ToxiShield: Promoting Inclusive Developer Communication through Real-Time Toxicity Filtering

MD AWSAF ALAM ANINDYA, Bangladesh University of Engineering and Technology, Bangladesh
SHOWVIK BISWAS, Bangladesh University of Engineering and Technology, Bangladesh
ANINDYA IQBAL, Bangladesh University of Engineering and Technology, Bangladesh
JAYDEB SARKER, University of Nebraska Omaha, USA
AMIANGSHU BOSU, Wayne State University, USA

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. To address this gap, we present ToxiShield, a browser extension for GitHub pull requests that is built using three modules: i) Toxicity Filter – to identify whether a text is toxic, ii) Communication coach – to facilitate just-in-time fine-grained toxicity categorization with explanations, and iii) The Reframer – that generates a revised, constructive alternative of a toxic text. For each module, we trained and evaluated multiple deep learning and Large Language Models (LLMs) to identify the best choice. A BERT-based binary detection model, trained on 38,761 code review samples, achieves 98% accuracy and an F1-score of 97% and is the selected one for the Toxicity Filter module. For the Communication Coach, prompt-tuned Claude 3.5 Sonnet achieved the best performance with 39%*MCC* and 42%*F*1 in multiclass toxicity classification with detailed reasoning. For Reframer, we evaluated five LLMs using a fine-tuning strategy on a dataset of 10,120 code review comments. The fine-tuned Llama 3.2 model achieves 95.27% style transfer accuracy, 97.03% fluency, 67.07% content preservation, and an 84% J-score. We further validated ToxiShield through a human evaluation using the Technology Acceptance Model with 10 participants, confirming its perceived usefulness and ease of adoption. ToxiShield sets a benchmark for advancing constructive communication in software engineering, driving inclusivity and healthier collaboration in open-source communities.

CCS Concepts: • **Software and its engineering → Collaboration in software development**; • **Collaboration in software development → Detoxification**.

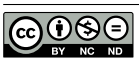
Additional Key Words and Phrases: toxicity, open source software, text style transfer, code review

ACM Reference Format:
Md Awsaf Alam Anindya, Showvik Biswas, Anindya Iqbal, Jaydeb Sarker, and Amiangshu Bosu. 2026. ToxiShield: Promoting Inclusive Developer Communication through Real-Time Toxicity Filtering. *Proc. ACM Softw. Eng.* 3, FSE, Article FSE123 (July 2026), 23 pages. https://doi.org/10.1145/3808130

1 Introduction

Successful Open Source Software (OSS) projects require strong coordination and teamwork among contributors [46, 50]. Developers primarily collaborate using text-based channels, including mailing lists, code reviews, version control systems, and bug trackers. Despite this need for collaboration,

Authors' Contact Information: Md Awsaf Alam Anindya, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, 1505114.maaa@ugrad.cse.buet.ac.bd; Showvik Biswas, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, 1805068@ugrad.cse.buet.ac.bd; Anindya Iqbal, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, anindya@cse.buet.ac.bd; Jaydeb Sarker, University of Nebraska Omaha, Omaha, USA, jsarker@unomaha.edu; Amiangshu Bosu, Wayne State University, Detroit, USA, amiangshu.bosu@wayne.edu.



ACM 2994-970X/2026/7-ARTFSE123
https://doi.org/10.1145/3808130

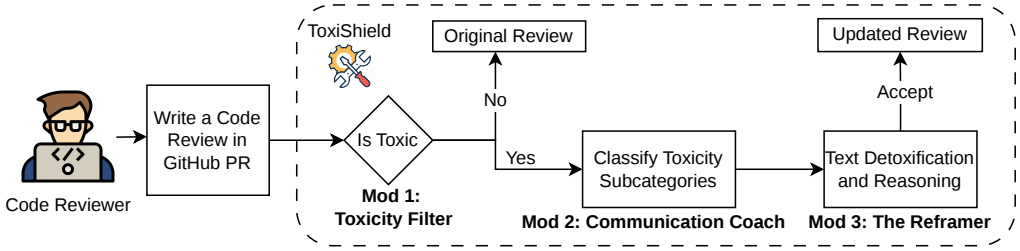


Fig. 1. Motivational Workflow of ToxiShield

research indicates that communication within OSS communities frequently becomes unprofessional and toxic [10, 21, 28, 39, 43–45].

