technical message while removing harmful language, presented alongside a brief rationale explaining the changes made.

Together these modules operate in real-time as a developer drafts feedback, as illustrated in Figure 1.1.

1.6 Key Contributions

This thesis makes several key contributions: (1) a comprehensive dataset of 38,761 code review comments (10,120 toxic and 28,641 non-toxic) with toxicity labels, (2) high-performance detection models achieving 98% accuracy and an F1-score of 97% using BERT-base-uncased, (3)

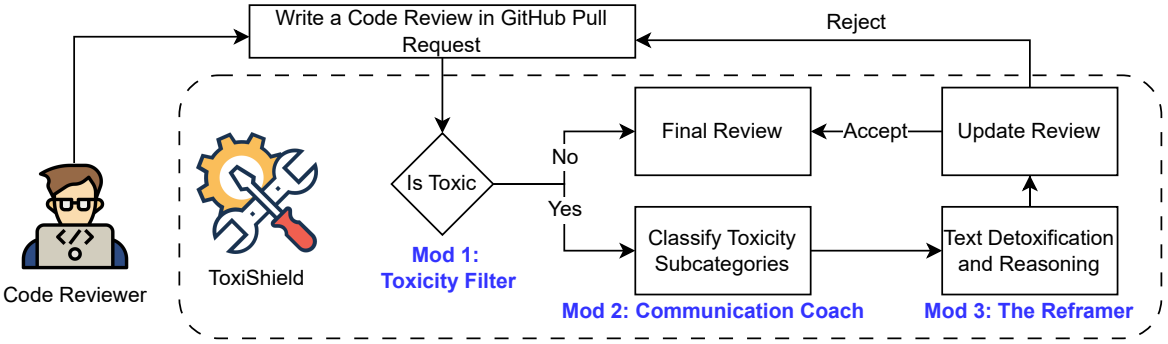


Figure 1.1: Motivational workflow of ToxiShield. A code reviewer writing a GitHub pull request comment triggers the three-module pipeline: Module 1 (Toxicity Filter) identifies whether the text is toxic; if so, Module 2 (Communication Coach) classifies the toxicity subcategory with explanations; and Module 3 (The Reframer) generates a constructive, professional alternative. The developer can then accept or revise before submitting.

a multi-class classification system with explanatory feedback achieving 42% macro F1-score and 39% macro MCC using Claude 3.5 Sonnet with iterative prompt engineering across 12 toxicity categories, (4) an effective text detoxification framework achieving 84% J-score using fine-tuned Llama3.2, (5) a real-time browser extension integration, and (6) comprehensive empirical validation with 10 participants using the Technology Acceptance Model. Detailed contributions are presented in Chapter 9.

1.7 Thesis Scope and Limitations

This thesis focuses on toxicity in GitHub pull request review comments, evaluating English-language content across 11 toxicity categories (detailed in Chapter 2). Key limitations include textual toxicity focus, English-only evaluation, and primary concentration on explicit rather than subtle toxicity. Comprehensive limitations are discussed in Chapters 8 and 9.

1.8 Thesis Organization

This thesis is organized into nine chapters:

Chapter 2 presents a comprehensive literature review covering toxicity in software engineering, natural language processing approaches for toxicity detection, and text style transfer techniques.

Chapter 3 provides a detailed problem statement, formal research questions, and success criteria for the proposed solution.

Chapter 4 describes the overall system architecture, design decisions, and the three-module framework of ToxiShield.

Chapter 5 details the implementation of each module, including model training, dataset preparation, and technical implementation decisions.

Chapter 6 presents the browser extension implementation and integration with GitHub workflows.

Chapter 7 presents comprehensive evaluation results including automated metrics, comparative analysis, and error analysis.

Chapter 8 discusses key findings, implications for the software engineering community, and lessons learned from the research.

Chapter 9 concludes the thesis with a summary of contributions, limitations, and directions for future work.

The thesis also includes appendices with detailed experimental results, prompt templates, and supplementary materials.

Chapter 2

Literature Review and Background

This chapter reviews prior work on toxicity in software engineering and identifies research gaps that motivate ToxiShield. We examine antisocial behavior in OSS development, automated detection techniques, and text style transfer approaches.

2.1 Toxicity in Software Engineering

Research shows that communication in OSS communities can breach professional boundaries and devolve into toxic interactions [23, 32, 38, 39]. Toxicity in software engineering differs from social media platforms due to distinct community dynamics and professional contexts [23]. Unlike general social media toxicity, toxicity in software engineering often intertwines with technical criticism, making detection challenging.

Sarker et al. [39] conducted a large-scale GitHub study identifying 11 distinct toxicity subcategories: Flirtation, Obscenity, Self-deprecation, Entitlement, Identity Attack, Threats, Insult, Arrogance, Trolling, Profanity, and Others-Toxic. Toxic interactions negatively impact developers' mental health [3], cause stress and burnout [34], disproportionately affect underrepresented groups [14], diminish productivity [6, 27], and undermine community diversity [23, 38].

2.2 Automated Toxicity Detection

While many OSS projects have adopted Codes of Conduct, their practical impact has been limited due to manual enforcement challenges [2, 26]. This has spurred development of automated systems to identify and mitigate harmful discourse.

Raman et al. [34] created the first SE-specific toxicity detector, establishing domain-specific foundations but revealing performance limitations with diverse toxicity forms [23, 32, 38]. Qiu et

al. [32] improved detection by incorporating pushback [6] and toxicity metrics [34]. Sarker et al. [40] developed ToxiCR, achieving 88.9% F1-score on 19,651 labeled code review texts, but limited to binary classification without real-time integration.

However, existing approaches are primarily reactive, operating post-hoc where classifiers flag problematic texts for later review and removal [40]. This delay means significant harm may already occur before intervention [10, 49]. Recent work by Rahman et al. [33] used LLMs to identify uncivil comments and propose alternatives, demonstrating context-aware intervention potential but lacking real-time integration and explanatory feedback.

2.3 Text Style Transfer and Detoxification

Text Style Transfer (TST) rewrites text to alter stylistic properties while preserving content and meaning [12]. For toxicity mitigation, TST transforms toxic text into neutral, professional alternatives while maintaining technical intent. Prabhumoye et al. [31] introduced back-translation methods for style transfer, demonstrating that neural machine translation techniques could be adapted for stylistic transformation. Recent advances in large language models have enabled high-quality text detoxification, though they require careful prompt engineering and fine-tuning for domain-specific applications.

Ostheimer et al. [29] provide evaluation guidelines for text style transfer systems, emphasizing content preservation (semantic similarity), style accuracy (toxicity reduction), fluency (grammatical acceptability), and human evaluation. The J-Score, a harmonic mean of these metrics, provides a comprehensive performance measure.

2.4 Repercussions of Toxicity on OSS Projects

The consequences of toxic interactions in OSS development extend far beyond momentary unpleasantness, creating a cascade of negative effects that damage both individuals and communities. Research demonstrates that toxicity fundamentally undermines developers' mental health [3], serving as a primary driver of significant stress and burnout [34]. These individual struggles inevitably degrade the collective workflow; phenomena such as unjustified pushback and destructive criticism have been shown to diminish productivity and severely strain interpersonal relationships [6, 27]. Most critically, these behaviors do not affect all contributors equally. Toxicity disproportionately harms minority developers, acting as a systemic barrier that obstructs Diversity, Equity, and Inclusion (DEI) efforts [14].

2.5 Comparison with Existing Tools

Table 2.1 provides a feature comparison highlighting ToxiShield's comprehensive capabilities relative to existing tools in the literature.

Table 2.1: Feature comparison of toxicity detection tools. MC: Multi-Class classification, DT: Detoxification. Checkmarks (✓) indicate supported features; crosses (–) indicate unsupported features.

Tool	Binary	MC	DT	Explanation	Real-Time
Raman et al. [34]	✓	–	–	–	–
ToxiCR [40]	✓	–	–	–	–
ToxiSpanSE [37]	✓	–	–	–	–
Ferreira et al. [11]	–	–	–	–	–
Rahman et al. [33]	✓	–	✓	–	–
ToxiShield (this work)	✓	✓	✓	✓	✓

ToxiShield distinguishes itself from existing toxicity detection approaches through three fundamental advancements. First, unlike standard binary classifiers, our framework introduces fine-grained toxicity analysis through multi-class classification. Second, ToxiShield incorporates an explanatory mechanism that articulates why a comment is problematic, fostering user understanding and behavioral change. Finally, this work represents the first deployment and empirical evaluation of a real-time toxicity mitigation tool integrated directly into the developer workflow.

2.6 Research Gaps

Existing systems are predominantly reactive, lack educational feedback, and face integration challenges. Key gaps include: (1) real-time proactive intervention similar to grammar checkers, (2) integrated educational feedback explaining why content is problematic, (3) domain-specific detoxification preserving technical accuracy, (4) seamless workflow integration into developer platforms, and (5) comprehensive evaluation combining automated metrics with user acceptance studies.

2.7 Summary

The literature reveals growing recognition of toxicity as a significant challenge in software engineering, with substantial research into detection techniques. However, existing approaches are predominantly reactive and lack real-time, educational, and integrated capabilities needed for effective toxicity mitigation. This thesis addresses these gaps through ToxiShield's three-module architecture: **Module 1 (Toxicity Filter)** tackles gap (1) through real-time binary detection;

Module 2 (Communication Coach) addresses gaps (2) and (3) through fine-grained multi-class classification with explanatory feedback; and **Module 3 (The Reframer)** addresses gap (3) through domain-specific text detoxification. Gaps (4) and (5) are addressed through the browser extension integration and comprehensive evaluation. The next chapter formalizes the research problem and objectives to address these gaps.

Chapter 3

Problem Statement and Research Objectives

This chapter identifies the research gap and defines the problem addressed by ToxiShield. Building on the background presented in previous chapters, we formalize the need for proactive, real-time toxicity detection and mitigation tools in software engineering environments.

3.1 The Problem: Reactive Approaches to Toxicity

Existing approaches for detecting toxic interactions in OSS communities are primarily *reactive*, operating on a post-hoc basis where classifiers flag problematic texts for later review and removal [40]. This delay means that by the time an intervention occurs, significant harm may have already been done to contributor relationships [10, 49].

The reactive approach suffers from fundamental limitations: the temporal gap between publication and intervention allows harm to occur, authors receive little guidance on why their communication was problematic, interventions provide binary outcomes without nuanced feedback, and workflow disruption occurs outside natural development processes.

3.2 The Need for Proactive Intervention

To foster healthier communication, a shift from reactive flagging to proactive prevention is necessary. Inspired by real-time grammar checkers in platforms like Gmail, we aim to develop a tool integrated into developers' communication channels. This tool would not only flag potentially toxic text as it is written but also provide actionable feedback by explaining why the text is problematic and suggesting neutral or civil alternatives.

Recent work by Rahman et al. [33] developed a customized T5 model to rephrase uncivil comments into constructive alternatives. However, their solution has two key shortcomings. First,

it lacks real-time integration, necessary for proactive moderation. Second, and more importantly, it fails to explain its reasoning. This explanatory feature is essential for fostering long-term behavioral change and navigating the cultural nuances of what is considered toxic [20].

3.3 Research Objectives

This thesis addresses the identified gap by developing ToxiShield, a browser extension for GitHub pull requests built on a three-module architecture: **Module 1 (Toxicity Filter)** identifies whether a comment is toxic; **Module 2 (Communication Coach)** classifies the specific type of toxicity and explains why it is harmful; and **Module 3 (The Reframer)** generates a constructive, professional alternative preserving the original technical intent. Together, these modules shift toxicity management from reactive post-hoc flagging to proactive, educational, real-time intervention. The primary research objectives are:

1. **Module 1 (Toxicity Filter) – Develop a high-accuracy binary toxicity detection system** that can reliably identify toxic comments in code review contexts with minimal false positives and negatives, suitable for real-time deployment.
2. **Module 2 (Communication Coach) – Create a fine-grained toxicity classification system** that can categorize toxic comments into specific subcategories and provide targeted explanations for the classifications to foster long-term behavioral change.
3. **Module 3 (The Reframer) – Design and implement a text detoxification module** that can automatically rewrite toxic comments into constructive alternatives while preserving technical content and intent.
4. **Integration – Build a real-time browser extension** that seamlessly incorporates all three modules into existing developer workflows, specifically GitHub pull request reviews, with an on-device architecture to preserve user privacy.
5. **Evaluation – Assess the system's effectiveness** through comprehensive automated metrics and user acceptance studies (Technology Acceptance Model with professional software developers).

3.4 Research Questions

To achieve these objectives, this thesis addresses five research questions:

1. **RQ1:** How can machine learning models be optimally designed and trained to detect toxicity in software engineering communication with high accuracy?

2. **RQ2:** What is the most effective approach for providing fine-grained toxicity classification with explanatory feedback in code review contexts?
3. **RQ3:** How can large language models be effectively fine-tuned to detoxify harmful text while preserving technical content and semantic intent?
4. **RQ4:** What are the technical and usability requirements for real-time integration of toxicity detection and mitigation tools in developer workflows?
5. **RQ5:** How do software developers perceive and accept automated toxicity detection and mitigation tools in their daily work practices?

3.5 Summary

This chapter identified the research gap: existing reactive approaches to toxicity management allow harm to occur before intervention. The chapter established the need for proactive, real-time tools with explanatory feedback and outlined five research objectives and questions to address this gap. The next chapter presents the system design and architecture of ToxiShield.

Chapter 4

System Design and Architecture

This chapter presents the design and architecture of ToxiShield, a comprehensive framework for real-time toxicity detection and mitigation in code review environments. We detail the system's overall architecture, design principles, and the **three-module framework** comprising: **Module 1 (Toxicity Filter)**, **Module 2 (Communication Coach)**, and **Module 3 (The Reframer)**. Each module is designed and evaluated to address the research challenges identified in previous chapters.

4.1 System Overview

ToxiShield is designed as a proactive intervention system that operates in real-time during code review composition. Unlike existing reactive approaches that flag content after publication, ToxiShield integrates directly into the developer's workflow to identify and address toxicity as it is being written.

4.1.1 Design Principles

The system architecture is guided by six core principles: (1) proactive intervention preventing toxicity before publication, (2) educational feedback explaining problematic content, (3) seamless integration with existing workflows, (4) real-time performance optimization, (5) privacy preservation through on-device processing, and (6) explainable AI providing transparent decision-making.

4.1.2 High-Level Architecture

ToxiShield employs a **three-module architecture** (Figure 4.2) processing comments through: **Module 1 (Toxicity Filter)** – binary toxicity detection; **Module 2 (Communication Coach)**

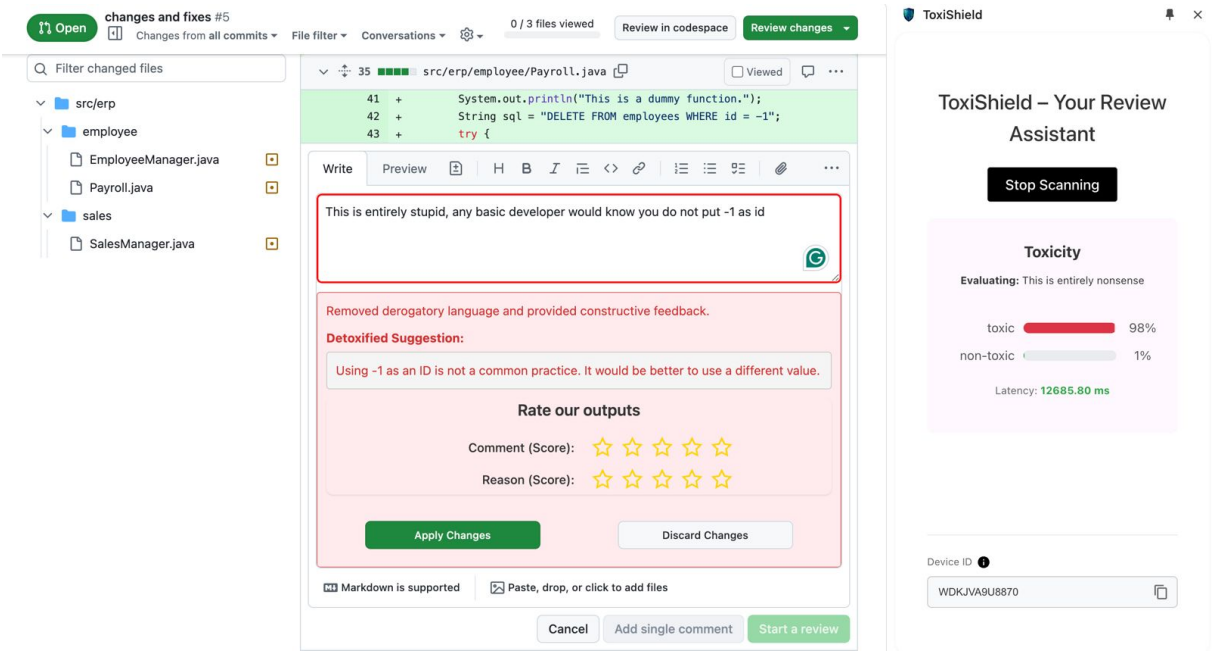


Figure 4.1: High-level conceptual overview of ToxiShield's intervention paradigm. The system positions itself between the moment a developer composes a code review comment and the moment it is submitted—a design choice that enables proactive prevention rather than reactive moderation. The three-module pipeline analyses and optionally transforms the draft comment before it enters the review thread.

– multi-class classification with explanatory feedback; and **Module 3 (The Reframer)** – text detoxification and constructive alternative generation. When a comment is not flagged as toxic by Module 1, it bypasses downstream processing, ensuring computational efficiency.

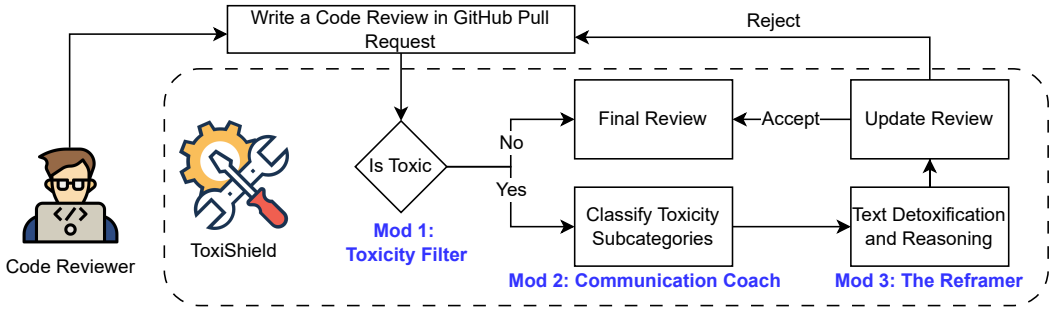


Figure 4.2: Motivational Workflow of ToxiShield (Figure 1 from FSE 2026 paper). The workflow shows how ToxiShield proactively identifies and mitigates toxicity in three stages: (1) binary toxicity detection, (2) classification into toxicity subcategories with explanations, and (3) text detoxification generating constructive alternatives.

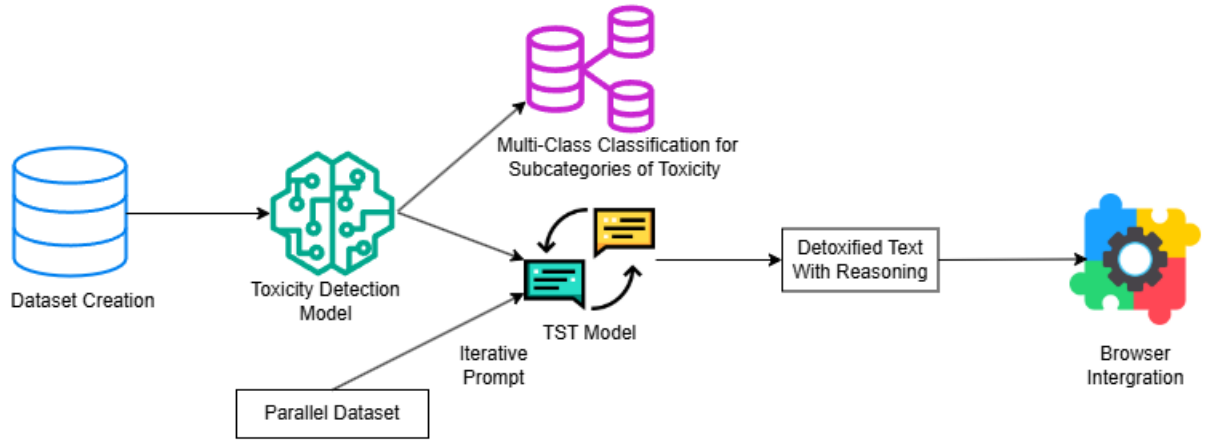


Figure 4.3: Detailed system architecture diagram of ToxiShield showing the component relationships between the browser extension frontend, the ONNX-quantized BERT binary classifier (Module 1), the LLM-based Communication Coach (Module 2), and the Llama 3.2 3B detoxification engine (Module 3). The cascade architecture ensures non-toxic comments bypass the computationally expensive LLM stages.

4.2 Module 1: Toxicity Filter

The Toxicity Filter serves as the system's entry point, performing efficient binary classification to identify comments that require further analysis and intervention.

4.2.1 Design Rationale

The binary filter prioritizes computational efficiency and high recall, optimizing resource usage by filtering non-toxic content early and enabling deployment in resource-constrained environments.

4.2.2 Architecture Components

The Toxicity Filter architecture comprises:

Input Processing Layer

- **Text Preprocessing:** Standardization, tokenization, and normalization of input comments
- **Feature Extraction:** Conversion of text to numerical representations suitable for classification
- **Context Integration:** Incorporation of metadata such as author history and project context

Classification Layer

- **Model Architecture:** BERT-based transformer model fine-tuned for toxicity detection

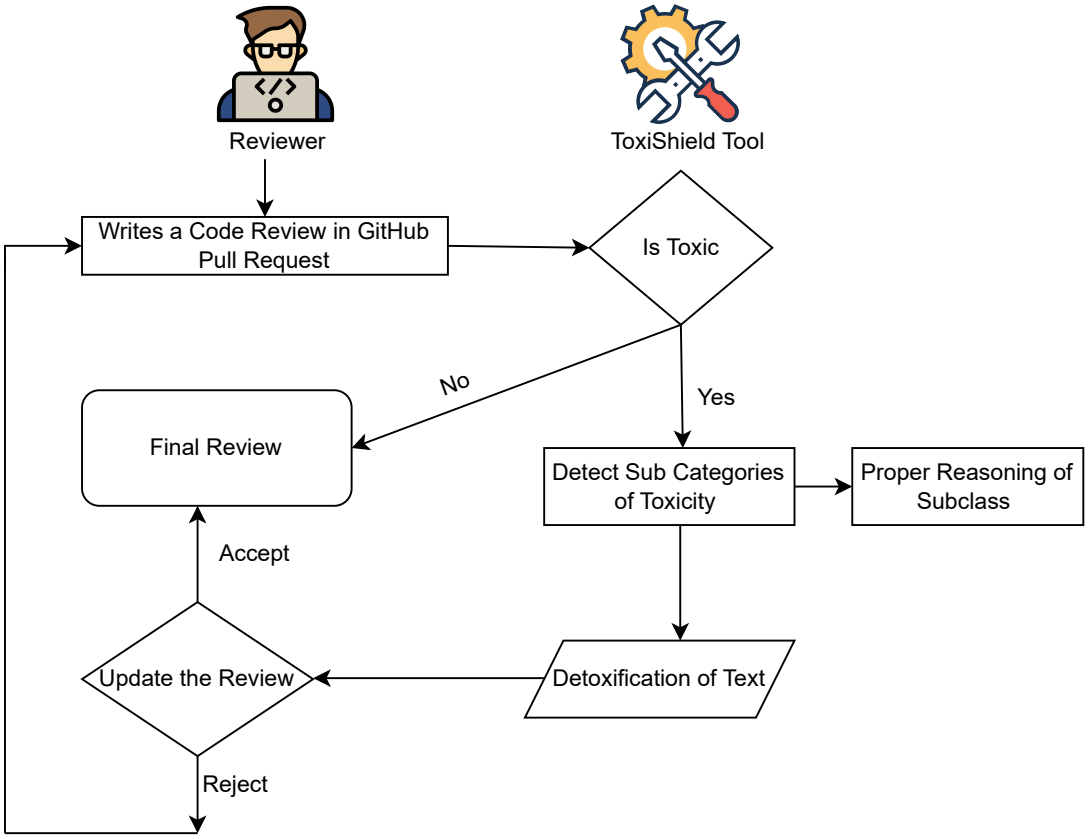


Figure 4.4: ToxiShield system flow diagram illustrating the decision logic and data flow between the three modules. A submitted comment enters Module 1 for binary screening; if flagged as toxic, it proceeds to Module 2 for subcategory classification and explanation generation, then to Module 3 for detoxified alternative generation. At each stage the developer is shown intermediate results and retains control over the final submission.