This hostility produces severe consequences. It creates significant barriers for new participants [39] and causes active contributors to leave the project [43]. Furthermore, toxic interactions disproportionately affect minority and junior developers [17]. Such behavior leads to emotional distress and burnout, which subsequently diminish developer productivity [38, 39]. Over time, these interpersonal conflicts degrade the overall health of the community by undermining diversity [17, 28] and sustaining ongoing disputes [8, 38]. Consequently, this toxic environment ultimately compromises the quality of the software produced [28].

To counter these issues, many OSS projects have adopted Codes of Conduct (CoC) to encourage respectful environments; however, their practical impact remains limited [2, 31]. The primary obstacle is enforcement. Current moderation typically relies on manual validation, a labor-intensive process that is unsustainable for large-scale communities [28]. The inadequacy of manual moderation 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 [39, 42, 45] but also nuanced antisocial behaviors, including incivility [14, 38], destructive criticism [17], sexism [6, 47], and pushback [8].

Nevertheless, existing approaches for managing toxic interactions in OSS are primarily reactive. They operate on a post hoc basis, using classifiers to flag problematic text for later review and removal [45]. This delay is critical because, by the time an intervention occurs, significant damage to contributor relationships may have already been done [38, 56]. To foster genuinely healthier communication, a paradigm shift is necessary from reactive flagging to proactive prevention. Inspired by real-time grammar checkers in platforms like Grammarly, we aim to develop a tool integrated directly into developers' communication channels. This tool would not only flag potentially toxic text during composition but also provide actionable feedback, explaining why the text is problematic and suggesting neutral or civil alternatives. While recent work by Rahman *et al.* moved in this direction by developing a customized Text-to-Text Transfer Transformer (T5) model to rephrase uncivil comments, their solution has two major limitations. First, it lacks the real-time integration necessary for proactive moderation. Second, and more importantly, it fails to explain its reasoning. This explanatory capability is essential for fostering long-term behavioral change and helping users navigate the cultural nuances of toxic communication [25].

To address these limitations, we propose **ToxiShield**, a comprehensive framework designed to proactively detect and mitigate toxicity in SE communication, with a specific focus on code review interactions. Unlike post hoc solutions, ToxiShield operates in real-time as a developer drafts feedback. The system identifies and mitigates potential toxicity through a three-stage process, as illustrated in Figure 1, comprising the following key modules.

Module 1: Toxicity Filter: Acting as the first line of defense, this module performs a binary classification to distinguish between toxic' and non-toxic' code review comments. When a comment is flagged as toxic, the filter automatically triggers a downstream workflow, routing the problematic text to specialized modules for granular analysis and remediation.

Module 2: Communication Coach: The Communication Coach functions as an automated pedagogical agent aimed at fostering long-term behavioral improvement. Upon receiving a comment identified as toxic, it conducts a detailed analysis to classify the text into specific subcategories of toxicity, such as Insult, Threat, or Obscenity. The module then generates a targeted explanation detailing precisely why the comment is problematic, thereby transforming a standard moderation event into a constructive learning opportunity for the user.

Module 3: The Reframer: This module serves as a generative remediation assistant designed to detoxify harmful discourse. When a comment is flagged, The Reframer analyzes the author's core intent and generates a revised, constructive alternative that preserves the technical message while adhering to professional standards. This suggested revision is presented to the user alongside a brief rationale, explaining the specific changes made and why the new version is more effective for maintaining a collaborative environment.

To identify the optimal architecture for each module, we trained and evaluated multiple machine learning models. Our evaluation reveals that for the primary detection task, lightweight models offer superior performance, surpassing even Large Language Models (LLMs) such as GPT-4o. Specifically, the *BERT-base-uncased* model achieved the highest performance for binary toxic text detection, attaining an accuracy of 98% and an $F1$-score of 97%. For the more granular task of classifying toxicity subcategories, Claude 3.5 Sonnet yielded the best overall results, achieving a macro $F1$-score of 42% (average $F1$-score of 75%) and a macro Matthews Correlation Coefficient (MCC) of 39% (average MCC of 66%). Finally, regarding detoxification, our fine-tuned Llama 3.2 model outperformed other state-of-the-art LLMs, achieving a J-Score of 84%. Beyond raw metrics, these techniques successfully generated high-quality rationales for the suggested changes, demonstrating the efficacy of language models in addressing complex linguistic challenges. We validated our approach through a comprehensive evaluation comprising both automated metrics and user studies to ensure the generated alternatives maintained the original context and intent while effectively reducing toxicity. A user study involving 10 software developers, evaluated via the Technology Acceptance Model (TAM), confirmed across four key metrics that our extension effectively aids in reducing toxicity in code reviews. The key contributions of this study are as follows:

- A curated dataset of toxic code reviews paired with their corresponding non-toxic alternatives.
- A robust multiclass toxicity classifier capable of categorizing specific types of toxicity.
- The development of **ToxiShield**, a browser extension-based tool for code review detoxification. Our replication package, including code, data, and experimental results, is available at [1].

The remainder of this paper is organized as follows. Sections 2, 3, and 4 detail the design and evaluation of the three core modules of ToxiShield, respectively. Section 5 presents the results of our integrated user evaluation. We discuss our findings in Section 6, address threats to validity in Section 7, review related works in Section 8, and conclude the paper in Section 9.

2 Module 1: Toxicity Filter

A robust binary toxicity detector serves as the foundational entry point of our pipeline, acting as a critical filter for all incoming comments. Effective binary filtration is essential to prevent the downstream classification and detoxification modules from being inundated with non-toxic content, thereby optimizing computational efficiency and ensuring system responsiveness. While prior research has introduced binary toxicity detectors for the SE domain [38, 39, 45], these approaches

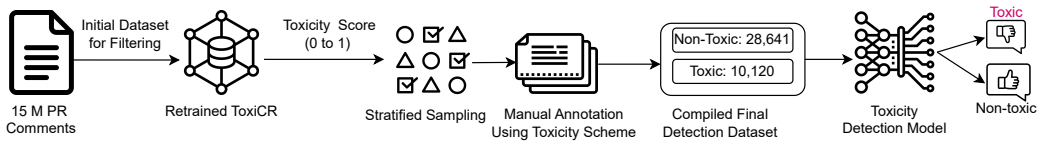


Fig. 2. Workflow of the Toxicity Filter Module where 15 M PR comments are selected from [44].

often prioritize offline classification accuracy over the latency constraints required for seamless integration into real-time developer workflows. For instance, the current state-of-the-art tool, ToxiCR [45], achieves strong performance metrics (88.9% F1-score, 95.8% accuracy); however, recent studies indicate that its efficacy diminishes when processing nuanced or sarcastic expressions [38], which are common in collaborative coding environments. Consequently, the primary objective of this module is to accurately classify code review comments as either toxic or non-toxic, addressing these limitations. Figure 2 illustrates the methodology adopted for Module 1, the design and evaluation of which are detailed in the following subsections.

2.1 Dataset Construction

Conceptualization of Toxicity in Code Reviews. Recent research within the OSS community has documented a broad spectrum of antisocial behaviors, ranging from general toxicity [28, 45] to specific manifestations such as destructive criticism [17], pushback [8], and incivility [13]. In this work, we target *toxicity* within code reviews. We adopt the operational definition of toxicity provided by Sarker *et* al. [44], whose taxonomy synthesizes prior SE findings [13, 28, 45] into 11 distinct categories. Therefore, within the context of this study, a text is considered toxic if it falls into one of the following categories: *Profanity, Trolling, Insult, Self-deprecation, Entitlement, Identity Attack, Threats, Obscenity, Arrogance, Flirtation*, or *Object-Directed Toxicity*. Consequently, behaviors such as *sarcasm* and *condescension* are not designated as toxic solely on the basis of their expression, as these behaviors may serve benign social functions or lack a standardized definition in SE taxonomies. We therefore label them as toxic in our binary dataset only when they meet the definitional criteria for one of the 11 core subcategories. For example, the comment *"Grandma, what long lines you have?"* is classified as toxic not merely due to its sarcastic tone, but because it constitutes a veiled insult devoid of constructive feedback.