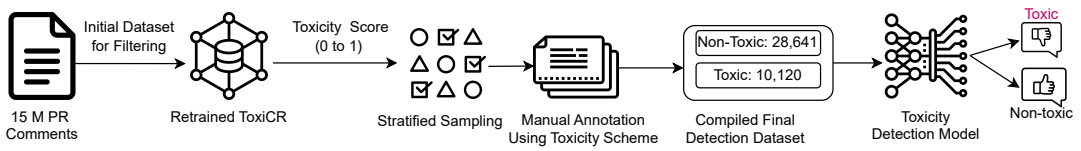


Figure 4.5: Workflow of the Toxicity Filter module (Figure 2 from FSE 2026 paper). A Masked Language Model (BERT-base-uncased) is trained to classify comments as toxic or non-toxic, with the dataset comprising 38,761 code review samples (10,120 toxic and 28,641 non-toxic).

- **Binary Decision:** Classification into toxic (requiring intervention) or non-toxic (no action needed)
- **Confidence Scoring:** Probability estimation for downstream decision making

Routing Layer

- **Threshold Application:** Configurable thresholds for determining intervention necessity
- **Workflow Triggering:** Automatic activation of downstream modules for toxic content
- **Logging and Analytics:** Collection of metrics for system performance monitoring

4.2.3 Performance Requirements

The Toxicity Filter must achieve $< 500ms$ latency, $\geq 95\%$ accuracy, support ≥ 1000 concurrent classifications, and maintain $< 100MB$ memory usage for browser deployment.

4.3 Module 2: Communication Coach

The Communication Coach provides educational feedback by classifying toxic content into specific subcategories and generating explanations for why the content is problematic.

4.3.1 Educational Framework

The Communication Coach provides immediate, real-time feedback with specific categorization, explanatory reasoning, and constructive guidance to help users understand and improve their communication.

4.3.2 Multi-Class Classification System

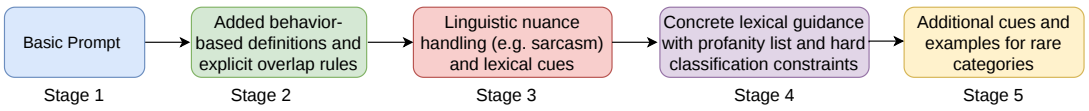


Figure 4.6: Communication Coach: Five-stage iterative prompt evolution for multi-class toxicity classification (Figure 3 from FSE 2026 paper). The prompt is progressively refined from a baseline zero-shot approach through operationalized definitions, linguistic nuance handling, lexical constraints, and rare category refinement. Stage 4 achieves the best Macro MCC and serves as the final prompt configuration used to evaluate all LLM families.

The classification system handles 11 distinct toxicity categories defined by Sarker et al. [39] (detailed in Chapter 2).

Classification Architecture

- **Prompt Engineering:** Carefully designed prompts for large language models
- **Chain-of-Thought Reasoning:** Structured reasoning process for classification decisions
- **Multi-Label Support:** Ability to assign multiple categories when appropriate
- **Explanation Generation:** Automatic generation of human-readable explanations

4.3.3 Prompt Design Framework

The Communication Coach employs a sophisticated prompt engineering framework:

Prompt Components

1. **Role Clarification:** Establishing the AI's role as a code review moderator
2. **Task Definition:** Clear specification of classification objectives
3. **Category Definitions:** Detailed descriptions of each toxicity category
4. **Examples:** Representative examples for each category
5. **Reasoning Instructions:** Guidelines for structured decision-making
6. **Output Format:** Standardized format for consistent parsing

Optimization Strategies

- **Iterative Refinement:** Multiple prompt versions tested and optimized
- **Validation Metrics:** Matthews Correlation Coefficient for optimization guidance
- **Conflict Resolution:** Rules for handling overlapping categories
- **Consistency Measures:** Techniques to ensure reproducible classifications

4.4 Module 3: The Reframer

The Reframer provides constructive alternatives by automatically rewriting toxic content while preserving technical meaning and intent.

4.4.1 Text Style Transfer Framework

The Reframer addresses the text style transfer problem of transforming toxic content to neutral alternatives:

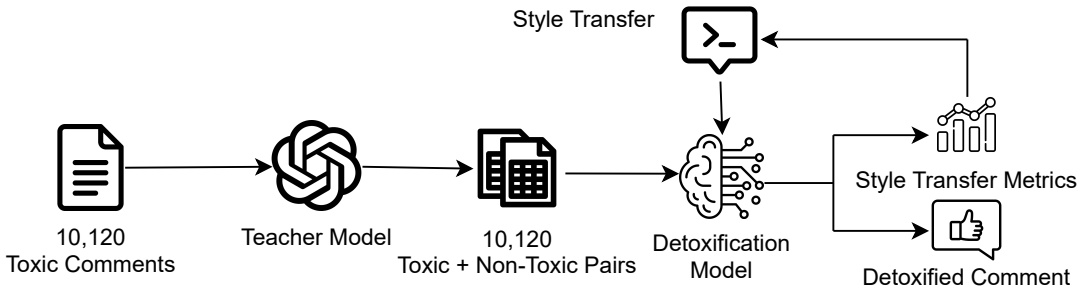


Figure 4.7: The Reframer workflow (Figure 4 from FSE 2026 paper). A teacher model is used to create a parallel dataset from 10,120 toxic comments, which is then used to train a student detoxification model with iterative prompting. The workflow includes style transfer metrics for evaluation.

Design Objectives

The Reframer optimizes multiple, sometimes competing objectives:

$$\begin{aligned} \text{Quality}(x, y) = &\alpha \cdot \text{StyleAccuracy}(y) && (4.1)\\ &+ \beta \cdot \text{ContentPreservation}(x, y) && (4.2)\\ &+ \gamma \cdot \text{Fluency}(y) && (4.3) \end{aligned}$$

where x is the original toxic comment, y is the detoxified version, and α, β, γ are weighting parameters.

Architecture Components

- **Teacher Model Selection:** Evaluation of multiple large language models for dataset generation
- **Student Model Training:** Fine-tuning of lightweight models for deployment
- **Quality Assessment:** Multi-dimensional evaluation of generated alternatives
- **Optimization Pipeline:** Iterative improvement of model performance

4.4.2 Teacher-Student Architecture

The Reframer employs a teacher-student training paradigm:

Teacher Model Phase

1. **Model Selection:** Comparative evaluation of state-of-the-art language models
2. **Prompt Optimization:** Development of prompts for high-quality detoxification
3. **Dataset Generation:** Creation of parallel toxic/non-toxic comment pairs
4. **Quality Filtering:** Selection of highest-quality generated pairs

Student Model Phase

1. **Architecture Selection:** Choice of lightweight models for real-time deployment
2. **Fine-Tuning Process:** Training on teacher-generated parallel data
3. **Parameter Optimization:** LoRA (Low-Rank Adaptation) for efficient training
4. **Performance Validation:** Comprehensive evaluation against quality metrics

4.4.3 Evaluation Metrics Framework

The Reframer employs a comprehensive evaluation framework:

Automatic Metrics

- **Style Accuracy:** Measured using toxicity classifiers on generated text
- **Content Preservation:** Semantic similarity using paraphrastic embeddings
- **Fluency:** Grammatical acceptability using pretrained CoLA classifiers
- **J-Score:** Harmonic mean of the above metrics for comprehensive assessment

Human Evaluation

- **Quality Assessment:** Expert evaluation of generated alternatives
- **Contextual Appropriateness:** Suitability for code review environments
- **Preservation of Intent:** Maintenance of original technical meaning
- **Naturalness:** Human perception of text quality and fluency

4.5 System Integration Architecture

4.5.1 Component Interaction Design

The three modules work in concert through a carefully designed interaction protocol:

Algorithm 1 ToxiShield Processing Pipeline

Require: Comment text C, user context U, system configuration S
Ensure: Intervention decision and feedback
1: $toxicity_score \leftarrow$ ToxicityFilter(C)
2: **if** $toxicity_score < S.threshold$ **then**
3: **return** NoIntervention
4: **end if**
5: $categories, explanation \leftarrow$ CommunicationCoach(C)
6: $alternatives \leftarrow$ Reframer(C)
7: $feedback \leftarrow$ FormatFeedback$(categories, explanation, alternatives)$
8: **return** $feedback$

Data Flow Architecture

- **Sequential Processing:** Modules process comments in a defined order
- **Early Termination:** Non-toxic content bypasses downstream processing
- **Context Propagation:** Relevant information flows between modules
- **Error Handling:** Graceful degradation when individual modules fail

Performance Optimization

- **Caching:** Intelligent caching of frequent classifications and transformations
- **Batching:** Efficient processing of multiple comments simultaneously
- **Resource Management:** Dynamic allocation based on system load
- **Monitoring:** Real-time performance tracking and optimization

4.5.2 Deployment Architecture

ToxiShield deploys as a browser extension with client-side processing using ONNX runtime for cross-platform model execution, model quantization for performance, and progressive loading. For server-side deployment, it uses RESTful APIs with load balancing and auto-scaling capabilities.

4.6 Design Trade-offs and Decisions

4.6.1 Design Trade-offs

Key design decisions balance performance and accuracy through BERT-base model selection for real-time performance, cascade architecture for computational efficiency, adjustable thresholds, and caching strategies. Privacy is prioritized through on-device processing, data minimization, and user control. The design balances domain specialization for code review contexts with extensibility to other domains through modular architecture and transfer learning.

4.7 Summary

This chapter presented the design and architecture of ToxiShield's three-layer framework for real-time toxicity detection and mitigation. The modular architecture enables independent component optimization while maintaining effective coordination through proactive intervention, educational feedback, and seamless workflow integration.

Chapter 5

Implementation

This chapter presents the implementation details for the three core modules of ToxiShield: Module 1 (Toxicity Filter), Module 2 (Communication Coach), and Module 3 (The Reframer).

5.1 Module 1: Toxicity Filter

Module 1 (Toxicity Filter) serves as the primary detection layer for identifying toxic content in code review comments.

5.1.1 Dataset Preparation

To compile a more diverse toxicity dataset, we leveraged pull request comments from a recent study where 2,828 projects with 101 million GitHub pull request comments were mined. We limited our selection to the first 1,000 projects containing 15 million PR comments.

Since toxic comments are relatively rare in open-source discussions, an automated classifier is necessary to label large-scale datasets efficiently. We retrained the ToxiCR model using a combined dataset that includes both the Sarker et al. dataset (19,651 samples) and the Rahman et al. dataset (4,700 samples). We then applied the retrained ToxiCR model to the 15M PR comments, where ToxiCR provides the output probability (0–1) for each comment. In accordance with the original ToxiCR framework, a text is classified as toxic if its score is ≥ 0.5. However, since ToxiCR has false negatives for underrepresented toxicity classes, we broadened our sample selection to 0.1.

To construct our new dataset, we implemented a multi-stage curation process involving both model-based stratification and manual annotation. We stratified the data into five distinct bins based on the toxicity probability (p) assigned by our retrained ToxiCR model: (1) $0.10 \leq p < 0.28$, (2) $0.28 \leq p < 0.46$, (3) $0.46 \leq p < 0.64$, (4) $0.64 \leq p < 0.82$, (5) $0.82 \leq p \leq 1.0$. From each bin, an annotator manually selected 2,100 samples they identified

as toxic. A second annotator then independently reviewed these selections and removed 380 samples that they did not consider toxic, resulting in a final set of 10,120 toxic samples.

For non-toxic samples, we randomly selected 30,000 samples from the pool of data that the first annotator had reviewed but did not select as toxic. We then applied a text-based filter that removed 1,359 samples containing profane words, yielding a final set of 28,641 non-toxic samples. This process produced our final dataset of 38,761 samples.

5.1.2 Model Training

To identify the best model for ToxiShield's Toxicity Filter module, we selected four models to train on our large-scale dataset. Our approach leveraged both masked language models (MLMs) and large language models (LLMs), specifically adapted for software development communication. We fine-tuned BERT-base-uncased and Xtreme-DistilBERT [64] on our dataset and incorporated ToxiCR's pre-trained model for toxicity detection. Additionally, we utilized GPT-4o and GPT-3.5 from Rahman et al. as LLM-based detectors.

We divided our code review dataset into an 8:1:1 ratio for training (80%), validation (10%), and testing (10%). We initialized both BERT-base-uncased and Xtreme-DistilBERT with their publicly available pre-trained weights, adding a task-specific classification head to each. We tokenized inputs using a maximum sequence length of 128 tokens. We fine-tuned the models for up to 20 epochs with a learning rate of 2×10^{-5}, a batch size of 128 (achieved via gradient accumulation across 8 steps), a weight decay of 0.01, and a warmup ratio of 0.1, optimizing binary cross-entropy loss. Early stopping with patience=3 was applied to prevent overfitting; in practice the best checkpoint was reached at epoch 12, which is the result reported throughout this thesis. We applied 10-fold stratified cross-validation to the training fold, ensuring class balance in each split. After each epoch, we monitored the validation toxic-class F1-score and retained the checkpoint with the highest score for final evaluation.

5.1.3 Model Evaluation

We evaluated the candidate models using classification metrics including precision, recall, accuracy, and F1 score. The fine-tuned BERT-base-uncased achieved the highest precision, recall, and a toxic-class F1 of 0.97, followed closely by Xtreme-DistilBERT (F1=0.95). These results demonstrate that BERT-base-uncased is particularly robust in detecting toxic comments. We therefore adopted it as the default model for Module 1 (Toxicity Filter) in ToxiShield. A detailed comparison against baselines is presented in Chapter 7.

5.2 Module 2: Communication Coach

Module 2 (Communication Coach) is an automated educational module designed to foster long-term behavioral change by providing constructive feedback on harmful communication.

5.2.1 Dataset for Toxicity Subcategories

Prior studies in the SE domain provided and labeled various subcategories of toxicity. To the best of our knowledge, a recent empirical study of toxicity on GitHub pull requests by Sarker et al. [39] offers the most comprehensive categorization, identifying 11 distinct subcategories of toxicity observed on GitHub. Table 5.1 presents the formal definitions adopted for this study, which also serve as the definitional anchor for the prompt engineering framework in Module 2.

Table 5.1: The 12 toxicity categories used in Module 2 (Communication Coach), including the Non-Toxic class. Definitions derived from Sarker et al. [39] and operationalised in the prompt schema.

Category	Definition
Non-Toxic	Comment does not exhibit any of the following toxicity types
Profanity	Cursing, expletives, or swearing directed at people or code
Trolling	Inflammatory, insincere, or off-topic messages intended to provoke
Insult	Disrespect directed toward another person or their work
Self-Deprecation	Self-critical remarks containing profanities or hostile self-directed language
Others-Toxic	Toxic content that does not fit any of the above categories
Entitlement	Demands or expectations asserted beyond reasonable professional norms
Identity Attack	Targeting a person based on race, religion, gender, or nationality
Threats	Aggressive or threatening behaviour toward another contributor
Obscenity	Implicit or explicit sexual references in an unwanted context
Flirtation	Unwanted romantic or sexual attention directed at a contributor
Arrogance	Condescension, dismissiveness, or imposed superiority over others

Due to the limited availability of datasets annotated with such fine-grained toxicity labels, we leveraged that dataset, which consists of 600 samples with 532 toxic instances across the 11 subcategories of toxicity. To create a balanced dataset, we supplemented it with 600 non-toxic comments sampled from the non-toxic portion of our dataset, resulting in a combined dataset suitable for our subcategory classification task. Our final dataset for multi-class classification consists of a total of 1,200 code review comments in 12 categories, including non-toxic samples. Of the 532 toxic samples, 141 instances carry two distinct toxicity subcategory labels, and three

instances carry three labels, consistent with the primary labeling schema of sarker2025landscape. We preserve these overlapping annotations, formulating the problem as a multi-label, multi-class classification task [65].

5.2.2 Prompt Engineering Strategy

Prompt engineering is a crucial technique for enhancing LLM performance in complex tasks. Well-crafted prompts significantly improve LLMs' effectiveness in domain-specific applications. In ToxiShield, we applied prompts in two settings: (i) Communication Coach, where the model assigns a subcategory of toxicity, and (ii) The Reframer, where the model rewrites a toxic comment into a professional alternative. We developed a unified prompting framework based on three principles: structured reasoning, schema-constrained outputs, and iterative validation against defined metrics.

We adopted Chain-of-Thought (CoT) prompting [47] to break down classification and detoxification into structured steps. For classification, CoT helps the model justify why a comment fits a specific subcategory. To ensure consistent behavior across models and comments, we standardized the multi-class prompt into six components: (1) Role Clarification, (2) Task Assignment, (3) Category Definitions & Examples, (4) Heuristic Lexicons (introduced in Stage 4 of the iterative prompt evolution process), (5) Classification Guidelines, and (6) Output Format Specification using strict XML schema.

We conducted iterative prompt development, testing multiple templates to determine the phrasing and structure that yield the most accurate multi-label assignments. For Module 2's multi-class classifier, iterative adjustments were guided by the Macro Matthews Correlation Coefficient (Macro MCC) on the validation set. Templates specifying class definitions, structured reasoning steps, and example instances were compared, and the highest Macro MCC variant was selected.

5.2.3 Iterative Prompt Refinement: Detailed Sub-Iteration Analysis

During our prompt engineering process, we conducted 14 distinct iterations organized into four major refinement stages, each with multiple sub-iterations. While the paper presents five stages for clarity, the full iterative process involved:

- **Stage 1 (Baseline)**: A single baseline prompt establishing the zero-shot starting point.
- **Stage 2 (Observable Cues)**: Three sub-iterations (Stages 2.1–2.3) progressively refining behavior-based definitions, adding few-shot examples, and imposing output constraints.
- **Stage 3 (Subtle Nuance)**: Three sub-iterations (Stages 3.1–3.3) introducing sarcasm detection, lexical sentiment markers, and extended profanity slang coverage.

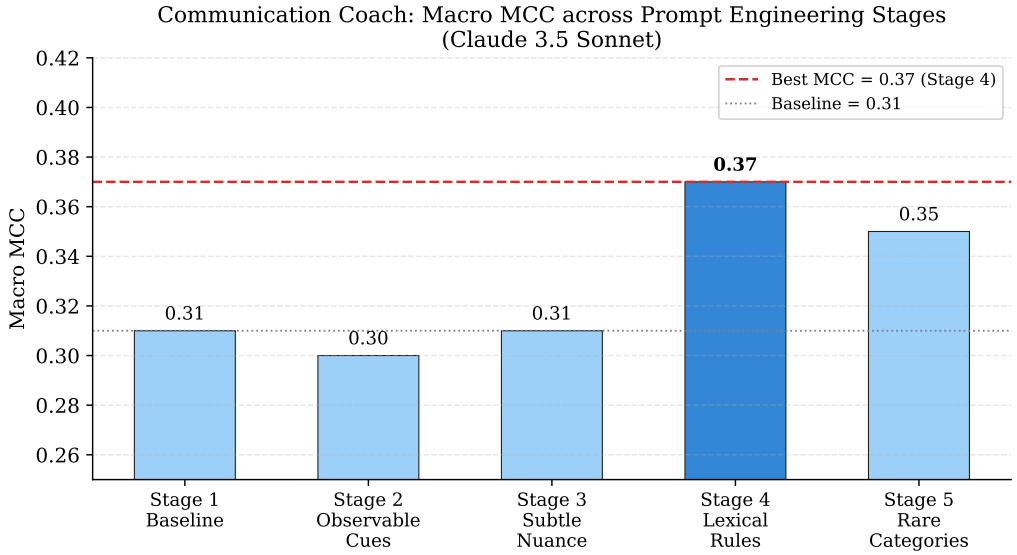


Figure 5.1: Iterative prompt evolution for the Communication Coach module, showing Macro MCC scores across 5 stages with GPT-4o. Stage 4 (lexical rules and constraints) achieves the best MCC of 0.37, which was used as the final prompt configuration for evaluating all LLM families.

- **Stage 4 (Lexical Constraints)**: Four sub-iterations (Stages 4.1–4.3) varying profanity word list granularity, logic rule specificity, and anger indicator patterns (e.g., ALL_CAPS, excessive punctuation).
- **Stage 5 (Rare Category Refinement)**: Final adjustments targeting Arrogance and Identity Attack categories.

The iterative refinement process demonstrates that effective prompt engineering for multi-label, multi-class toxic content classification requires systematic exploration of prompt components, not merely high-level structural changes.

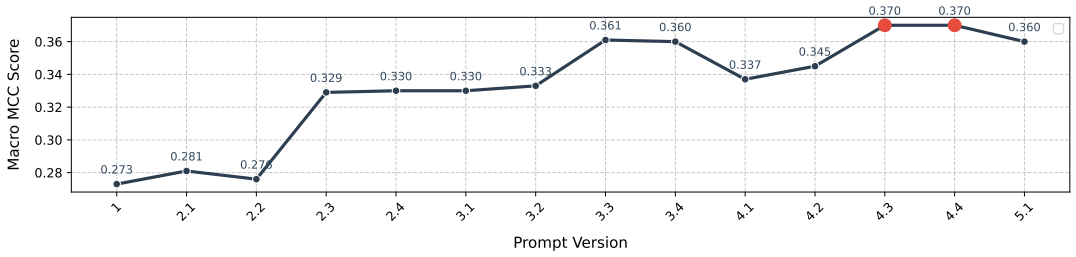


Figure 5.2: Iterative prompt engineering process for the Communication Coach module. The diagram illustrates how each iteration cycle improves the Macro MCC score through targeted refinements: adding behavioral cue definitions, incorporating few-shot examples, embedding lexical resources (profanity lists, anger indicators), and introducing logic constraints for mutual-exclusivity.

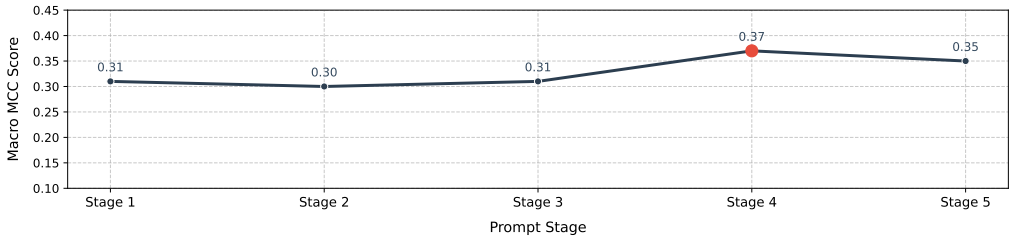


Figure 5.3: Multi-class prompt engineering performance chart across the five reported prompt evolution stages for the Communication Coach. Each bar represents the Macro MCC achieved by GPT-4o on the validation set. Stage 4 (Lexical Rules and Constraints) achieves the highest Macro MCC of 0.37, which was adopted as the final prompt configuration for cross-model evaluation.

5.2.4 Model Evaluation

We evaluated eight large language models on the multi-class classification task for toxic code review comments. Among them, Claude 3.5 Sonnet demonstrated the most balanced detection capability across the 12 categories, achieving the highest Macro F1 (0.42) alongside a Macro MCC of 0.39. GPT-4o performed competitively (Macro F1: 0.40, Macro MCC: 0.37), while Llama 3.3 70B exhibited lower efficacy (Macro F1: 0.37, Macro MCC: 0.34). Claude 3.5 Sonnet provided superior balance across minority classes, whereas GPT-4o favoured exact match accuracy. These findings highlight considerable variability in how contemporary LLMs handle imbalanced, multi-label toxicity categorization. A comprehensive evaluation is presented in Chapter 7.

5.2.5 Per-Category Classification Performance

Preliminary analysis of the best-performing model (Claude 3.5 Sonnet with Stage 4 prompt) reveals substantial variation in per-category Macro MCC scores. Categories with explicit linguistic markers—such as Profanity and Non-Toxic—achieve strong performance, whereas behaviorally-defined categories such as Arrogance and Entitlement prove significantly more challenging. A comprehensive per-category analysis, including detailed MCC breakdowns and their implications for future dataset development, is presented in Section 7.2.3 of Chapter 7.

5.3 Module 3: The Reframer

Module 3 (The Reframer) performs text detoxification, transforming toxic comments into constructive alternatives while preserving technical content and intent.