Dataset Selection To the best of our knowledge, the largest existing binary toxicity dataset in the SE domain is the ToxiCR corpus [45], which contains 19,651 code review comments. However, this dataset exhibits a significant class imbalance; it is heavily skewed towards profanity (comprising nearly half of the samples), leaving other forms of toxicity underrepresented. Furthermore, alternative datasets with toxicity labels are limited in scale [9, 39]. To construct a more diverse and representative dataset, we leveraged the corpus from a recent study by Sarker *et* al. [44], which mined 101 million comments across 2,828 GitHub-based OSS projects. Aligning with our specific focus on code reviews, we isolated Pull Request (PR) comments from this repository. After excluding 86 million issue comments, we were left with approximately 15 million PR comments.

Dataset Filtering Given the sparsity of toxic comments in OSS discussions [37, 43], automated classification is essential for efficient large-scale dataset curation. To improve detection capabilities, we retrained the ToxiCR model using a composite dataset merging the Sarker *et* al. corpus (19,651 samples) [45] and the Rahman *et* al. dataset (4,700 samples) [38]. This retraining process significantly enhanced the model's robustness and calibration regarding the nuances of code review discourse. We then deployed this retrained model across the 15 million PR comments, generating a probability score (0 to 1) for each entry. While the standard ToxiCR framework classifies text as toxic at a

Table 1. Dataset Annotation with Examples: 10,120 toxic and 28,641 non toxic samples in the final dataset.

Probability	Example of PR comments	Rationale
$0.10 \leq p < 0.28$	Please try to understand the code you're commenting on.	Represents Condescension (Implicit Toxicity)
$0.28 \leq p < 0.46$	Oh, it's called Trainwreck. Surprised you have you not heard of it?	Represents Sarcasm
$0.46 \leq p < 0.64$	you are a liar!?	Insulting comment
$0.64 \leq p < 0.82$	That's just wacky. As I said in the issue report, bizarre that it would even miscompile in debug mode in MSVC.	Expresses Object-Directed Toxicity (Dismissive Tone)
$0.82 \leq p \leq 1.0$	Yep, fuck those technomancers, always hated.	Expresses Profanity

threshold of ≥ 0.5 [45], prior work indicates this threshold yields a high false negative rate for underrepresented toxicity classes [38]. Consequently, we expanded our candidate search space to include probabilities ranging from 0.10 to 1.0. This broader range allows us to capture a more diverse spectrum of toxicity inherent to code reviews.

Dataset Labeling To mitigate potential labeling bias, two annotators were trained on the conceptual framework detailed in Section 2.1. We employed a stratified sampling strategy, dividing the unlabeled data into five distinct bins based on the toxicity probability (p) assigned by the model (see Table 1). Because the model tends to assign high probabilities primarily to explicit toxicity (e.g., profanity), this stratification enabled us to systematically target implicit or subtle toxicity in the lower-probability bins. From each bin, the first annotator manually reviewed entries until 2,100 toxic samples were identified, resulting in an initial corpus of 10,500 candidates. A second annotator independently validated these selections, removing 380 instances deemed non-toxic. This rigorous validation process yielded a final, high-quality dataset of 10,120 toxic samples.

Non-toxic Samples To compile the non-toxic component of the dataset, we randomly selected 30,000 samples from the pool of 15 million PR comments, which were subsequently reviewed and confirmed as non-toxic by the first annotator. To further ensure data purity, we applied a keyword-based filtering process to these candidates, removing 1,359 samples containing explicit profanity. This rigorous validation yielded a final set of 28,641 non-toxic samples. Combining these with our toxic corpus resulted in a comprehensive dataset of 38,761 samples, comprising **10,120 toxic** and **28,641 non-toxic** PR reviews. Crucially, by selecting samples across the full spectrum of ToxiCR's [45] probability distribution, we mitigate the risk of inheriting the model's existing biases in our subsequent evaluation.