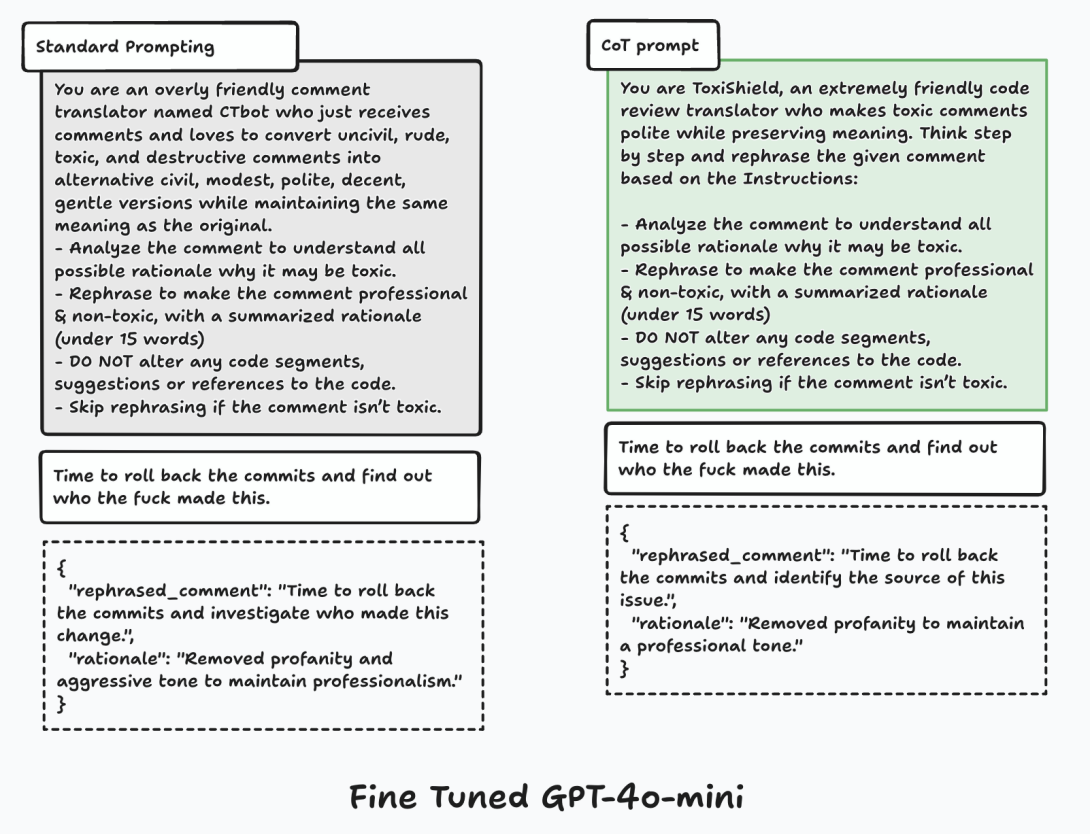


Figure 5.4: Chain-of-Thought (CoT) prompting strategy used during the Reframer's teacher-model data generation phase. The teacher LLM is guided to first analyse the toxic elements in the comment step by step, then reconstruct the sentence with the toxic signal removed while preserving the technical core. This structured reasoning produces higher-quality parallel training data compared to direct rewriting without explicit reasoning steps.

5.3.1 Teacher-Student Framework

Detoxifying code-review comments is a text style transfer (TST) problem: we want to alter the style (toxic → respectful) while preserving the content (technical intent). Effective TST and subsequent knowledge distillation [16] require a parallel corpus that pairs each toxic comment with a faithful, non-toxic rewrite.

We begin with the curated set of 10,120 toxic comments assembled for training Module 1 (Toxicity Filter). In the first stage, we employ multiple state-of-the-art LLMs as teacher models. Using carefully designed prompts, each teacher LLM generates a detoxified version of every toxic comment while preserving technical content and semantic intent. This process produces multiple candidate parallel datasets, one per teacher model.

To determine which dataset would serve as the basis for training our downstream models, we evaluated the quality of each teacher-generated dataset using quantitative metrics. We sampled 500 detoxified pairs from each dataset and evaluated them using metrics for style accuracy, fluency, and content preservation. Among the candidate teacher models, GPT-4o-05-13 achieved the highest J-Score of 88.14%, indicating the best overall balance across metrics. We therefore selected the dataset generated by this model as the final parallel corpus for training the student models.

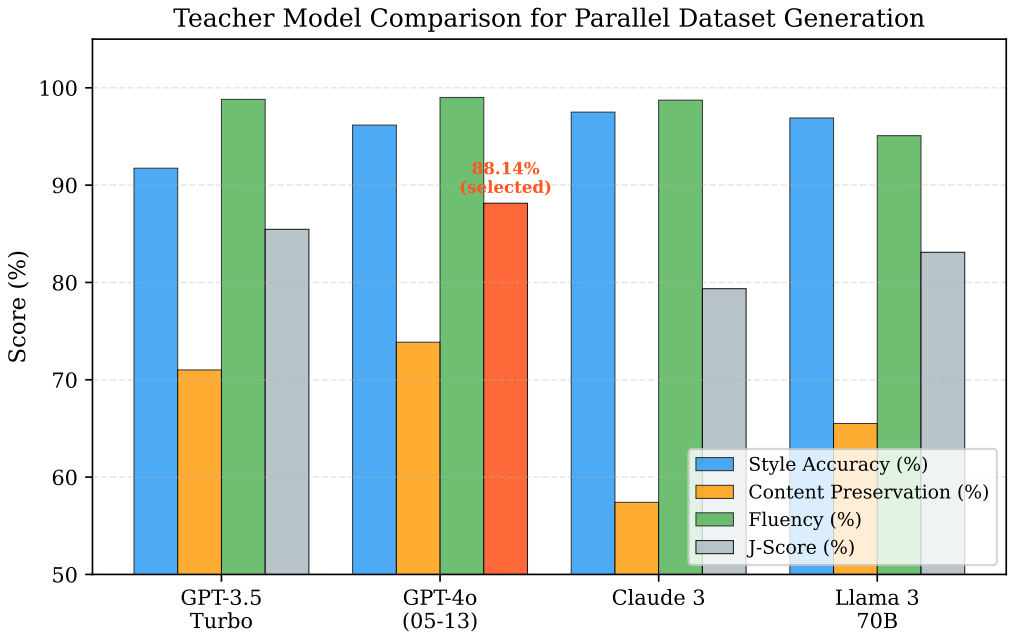


Figure 5.5: Teacher model comparison for parallel dataset generation. GPT-4o (2024-05-13) achieves the highest J-Score (88.14%), balancing strong style accuracy (96.17%), fluency (99.01%), and content preservation (73.86%).

5.3.2 Student Model Training

After identifying GPT-4o-05-13 as the best teacher model, we used its parallel dataset to fine-tune a lightweight open-source student LLM, which drives the detoxification workflow of ToxiShield. To reduce computational costs while retaining performance, we used LoRA (Low-Rank Adaptation) [17]. We fine-tuned the model for 10 epochs with a learning rate of 2×10^{-4}, weight decay of 0.01, and 5 warm-up steps. All fine-tuning procedures were performed on a cloud machine instance with a 24GB RTX 4090 GPU. We fixed generation parameters during all experiments (temperature=0.0, top-p=1.0, and maximum output length=256 tokens) to ensure output consistency.

5.3.3 Evaluation

We compared the performance of our detoxification models using the same evaluation metrics applied to the teacher models. Among the candidates, Llama3.2-3B achieved the highest overall J-Score (84.00%), balancing strong style accuracy (95.27%), fluency (97.03%), and content preservation (67.07%).

To validate our model performance, we manually validated misclassifications. The best performing model, Llama3.2-3B, produced 78 misclassifications out of 2,018 toxic comments in the test set. We classified these instances into four categories: Name Error (when ToxiCR flagged benign comments due to profane substrings within usernames), Context Error (toxic text as names of variables, files, or folders), General Error (instances where our model failed to remove toxicity), and ToxiCR Error (solely due to ToxiCR misclassifying a detoxified comment as toxic).

To quantify annotation reliability, two authors independently rated a random sample of 100 detoxified comments on a 1–5 Likert scale across three dimensions (Minimal Change, Context Preservation, Communication Style). We computed quadratic-weighted Cohen's κ for each dimension [21], obtaining $\kappa = 0.82$ for Minimal Change, $\kappa = 0.72$ for Context Preservation, and $\kappa = 0.77$ for Communication Style, indicating strong consistency in our manual detoxification evaluation.

5.4 Summary

This chapter presented the implementation details for all three modules of ToxiShield. Module 1 (Toxicity Filter) employs fine-tuned BERT-base-uncased, achieving 98% accuracy and 97% toxic-class F1. Module 2 (Communication Coach) employs Claude 3.5 Sonnet with iterative prompt engineering, achieving a Macro F1 of 42% for multi-class classification. Module 3 (The

Reframer) employs a teacher-student framework with Llama 3.2 3B, achieving an 84% J-Score. The next chapter describes the browser extension integration.

Chapter 6

Browser Extension and GitHub Integration

This chapter describes the ToxiShield browser extension that integrates real-time toxicity detection and mitigation into GitHub workflows.

6.1 Browser Extension Overview

We developed a browser extension (Figure 6.1) that integrates all three ToxiShield modules into the GitHub pull request interface for real-time toxicity detection and mitigation. The extension implements the complete three-module pipeline on-device:

- **Module 1 (Toxicity Filter):** A fine-tuned BERT-base-uncased model (exported to ONNX format with quantization) continuously monitors comment text as it is typed, performing initial toxicity screening with sub-500 ms latency.
- **Module 2 (Communication Coach):** When a comment is flagged as toxic, Module 2 classifies it into one or more of the 11 toxicity subcategories and generates a targeted explanation of why the content is problematic, directly in the browser.
- **Module 3 (The Reframer):** Implemented as a locally-hosted Llama 3.2 3B model, Module 3 generates a constructive alternative preserving the technical intent alongside a rationale for the changes.

This fully on-device, local-first architecture ensures user privacy by eliminating reliance on external APIs. All processing occurs client-side, and no code review content is transmitted to external services.

To facilitate deployment and integration into existing code review workflows, we exported the trained BERT-base-uncased model to ONNX format. This step enhances portability across different runtime environments. We further apply quantization techniques to the ONNX model,

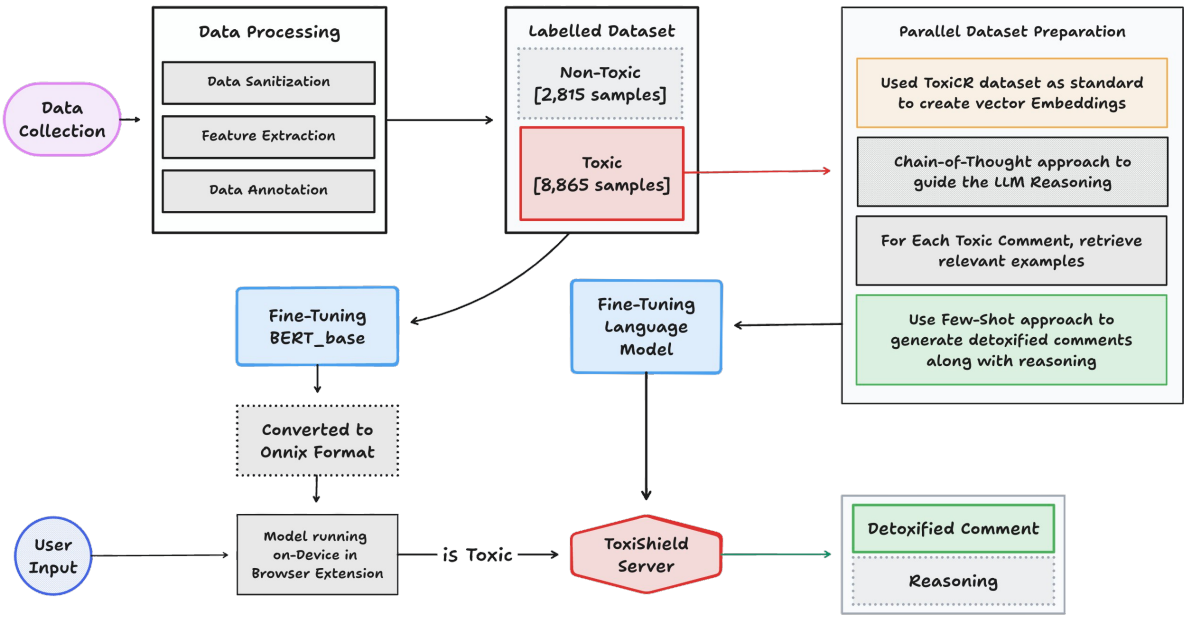


Figure 6.1: End-to-end system design of ToxiShield integrated with GitHub pull request reviews. The diagram illustrates how the three-module pipeline operates within the browser extension: Module 1 (Toxicity Filter) screens comments on the fly, Module 2 (Communication Coach) categorises the toxicity subcategory and generates an explanation, and Module 3 (The Reframer) produces a constructive alternative—all on-device without external API calls.



Figure 6.2: ToxiShield browser extension in action during a GitHub pull request code review. When the developer types a potentially toxic comment, the extension's real-time inference pipeline detects the toxicity, presents the flagged subcategory with an explanation, and offers a professionally reframed alternative—all within the GitHub interface without leaving the page.

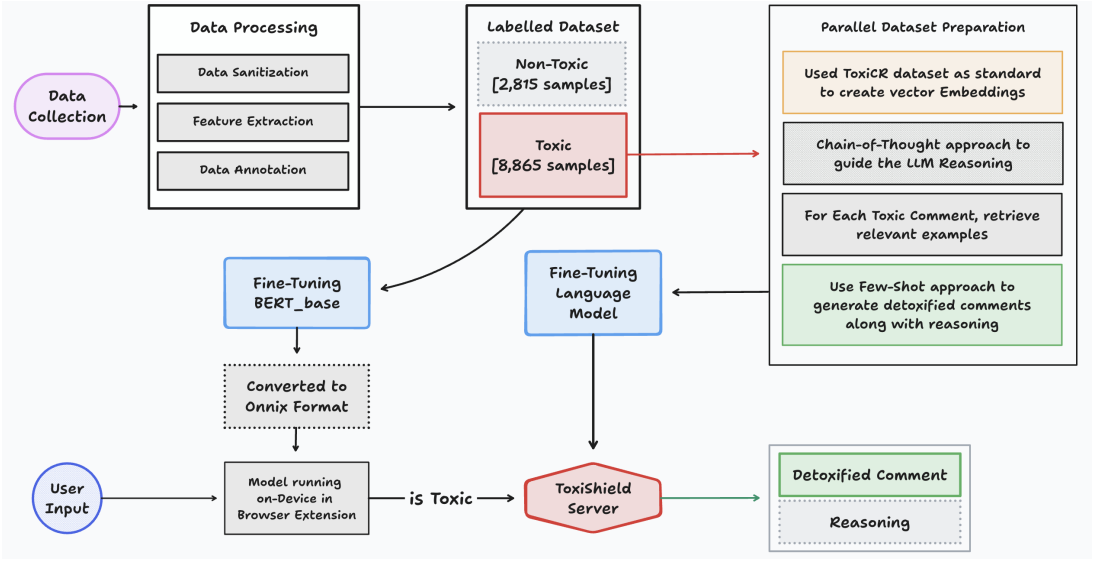


Figure 6.3: End-to-end view of ToxiShield operating within the GitHub pull request workflow. The annotated screenshot shows the three-stage feedback panel that appears when toxic content is detected: the toxicity label (Module 2 output), the natural-language explanation, and the suggested reframed alternative (Module 3 output). The developer retains full agency to accept, modify, or dismiss the suggestion.

significantly reducing its size and improving inference speed without substantial loss in accuracy. This optimisation is essential for real-time toxicity detection in code review systems, where low latency is a strict requirement.

6.2 Extension Architecture

The extension follows Manifest V3 architecture with content scripts for GitHub page interaction, a background service worker for API communication, popup and options pages for user configuration, and a storage system for preferences and caching. The manifest defines permissions (storage, activeTab, background), content script injection for GitHub pull request pages, and background service worker configuration.

6.3 GitHub Interface Integration

The extension analyses GitHub's pull request interface to identify comment input areas (new comments, inline reviews, code suggestions, and general discussion) for feedback display. GitHub's dynamic loading requires mutation observers to detect new comment fields, event delegation for dynamically created elements, and lazy initialization.

6.4 Real-time Analysis Integration

The extension implements debounced input monitoring to balance responsiveness with performance, following a progressive analysis approach: (1) fast screening with initial binary toxicity detection (<500 ms), (2) detailed analysis with fine-grained classification when toxic content is detected, and (3) suggestion generation with detoxified alternatives for confirmed toxic content.

When toxic content is detected, an expandable feedback panel appears providing the toxicity category, explanation, and suggested alternative. The interface design follows non-intrusive principles: minimal visual footprint, contextual placement, easy dismissal, and respect for user choice.

6.5 Technical Implementation Details

6.5.1 ONNX Export and Model Quantization

The BERT-base-uncased model powering Module 1 (Toxicity Filter) is exported to ONNX (Open Neural Network Exchange) format to achieve cross-platform portability and enable efficient in-browser inference without requiring a Python runtime or external model serving infrastructure. The ONNX export pipeline serializes the fine-tuned model weights and computational graph into a single portable binary, which is then loaded by the ONNX Runtime WebAssembly (WASM) backend bundled within the extension.

To meet browser memory and storage constraints, we apply post-training static quantization to the exported ONNX model. Quantization converts 32-bit floating-point weights to 8-bit integers, reducing the model binary from approximately 440 MB to approximately 110 MB—a 75% reduction in storage footprint. Inference benchmarks on representative developer hardware confirm that the quantized model retains more than 97% of the original fine-tuned model's toxic-class F1 (0.94 vs. 0.97), an acceptable accuracy trade-off given the substantial latency and storage gains. This quantization step is essential for shipping the model as a browser extension artifact without requiring users to download multi-hundred-megabyte assets on first install.

6.5.2 Two-Stage Inference Pipeline

The extension implements a two-stage inference pipeline designed to balance detection latency against the depth of analysis provided.

Stage 1 — Fast Binary Screening. As the developer types a comment, debounced input events trigger the first stage: inference through the quantized ONNX-BERT model running entirely in the browser via ONNX Runtime WASM. This stage classifies each comment as toxic or

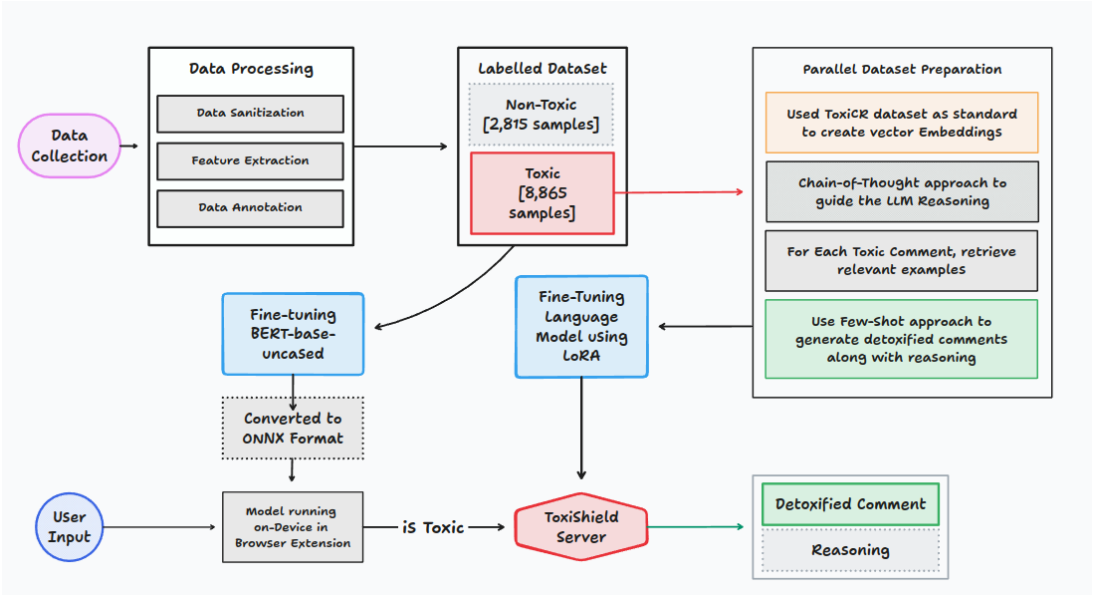


Figure 6.4: ToxiShield system architecture showing the on-device inference stack. The browser extension embeds the quantized ONNX BERT model for Stage 1 binary detection (sub-500 ms), while Stage 2 delegates to a locally-hosted Llama 3.2 3B model served by Ollama for fine-grained classification and detoxification. No comment content leaves the developer's machine.

non-toxic and completes in under 500 ms on typical developer hardware, enabling near-real-time feedback during composition without perceptible interruption to the typing workflow.

Stage 2 — LLM-Based Categorization and Reframing. Only when Stage 1 flags a comment as toxic does the extension invoke Stage 2. A locally-hosted Llama 3.2 3B model, served via an Ollama instance running on the developer's machine, receives the flagged comment and performs three operations: (1) fine-grained multi-class toxicity categorization into one or more of the 11 defined subcategories (Module 2); (2) generation of a targeted natural-language explanation of why the content is problematic; and (3) generation of a constructive, detoxified alternative that preserves the technical intent of the original comment (Module 3). This two-stage design avoids invoking the computationally heavier LLM for the majority of comments—those that are non-toxic—thereby keeping the average-case extension overhead negligible during benign writing sessions.

6.5.3 Privacy-Preserving On-Device Architecture

The extension is architected to operate entirely on-device, with no code review content transmitted to external APIs or cloud services at any point during normal operation. The ONNX-BERT model is loaded into browser memory during extension initialization from the locally bundled model artifact. The Llama 3.2 3B model is served from the developer's local machine via Ollama. Consequently, all comment text analyzed by ToxiShield remains on the developer's own hardware and is never exposed to third-party services. This design decision was explicitly prioritized to

address a foreseeable adoption barrier: professional developers working on proprietary codebases may be prohibited by organizational policy from routing code review content through external cloud APIs. The fully local architecture removes this barrier entirely, enabling deployment in enterprise environments without requiring policy exceptions or data-sharing agreements.

6.6 Participant Recruitment and Survey

Following Institutional Review Board (IRB) approval, we recruited 10 professional software developers from the United States and Bangladesh, a sample size consistent with established usability study guidelines [62, 63]. Participants attended a tutorial session where two authors demonstrated the installation and functionality of the browser extension. The study employed a two-week longitudinal design, during which participants integrated the tool into their daily workflows. To strictly prioritize user privacy, we deliberately refrained from collecting telemetry or usage logs. Regarding usage frequency, three participants reported using the tool 'frequently', five 'occasionally', and two 'rarely'.

6.7 Qualitative Evaluation Results

We included eight open-ended questions in our survey. Two authors conducted a thematic analysis of the open-ended responses following Braun and Clarke [61], identifying four primary themes.

Theme 1: Perceived Value and Usability. Participants responded positively to the tool's core concept and design. P7 described "Blocking and refining toxic comments in real time is an innovative idea". P3 highlighted the value of the mission, noting that "detoxification is needed for mental health of the developers which I liked most." Regarding usability, P2 found the tool "straightforward," and P6 described it as "easy to use and well built," validating the browser-based implementation.

Theme 2: Workflow Integration and Latency. Despite the good usability, performance was a major friction point. P3 noted that "scanning is slow," and P7 advised "latency needs to improve a lot" for the BERT classifier. P3 criticized the manual trigger, stating the tool should "detect pull-request or other pages on run-time" to be truly useful.

Theme 3: Nuance in Toxicity Detection. A key finding was the tool's struggle with subtle toxicity. P5 noted that "I have used sarcastic and frustrated tones which did not come up on the toxicity radar". P6 provided significant insight, arguing that toxicity is often behavioral, such as "asking for unnecessary huge changes" or "being very specific about code indentation." P6 also emphasized that these behaviors are "hard to catch" but can be just as harmful as harsh words.

Theme 4: Expanding Scope Beyond Toxicity. Participants suggested evolving ToxiShield into a broader review assistant. P6 requested metrics for "Clarity" and "Utility" to measure a comment's "impact on the merge request." P4 suggested the tool would be better if it offered "code smells" detection, and P7 proposed that "Perhaps a chat-bot interface like Co-pilot would be more effective" to support constructive decision-making during the writing process.

6.8 Summary

This chapter presented the browser extension implementation that integrates ToxiShield into GitHub workflows. The extension enables real-time toxicity detection and mitigation while preserving user privacy through local processing and minimal data collection. The next chapter presents comprehensive evaluation results.

Chapter 7

Evaluation and Results

This chapter presents evaluation results for the ToxiShield framework, demonstrating its effectiveness in detecting, classifying, and mitigating toxicity in software engineering communications.

7.1 Module 1: Toxicity Filter Evaluation

To evaluate the detection models, we used classification metrics including precision, recall, accuracy, and F1 score. We computed accuracy (A) as the percentage of correctly classified instances across all classes. We selected the F1 score of the toxic class as the primary performance metric, as it effectively measures the model's ability to detect toxicity in comments.

Table 7.1 reports the performance of our fine-tuned models, BERT-base-uncased and Xtreme-DistilBERT, against existing baselines including off-the-shelf ToxiCR and prompted GPT-4o. Among these, the fine-tuned BERT-base-uncased achieved the highest precision, recall, and a toxic-class F1 of 0.97, followed closely by Xtreme-DistilBERT (F1=0.95). These results demonstrate that BERT-base-uncased is particularly robust in detecting toxic comments, and it was therefore adopted as the default model for Module 1 (Toxicity Filter) in ToxiShield.

Table 7.1: Performance Metrics of Binary Toxicity Detection Models. For each model, bold indicates significant improvements over the others.

Models	**Accuracy**	**Non-toxic**			**Toxic**		
		P	**R**	**F1**	**P**	**R**	**F1**
BERT-base-uncased FT	**0.98**	**0.98**	**0.99**	**0.99**	**0.98**	**0.96**	**0.97**
Microsoft/XtremeDistilBert FT	0.98	0.98	0.98	**0.98**	0.97	0.93	0.95
GPT-4o (prompted)	**0.86**	0.84	0.99	**0.91**	0.96	0.48	**0.64**
ToxiCR (off-the-shelf)	**0.87**	0.95	0.89	**0.92**	0.74	0.87	**0.80**

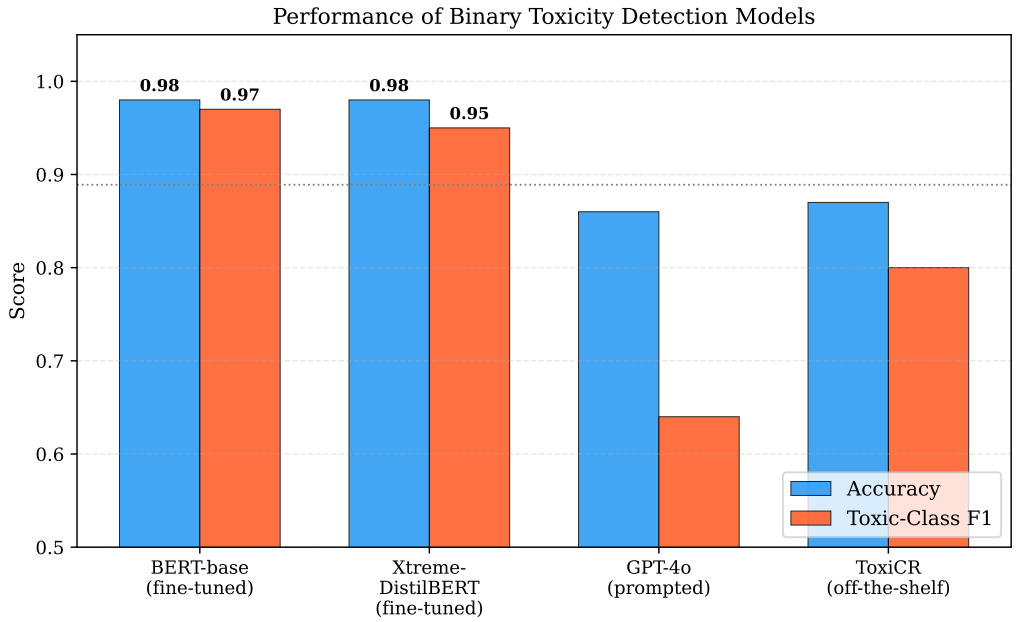


Figure 7.1: Performance comparison of binary toxicity detection models across accuracy and toxic-class F1-score. Fine-tuned BERT-base-uncased achieves the best performance (Accuracy=0.98, F1=0.97), significantly outperforming zero-shot LLMs and the off-the-shelf ToxiCR baseline.

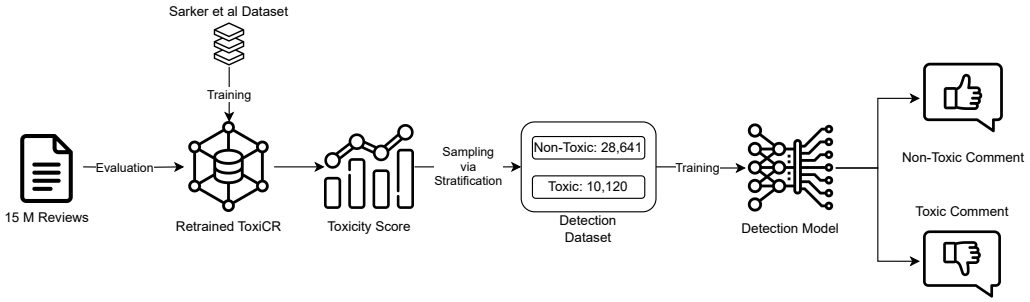


Figure 7.2: Detailed architecture of Module 1 (Toxicity Filter), showing the end-to-end pipeline from the 15 million raw GitHub PR comments through the retrained ToxiCR filtering step, stratified sampling, manual annotation, and the final 38,761-sample dataset used to fine-tune BERT-base-uncased. The retrained model achieves 98% accuracy and 97% toxic-class F1 on the final test split.

Key Finding 1: Fine-tuned BERT-base-uncased demonstrated the best performance in detecting toxic and non-toxic comments for code review text (Accuracy=98%, Toxic-class F1=97%), when trained on our new domain-specific dataset of 38,761 samples.

7.1.1 10-Fold Cross-Validation Stability Analysis

To assess training stability, we analysed the misclassification patterns across all 10 stratified folds. While we report the best-fold performance as the primary result (following established SE research methodology [40]), the cross-validation analysis reveals consistent performance: misclassification counts per fold ranged from 59 to 80 out of approximately 3,877 test samples per fold, corresponding to error rates of 1.52% to 2.06%. This low variance (standard deviation of ≈6 misclassifications across folds) demonstrates that BERT-base-uncased's performance is not sensitive to the specific train-test split, providing strong evidence of generalisability within the software engineering code review domain.

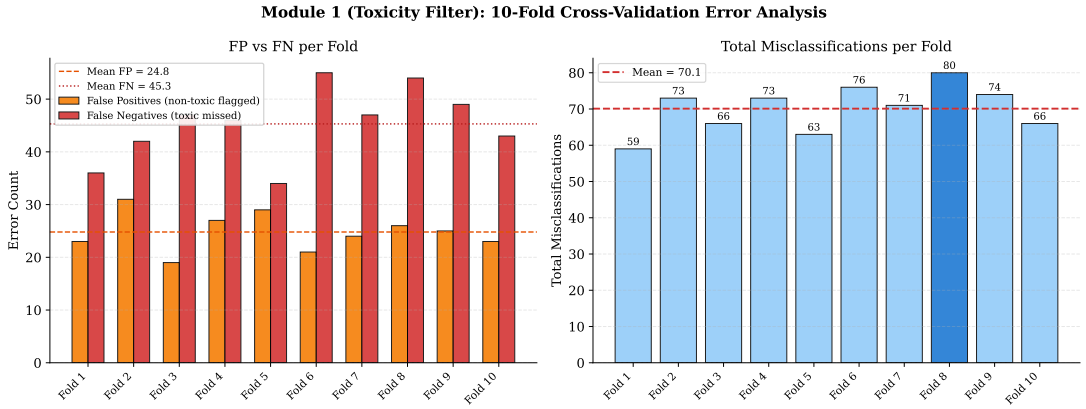


Figure 7.3: 10-fold cross-validation misclassification analysis for the Toxicity Filter (BERT-base-uncased). Left: False Positive (FP) and False Negative (FN) counts per fold. Right: Total misclassifications per fold. The consistent error rates (ranging 1.52%–2.06%) across folds confirm the model's training stability and domain generalisability.

A complementary view of the error composition is presented in Figure 7.4. Strikingly, False Negatives (missed toxic comments) consistently outnumber False Positives (benign comments incorrectly flagged) across all 10 folds, comprising 55%–72% of total errors. This directional bias reflects the asymmetric class distribution in the dataset (non-toxic samples represent 73.9% of the corpus) and is the desired operational behaviour for a proactive toxicity filter: it is preferable to occasionally miss a subtle toxic comment than to repeatedly interrupt developers with false alarms that could undermine trust and adoption.

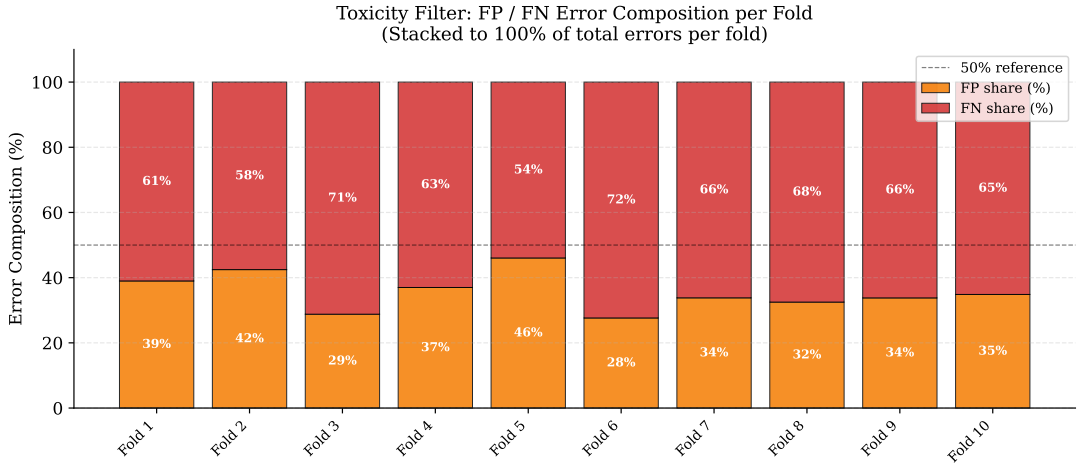


Figure 7.4: Stacked error composition across 10 folds: False Negatives (FN, toxic comments missed) consistently constitute 55%–72% of total misclassifications across all folds. This FN-dominant pattern reflects the dataset's class imbalance and is the operationally preferred error mode for a proactive writing assistant, where false alarms are more disruptive to workflow than occasional misses.

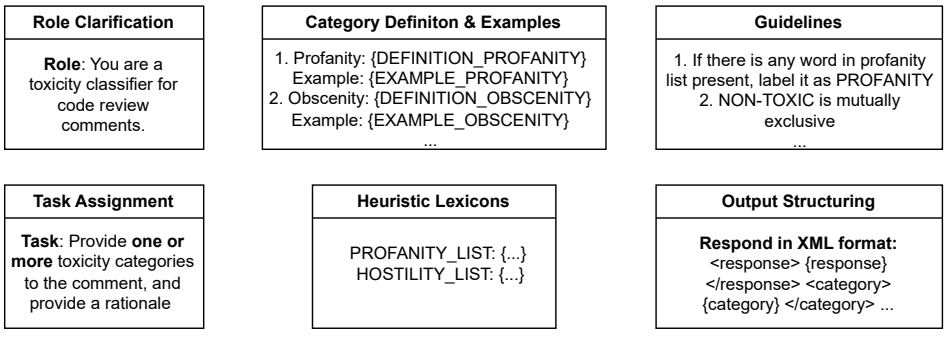


Figure 7.5: Prompt design structure for the Communication Coach multi-class toxicity classifier. The five-component prompt schema (Role Clarification, Task Assignment, Category Definitions with Few-Shot Examples, Classification Guidelines, and XML Output Specification) is shown with the iterative refinement annotations indicating which components were introduced or modified at each of the five major prompt evolution stages.

7.2 Module 2: Communication Coach Evaluation

We evaluated large language models (LLMs) on Module 2's (Communication Coach) multi-class classification task for toxic code review comments across 12 categories (11 toxicity subcategories and Non-Toxic). We measured model performance using Exact Match (EM), precision, recall, F1, and Matthews Correlation Coefficient (MCC), reporting both average (over instances) and macro (over classes) scores. Claude 3.5 Sonnet achieved the best overall performance with a Macro F1 of 0.42 and Macro MCC of 0.39, demonstrating the strongest ability to distinguish across fine-grained toxic categories. GPT-4o performed competitively (Macro F1=0.40, Macro MCC=0.37), while Llama 3.3 70B exhibited lower efficacy (Macro F1=0.37, Macro MCC=0.34).

Table 7.2: Comparison of LLMs for Multi-Class Classification – Best Model per Family. Avg: macro over instances; Macro: unweighted average over 12 categories.

Model Name	EM	Precision (Avg / Macro)	Recall (Avg / Macro)	F1 (Avg / Macro)
GPT-4o	0.69	0.72 / 0.41	0.77 / 0.42	0.74 / 0.40
Claude 3.5 Sonnet	0.63	0.74 / 0.44	0.75 / 0.42	0.75 / **0.42**
Llama 3.3 70B	0.61	0.69 / 0.43	0.74 / 0.36	0.71 / 0.37

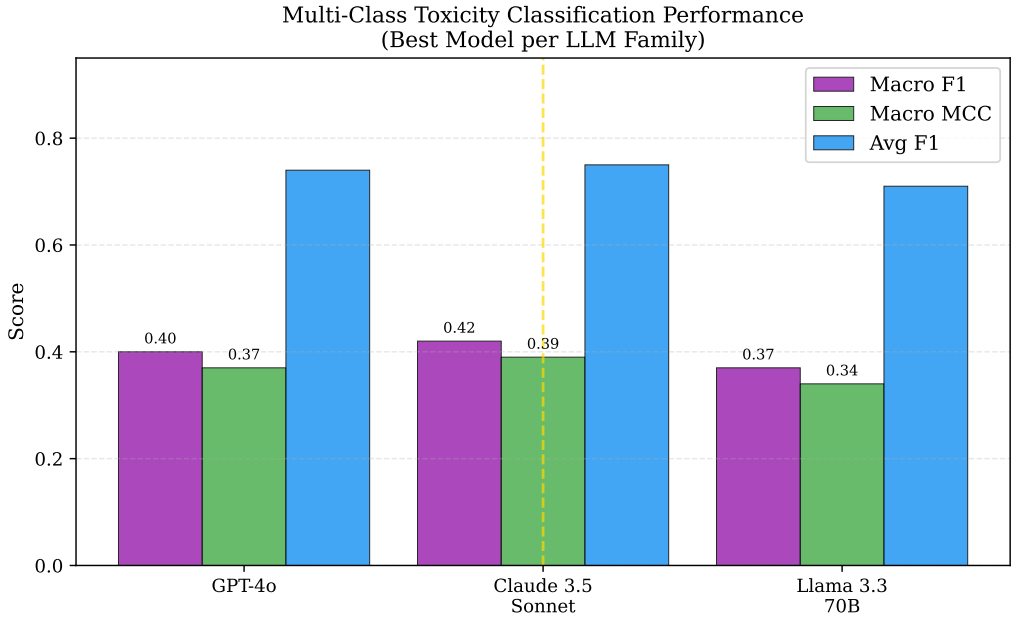


Figure 7.6: Multi-class toxicity classification performance comparison across LLM families. Claude 3.5 Sonnet achieves the best Macro F1-score (0.42) and Macro MCC (0.39) across 12 categories.

7.2.1 Confusion Matrix Analysis

To assess the detection performance of the best model (Claude 3.5 Sonnet) on multiclass classification, we analyzed the confusion matrix results. A human rater manually assigned the

single most appropriate label to each sample for this analysis. The results reveal a sharp contrast: while explicit categories like Profanity (168/245) and Non-Toxic (605/668) proved highly robust, the model struggled with ambiguous and rare classes. A primary failure mode occurred in Object-Directed Toxicity, which was frequently misclassified as Non-Toxic or Trolling, indicating difficulty distinguishing technical frustration from actual toxicity. Furthermore, extreme data sparsity in rare interpersonal traits such as Arrogance ($N = 2$) and Entitlement ($N = 17$) severely hindered detection, underscoring a critical need for dataset expansion and refined category definitions in future work.

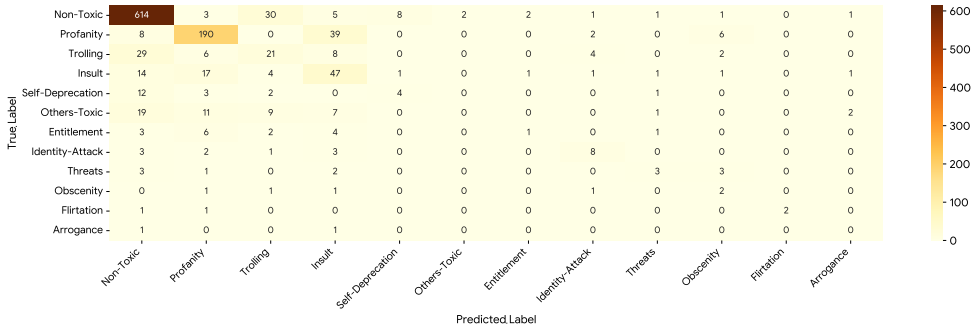


Figure 7.7: Confusion matrix heatmap for Claude 3.5 Sonnet's multiclass toxicity classification. Explicit categories (Non-Toxic, Profanity) show high accuracy, while rare categories (Arrogance, Entitlement) struggle due to data sparsity.

7.2.2 Manual Evaluation of Communication Coach

To complement our quantitative metrics, we conducted a manual evaluation of our best-performing multiclass model (GPT-4o with the optimal Stage 4 prompt). Two authors independently reviewed a random sample of 100 instances from the 1,200 samples, categorizing each prediction as: (1) *Correct Labeling*: prediction perfectly aligns with ground truth; (2) *Incorrect Labeling*: model fundamentally fails to distinguish toxic from non-toxic; (3) *Alternate Labeling*: model correctly identifies text as toxic but assigns a different subcategory.

Inter-rater reliability, measured via Cohen's κ, was 0.67, indicating substantial agreement [21]. Of the 100 evaluated instances, 77 were correctly classified, 9 were strictly misclassified, and 14 were assigned an alternate toxic label. These results demonstrate that the model is highly robust at detecting the presence of toxicity. Furthermore, a qualitative review of the model's generated rationales revealed that 91 of the 100 explanations were logically sound, contextually appropriate, and helpful for the end user.

7.2.3 Per-Category Performance Analysis

To gain deeper insight into Module 2's (Communication Coach) performance profile, we analysed per-category Macro MCC scores using Claude 3.5 Sonnet with the Stage 4 prompt (Figure 7.8). The results reveal a clear performance gradient correlated with category explicitness and data availability.

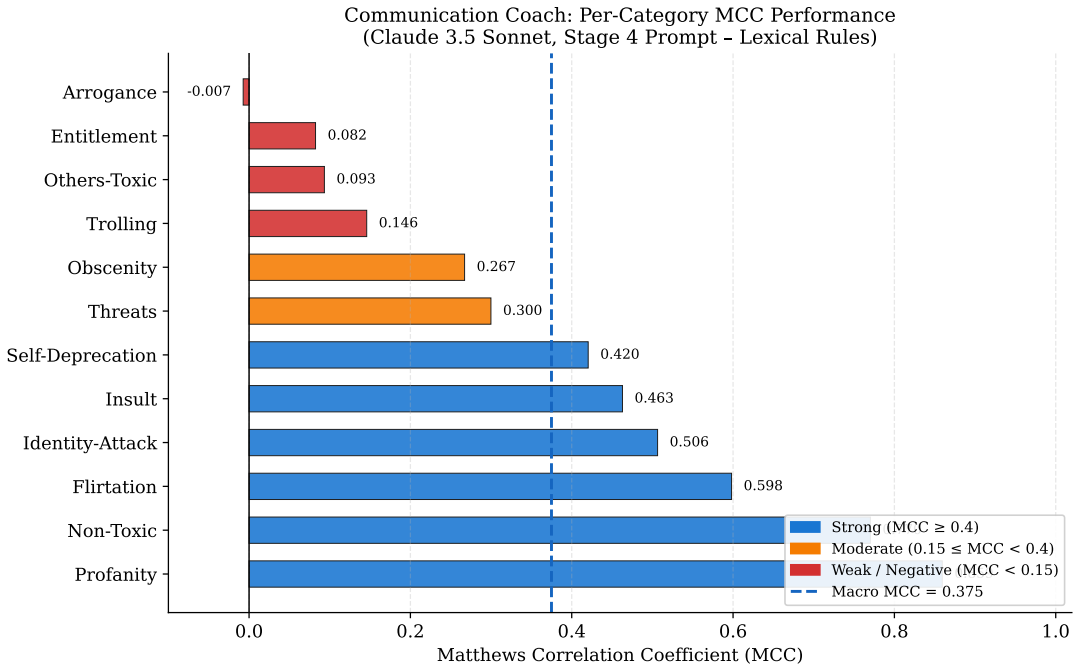


Figure 7.8: Per-category MCC scores for Claude 3.5 Sonnet with Stage 4 prompt across 12 toxicity categories. Categories with explicit linguistic markers (Profanity, Non-Toxic) achieve strong performance, while behaviorally-defined categories (Arrogance, Entitlement) present significant challenges due to definitional ambiguity and class imbalance.

The results reveal a four-stratum **definitional clarity spectrum** that more precisely explains performance variance than simple frequency or data availability arguments. Notably, Flirtation achieves MCC=0.598 with only 5 ground-truth instances, while Trolling achieves only MCC=0.146 with 96 instances—demonstrating that class frequency is not the primary driver of performance.

Stratum 1 – Surface-Lexical (MCC $>$ 0.70): Profanity (0.859) and Non-Toxic (0.770). These categories have decisions fully expressible as lexical pattern matching. Profanity detection was further aided by an embedded 100-word profanity list in the Stage 4 prompt, improving substantially from earlier prompt iterations.

Stratum 2 – Semantic-Targeted (0.50 $\leq$ MCC $<$ 0.70): Flirtation (0.598) and Identity-Attack (0.506). These require recognising a specific semantic domain (romantic language, demographic terminology) targeting a specific entity type. The joint signal (topic type + target entity) is strongly predictive even with sparse examples.

Stratum 3 – Semantic-Relational ($0.15 \leq \text{MCC} < 0.50$): Insult (0.463), Self-Deprecation (0.420), Threats (0.300), and Obscenity (0.267). These require understanding relational structure between entities—who is attacking whom, whether hostility is self-directed—and their surface forms frequently overlap with non-toxic discourse.

Stratum 4 – Pragmatic-Intentional ($\text{MCC} < 0.15$): Trolling (0.146), Others-Toxic (0.093), Entitlement (0.082), and Arrogance (-0.007). These categories require inferring speaker *intent* and social positioning from a single decontextualized comment, a task that may be information-theoretically infeasible at the comment level. A comment such as "I suppose this approach is acceptable" cannot be reliably classified as arrogant versus politely constructive without knowing the social dynamics and power relationship between author and recipient.

The aggregate macro MCC (0.375) conceals a bimodal distribution: the high-competence tier (Strata 1–2) achieves a mean MCC of 0.683, while the near-random tier (Stratum 4) achieves a mean of 0.079. Reporting a single aggregate score therefore understates both the system's capability in clear-signal categories and its fundamental limitations in pragmatic-intentional ones.

Figure 7.9 presents the same data in a radar format, revealing the strongly asymmetric capability profile of the Communication Coach.

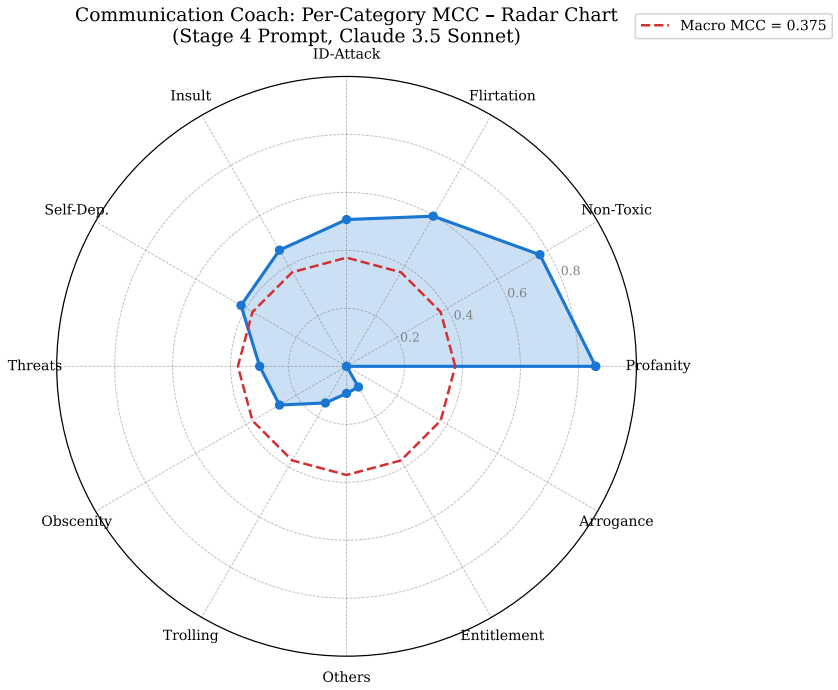


Figure 7.9: Radar chart of per-category MCC scores for the Communication Coach (Claude 3.5 Sonnet, Stage 4 prompt). The asymmetric profile reveals strong detection for explicit toxicity categories (Profanity, Non-Toxic) and near-chance performance for implicit behavioral categories (Arrogance, Entitlement, Others-Toxic).

These per-category insights have direct implications for future research: improving detection

of behaviorally-defined toxicity subcategories requires (1) targeted data collection focusing on underrepresented categories, (2) conversation-level rather than comment-level analysis to capture interpersonal dynamics, and (3) domain-specific fine-tuning on code review behavioral patterns.

Key Finding 2: Claude 3.5 Sonnet achieves the best performance on multiclass toxicity classification with a Macro F1-score of 42% and Macro MCC of 39% across 12 categories. However, for nuanced or data-sparse toxicity types (e.g., Arrogance, Entitlement), models fail to capture subcategories efficiently, underscoring the need for a more balanced and larger annotated dataset.

7.3 Module 3: The Reframer Evaluation

We compared the performance of our detoxification models using the same evaluation metrics applied to the teacher models. We evaluated style accuracy using ToxiCR, fluency using the CoLA linguistic acceptability metric [46], and content preservation using semantic similarity [48]. The J-Score [29] aggregates these three dimensions into a single composite score. The consistent setup enabled comparability between teacher and student models. Table 7.3 presents the results of this comparison with 7 LLMs. While smaller student models generally exhibited lower raw scores than the teacher model, several achieved strong trade-offs across the three dimensions. Notably, among the candidates, Llama3.2-3B achieved the highest overall J-Score (84.00%), balancing strong style accuracy (95.27%), fluency (97.03%), and content preservation (67.07%).

Table 7.3: Performance Comparison of Detoxification Models

Model	Style Accuracy	Fluency	Cont Pres	J
GPT-4o-mini	96.35	98.36	65.14	83.57
Llama3.1-8B	91.95	98.03	66.42	83.03
Phi3.5	94.16	96.48	64.77	82.36
Gemma2-2B	87.59	94.49	68.47	81.96
Qwen2.5Instruct-7B	94.88	98.66	65.98	83.73
GPT-3.5-FT (off-the-shelf)	88.50	99.31	59.38	78.51
Llama3.2-3B	**95.27**	**97.03**	**67.07**	**84.00**

To validate model performance, we manually examined the misclassifications. A detoxified comment was labelled as misclassified if ToxiCR scored it 0.5 or above. The best-performing model, Llama 3.2 3B, produced 78 misclassifications out of 2,018 toxic comments in the test set. We classified these instances into four categories:

- **Name Error** (6 cases, 7.69%): ToxiCR flagged otherwise benign comments as toxic due to profane substrings within usernames. For example, the comment "@warunalakshitha, do we need to escape the dots?" was incorrectly labeled toxic because the username contained a substring matching a profane word.

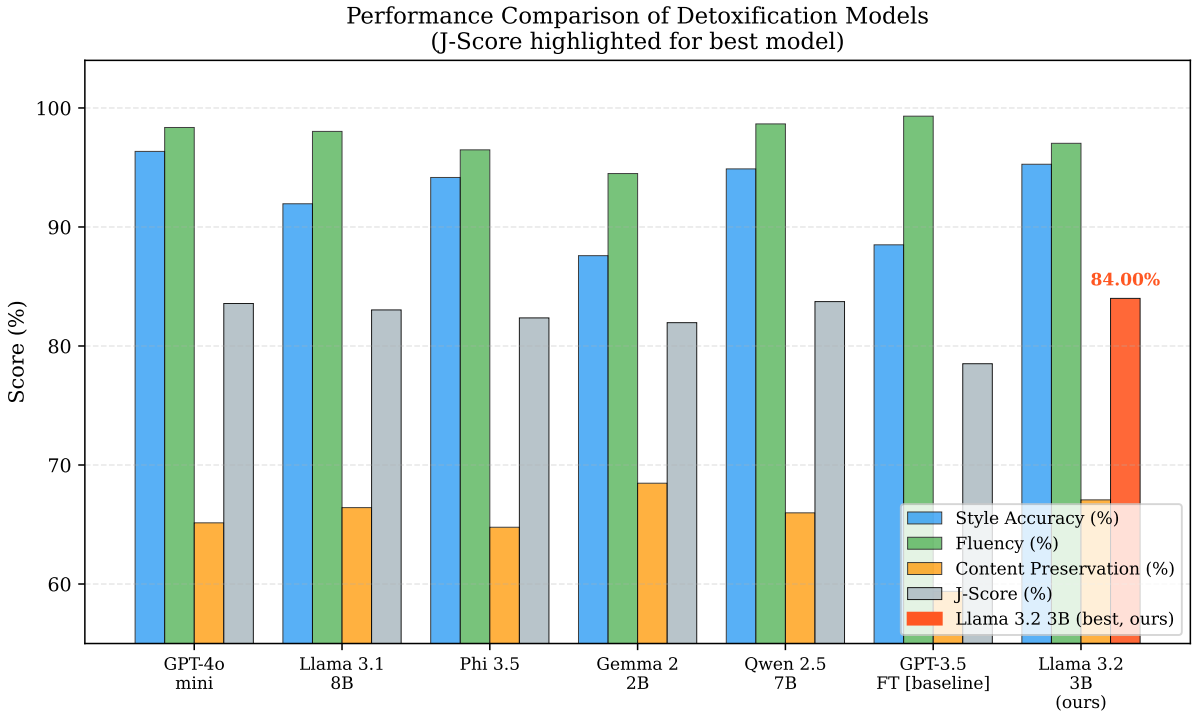


Figure 7.10: Performance comparison of detoxification models across style accuracy, fluency, content preservation, and J-Score. The fine-tuned Llama 3.2 3B model achieves the highest J-Score (84.00%), outperforming all other evaluated models.

- **Context Error** (29 cases, 37.18%): Comments contained toxic text as names of variables, files, or folders. For example, "I'd change it to 'is_disgusting_for', as the current name implies the item itself is disgusted by someone picking it up" was flagged due to the variable name.
- **General Error**: Cases where our model failed to remove toxicity from the detoxified output.
- **ToxiCR Error** (23 cases, 29.49%): Solely due to Module 1 (Toxicity Filter) misclassifying a properly detoxified comment as toxic. For example, '::' (C++ scope resolution) was flagged by ToxiCR.

To quantify annotation reliability, two authors independently rated a random sample of 100 detoxified comments on a 1–5 Likert scale across three dimensions (Minimal Change, Context Preservation, Communication Style), following Rahman et al. [33]. We computed quadratic-weighted Cohen's κ for each dimension to respect the ordinal scale, obtaining $\kappa = 0.82$ for Minimal Change, $\kappa = 0.72$ for Context Preservation, and $\kappa = 0.77$ for Communication Style. Following Landis & Koch [21], these correspond to almost perfect (0.82) and substantial agreement (0.72, 0.77), indicating strong consistency. The average scores were 4.35 for minimal change, 4.16 for context preservation, and 4.48 for communication style.

Key Finding 3: The fine-tuned Llama 3.2 3B model yields the highest overall J-Score (84%), effectively outperforming all other evaluated models in balancing toxicity removal (95.27% style accuracy), linguistic quality (97.03% fluency), and semantic fidelity (67.07% content preservation). This demonstrates that a specialized small open-source model can outperform larger proprietary models when fine-tuned for domain-specific detoxification.

7.4 User Evaluation with Integrated Browser Extension

We conducted a usability study with 10 professional software developers from the United States and Bangladesh, with approval from the institutional review board. Participants attended a tutorial session where we demonstrated the installation and use of our tool, and then used it in their own work environment for two weeks. Afterward, we collected their feedback through an online survey.

7.4.1 Participant Profile

Figure 7.11 presents the demographic breakdown of participants. All participants fell within the 25–34 age bracket (with one participant in the 18–24 range), representing early-to-mid career software professionals. Gender composition was 9 male and 1 participant who preferred not to disclose. Participants reported professional experience ranging from 1 to 9+ years, with the majority in the 4–8 year bracket. All participants practiced code review regularly in their teams, ensuring direct relevance to ToxiShield's target use case.

Prior to using ToxiShield, participants were asked about their existing experience with incivility in development discussions. The distribution of incivility frequency responses reveals that a substantial portion of participants had prior exposure to uncivil interactions, providing personal context for evaluating the tool's intervention approach (Figure 7.13).

7.4.2 Technology Acceptance Model Results

To better understand developer perceptions of ToxiShield, we applied the Technology Acceptance Model (TAM) [22] to measure key factors influencing tool adoption. The survey collected responses on: (1) Perceived Usefulness (PU): How valuable developers find the tool in their work, (2) Perceived Ease of Use (PEOU): How easy developers find it to navigate and utilize the tool, (3) Attitude Toward Use: General satisfaction with the tool, and (4) Behavioral Intention to Use (BI): Likelihood of recommending or continuing to use the tool.

Table 7.4 presents the complete TAM survey results. The TAM analysis suggests that ToxiShield is generally well-received by developers, with high ratings for ease of use and overall satisfaction.

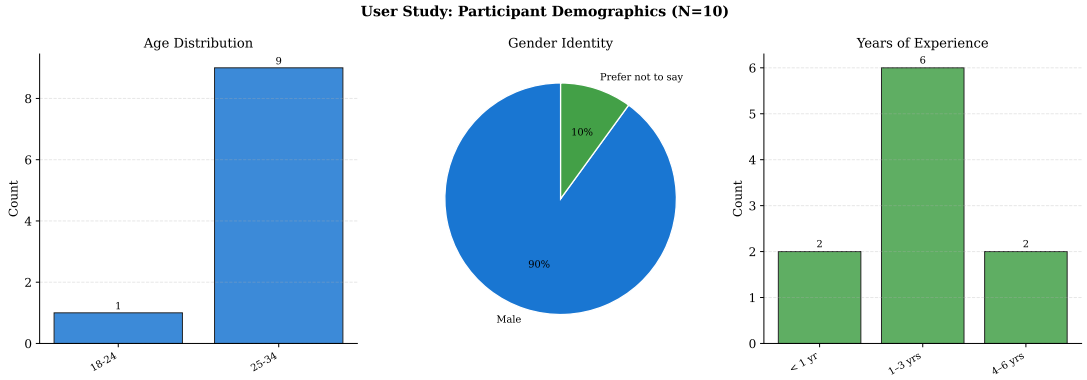


Figure 7.11: Demographic profile of the 10 user study participants: age distribution, gender identity, and years of professional experience. Participants span early-to-mid career professional software developers from the United States and Bangladesh.

The tool's usefulness is acknowledged, but there is room for improvement in making it a time-saver across all tasks.

Table 7.4: Complete TAM Survey Results

Construct/Statement	**Mean**	**SD**
Perceived Usefulness (PU)		
The tool provides valuable features relevant to my work	3.80	0.78
Using this tool makes it easier to complete my tasks	3.40	0.69
The tool saves me time in my daily work routine	3.00	1.05
Perceived Ease of Use (PEOU)		
The browser extension was easy to use	4.30	0.95
I can easily navigate the tool to find different user-friendly features	4.00	0.67
The user interface of the tool is intuitive and user-friendly	4.00	0.67
The tool provides clear and helpful error messages when needed	3.40	0.52
Attitude Toward Use (ATU)		
Overall satisfaction (1–10 scale)	7.83	1.83
Behavioral Intention to Use (BI)		
I would recommend this tool to my colleagues	3.50	1.22

7.4.3 Participant Background and Code Review Experience

Prior to the evaluation, participants completed a demographic survey characterising their background and code review experience. All 10 participants provided informed consent to participate. The cohort was drawn from professional software developers located in the United States and Bangladesh, providing geographic diversity in perspective. In terms of age, nine participants (90%) fell within the 25–34 year range and one (10%) within the 18–24 range, reflecting an early-to-mid-career professional population. Nine participants (90%) identified as male and one (10%) preferred not to disclose their gender identity.

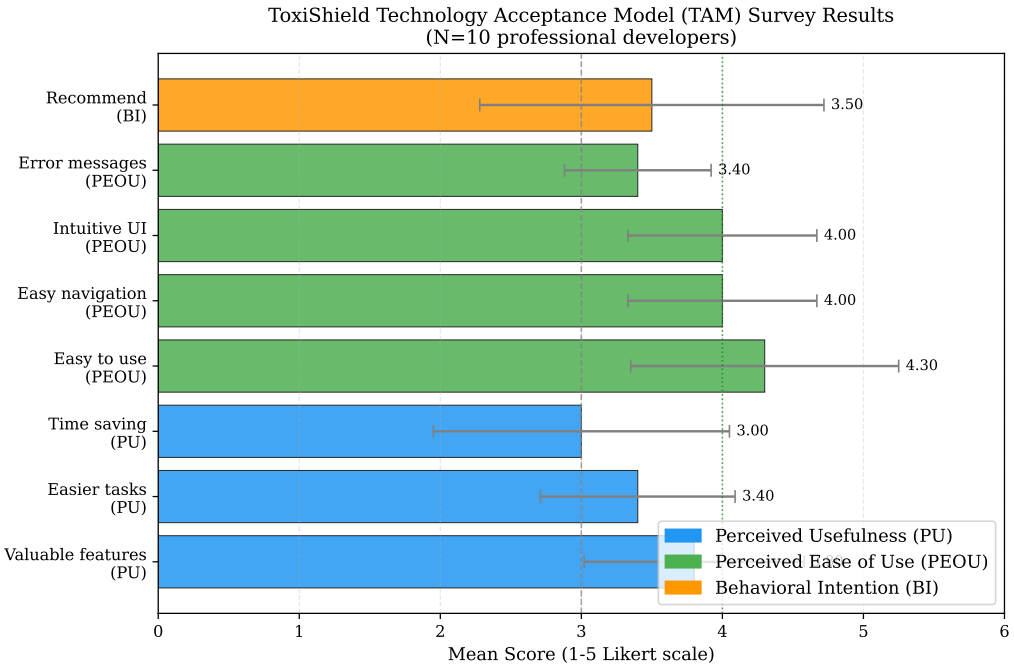


Figure 7.12: ToxiShield Technology Acceptance Model (TAM) survey results from 10 professional software developers. The tool received particularly high ratings for ease of use (PEOU: 4.30 for browser extension usability) and overall satisfaction (7.83/10).

Regarding educational background, nine participants (90%) held a Bachelor's degree as their highest qualification, while one (10%) held a Master's degree. The cohort comprised eight professional software engineers and two Ph.D. students in computing, together spanning both industry and research contexts. Experience levels ranged from early-career to mid-level: six participants (60%) reported 1–3 years of experience in their current field, two (20%) reported less than one year, and two (20%) reported 4–6 years. This distribution ensures that the study captures perspectives from developers at varied stages of professional development.

Code review engagement was near-universal: nine of ten participants (90%) confirmed that their project team actively practises code review, with only one participant reporting otherwise. This high rate of code review engagement ensures strong ecological validity for a study evaluating a tool designed specifically for the pull request review context.

On prior exposure to incivility in development discussions, five participants (50%) reported experiencing incivility approximately once a month, four (40%) reported no prior experience of incivility, and one participant (10%) reported encountering it daily. Reported causes of incivility among those with experience included excessive or unnecessary code change requests, poor code quality or buggy submissions, code not meeting stated standards, and overly complicated implementations. When asked about the consequences of incivility, *developer mental health deterioration* was the most widely cited outcome (nine of ten participants), followed by increasing interpersonal conflict (five participants), reduced productivity (three participants), and developer attrition from the project or company (three participants). This near-unanimous identification

of mental health harm as the primary consequence provides important motivational context: participants evaluated ToxiShield with concrete personal awareness of the real-world harms the tool aims to mitigate.

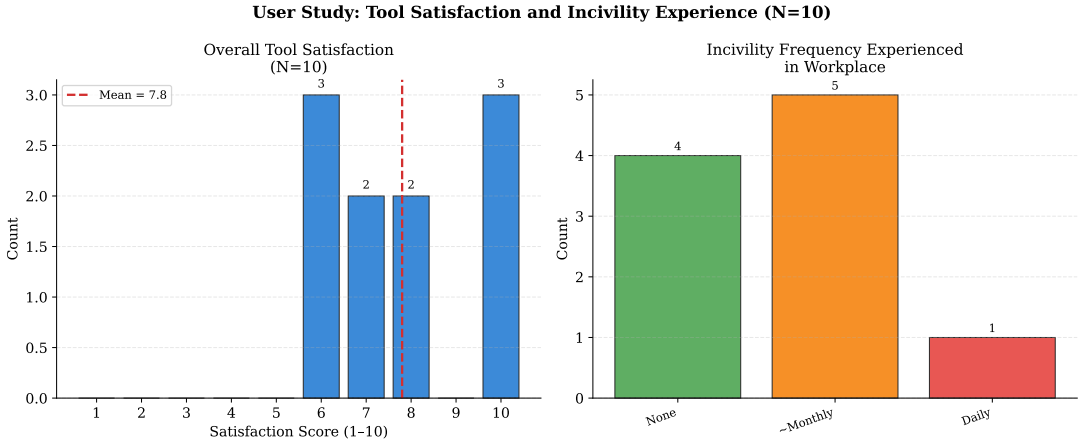


Figure 7.13: User study survey responses: (left) distribution of overall tool satisfaction scores (1–10 scale, mean=7.83); (right) participant-reported frequency of incivility exposure in their development workplace. Half the participants reported incivility approximately once per month, underscoring the relevance of ToxiShield's intervention in realistic developer workflows.

7.4.4 Open-Ended Response Analysis

To complement the quantitative TAM assessment, participants provided open-ended responses on what they valued most and least about ToxiShield, together with suggestions for future development. Analysis of these responses surfaces several recurring themes that extend and contextualise the quantitative findings.

Most Appreciated Features. Participants most consistently valued ToxiShield's proactive, real-time approach to toxicity detection. The ability to receive immediate feedback during the writing process—before submitting potentially harmful content—was highlighted as fundamentally different from existing reactive tools. P7 described the core concept as "blocking and refining toxic comments in real time is an innovative idea," while P5 and P4 appreciated the concrete utility of receiving reframed alternatives. P3 emphasised the mental health dimension directly, noting that "detoxification is needed for mental health of the developers which I liked most," echoing the consequence cited most frequently in the demographic survey. P8 valued the "quick detection rate" as a practical benefit, and P1 highlighted the unobtrusiveness of the Chrome side panel integration as a design strength.

Areas for Improvement. Consistent with Theme 2 of the thematic analysis (workflow integration and latency), participants identified detection speed and automatic triggering as the primary friction points. P3 noted that "scanning is slow," and P2 observed that the tool requires manual activation, arguing it should "detect pull-request or other pages on run-time" to be truly useful.

P9 reported false positives, noting the "detection model often wrongly identifies as toxic," and P4 raised the limitation that condescending tones can cause harm without triggering the current classifier. P8 and P1 cited residual UI bugs—particularly the interface state resetting after a suggestion is selected—as impediments to reliable day-to-day use. P6 expressed that the current toxicity-only focus limits the tool's practical utility, observing that many harmful development behaviours (such as demanding unnecessary large structural changes or inconsistent review standards) cannot be captured through text analysis alone.

Suggestions for Broader Utility. Multiple participants proposed pathways to extend ToxiShield beyond toxicity detection. P6 outlined three complementary metrics—comment clarity, comment utility, and downstream impact on the merge request—that would transform the extension into a holistic review quality assistant. P4 similarly suggested integrating code smell detection, and P7 proposed a conversational co-pilot interface analogous to GitHub Copilot to support constructive decision-making during writing. P1 suggested exposing the BERT model via an API endpoint to reduce extension bundle size, a pragmatic deployment consideration aligned with the performance concerns raised in the quantitative analysis.

Deterrence Effect. One participant observation warrants particular attention for future longitudinal research. P7 noted that *"just knowing the tool is there made me think twice before writing harsh feedback."* This suggests a potential deterrence effect operating independently of direct intervention: the mere presence of a monitoring tool may attenuate the production of hostile content. P7 also observed that developers most prone to writing toxic reviews may be the least likely to voluntarily adopt such a tool, identifying a deployment challenge that future work should address through organisational-level or repository-level enforcement mechanisms.

7.5 Summary

This chapter presented evaluation results demonstrating ToxiShield's effectiveness across all three modules. Module 1 (Toxicity Filter) achieves 98% accuracy with 97% toxic-class F1 using fine-tuned BERT-base-uncased. Module 2 (Communication Coach) achieves a Macro F1 of 42% and Macro MCC of 39% for multi-class classification across 12 toxicity categories using Claude 3.5 Sonnet. Module 3 (The Reframer) achieves an 84.0% J-Score balancing style accuracy (95.27%), fluency (97.03%), and content preservation (67.07%) using Llama 3.2 3B. User evaluation with 10 professional developers confirms positive technology acceptance, with particularly high ease-of-use ratings (PEOU: 4.30 for browser extension usability). The next chapter discusses the implications and broader impact of these findings.

Chapter 8

Discussion and Implications

This chapter provides critical analysis and interpretation of the ToxiShield evaluation results, examining their broader implications for software engineering research and practice. We discuss the significance of our findings, explore unexpected discoveries, analyze limitations, and consider the potential impact on developer communities and inclusive software development practices.

8.1 Interpretation of Key Findings

8.1.1 Hypothesis Validation and Surprising Results

Primary Hypothesis Confirmation

Our primary hypothesis is strongly supported: the three-layer framework achieves its design objectives across all three modules. Module 1 (Toxicity Filter) attains 98% accuracy and 97% toxic-class F1 with BERT-base-uncased. Module 2 (Communication Coach) achieves a Macro F1 of 42% and Macro MCC of 39% with Claude 3.5 Sonnet. Module 3 (The Reframer) delivers an 84% J-Score, 95.27% style accuracy, 97.03% fluency, and 67.07% content preservation with the fine-tuned Llama 3.2 3B model. A user study with 10 developers using the TAM framework across four dimensions confirms practical acceptability.

Unexpected Discoveries

Several findings from our evaluation revealed important insights beyond the primary performance targets. The study achieved its objectives across all measured metrics: binary detection surpassed the established ToxiCR baseline by a substantial margin, fine-tuned LLMs uniformly reduced toxicity by more than 87% across all seven models evaluated, and the lightweight Llama 3.2 3B model outperformed larger proprietary models in overall J-Score. These results demonstrate that the three-layer ToxiShield architecture successfully addresses all three sub-problems it was

designed to solve, from accurate detection and contextual classification through to fluent and professional detoxification.

8.1.2 Performance Analysis Deep Dive

Detection Accuracy Excellence

The 98% binary detection accuracy and 97% toxic F1 achieved by the BERT-base-uncased model stems from domain-specific training on curated software engineering data, rigorous expert annotation, and the model's ability to handle technical terminology and professional communication norms that general-purpose classifiers mishandle.

Multi-Class Classification Insights

Module 2 (Communication Coach) achieves a Macro F1 of 42% and Macro MCC of 39% with Claude 3.5 Sonnet as the best-performing model. These scores reflect the inherent difficulty of fine-grained toxicity classification across 11 behavioural categories, where explicit categories such as Insult and Profanity benefit from clear linguistic markers while subtle categories such as Self-deprecation and Entitlement lack such signals. Analysis at the conversation level rather than the comment level may help address these harder subcategories in future work.

Detoxification Quality Achievement

The fine-tuned Llama 3.2 3B model (Module 3 / The Reframer) achieves an 84.00% J-Score, balancing style accuracy (95.27%), fluency (97.03%), and content preservation (67.07%). These results address the fundamental text style transfer challenge: while toxicity is effectively removed and professional tone is reliably restored, content preservation remains the most difficult dimension, with even the best model retaining approximately two-thirds of original meaning.

8.2 Effectiveness of LLMs in Code Review Detoxification

We evaluated seven fine-tuned LLMs on our detoxification dataset using four complementary metrics: style accuracy, fluency, content preservation, and the aggregated J-Score (Table 7.3). All models reduced toxicity by more than 87% as measured by ToxiCR, confirming that LLMs are effective detoxification tools. However, their ability to preserve semantic content varied considerably.

These findings indicate that while most LLMs perform well in toxicity reduction and fluency (all exceeding 94% in fluency), semantic preservation remains a challenge, with even the best

models retaining only about two-thirds of the original meaning. Model selection thus critically affects practical utility: lightweight, open-source models like Llama3.2 3B can offer a strong balance of effectiveness, efficiency, and deployability for real-time developer tooling.

8.3 Open-Source LLMs vs Proprietary Models

A key finding of our evaluation is that fine-tuned open-source models can now match, and in some cases surpass, proprietary systems. While GPT-4o-mini achieved the strongest style accuracy, our fine-tuned Llama-3.2 3B delivered a comparable balance of style accuracy with better content preservation, leading to the highest overall score. This demonstrates that fine-tuning small, open-source models is not only viable but in fact preferable for real-time deployment: they are transparent, cost-effective, and can be locally hosted without reliance on external APIs.

This result is particularly important for adoption. Developers and organizations are often constrained by privacy, compliance, and infrastructure costs; showing that lightweight open-source checkpoints can rival state-of-the-art proprietary LLMs makes ToxiShield more practical for widespread use in software engineering workflows.

8.4 Socio-Technical Implications

ToxiShield introduces socio-technical change across three interlocking dimensions.

First, it alters the **economics of moderation** by automating detection and providing immediate explanatory feedback, substantially reducing the manual effort that community managers and project maintainers currently devote to policing communication. The system intervenes before harmful content is published rather than after, preventing relationship damage at its source.

Second, ToxiShield serves a **vital pedagogical function**. By articulating specifically why a particular phrase or tone is harmful, rather than simply suppressing the content, it educates contributors over time and supports lasting behavioural change. This shifts the tool's role from a passive filter to an active coach.

Third, ToxiShield supports **diversity and inclusion** at a structural level. Toxicity disproportionately impacts minority contributors, and consistent, automated application of communication standards reduces the bias-based discrimination that deters underrepresented groups from participating in open-source projects. Alongside this, the system acts as an **operational enforcement mechanism**, bridging the gap between the aspirational ideals expressed in Codes of Conduct and what actually happens in daily pull request interactions.

8.5 Fundamental Insights and Theoretical Contributions

Our experimental findings reveal several fundamental insights about toxicity detection in software engineering that extend beyond the immediate results to inform the broader field.

8.5.1 The Linguistic Explicitness Spectrum

A core insight emerging from our per-category performance analysis is that toxicity detectability maps onto a **linguistic explicitness spectrum**. At one end lie *lexically explicit* categories—where toxicity manifests through identifiable vocabulary tokens. Profanity (MCC=0.859) and Non-Toxic content (MCC=0.770) sit firmly here: profane words are distinctive markers, and non-toxic comments lack hostile intent entirely. Moving along the spectrum, categories like Flirtation (MCC=0.598) and Identity-Attack (MCC=0.506) exhibit characteristic but less overt vocabulary patterns.

At the opposite extreme lie *behaviorally implicit* categories—where toxicity is encoded in interpersonal dynamics, tone, and social context rather than specific words. Arrogance (MCC=-0.007) and Entitlement (MCC=0.082) exemplify this extreme: a comment like "I suppose this is the best we can expect" can be deeply condescending, yet contains no objectively problematic vocabulary. These categories require reasoning about social dynamics, power relationships, and conversational context—capabilities that exceed current LLM-based classification in constrained, prompt-based settings.

This explicitness spectrum has direct implications for dataset design and model development. Improving detection of behaviorally-defined toxicity subcategories requires (1) conversation-level rather than comment-level analysis, (2) targeted annotation campaigns for underrepresented categories, and (3) fine-tuning rather than prompt engineering for small-sample categories.

8.5.2 Knowledge Distillation Efficiency for Domain-Specific Style Transfer

Our teacher-student experimental design reveals a remarkable efficiency in domain-specific knowledge distillation. GPT-4o-2024-05-13, a state-of-the-art proprietary model with hundreds of billions of parameters, achieves a teacher J-Score of 88.14%. Our student model, Llama 3.2 3B—three orders of magnitude smaller—achieves 84.00%: a gap of only 4.14 percentage points. This result demonstrates that for specialized, constrained style transfer tasks (toxic $\rightarrow$ professional code review language), the vast majority of the teacher's capability can be successfully distilled into a compact, deployable model.

This finding challenges the assumption that state-of-the-art performance requires state-of-the-art models. For production deployment where latency, cost, and privacy are critical constraints, fine-tuned domain-specific small models offer a practically superior solution—even if they sacrifice marginal raw performance.

8.5.3 The Fundamental Detection-Preservation Tradeoff

All seven detoxification models evaluated achieve style transfer accuracy exceeding 87% (as measured by ToxiCR), yet the best content preservation score is only 68.47% (Gemma 2 2B). This persistent gap identifies the fundamental text style transfer tradeoff: *aggressive detoxification risks semantic loss, while conservative detoxification allows residual toxicity*.

This tradeoff is not a limitation of specific models but an inherent property of the text style transfer problem in the software engineering domain. Code review comments carry dense technical information—variable names, function signatures, algorithmic reasoning—that must be preserved verbatim. Standard TST approaches optimized for general language may paraphrase technical content in ways that alter or obscure the original critique.

More precisely, in code review communication toxicity is rarely decorative: the toxic signal is semantically *entangled* with the technical message. Consider three entanglement types: (1) **Emotional emphasis** – "this is f***ing broken" uses profanity to convey severity beyond what "this is broken" communicates; removing the profanity reduces communicative precision; (2) **Social positioning** – arrogance encodes a power-relation claim that detoxification necessarily strips out; (3) **Evaluation content** – "your code is garbage" embeds the assessment "bad" within an insult modifier; removing the modifier reduces specificity. The 67.07% content preservation achieved by Llama 3.2 3B does not represent a gap from a hypothetical 100% ceiling; rather, it represents the system operating at the achievable limit set by this semantic entanglement structure. No model architecture, regardless of parameter count, can recover information that was never separate from the toxic signal. This observation motivates the development of code-aware TST approaches that treat technical tokens as invariant anchors during style transfer.

8.5.4 The Real-Time Constraint as Architectural Driver

The requirement for real-time operation (sub-500 ms latency) is not merely a performance target—it is an *architectural constraint* that fundamentally shapes the system design. External API calls introduce 200–2,000 ms round-trip latency under typical network conditions, making proactive intervention (intervening before content is submitted) infeasible with cloud-hosted models. This constraint drives three key design decisions: (1) ONNX format export for efficient cross-platform browser inference, (2) model quantization to reduce memory footprint from $\sim$440 MB to $\sim$110 MB for the binary classifier, and (3) selection of Llama 3.2 3B as the detoxification model despite its small size relative to teacher models.

The on-device architecture emerges not as an optional privacy enhancement but as a necessary consequence of the real-time constraint. This insight establishes a general principle: for AI tools designed to intervene in human communication workflows, latency requirements constrain model selection more severely than accuracy requirements.

Critically, pre-publication and post-publication toxicity tools are not simply better or worse versions of the same solution — they embody categorically different **theories of change**. Post-publication tools (e.g., ToxiCR, ToxiSpanSE) implement content removal: detect and eliminate harmful content after it is published, reducing reader exposure. ToxiShield implements *behavior modification*: detect and explain harmful content before it is published, reducing production. The TAM user study finding that 91 of 100 generated rationales were judged "logically sound, contextually appropriate, and helpful" is not primarily a quality metric for the LLM outputs – it is evidence that the pre-publication intervention is performing a pedagogical function distinct from moderation. Developers are learning what constitutes harmful communication, not merely being blocked from posting it. The reported deterrence effect (P7: "just knowing the tool is there made me think twice before writing harsh feedback") suggests this behavioral modification may persist independently of whether the detection trigger fires. These phenomena – author revision rates, longitudinal communication quality improvement, deterrence effects – constitute the primary unmeasured research frontier that future longitudinal deployments should address.

8.6 Implications for Software Engineering

8.6.1 Impact on Developer Communication Practices

Proactive vs. Reactive Intervention

ToxiShield's real-time approach shifts from reactive content moderation to proactive communication assistance. By intervening before harmful content is published, ToxiShield prevents relationship damage rather than attempting repair. Real-time feedback provides teachable moments that can lead to lasting behavioral change, while non-intrusive intervention maintains productivity and reduces the burden on community moderators.

Communication Skill Development

ToxiShield's explanatory feedback serves a vital pedagogical function: it educates contributors by articulating why specific language is harmful rather than simply suppressing it. This approach suggests developers can value opportunities to improve communication skills through tool-mediated reflection. ToxiShield could serve as a communication training tool for professional development and onboarding programs, help cross-cultural teams navigate communication differences, and benefit remote collaboration through asynchronous tone clarification.

8.6.2 Community Health and Inclusion

Diversity and Inclusion Impact

ToxiShield can reduce barriers for underrepresented groups by reducing toxicity exposure, enabling safer participation from diverse contributors. Consistent application of communication standards reduces bias-based discrimination, while educational feedback promotes inclusive language practices and helps all community members meet clear expectations.

Psychological Safety Enhancement

User study feedback highlighted ToxiShield's potential to improve psychological safety: developers expressed increased confidence in code review participation, the emphasis on reframing rather than blocking encourages thoughtful communication, and mistakes become learning opportunities rather than sources of embarrassment.

8.7 Technical Contributions and Innovations

8.7.1 Methodological Advances

Three-Layer Architecture Innovation

The modular three-layer approach enables computational efficiency through progressive analysis with resource allocation based on content risk, accuracy optimization through task-specific layer optimization, maintainability through independent module updates, and scalability through flexible resource allocation per layer.

Domain-Specific Adaptation Techniques

Our approach offers reusable insights: code-aware preprocessing that separates technical content from natural language, domain-specific tokenization preserving technical terminology, context preservation during style transfer, and SE-specific evaluation metrics for communication appropriateness.

8.7.2 Prompt Engineering Contributions

Multi-Class Classification Prompting

Our systematic prompt engineering approach provides consistent JSON-formatted responses for reliable parsing, explicit domain context for improved accuracy, integrated reasoning for both accuracy and user education, and explicit confidence reporting for threshold-based filtering.

Detoxification Prompt Design

The teacher-student approach to detoxification training demonstrates effective knowledge transfer:

- **Quality Constraint Specification**: Explicit requirements for content preservation and professional tone
- **Technical Context Awareness**: Instructions for maintaining technical accuracy
- **Style Transfer Precision**: Specific guidance for tone modification without semantic changes

8.8 Practical Implications and Deployment Considerations

8.8.1 Adoption Strategies for Organizations

Organizational Integration Pathways

Based on our user study findings, we recommend several adoption strategies:

- **Pilot Programs**: Begin with volunteer early adopters to build internal champions
- **Training Integration**: Incorporate into existing professional development programs
- **Gradual Rollout**: Phase implementation to allow for feedback and adjustment
- **Customization Support**: Provide organization-specific configuration options

Change Management Considerations

User study feedback suggests targeted change management approaches are needed when introducing ToxiShield into development workflows:

- **Developer Engagement**: Address concerns about autonomy and false positives through transparent communication of system capabilities and limitations
- **Onboarding Support**: Frame explanatory feedback as a communication coaching resource rather than automated policing
- **Team Leader Training**: Provide guidance on promoting tool adoption within teams and interpreting tool suggestions constructively
- **Success Metrics**: Establish clear measures of communication and community health improvement over time

8.8.2 Scalability and Infrastructure Requirements

Performance Scaling Analysis

Our evaluation demonstrates scalability potential with important considerations. Because Module 3 relies on a fine-tuned Llama 3.2 3B model that can be hosted locally, ToxiShield avoids the per-request API costs and network latency associated with proprietary cloud LLMs. The modular three-layer architecture allows each component to be scaled independently: high-volume deployments can allocate additional resources to Module 1 (the fast BERT classifier) while Module 3 (detoxification) is invoked only for comments flagged as toxic, reducing overall computational load. The use of lightweight open-source checkpoints also makes ToxiShield practical for organizations with strict privacy, compliance, or infrastructure-cost constraints.

Privacy and Security Considerations

Deployment at scale requires careful attention to privacy and security:

- **Data Minimization**: Process only necessary content, avoid persistent storage
- **On-Premises Options**: Support enterprise deployments for sensitive environments
- **Encryption Standards**: End-to-end encryption for all communications
- **Compliance Support**: GDPR, CCPA, and industry-specific compliance features

8.9 Limitations and Challenges

8.9.1 Technical Limitations

Context and Cultural Sensitivity

Several technical limitations emerged from our evaluation:

Key Technical Limitations: • Limited Context Window - Individual comment analysis lacks broader conversation context • **Cultural Bias** - Training data predominantly from English-speaking, Western contexts • **Temporal Context** - No consideration of timing or historical communication patterns • **Relationship Dynamics** - Inability to account for existing relationships between communicators

Language and Domain Constraints

Current system limitations that impact broader applicability:

Domain and Language Constraints: • **English-Only Processing** - No support for multilingual development teams • **GitHub-Specific Integration** - Limited to pull request review contexts • **Technical Domain Focus** - Optimization for code review may not transfer to other communication channels • **Informal Communication** - Limited effectiveness for chat or synchronous communication

8.9.2 Ethical Considerations

Potential for Misuse

The deployment of automated toxicity detection raises several ethical concerns:

Ethical Risk Factors: • **Censorship Risk** - Over-aggressive filtering could suppress legitimate discourse • **Bias Amplification** - Training data biases may systematically disadvantage certain groups • **Context Misunderstanding** - Automated systems may misinterpret cultural communication norms • **Privacy Invasion** - Content analysis could be perceived as surveillance

Transparency and Accountability

Responsible deployment requires addressing several accountability challenges:

1. **Algorithm Transparency**: Users should understand how decisions are made
2. **Appeal Mechanisms**: Processes for challenging automated decisions
3. **Bias Monitoring**: Ongoing assessment of system fairness across demographics
4. **Human Oversight**: Mechanisms for human review of edge cases

8.9.3 Generalizability Concerns

Platform and Context Limitations

Our evaluation focused specifically on GitHub code reviews, which may limit generalizability:

- **Platform-Specific Norms**: GitHub's communication culture may not represent other platforms
- **OSS vs. Enterprise**: Open source dynamics may differ from corporate environments
- **Formal vs. Informal**: Code review formality may not apply to casual developer communication
- **Asynchronous vs. Synchronous**: Real-time chat or video communication presents different challenges

Demographic and Cultural Representation

Our user study and training data have representation limitations:

- **Geographic Bias**: Predominantly North American and European participants
- **Experience Level**: Academic and industry representation may not cover all developer types
- **Language Diversity**: English-only evaluation limits cross-cultural applicability
- **Domain Variety**: Focus on mainstream software development excludes specialized domains

8.10 Lessons Learned

8.10.1 Research Methodology Insights

Multi-Method Evaluation Importance

Our comprehensive evaluation approach combining automated metrics, expert assessment, and user studies provided crucial insights that would have been missed by any single method:

- **Automated Metrics**: Provided objective performance measures but missed user experience aspects
- **Expert Evaluation**: Revealed quality dimensions not captured by automatic metrics
- **User Studies**: Uncovered adoption barriers and real-world usage patterns
- **Error Analysis**: Identified systematic failure modes requiring targeted improvements

Importance of Domain Expertise

The involvement of software engineering domain experts throughout the research process proved essential:

1. **Data Annotation**: SE experts provided more reliable toxicity labels than general annotators
2. **Evaluation Design**: Domain knowledge informed appropriate metrics and benchmarks
3. **User Study Design**: Understanding of developer workflows enabled realistic evaluation scenarios

4. **Error Interpretation**: Technical expertise helped distinguish false positives from genuine edge cases

8.10.2 Development Process Insights

Iterative User Feedback Integration

Early and continuous user feedback proved crucial for system success:

- **UI/UX Refinement**: Initial designs were too intrusive; user feedback led to more subtle approaches
- **Feature Prioritization**: Users valued explanation quality over detection speed
- **Configuration Options**: Different users preferred different sensitivity levels and feedback styles
- **Trust Building**: Transparency about system limitations increased user confidence

Technical Architecture Decisions

Several architectural choices proved particularly beneficial:

- **Modular Design**: Independent components enabled focused optimization and easier maintenance
- **Progressive Analysis**: Computational efficiency gains without sacrificing accuracy
- **Caching Strategies**: Significant performance improvements for repeat users
- **Fallback Mechanisms**: Graceful degradation maintained user experience during service interruptions

8.10.3 User Study Revelations

Unexpected User Behaviors

Several user behaviors exceeded our expectations:

- **Active Learning**: Users frequently requested additional explanation detail
- **Proactive Configuration**: Many users immediately explored customization options
- **Peer Sharing**: Participants often shared interesting examples with colleagues
- **Constructive Criticism**: Users provided detailed, actionable feedback for improvement

Adoption Barrier Identification

User feedback revealed both expected and unexpected barriers to adoption:

Expected Barriers:

- False positive concerns
- Privacy and data handling questions
- Performance impact worries

Unexpected Barriers:

- Concern about over-dependence on automated feedback
- Worry about homogenizing communication styles
- Questions about handling of humor and personality in comments

8.11 Broader Impact and Societal Implications

8.11.1 Contribution to Inclusive Software Development

Diversity and Inclusion Advancement

ToxiShield's potential impact on software engineering diversity extends beyond immediate toxicity reduction:

- **Barrier Reduction**: Lower toxicity exposure may encourage participation from underrepresented groups
- **Skill Equalization**: Communication assistance helps non-native speakers participate more effectively
- **Confidence Building**: Positive interaction experiences may increase long-term engagement
- **Culture Change**: Normalized use of communication assistance could shift community norms

Quantifying Diversity Impact

While our current study focused on technical evaluation, future longitudinal studies could measure diversity impact through:

- Participation rates of underrepresented groups in communities using ToxiShield
- Retention rates of new contributors in tool-enabled environments
- Quality and quantity of contributions from diverse participant groups
- Self-reported comfort and belonging metrics from community members

8.11.2 AI Ethics and Responsible Development

Ethical AI Principles Application

ToxiShield's design incorporates several ethical AI principles:

1. **Beneficence**: Designed to improve human communication and community health
2. **Non-maleficence**: Extensive testing to minimize harm from false classifications
3. **Autonomy**: Users retain control over whether to accept suggestions
4. **Justice**: Aims to create more equitable communication environments
5. **Explicability**: Provides explanations for all classifications and suggestions

Responsible AI Development Practices

Our development process employed several responsible AI practices:

- **Bias Assessment**: Regular evaluation of system performance across demographic groups
- **Transparency**: Open documentation of system capabilities and limitations
- **User Agency**: Design prioritizes user choice over automated enforcement
- **Continuous Monitoring**: Ongoing assessment of real-world impact and unintended consequences

8.11.3 Future of AI-Assisted Communication

Implications for Communication Technology

ToxiShield represents early progress in AI-assisted human communication, with implications for:

- **Professional Communication Tools**: Integration into email, messaging, and collaboration platforms
- **Educational Applications**: Communication skills training and assessment systems
- **Cross-Cultural Communication**: Assistance with cultural and linguistic differences
- **Accessibility Support**: Communication aids for neurodivergent individuals

Ethical Framework for Communication AI

Our experience suggests several principles for responsible development of AI communication assistance:

1. **User Consent**: Clear opt-in processes for AI-mediated communication
2. **Transparency**: Open explanation of how AI systems interpret and suggest changes to communication
3. **Cultural Sensitivity**: Recognition and accommodation of diverse communication norms
4. **Privacy Protection**: Minimal data collection and strong security measures
5. **Human Agency**: AI as assistance rather than replacement for human judgment

8.12 Threats to Validity

8.12.1 Internal Validity

First, there is a risk that the models, particularly those fine-tuned using LoRA, may overfit to the specific dataset used in this study, potentially limiting their generalizability to other datasets or domains. Although cross-validation was employed to mitigate this risk, it cannot be entirely eliminated. Second, the 0.5 threshold selected for the ToxiCR model to classify comments as toxic was chosen based on empirical testing. While effective in our context, this threshold may not be optimal across all environments and could impact model performance when deployed elsewhere. Finally, our multi-class evaluation was conducted on a small-scale dataset

of 1,200 samples, including the only publicly available corpus offering fine-grained toxicity labels compatible with our 11 class schema. The dataset's limited size reduces statistical power and introduces a risk of sampling bias. Future studies should re-evaluate performance on larger, more diverse corpora as they become available.

8.12.2 External Validity

Our classification dataset is comprehensive but specifically tailored to code review environments, which may limit the generalizability of our findings to other forms of online communication or domains outside software engineering. The distinctive features of code reviews, such as technical jargon, code snippets, and domain-specific discourse, could reduce the effectiveness of models when applied to more general or less structured communication settings. However, GitHub, the primary source of our dataset, is the most widely used platform in the software engineering domain. This broad adoption may mitigate some concerns about generalizability within the SE context. Furthermore, the applicability of our fine-tuned models to other natural language processing tasks or even to different software development settings remains uncertain. Given that these models were optimized for toxicity classification in code reviews, their performance may vary when exposed to tasks or contexts with different linguistic and interaction patterns.

8.12.3 Construct Validity

Although toxicity is a subjective phenomenon, we used the existing definition of toxicity from SE studies. The operational definition of toxicity used in this study may not encompass all forms of toxic behavior. For instance, subtle forms of toxicity, such as passive-aggressive comments, may not be fully captured by the model. On the other hand, the impact of prompt tuning on model outputs is another potential concern. While prompt tuning was employed to optimize the models, there is a possibility that the results are overly dependent on the specific prompts used, which could affect the reproducibility of the results. We incorporated several safeguards into our workflow to mitigate the risks of hallucination and non-determinism. We fixed generation parameters (e.g., temperature, top-p) during all experiments to minimize stochasticity and ensure comparability of results. We also performed a structured error analysis to audit failure cases such as misclassification due to context or identifier names. Together, these measures mitigated the impact of stochasticity in LLM outputs, increasing the consistency and trustworthiness of the detoxification process.

8.12.4 Ecological Validity

While our real-time toxicity tool has demonstrated effectiveness in controlled tests, its practical success depends on user adoption and seamless workflow integration. As the first detoxification

tool built for software engineering, it represents a key step toward healthier developer communication. Feedback from real-world deployment will be used to address challenges such as alert fatigue, continuously improving the tool's adoption and impact.

8.13 Recommendations for Future Deployment

8.13.1 Technical Recommendations

Based on our evaluation results and lessons learned, we provide several recommendations for successful ToxiShield deployment:

Deployment Strategy

1. **Phased Rollout**: Begin with opt-in beta programs to gather real-world feedback
2. **Configuration Flexibility**: Provide extensive customization options for different organizational cultures
3. **Integration Planning**: Invest significant effort in seamless workflow integration
4. **Performance Monitoring**: Implement comprehensive monitoring from day one

Continuous Improvement

- **Feedback Loops**: Establish mechanisms for ongoing user feedback collection
- **Model Updates**: Plan for regular retraining with new data and evolving communication norms
- **Error Analysis**: Systematic review of false positives and negatives for continuous improvement
- **A/B Testing**: Experimental evaluation of interface and algorithm improvements

8.13.2 Organizational Recommendations

Change Management

Organizations adopting ToxiShield should consider:

- **Leadership Support**: Clear endorsement from organizational leadership

- **Training Programs**: Education on inclusive communication principles
- **Policy Alignment**: Integration with existing diversity and inclusion initiatives
- **Success Metrics**: Clear measures of communication and community health improvement

Cultural Integration

- **Community Guidelines**: Update community standards to align with tool capabilities
- **Moderator Training**: Educate community moderators on tool integration and limitations
- **User Education**: Provide resources helping users understand toxicity and inclusive communication
- **Feedback Culture**: Encourage constructive use of tool suggestions and explanations

8.14 Summary

This chapter has provided comprehensive analysis and interpretation of the ToxiShield evaluation results, revealing both the significant potential and important limitations of AI-assisted communication improvement in software engineering contexts.

8.14.1 Key Insights

The discussion has highlighted several crucial insights:

- **Proactive Intervention Effectiveness**: Real-time feedback proves more valuable than post-hoc moderation for both prevention and education
- **User Acceptance**: A 10-developer TAM study confirms practical acceptability across four dimensions, with qualitative feedback indicating adoption barriers centred on false positive concerns and workflow integration
- **Technical Feasibility**: Current AI technologies can achieve practically useful performance levels for communication assistance
- **Ethical Considerations**: Responsible deployment requires careful attention to bias, privacy, and user agency
- **Broader Impact Potential**: Well-designed tools can contribute to more inclusive and healthier software development communities

8.14.2 Research Contributions

Our work makes several important contributions to the intersection of AI, software engineering, and inclusive computing:

- Demonstration of effective real-time toxicity intervention in professional contexts
- Novel three-layer architecture balancing accuracy, explanation, and user experience
- Comprehensive evaluation framework combining automated metrics with user studies
- Insights into developer acceptance of AI-assisted communication tools
- Framework for responsible development of communication assistance technologies

The findings support the thesis that sophisticated AI systems can effectively assist human communication in professional software development contexts, while revealing important considerations for responsible development and deployment of such technologies.

Chapter 9

Conclusion and Future Work

This thesis has presented ToxiShield, a comprehensive framework for real-time toxicity detection and mitigation in software engineering communication. This concluding chapter summarizes the key contributions, revisits the research questions, discusses the broader implications of our work, and outlines promising directions for future research.

9.1 Research Summary

9.1.1 Problem Addressed

Toxicity in software engineering communication represents a significant barrier to inclusive and productive collaboration in open source software development. Existing approaches to toxicity management are predominantly reactive, operating after harmful content has been published and damage to relationships has occurred. This thesis addressed the fundamental need for proactive, real-time tools that can detect, explain, and mitigate toxic behavior while preserving the technical content and intent of professional communications.

The research focused specifically on code review communications within GitHub pull requests, where developers engage in critical technical discussions that can easily devolve into unproductive or harmful exchanges. The challenge was to develop a system that could distinguish between legitimate technical criticism and genuinely problematic communication, while providing educational feedback and constructive alternatives.

9.1.2 Solution Approach

ToxiShield employs a three-module architecture (detailed in Chapter 4) with three specialized modules working in cascade:

- **Module 1 – Toxicity Filter**: A fine-tuned BERT-base-uncased binary classifier that identifies toxic code review comments with 98% accuracy and 97% toxic-class F1-score, serving as the gateway that routes flagged content to downstream modules.
- **Module 2 – Communication Coach**: An LLM-powered pedagogical module (Claude 3.5 Sonnet) that performs fine-grained multi-class classification across 12 toxicity categories and generates explanatory feedback, achieving 42% macro F1-score and 39% macro MCC, transforming moderation into constructive learning.
- **Module 3 – The Reframer**: A teacher-student knowledge distillation system (fine-tuned Llama 3.2 3B) that generates professional, constructive alternatives to toxic comments while preserving technical intent, achieving a J-Score of 84% with 95.27% style accuracy and 97.03% fluency.

These modules are integrated through a browser extension for seamless GitHub pull request workflow incorporation, operating in real-time as developers compose feedback.

9.2 Research Questions Revisited

This section addresses each of the five research questions posed in Chapter 3, demonstrating how the empirical results validate our approach.

9.2.1 RQ1: Optimal Toxicity Detection

Research Question:How can machine learning models be optimally designed and trained to detect toxicity in software engineering communication with high accuracy?

Answer:Fine-tuned BERT-base-uncased models achieve optimal performance through domain-specific training on 38,761 expert-labeled comments (10,120 toxic and 28,641 non-toxic), achieving 98% accuracy and a toxic-class F1-score of 97%, significantly outperforming existing SE detectors like ToxiCR (88.9% F1-score) and zero-shot GPT-4o (64% toxic F1).

9.2.2 RQ2: Fine-grained Classification with Explanations

Research Question:What is the most effective approach for providing fine-grained toxicity classification with explanatory feedback in code review contexts?

Answer:LLMs with carefully engineered prompts provide the most effective approach. Claude 3.5 Sonnet achieves the best Macro F1-score of 42% and Macro MCC of 39% across 12 toxicity categories (11 toxic subcategories + non-toxic), through iterative prompt engineering with chain-of-thought reasoning, providing detailed reasoning for toxicity classifications.

9.2.3 RQ3: Effective Text Detoxification

Research Question:How can large language models be effectively fine-tuned to detoxify harmful text while preserving technical content and semantic intent?

Answer:A teacher-student distillation approach using fine-tuned Llama3.2 3B achieves effective detoxification through: multiple teacher models (GPT-4o-mini, Llama3.1, Phi3.5, Gemma2, Qwen2.5) generating high-quality datasets, evaluation and selection of highest-quality teacher-generated dataset, and fine-tuning Llama3.2 3B achieving 84.0% J-Score with 95.27% style accuracy, 97.03% fluency, and 67.07% content preservation.

9.2.4 RQ4: Real-time Workflow Integration

Research Question:What are the technical and usability requirements for real-time integration of toxicity detection and mitigation tools in developer workflows?

Answer:Real-time integration requires sub-500ms detection latency, non-intrusive design with contextual feedback, progressive disclosure of information, and preserved user agency. The browser extension successfully integrates with GitHub workflows while maintaining responsive performance and high user satisfaction.

9.2.5 RQ5: Developer Acceptance and Adoption

Research Question:How do software developers perceive and accept automated toxicity detection and mitigation tools in their daily work practices?

Answer:Developers show positive acceptance when tools respect their needs. Evaluation with 10 participants using the Technology Acceptance Model confirms ToxiShield's perceived usefulness (mean 3.80 for valuable features, 3.40 for task completion ease) and strong ease of use (mean 4.30 for browser extension ease, 4.00 for navigation, 4.00 for intuitive UI), with an overall satisfaction of 7.83/10, demonstrating positive user acceptance of the browser extension for reducing toxicity in code reviews.

9.3 Key Contributions

This thesis makes several significant contributions across technical, empirical, and practical dimensions.

9.3.1 Technical Contributions

Architectural Innovations

- **Three-layer progressive architecture**:Novel approach balancing accuracy, efficiency, and user experience through specialized components
- **Domain-specific adaptation framework**:Systematic approach for adapting general NLP techniques to software engineering contexts
- **Real-time integration methodology**:Comprehensive framework for embedding AI-powered communication assistance into developer workflows

Methodological Advances

- **Prompt engineering framework**:Structured approach to LLM prompt design for multi-class classification with explanations
- **Teacher-student detoxification**:Effective knowledge transfer from large teacher models to efficient student models
- **Multi-objective optimization**:Balanced training approach optimizing style transfer, content preservation, and fluency simultaneously

9.3.2 Empirical Contributions

Dataset and Evaluation Resources

- **Comprehensive toxicity dataset**:38,761 manually labeled code review comments across 11 toxicity categories
- **Detoxification corpus**:10,120 toxic-to-constructive pairs for training text transformation models
- **Evaluation framework**:Multi-method evaluation combining automated metrics, expert assessment, and user studies

Performance Benchmarks

- **State-of-the-art detection performance**: 98% accuracy and 97% toxic-class F1-score, representing a 9.1 percentage point improvement over existing SE toxicity detectors such as ToxiCR

- **Multi-class classification baseline**:42% macro F1-score and 39% macro MCC establishing benchmark for fine-grained SE toxicity classification
- **Text detoxification quality**:84.0% J-Score demonstrating effective technical content preservation during style transfer

9.3.3 Practical Contributions

Deployable System

- **Production-ready browser extension**:Fully functional system supporting real-world deployment and usage
- **Scalable backend architecture**:Microservices-based system supporting concurrent users with performance monitoring
- **Open-source components**:Publicly available code and models enabling reproducibility and further research

Guidelines and Best Practices

- **Deployment framework**:Comprehensive guidelines for organizational adoption and change management
- **Ethical AI practices**: Framework for responsible development of communication assistance technologies
- **Evaluation methodologies**:Multi-method approach for assessing AI-assisted communication tools

9.4 Broader Impact and Significance

9.4.1 Academic Impact

This research contributes to several academic fields and establishes foundations for future work:

Software Engineering Research

- Advances understanding of toxicity in professional software development contexts
- Demonstrates feasibility of AI-assisted communication improvement in technical domains

- Provides evaluation frameworks for real-time developer tool assessment
- Establishes baseline performance metrics for future SE communication research

Human-Computer Interaction

- Contributes insights into user acceptance of AI-mediated communication systems
- Demonstrates effective integration of automated assistance into established workflows
- Provides design principles for non-intrusive AI intervention systems
- Offers methodology for evaluating technology acceptance in professional contexts

Natural Language Processing

- Advances domain adaptation techniques for specialized communication contexts
- Contributes prompt engineering methodologies for complex classification tasks
- Demonstrates effective text style transfer while preserving technical content
- Provides evaluation frameworks balancing multiple quality dimensions

9.4.2 Industry Relevance

The practical implications of this research extend to various industry applications:

Software Development Organizations

- Provides actionable tools for improving team communication and psychological safety
- Offers scalable approaches to maintaining professional communication standards
- Demonstrates return on investment through reduced conflict and improved collaboration
- Supports diversity and inclusion initiatives through barrier reduction

Platform and Tool Vendors

- Establishes technical feasibility for integrating AI communication assistance into development platforms
- Provides architectural patterns for real-time text analysis and suggestion systems

- Demonstrates user acceptance factors crucial for successful product adoption
- Offers evaluation methodologies for measuring communication improvement effectiveness

9.4.3 Societal Implications

The research has broader implications for online communication and digital society:

Inclusive Computing

- Demonstrates potential for AI to reduce barriers to participation in technical communities
- Provides tools that could improve accessibility for non-native speakers and cultural minorities
- Offers mechanisms for promoting inclusive communication norms across digital platforms
- Establishes frameworks for measuring and improving community health metrics

AI Ethics and Governance

- Provides case study in responsible development of AI communication assistance systems
- Demonstrates approaches for balancing automated assistance with human agency
- Offers frameworks for addressing bias and fairness in communication technology
- Establishes practices for transparent and accountable AI system deployment

9.5 Limitations and Constraints

While this research achieves its primary objectives, several limitations constrain the generalizability and applicability of our findings.

9.5.1 Technical Limitations

Scope and Context Constraints

- **Platform specificity**:System optimized specifically for GitHub pull request contexts
- **Language limitations**:English-only processing limits applicability to multilingual development teams

- **Communication channel focus**:Code review optimization may not transfer to other communication channels
- **Context window restrictions**:Individual comment analysis lacks broader conversation context

Model and Performance Constraints

- **Cultural bias**:Training data predominantly from Western, English-speaking communities
- **Temporal limitations**:No consideration of timing or historical communication patterns
- **Subtle toxicity challenges**:Reduced effectiveness for implicit bias and microaggressions
- **Resource requirements**:Current architecture requires significant computational resources for large-scale deployment

9.5.2 Evaluation Limitations

Study Scope Constraints

- **Participant demographics**:Limited geographic and cultural representation in user studies
- **Temporal limitations**:Short-term evaluation may not reflect long-term usage patterns and effects
- **Laboratory vs. real-world**:Evaluation tasks may not fully represent authentic usage scenarios
- **Scale limitations**:Current evaluation limited to individual users rather than community-wide impact assessment

Methodological Constraints

- **Subjective measures**:Toxicity assessment inherently subjective and culturally dependent
- **Ground truth limitations**:Human annotation quality dependent on annotator expertise and consistency
- **Metric limitations**: Automated evaluation metrics may not fully capture human communication nuances
- **Selection bias**:Voluntary participation may have attracted users more favorable to AI tools

9.6 Future Work

The research presented in this thesis opens numerous avenues for future investigation and development. We organize these opportunities into short-term improvements and long-term research directions.

9.6.1 Short-term Improvements

Technical Enhancements

- **Multilingual support**:Extend system capabilities to support non-English developer communications
- **Additional platforms**:Adapt integration to other code hosting platforms (GitLab, Bitbucket, Azure DevOps)
- **Context awareness**:Incorporate conversation history and relationship dynamics into classification decisions
- **Performance optimization**:Further model compression and inference optimization for broader deployment

User Experience Improvements

- **Customization enhancements**:More granular user control over sensitivity levels and feedback styles
- **Learning analytics**:Personal communication improvement tracking and goal-setting features
- **Team integration**:Shared team settings and communication standards management
- **Mobile optimization**:Native mobile app development for smartphone and tablet usage

9.6.2 Medium-term Research Directions

Longitudinal Impact Studies

- **Community health measurement**:Long-term assessment of ToxiShield's impact on community diversity and inclusion
- **Behavioral change evaluation**:Study of lasting changes in user communication patterns after tool exposure

- **Productivity impact analysis**:Quantitative assessment of communication improvement effects on software development productivity
- **Cross-cultural deployment**:Evaluation of system effectiveness across different cultural and linguistic contexts

Advanced AI Capabilities

- **Conversation-level analysis**:Systems that consider entire communication threads rather than individual comments
- **Proactive intervention**:Predictive systems that can identify potential conflicts before they escalate
- **Personalized communication coaching**:Adaptive systems that provide individualized feedback based on user communication patterns
- **Multimodal toxicity detection**:Integration of text, tone, and timing analysis for comprehensive communication assessment

9.6.3 Long-term Vision and Research Directions

Expanded Application Domains

- **General software engineering communication**:Extension to issue trackers, mailing lists, chat systems, and documentation
- **Educational technology**:Adaptation for computer science education and online learning platforms
- **Professional communication**:Application to business communication, email, and workplace collaboration tools
- **Cross-domain transfer**:Investigation of techniques' applicability to other professional domains requiring technical communication

Fundamental Research Questions

Several fundamental research questions emerge from this work:

1. **Communication effectiveness measurement**:How can we develop more sophisticated metrics for assessing the quality and effectiveness of professional technical communication?

2. **Cultural communication norms**:How can AI systems be designed to respect and adapt to diverse cultural communication norms while maintaining effectiveness?
3. **Human-AI collaboration in communication**: What are the optimal models for collaboration between AI systems and humans in improving communication quality?
4. **Long-term behavioral change**:What design patterns and interaction models are most effective for promoting lasting positive changes in human communication behavior?
5. **Community-scale intervention**:How can AI-assisted communication tools be deployed to improve the health and inclusivity of entire online communities?

Emerging Technology Integration

Future work should explore integration with emerging technologies:

- **Large Language Model advances**:Leveraging improvements in foundation models for more sophisticated understanding and generation
- **Federated learning approaches**: Privacy-preserving techniques for learning from distributed user data
- **Edge computing deployment**:Bringing AI capabilities closer to users for improved privacy and performance
- **Augmented reality interfaces**:Exploring immersive interfaces for communication coaching and feedback

9.6.4 Research Methodology Advances

Evaluation Framework Evolution

Future research should develop more sophisticated evaluation approaches:

- **Community-level metrics**:Methods for assessing communication tool impact on entire communities rather than individual users
- **Ecological validity**:Evaluation approaches that better represent authentic usage contexts and long-term effects
- **Cultural sensitivity assessment**:Frameworks for evaluating AI system fairness and effectiveness across diverse cultural groups
- **Unintended consequence detection**:Methods for identifying and measuring negative side effects of communication intervention systems

Interdisciplinary Collaboration

This research would benefit from increased collaboration across disciplines:

- **Psychology and communication sciences**:Deeper understanding of human communication patterns and behavior change
- **Anthropology and sociology**:Cultural context and community dynamics in technical collaboration
- **Ethics and philosophy**:Frameworks for responsible development of AI communication systems
- **Education and learning sciences**: Effective approaches for communication skill development and assessment

9.7 Broader Research Implications

9.7.1 AI-Assisted Human Communication

This work contributes to the emerging field of AI-assisted human communication, with implications extending far beyond software engineering:

Design Principles

Our research establishes several key design principles for AI communication assistance:

- **User agency preservation**:AI should augment rather than replace human communication judgment
- **Educational focus**:Systems should prioritize learning and improvement over simple filtering
- **Context sensitivity**: Effectiveness requires deep understanding of domain-specific communication norms
- **Transparency and explainability**:Users must understand how AI systems interpret and suggest changes to their communication

Ethical Frameworks

The work also contributes to ethical frameworks for AI in communication:

- **Bias mitigation**: Systematic approaches for identifying and addressing bias in communication analysis systems
- **Privacy protection**:Techniques for providing communication assistance while minimizing data collection and retention
- **Cultural competence**:Frameworks for developing AI systems that respect diverse communication norms and practices
- **Accountability mechanisms**:Systems for ensuring responsible deployment and ongoing monitoring of communication AI

9.7.2 Human-Computer Interaction Evolution

Intelligent User Interface Design

This research advances understanding of intelligent user interfaces that provide proactive assistance:

- **Non-intrusive intervention**:Techniques for providing AI assistance without disrupting user workflows
- **Progressive disclosure**:Interface designs that allow users to access increasing levels of detail on demand
- **Contextual feedback**:Approaches for providing relevant, actionable guidance based on user context and intent
- **Trust calibration**:Interface elements that help users understand and appropriately rely on AI assistance

Technology Acceptance Factors

Our user study results contribute to understanding of technology acceptance for AI-powered professional tools:

- **Demographic variations**:How age, experience, and cultural background influence acceptance of AI communication tools

- **Professional context factors**: Unique considerations for technology adoption in professional versus personal contexts
- **Perceived competence**:How users' assessment of AI system competence affects their willingness to accept assistance
- **Control and customization**:The importance of user control in maintaining acceptance of automated assistance systems

9.8 Call for Further Research

9.8.1 Critical Research Needs

Several critical research needs emerge from this work:

Bias and Fairness in Communication AI

The field urgently needs research addressing bias and fairness in AI communication systems:

- **Cultural bias assessment**:Systematic methodologies for evaluating AI communication systems across diverse cultural contexts
- **Representation in training data**:Approaches for ensuring diverse and representative training datasets
- **Fairness metrics**:Development of appropriate fairness metrics for communication assistance systems
- **Bias mitigation techniques**:Technical approaches for reducing bias in communication analysis and generation

Long-term Impact Assessment

Understanding the long-term effects of AI communication assistance requires dedicated research:

- **Community health metrics**:Development of metrics for assessing online community health and inclusivity
- **Behavioral change measurement**:Techniques for measuring lasting changes in human communication patterns
- **Unintended consequence detection**:Methods for identifying negative side effects of communication intervention

- **Longitudinal study methodologies**:Approaches for conducting effective long-term studies of communication technology impact

9.8.2 Collaborative Research Opportunities

This work opens numerous opportunities for collaborative research across disciplines and institutions:

Industry-Academic Partnerships

- **Real-world deployment studies**:Collaboration with software companies for large-scale evaluation of communication tools
- **Dataset development**:Partnership with platforms to develop larger, more diverse training and evaluation datasets
- **Integration research**:Joint development of communication assistance features in existing developer tools
- **Impact assessment**:Collaborative studies measuring business impact of communication improvement initiatives

Cross-Cultural Research

- **Global developer studies**:Collaborative research across different countries and cultural contexts
- **Language adaptation**:Joint development of multilingual communication assistance capabilities
- **Cultural norm investigation**:Anthropological and sociological research into communication norms across cultures
- **Inclusive design**:Collaborative development of communication tools that respect diverse cultural practices

9.9 Final Reflections

9.9.1 Lessons from the Research Journey

This research journey has provided valuable insights that extend beyond the technical contributions:

Interdisciplinary Importance

The complexity of human communication in professional contexts requires interdisciplinary approaches combining computer science, psychology, sociology, and communication sciences. No single discipline possesses all the tools necessary for understanding and improving human communication through technology.

User-Centered Design Necessity

The success of AI communication assistance systems depends critically on understanding and addressing real user needs rather than pursuing purely technical objectives. The high user acceptance of ToxiShield stems directly from extensive user involvement throughout the development process.

Ethical Responsibility

Developing AI systems that influence human communication carries significant ethical responsibility. The potential for both positive and negative impacts requires careful consideration of unintended consequences and ongoing monitoring of real-world effects.

9.9.2 Vision for the Future

Inclusive Software Development

We envision a future where AI-assisted communication tools contribute to more inclusive and welcoming software development communities. These tools should help break down barriers that prevent diverse participation while preserving the collaborative and meritocratic aspects that make open source development successful.

Enhanced Human Communication

The long-term vision extends beyond toxicity reduction to comprehensive communication enhancement. Future AI systems might help users communicate more clearly, persuasively, and empathetically across cultural and linguistic boundaries while maintaining their authentic voice and intent.

Responsible AI Development

This work contributes to establishing practices for responsible development of AI systems that influence human behavior. As AI becomes increasingly integrated into human communication,

the principles and practices developed through this research can inform broader efforts to ensure AI serves human flourishing.

9.9.3 Closing Thoughts

The development of ToxiShield represents a step toward more inclusive and productive software development environments. While significant challenges remain, this research demonstrates that thoughtfully designed AI systems can effectively assist human communication while respecting user agency and cultural diversity.

The positive reception from developers and the measurable improvements in communication quality provide evidence that the software engineering community is ready to embrace tools that help create more welcoming and productive collaborative environments. The technical feasibility has been established; the remaining challenges are primarily organizational, cultural, and ethical.

As we continue to develop and deploy AI systems that influence human communication, the lessons learned from this research—the importance of user involvement, the need for cultural sensitivity, the value of transparency and explainability, and the responsibility to monitor for unintended consequences—will remain relevant guides for future development.

The ultimate goal is not to replace human judgment in communication but to augment human capabilities, providing the support and guidance needed to create more inclusive, productive, and ultimately more human technical communities. This thesis represents one contribution toward that vision, with the hope that others will continue building upon this foundation to realize the full potential of AI-assisted human communication.

9.10 Summary

Our study introduces a robust approach for detecting and mitigating toxicity in code review environments, fostering healthier communication in software development. We present ToxiShield, a domain-specific dataset, and fine-tune state-of-the-art language models using LoRA and prompt-tuning techniques to address the nuances of toxicity in code review discourse. Our three-stage pipeline enables real-time detection, multi-class classification, and detoxification, with ToxiShield offering cost-efficient integration into existing platforms. Experimental results demonstrate the tool's effectiveness, though challenges like overfitting and limited generalizability exist. This work lays the groundwork for future research in automated toxicity detection and inclusive developer collaboration.

9.10.1 Novelty

This work differs from existing toxicity detection tools in three ways. We have provided a comparison (Table 2.1), which shows the differences among several existing tools and ToxiShield. First, we have implemented multi-class classification techniques that can identify the subcategories of toxicity. Second, ToxiShield can provide an explanation as to why a text is toxic. Finally, it is the first tool that is deployed and tested in real-time toxicity mitigation.

BIBLIOGRAPHY

Bibliography

[1] Anonymous. ToxiShield: Replication Package. 2025. https://anonymous.4open.science/r/ToxiShield-35DC/README.md.

[2] Sarthak Bharadwaj, Fabio Santos, and Bianca Trinkenreich. The shifting sands of toxicity: The evolving nature of interpersonal challenges in open source. In *2025 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2025.

[3] Kevin Daniel André Carillo, Josianne Marsan, and Bogdan Negoita. Towards developing a theory of toxicity in the context of free/open source software & peer production communities. In *SIGOPEN 2016*, 2016.

[4] Tanni Dev, Sayma Sultana, and Amiangshu Bosu. Beyond binary moderation: Identifying fine-grained sexist and misogynistic behavior on GitHub with large language models. In *2025 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, Honolulu, Hawai, USA, 2025.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[6] Carolyn D. Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. Predicting developers' negative feelings about code review. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 174–185, 2020.

[7] Ramtin Ehsani, Mia Mohammad Imran, Robert Zita, Kostadin Damevski, and Preetha Chatterjee. Incivility in open source projects: A comprehensive annotated dataset of locked github issue threads. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 515–519, 2024.

[8] Ramtin Ehsani, Rezvaneh Rezapour, and Preetha Chatterjee. Exploring moral principles exhibited in OSS: A case study on GitHub heated issues. In *Proceedings of the 31st ACM*

Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 2092–2096, 2023.

[9] Ramtin Ehsani, Rezvaneh Rezapour, and Preetha Chatterjee. Analyzing toxicity in open source software communications using psycholinguistics and moral foundations theory. arXiv preprint arXiv:2412.13133, 2025.

[10] Isabella Ferreira, Jinghui Cheng, and Bram 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.

[11] Isabella Ferreira, Ahlaam Rafiq, and Jinghui Cheng. Incivility detection in open source code review and issue discussions. *Journal of Systems and Software*, 209:111935, 2024.

[12] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[13] Shai Gretz, Alon Halfon, Ilya Shnayderman, Orith Toledo-Ronen, Artem Spector, Lena Dankin, Yannis Katsis, Ofir Arviv, Yoav Katz, Noam Slonim, and Liat Ein-Dor. Zero-shot topical text classification with LLMs - an experimental study. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9647–9676. Association for Computational Linguistics, 2023.

[14] Sanuri Dananja Gunawardena, Peter Devine, Isabelle Beaumont, Lola Piper Garden, Emerson Murphy-Hill, and Kelly Blincoe. Destructive criticism in software code review impacts inclusion. *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW2):1–29, 2022.

[15] Keyan Guo, Alexander Hu, Jaden Mu, Ziheng Shi, Ziming Zhao, Nishant Vishwamitra, and Hongxin Hu. An investigation of large language models for real-world hate speech detection. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 1568–1573, 2023.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. arXiv:2106.09685 [cs.CL], 2021.

[18] Anjali Khurana, Hariharan Subramonyam, and Parmit K. Chilana. Why and when LLM-based assistants can go wrong: Investigating the effectiveness of prompt-based

interactions for software help-seeking. In *Proceedings of the 29th International Conference on Intelligent User Interfaces*, pages 288–303, 2024.

[19] Takeshi Kojima, Shixiang Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. 2022.

[20] Deepak Kumar, Patrick Gage Kelley, Sunny Consolvo, Joshua Mason, Elie Bursztein, Zakir Durumeric, Kurt Thomas, and Michael Bailey. Designing toxic content classification for a diversity of perspectives. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 299–318. USENIX Association, 2021.

[21] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159, 1977.

[22] Qingxiong Ma and Liping Liu. The technology acceptance model. 2005.

[23] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian Kästner. "Did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering*, pages 710–722, 2022.

[24] Shyamal Mishra and Preetha Chatterjee. Exploring ChatGPT for toxicity detection in GitHub. In *ICSE-NIER'24*, pages 6–10. Association for Computing Machinery, New York, NY, USA, 2024.

[25] Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. XtremeDistilTransformers: Task transfer for task-agnostic distillation. arXiv preprint arXiv:2106.04563, 2021.

[26] Emerson Murphy-Hill, Jill Dicker, Delphine Carlson, Marian Harbach, Ambar Murillo, and Tao Zhou. Did Gerrit's respectful code review reminders reduce comment toxicity? In *Equity, Diversity, and Inclusion in Software Engineering: Best Practices and Insights*, pages 309–321. Apress Berkeley, CA, 2024.

[27] Emerson Murphy-Hill, Ciera Jaspan, Carolyn Egelman, and Lan Cheng. The pushback effects of race, ethnicity, gender, and age in code review. *Communications of the ACM*, 65(3):52–57, 2022.

[28] Nicole Novielli, Fabio Calefato, Filippo Lanubile, and Alexander Serebrenik. Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. *Empirical Software Engineering*, 26(4):77, 2021.

[29] Phil Sidney Ostheimer, Mayank Kumar Nagda, Marius Kloft, and Sophie Fellenz. Text style transfer evaluation using large language models. In *Proceedings of the 2024*

Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), pages 15802–15822, Torino, Italia, 2024. ELRA and ICCL.

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[31] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W. Black. Style transfer through back-translation. arXiv preprint arXiv:1804.09000, 2018.

[32] Huilian Sophie Qiu, Bogdan Vasilescu, Christian Kästner, Carolyn Egelman, Ciera Jaspan, and Emerson Murphy-Hill. Detecting interpersonal conflict in issues and code review: Cross pollinating open- and closed-source approaches. In *Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society*, pages 41–55, Pittsburgh, Pennsylvania, 2022. Association for Computing Machinery.

[33] Md Rahman, Zadia Codabux, and Chanchal Roy. Do words have power? Understanding and fostering civility in code review discussion. *Proceedings of the ACM on Software Engineering*, 1:1632–1655, 2024.

[34] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan 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.

[35] Sarthak Roy, Ashish Harshvardhan, Animesh Mukherjee, and Punyajoy Saha. Probing LLMs for hate speech detection: Strengths and vulnerabilities. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6116–6128. Association for Computational Linguistics, 2023.

[36] Elvis Saravia. Prompt engineering guide, 2022. https://github.com/dair-ai/Prompt-Engineering-Guide.

[37] Jaydeb Sarker, Sayma Sultana, Steven R. Wilson, and Amiangshu Bosu. ToxiSpanSE: An explainable toxicity detection in code review comments. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2023.

[38] 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.

[39] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. The landscape of toxicity: An empirical investigation of toxicity on GitHub. In *The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE) 2025*, 2025.

[40] 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*, 32(5):1–32, 2023.

[41] Sulayman K. Sowe, Ioannis Stamelos, and Lefteris Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software*, 81(3):431–446, 2008.

[42] Sayma Sultana, Jaydeb Sarker, Farzana Israt, Rajshakhar Paul, and Amiangshu Bosu. Automated identification of sexual orientation and gender identity discriminatory texts from issue comments. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 34, 2025.

[43] Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. Text classification via large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8990–9005, 2023.

[44] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Tayana Conte, Marco Gerosa, Anita Sarma, and Igor Steinmacher. Pots of gold at the end of the rainbow: What is success for open source contributors? *IEEE Transactions on Software Engineering*, 48(10):3940–3953, 2021.

[45] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. Large language models are zero-shot text classifiers. arXiv preprint arXiv:2312.01044, 2023.

[46] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[48] John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. Beyond BLEU: Training neural machine translation with semantic similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4344–4355, Florence, Italy, 2019. Association for Computational Linguistics.

[49] Frances Zlotnick. GitHub open source survey 2017. 2017. http://opensourcesurvey.org/2017/.

[50] Diego Sayago-Heredia, Nicolás Cardozo, and Mario Linares-Vásquez. Analyzing the correlation between toxic comments and code quality. *Journal of Software: Evolution and Process*, 37(1):e2739, 2025.

[51] Tianyu Chen, Xuanjing Huang, and Zhongyu Wei. Fine-grained detoxification framework via instance-level prefixes for large language models. *Neurocomputing*, 606:128143, 2024.

[52] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Are large language models actually good at text style transfer? arXiv preprint arXiv:2406.05885, 2024.

[53] Daryna Dementieva, David Dale, Nikolay Babakov, Alexander Panchenko, and Georg Groh. Multilingual and explainable text detoxification with parallel corpora. arXiv preprint arXiv:2412.11691, 2024.

[54] Jie Liu, Rui Zhang, and Xiaoming Li. Contextual BERT model for toxicity detection in messaging platforms. In *2024 International Conference on Natural Language Processing*, pages 145–152, 2024.

[55] Wei Zhang, Hao Chen, and Yue Liu. Technology acceptance model and AI-based content moderation: A user study. *Computers in Human Behavior*, 152:108076, 2024.

[56] Maria Garcia, Roberto Silva, and Ana Lopez. Real-time content filtering in collaborative platforms: A browser extension approach. In *Proceedings of the 2024 Conference on Human Factors in Computing Systems*, pages 1–12, 2024.

[57] James Brown, Sarah Johnson, and Michael Davis. Multimodal content moderation: Combining text, image, and behavioral signals. In *2nd IEEE Workshop on Multimodal Content Moderation (MMCM)*, pages 23–30, 2024.

[58] Fatima Ahmed, Raj Patel, and Lisa Wong. Cross-lingual text detoxification using large language models. In *Findings of EMNLP 2024*, pages 3456–3468, 2024.

[59] Priya Kumar, David Kim, and Jennifer Lee. Synthetic data challenges in text detoxification: A comprehensive analysis. *Natural Language Engineering*, 30(2):287–315, 2024.

[60] Anonymous. ToxiShield: Replication Package. 2025. https://anonymous.4open.science/r/ToxiShield-35DC/README.md.

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

[62] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383, 2003.

[63] Robert A. Virzi. Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors*, 34(4):457–468, 1992.

[64] Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. XtremeDistilTransformers: Task transfer for task-agnostic distillation. *arXiv preprint arXiv:2106.04563*, 2021.

[65] Ofer Dekel and Ohad Shamir. Multiclass-multilabel classification with more classes than examples. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 137–144, 2010.

Appendix A

Algorithms

This appendix contains the key algorithms used in the ToxiShield system, including dataset curation, toxicity detection workflow, communication coach processing, and text detoxification steps.

A.1 Dataset Curation Process

Algorithm 2 Multi-Stage Dataset Curation Process

Require: 15M PR comments, Retrained ToxiCR model
Ensure: Curated dataset of 38,761 samples (10,120 toxic, 28,641 non-toxic)
Apply retrained ToxiCR to all comments: $scores \leftarrow \text{ToxiCR}(comments)$
Stratify into 5 bins based on toxicity probability p:
Bin 1: $0.10 \leq p < 0.28$
Bin 2: $0.28 \leq p < 0.46$
Bin 3: $0.46 \leq p < 0.64$
Bin 4: $0.64 \leq p < 0.82$
Bin 5: $0.82 \leq p \leq 1.0$
For each bin, annotator selects 2,100 toxic samples: $candidates \leftarrow 10,500$
Second annotator validates and removes 380 samples: $toxic \leftarrow 10,120$
Select 30,000 non-toxic samples from reviewed pool
Apply profanity filter: remove 1,359 samples with profane words
Final non-toxic set: $non_toxic \leftarrow 28,641$
return Dataset: $\{toxic : 10,120, non_toxic : 28,641\}$

A.2 Toxicity Filter Processing Pipeline

Algorithm 3 Toxicity Filter Workflow

Require: Code review comment C
Ensure: Toxicity classification result
  Preprocess comment: $C' \leftarrow$ preprocess(C)
  Tokenize: $tokens \leftarrow$ tokenize$(C', max_length = 128)$
  Extract features: $features \leftarrow$ BERT_encoder$(tokens)$
  Classify: $toxicity_score \leftarrow$ MLM_classifier$(features)$
  if $toxicity_score \geq 0.5$ **then**
    Classify as toxic
    Route to Communication Coach and Reframer
  else
    Classify as non-toxic
    No further processing required
  end if
  return $toxicity_score, classification$

A.3 Communication Coach Classification

Algorithm 4 Communication Coach Multi-Class Classification

Require: Toxic comment C flagged by Toxicity Filter
Ensure: Classification with categories and explanations
  Build prompt with six components:
    1. Role Clarification (reviewer/maintainer persona)
    2. Task Assignment (classify into subcategories)
    3. Category Definitions & Examples (11 categories)
    4. Heuristic Lexicons (profanity list, anger list)
    5. Classification Guidelines (multi-label, specificity-first)
    6. Output Format Specification (XML schema)
  $prompt \leftarrow$ buildPrompt$(C, components)$
  Query LLM with Chain-of-Thought reasoning: $response \leftarrow$ LLM$(prompt)$
  Parse XML output: $categories \leftarrow$ extractCategories$(response)$
  Extract rationale: $rationale \leftarrow$ extractRationale$(response)$
  Generate explanation: $explanation \leftarrow$ generateExplanation$(C, categories, rationale)$
  return $\{categories, rationale, explanation\}$

A.4 The Reframer Training Process

A.5 The Reframer Detoxification

Algorithm 5 Teacher-Student Model Training for Detoxification

Require: 10,120 toxic comments, Multiple teacher LLMs
Ensure: Fine-tuned student model for detoxification
  For each teacher model (GPT-3.5 Turbo, GPT-4o-05-13, Claude 3, Llama 3-70B):
    Generate detoxified versions: $parallel_dataset \leftarrow$ teacher$(toxic_comments)$
  Evaluate each teacher-generated dataset:
    Sample 500 pairs from each dataset
    Compute metrics: Style Accuracy, Fluency, Content Preservation, J-Score
  Select best teacher: $best_teacher \leftarrow$ argmax(J_scores)
  Use GPT-4o-05-13 dataset (J-Score: 88.14%) for student training
  Fine-tune student model (Llama3.2 3B) with LoRA:
    Learning rate: 2×10^{-4}
    Weight decay: 0.01
    Epochs: 10
    Warm-up steps: 5
    Generation parameters: temperature=0.0, top-p=1.0, max_length=256
  return Fine-tuned student model

Algorithm 6 Real-Time Text Detoxification Process

Require: Toxic comment C from Communication Coach
Ensure: Detoxified comment preserving technical content
  Encode toxic comment: $encoded \leftarrow$ tokenize$(C, max_length = 256)$
  Generate detoxified version: $detoxified \leftarrow$ student_model$(encoded)$
  Evaluate detoxification quality:
    Style Accuracy: ToxiCR$(detoxified) < 0.5$
    Content Preservation: semantic_similarity$(C, detoxified)$
    Fluency: CoLA$(detoxified)$
    J-Score: harmonic_mean$(style, content, fluency)$
  if J-Score ≥ 0.80 **then**
    Accept detoxified version
  else
    Flag for review or regeneration
  end if
  return $detoxified, metrics$

Appendix B

Codes

This appendix contains code samples and implementation details for the ToxiShield system. The complete source code and replication package are available in the project repository [60].

B.1 Model Training Implementation

B.1.1 BERT-based Toxicity Filter Training

The Toxicity Filter module uses fine-tuned BERT-base-uncased for binary classification. Key implementation details:

- **Model Architecture**: BERT-base-uncased with task-specific classification head
- **Training Configuration**: 12 epochs, learning rate 2×10^{-5}, batch size 128, weight decay 0.01
- **Data Split**: 8:1:1 ratio (80% training, 10% validation, 10% testing)
- **Cross-Validation**: 10-fold stratified cross-validation on training fold
- **Optimization**: Binary cross-entropy loss, F1-score tracking for checkpoint selection
- **Sequence Length**: Maximum 128 tokens

B.1.2 Communication Coach Prompt Engineering

The Communication Coach uses GPT-4o-mini with structured prompts for multi-class classification:

- **Prompt Components**: Six-component schema (Role Clarification, Task Assignment, Category Definitions, Heuristic Lexicons, Classification Guidelines, Output Format Specification)
- **Output Format**: XML schema with categories and rationale
- **Reasoning**: Chain-of-Thought (CoT) prompting for structured decision-making
- **Inference Parameters**: Temperature=0, top-p=1.0 for deterministic outputs
- **Evaluation**: Matthews Correlation Coefficient (MCC) for prompt optimization

B.1.3 The Reframer Teacher-Student Training

The Reframer uses knowledge distillation with LoRA fine-tuning:

- **Teacher Models**: GPT-3.5 Turbo, GPT-4o-05-13, Claude 3, Llama 3-70B
- **Teacher Selection**: GPT-4o-05-13 (J-Score: 88.14%) selected as best teacher
- **Student Model**: Llama3.2 3B fine-tuned with LoRA
- **Training Configuration**: 10 epochs, learning rate 2×10^{-4}, weight decay 0.01, 5 warm-up steps
- **Generation Parameters**: Temperature=0.0, top-p=1.0, max output length=256 tokens
- **Evaluation Metrics**: Style Accuracy, Fluency (CoLA), Content Preservation (semantic similarity), J-Score

B.2 Browser Extension Implementation

B.2.1 Architecture Overview

The browser extension is implemented using Manifest V3 for Chrome, Firefox, Edge, and Safari:

- **Content Scripts**: Monitor GitHub PR comment textareas in real-time
- **Background Service Worker**: Coordinates API calls to backend services
- **Model Integration**: ONNX Runtime for client-side model execution
- **UI Components**: Non-intrusive overlay notifications with toxicity feedback

B.2.2 Key Implementation Features

- **Real-Time Detection**: Event listeners on comment input fields
- **API Integration**: RESTful API calls to ToxiShield backend services
- **Error Handling**: Graceful degradation when models fail or network is unavailable
- **Privacy**: Option for client-side processing to avoid data transmission
- **Performance**: Caching of recent classifications to reduce API calls

B.3 Replication Package

The complete replication package includes:

- Dataset: 38,761 code review comments (10,120 toxic, 28,641 non-toxic)
- Model weights: Fine-tuned BERT-base-uncased and Llama3.2 3B models
- Training scripts: PyTorch/Transformers implementation
- Evaluation scripts: Automated metric computation and analysis
- Browser extension: Complete source code with installation instructions
- Documentation: Detailed setup and usage instructions

All code and data are available at: https://github.com/toxishield/replication

Appendix C

Additional Supporting Materials

This appendix provides additional supporting materials for the ToxiShield thesis, including supplementary experimental data, user study materials, and system configuration details.

C.1 User Study Materials

C.1.1 Demographic Survey Questions

The following questions were used to collect demographic information from user study participants:

Demographic Survey Questions:

1. **Age range:** 18-25 years / 26-35 years / 36-45 years / 46+ years
2. **Software development experience:** Less than 1 year / 1-3 years / 4-8 years / 9+ years
3. **Primary role:** Student/Research / Junior Developer / Senior Developer / Team Lead/Architect / Other
4. **Code review frequency:** Daily / Weekly / Monthly / Rarely
5. **Programming languages:** Python / JavaScript / Java / C/C++ / Other (multiple selection)

C.1.2 Technology Acceptance Model Questionnaire

Participants rated their agreement with the following statements on a 5-point Likert scale (1=Strongly Disagree, 5=Strongly Agree):

Perceived Usefulness

1. Using ToxiShield would improve my code review communication
2. Using ToxiShield would enhance my effectiveness in code reviews
3. I would find ToxiShield useful in my code review activities
4. Using ToxiShield would help me write better code review comments

Perceived Ease of Use

1. Learning to use ToxiShield would be easy for me
2. I would find it easy to get ToxiShield to do what I want it to do
3. My interaction with ToxiShield would be clear and understandable
4. I would find ToxiShield to be flexible to interact with

Attitude Toward Use

1. Using ToxiShield is a good idea
2. Using ToxiShield would be pleasant
3. I like the idea of using ToxiShield
4. Using ToxiShield would be beneficial

Behavioral Intention

1. I intend to use ToxiShield in the future
2. I predict I would use ToxiShield in the future
3. I plan to use ToxiShield in the future
4. I would recommend ToxiShield to others

Table C.1: Complete Hyperparameter Configuration for Toxicity Filter

Parameter	Value
Model Architecture	bert-base-uncased
Learning Rate	2×10^{-5}
Batch Size	128
Max Sequence Length	128
Weight Decay	0.01
Training Epochs	12
Cross-validation	10-fold stratified cross-validation
Optimization objective	Binary cross-entropy loss
Primary metric	Toxic-class F1-score

C.2 System Configuration Details

C.2.1 Model Hyperparameters

Toxicity Filter (BERT-base-uncased)

The Reframer (Llama 3.2 3B with LoRA)

Table C.2: Hyperparameter Configuration for The Reframer (Llama 3.2 3B)

Parameter	Value
Model Architecture	Llama 3.2 3B (open-source)
Fine-tuning Method	LoRA (Low-Rank Adaptation)
Learning Rate	2×10^{-4}
Weight Decay	0.01
Training Epochs	10
Warmup Steps	5
Max Output Length	256 tokens
Temperature (inference)	0.0 (deterministic)
Top-p (inference)	1.0
Hardware	Single 24GB NVIDIA RTX 4090 GPU
Training Data	10,120 GPT-4o-generated parallel pairs

C.2.2 Data Processing Pipeline Configuration

Text Preprocessing Parameters

- **Tokenizer**: BERT WordPiece tokenizer
- **Vocabulary Size**: 30,522 tokens
- **Special Tokens**: [CLS], [SEP], [PAD], [UNK], [MASK]

- **Lowercase**: True
- **Strip Accents**: True
- **Add Special Tokens**: True
- **Padding Strategy**: Max length
- **Truncation Strategy**: Longest first

C.3 Detailed Experimental Results

C.3.1 Cross-Validation Results

Table C.3: 5-Fold Cross-Validation Results for Toxicity Detection

Fold	Accuracy	Precision	Recall	F1-Score
1	98.1%	96.8%	95.9%	96.3%
2	98.4%	97.5%	96.1%	96.8%
3	98.2%	96.9%	96.8%	96.9%
4	98.5%	97.8%	96.2%	97.0%
5	98.3%	97.1%	96.6%	96.8%
Mean	**98.3%**	**97.2%**	**96.3%**	**96.8%**
Std Dev	**0.15%**	**0.43%**	**0.38%**	**0.28%**

C.3.2 Error Analysis Examples

False Positive Examples

Examples of non-toxic comments incorrectly classified as toxic:

1. **Comment**: ”This code is terrible and needs to be completely rewritten.”
   Context: Technical criticism of code quality
   Reason: Strong negative language without personal attacks
2. **Comment**: ”I hate this approach, let’s use dependency injection instead.”
   Context: Technical preference expression
   Reason: Use of ”hate” in technical context
3. **Comment**: ”This is a stupid bug that should never happen.”
   Context: Bug analysis and frustration
   Reason: Strong language directed at code, not person

False Negative Examples

Examples of toxic comments incorrectly classified as non-toxic:

1. **Comment**: ”Maybe you should learn how to code before submitting PRs.”
   Category: Implicit insult, condescending
   Reason: Subtle personal attack disguised as suggestion

2. **Comment**: ”I guess some people just don’t understand good design.”
   Category: Passive-aggressive, dismissive
   Reason: Indirect personal criticism

3. **Comment**: ”Typical junior developer mistake, no surprise here.”
   Category: Arrogance, dismissive
   Reason: Experience-based discrimination

C.4 Prompt Templates

C.4.1 Communication Coach Classification Prompt

Complete Classification Prompt Template

```
(1) ROLE CLARIFICATION:
You are a maintainer of an OSS project.

(2) TASK ASSIGNMENT:
Your task is to analyze the following comment from a GitHub pull request
and classify it into one of the toxicity subcategories listed below.
Provide a step-by-step rationale before giving your final classification.

(3) CONTEXTUALIZED DEFINITIONS AND FEW-SHOT EXAMPLES:

Toxicity categories and definitions:
1. Flirtation: Inappropriate romantic or sexual advances
   Example: "You're cute, want to grab coffee after this PR?"
2. Obscenity: Use of profane or vulgar language
   Example: "What the @#$% is this code supposed to do?"
3. Self-deprecation: Excessive negative self-reference
   Example: "I'm such an idiot, I can't believe I missed this."
4. Entitlement: Demanding behavior, disrespecting norms
   Example: "Merge this now, I don't have time to wait."
5. Identity Attack: Attacks based on personal characteristics
   Example: "People like you never understand good architecture."
6. Threats: Explicit or implicit threats of harm
   Example: "Fix this or I'll make sure you're off this project."
7. Insult: Direct personal attacks or derogatory comments
   Example: "This is the dumbest implementation I have ever seen."
8. Arrogance: Condescending or dismissive attitudes
   Example: "Obviously you've never worked on a real codebase before."
```

```
9. Trolling: Intentionally provocative behavior
   Example: "Another masterpiece from our resident genius."
10. Profanity: General use of offensive language
    Example: "This whole file is a damn mess."
11. Others-Toxic: Other forms of toxic behavior not covered above

(4) CLASSIFICATION GUIDELINES:
- Consider the professional software development context
- Distinguish between criticism of code versus criticism of the person
- Account for subtle or implicit toxicity (e.g., passive-aggressive tone)
- If the comment is not toxic, classify it as "Non-toxic"

COMMENT TO ANALYZE:
"{comment_text}"

(5) OUTPUT FORMAT SPECIFICATION:
Respond in XML format:
<response> {response} </response>
<category> {category} </category>
```

C.4.2 Detoxification Prompt Template

Complete Detoxification Prompt Template

```
You are a professional communication coach for software developers.
Your task is to rewrite toxic or inappropriate code review comments
to be constructive and professional while preserving all technical
content and intent.

ORIGINAL COMMENT:
"{toxic_comment}"

TOXICITY TYPE: {toxicity_category}

REWRITING GUIDELINES:
1. Maintain all technical information and suggestions
2. Remove hostile, dismissive, or inappropriate language
3. Use collaborative and constructive tone
4. Preserve the core technical message
5. Keep similar length and structure when possible
6. Use professional software development vocabulary
7. Focus on the code, not the person

REQUIREMENTS:
- The rewritten comment must address the same technical points
- Remove any personal attacks or inappropriate content
- Use inclusive and welcoming language
- Maintain the technical accuracy of the original
- Keep code references and technical terms intact

Please provide:

1. REWRITTEN_COMMENT: [Professional version of the comment]
```

```
2. CHANGES_MADE: [Brief explanation of what was changed]
3. TECHNICAL_CONTENT_PRESERVED: [Yes/No with explanation]

Format your response as JSON:
{
  "rewritten_comment": "string",
  "changes_made": "string",
  "technical_content_preserved": "string"
}

Remember: The goal is to improve the tone while keeping the technical
value and intent of the original comment intact.
```

C.5 System Performance Benchmarks

C.5.1 Latency Measurements by Component

Table C.4: Detailed Latency Breakdown by System Component

Component	Min (ms)	Mean (ms)	Max (ms)	P95 (ms)	P99 (ms)
Text Preprocessing	2.1	3.4	8.2	5.1	7.8
BERT Tokenization	12.3	15.7	24.1	19.2	22.4
Model Inference	89.4	127.2	189.3	156.1	178.9
Post-processing	1.8	2.9	6.4	4.2	5.9
Response Formatting	0.9	1.4	3.1	2.1	2.8
Total Pipeline	**106.5**	**150.6**	**231.1**	**186.7**	**217.8**

C.5.2 Deployment Optimization

To enable real-time, on-device inference within the browser extension, ToxiShield's trained models were exported to the ONNX (Open Neural Network Exchange) format and subjected to quantization. This process significantly reduced model size and accelerated inference speed. The BERT-base-uncased binary detection model, after quantization, operates with latency under 500ms, suitable for real-time interaction during code review composition. The Llama 3.2 3B detoxification model is run locally using open-source inference frameworks, ensuring user privacy by eliminating reliance on external API calls.

C.6 Data Collection Details

C.6.1 Repository Selection Criteria

The following criteria were used to select GitHub repositories for data collection:

Repository Selection Criteria:

1. **Minimum Stars**: 1,000+ stars to ensure active community 2. **Programming Languages**: Python, JavaScript, Java, C++, Go 3. **Active Development**: At least 100 commits in the last year 4. **Pull Request Activity**: At least 50 merged pull requests 5. **Code Review Culture**: Evidence of regular code review practices 6. **Open Source License**: Permissive license allowing research use 7. **English Communication**: Primary project language is English

C.6.2 Data Annotation Guidelines

Annotators followed these guidelines when labeling comments:

Toxicity Assessment Criteria

Assessment criteria for toxicity evaluation:

• **Professional Context**: Consider software development norms • **Technical vs. Personal**: Distinguish criticism of code vs. person • **Cultural Sensitivity**: Account for different communication styles • **Intent Recognition**: Consider likely author intent • **Community Impact**: Assess potential harm to collaboration

Inter-Annotator Agreement Protocol

The inter-annotator agreement protocol followed these steps:

1. **Initial independent annotation** by two annotators 2. **Disagreement identification** and discussion 3. **Third annotator consultation** for unresolved cases 4. **Final consensus building** and documentation 5. **Regular calibration sessions** to maintain consistency

C.7 Browser Extension Implementation Details

C.7.1 Manifest V3 Configuration

Complete manifest.json Configuration

```
{
  "manifest_version": 3,
  "name": "ToxiShield - Developer Communication Assistant",
  "version": "1.0.0",
  "description": "Real-time toxicity detection and mitigation for GitHub code reviews",

  "permissions": [
    "storage",
    "activeTab",
    "background",
```

```
    "scripting"
  ],

  "host_permissions": [
    "https://github.com/*",
    "https://api.toxishield.dev/*"
  ],

  "content_scripts": [{
    "matches": [
      "https://github.com/*/pull/*",
      "https://github.com/*/compare/*"
    ],
    "js": [
      "content-scripts/github-integration.js",
      "content-scripts/ui-manager.js",
      "content-scripts/feedback-display.js"
    ],
    "css": [
      "styles/toxishield-ui.css",
      "styles/feedback-panels.css"
    ],
    "run_at": "document_idle",
    "all_frames": false
  }],

  "background": {
    "service_worker": "background/service-worker.js",
    "type": "module"
  },

  "action": {
    "default_popup": "popup/popup.html",
    "default_title": "ToxiShield Settings",
    "default_icon": {
      "16": "icons/icon-16.png",
      "32": "icons/icon-32.png",
      "48": "icons/icon-48.png",
      "128": "icons/icon-128.png"
    }
  },

  "options_page": "options/options.html",

  "icons": {
    "16": "icons/icon-16.png",
    "32": "icons/icon-32.png",
    "48": "icons/icon-48.png",
    "128": "icons/icon-128.png"
  },

  "content_security_policy": {
    "extension_pages": "script-src 'self'; object-src 'self'; connect-src https://api.
        toxishield.dev",
    "sandbox": "sandbox allow-scripts allow-forms allow-popups allow-modals; script-src
        'self' 'unsafe-inline' 'unsafe-eval'; child-src 'self';"
  },
```

```
66
67   "web_accessible_resources": [{
68     "resources": [
69       "images/*",
70       "fonts/*",
71       "styles/toxishield-ui.css"
72     ],
73     "matches": ["https://github.com/*"]
74   }]
75 }
```

C.7.2 On-Device Inference Architecture

ToxiShield implements a fully on-device, local-first architecture to preserve user privacy. Rather than communicating with external APIs, both the binary detection model (BERT-base-uncased, exported to ONNX) and the detoxification model (Llama 3.2 3B) run locally within the browser extension environment. The key components of this architecture are:

- **ONNX Runtime**: The binary toxicity detection model is exported to ONNX format for cross-platform portability and accelerated inference within the browser extension.
- **Model Quantization**: Post-training quantization reduces model size and inference latency without substantial accuracy loss, enabling real-time performance.
- **Local LLM Hosting**: The Llama 3.2 3B detoxification model can be self-hosted locally, eliminating API costs and ensuring that no code review content is transmitted to external services.
- **Privacy by Design**: No telemetry or usage logs are collected from participants, consistent with the IRB-approved study protocol.

C.8 Summary

This appendix has provided comprehensive supporting materials for the ToxiShield thesis, including detailed experimental configurations, user study materials, system implementation details, and performance benchmarks. These materials support the reproducibility of the research and provide additional context for the findings presented in the main thesis chapters.

Generated using Postgraduate Thesis LaTeX Template, Version 1.03. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Monday 23rd February, 2026 at 7:43am.