2.2 Model Training

To identify the optimal architecture for ToxiShield's Toxicity Filter, we evaluated a diverse set of candidate models, detailed in Table 2, using the dataset described in Section 2.1. Our selection strategy encompassed both Masked Language Models (MLMs) and Large Language Models (LLMs). Specifically, we fine-tuned *BERT-base-uncased* [7] and *Xtreme-DistilBERT* [30], and incorporated the pre-trained ToxiCR model [45] to serve as a domain-specific baseline. Additionally, we employed GPT-4o and GPT-3.5 as LLM-based detectors, following the benchmarking methodology of Rahman *et al.* [38]. For the experimental setup, the dataset was partitioned into training (80%), validation (10%), and testing (10%) sets. We initialized the MLMs with their publicly available pre-trained weights and appended a task-specific classification head to each. Input sequences were tokenized with a maximum length of 128 tokens. Fine-tuning was conducted over 12 epochs using

Table 2. Performance of binary toxicity detection models. Bold indicates the highest score per metric.

Models	Non-toxic			Toxic			Accuracy
	P_0	R_0	$F1_0$	P_1	R_1	$F1_1$	
BERT-base-uncased (fine-tuned)	**0.98**	**0.99**	**0.99**	**0.98**	**0.96**	**0.97**	**0.98**
Microsoft/Xtreme DistilBert (fine-tuned)	**0.98**	0.98	0.98	0.97	0.93	0.95	**0.98**
GPT-4o (prompted)	0.84	0.99	0.91	0.96	0.48	0.64	0.86
ToxiCR (off-the-shelf)	0.95	0.89	0.92	0.74	0.87	0.80	0.87

a learning rate of 2×10^{-5}, a batch size of 128, and a weight decay of 0.01, with the objective of minimizing binary cross-entropy loss. To ensure robustness, we applied 10-fold stratified cross-validation within the training split and monitored per-epoch validation performance to select the checkpoint that maximized the toxic-class F1-score for final testing.

2.3 Model Evaluation

To evaluate our detection models, we employed standard classification metrics: precision, recall, accuracy, and the F1-score. Given the critical necessity of accurately identifying toxic content in code reviews, we designated the F1-score of the toxic class as our primary performance metric. Table 2 presents a comparative performance analysis of our fine-tuned models, *BERT-base-uncased* and *Xtreme-DistilBERT*, against established baselines, including the off-the-shelf ToxiCR model and a zero-shot prompted GPT-4o. Among the evaluated architectures, the fine-tuned *BERT-base-uncased* model demonstrated superior performance, achieving the highest precision (0.98) and recall (0.96), with a toxic-class F1-score of 0.97. *Xtreme-DistilBERT* followed closely with an F1-score of 0.95. These results underscore the robustness of *BERT-base-uncased* in identifying toxic discourse. Consequently, we selected it as the default detection engine for ToxiShield's Toxicity Filter module.

2.4 Error Analysis of Toxicity Filter Module

We conducted an error analysis on our best model, *BERT-base-uncased*, which misclassified 59 of 3,877 test samples (23 False Positive (FPs), 36 False Negatives (FNs)). Using an open-coding approach based on established SE research [45, 47], two authors categorized these failures into four types.

i) Pragmatic and Contextual Misinterpretation: The model often failed to detect sarcasm, passive-aggressiveness, and mockery, accounting for most misclassifications (65% of FPs, 81% of FNs). Positive words frequently masked toxic intent, causing FNs. For instance, the model prioritized "hilarious" over the mocking tone in: "This means you can make pizza-flavoured ice cream, you tell me that isn't a hilarious notion". Conversely, it detected FPs by over-indexing on negative words during valid technical criticism, incorrectly flagging: "this thing is still just terrible, average of 9s".

ii) Domain-Specific Terminology Misinterpretation: These errors arise when standard technical identifiers, jargon, or references are misidentified as general English profanity or aggression. This category accounted for 26% of FPs and 11% of FNs. A PR comment *"convert anal.autoname into 'int'"*, which was incorrectly flagged as toxic (FP). The model failed to recognize *"anal"* as a standard abbreviation for "*analysis*" within the software domain context.

iii) Misinterpretation of Self-Deprecation: The model occasionally conflated benign self-deprecation, a common, non-toxic behavior in code reviews [22, 28, 44, 45], with hostility (9% of FPs, 5% of FNs). For instance, it incorrectly flagged "i dig it. will do when I'm not drunk" (FP), likely reacting to the keyword "drunk" rather than the humor.

iv) Obfuscation and Tokenization Failures: We identified a single instance (3% of FNs) where the model failed due to adversarial text modification designed to evade filters. The comment "*Yes*

there is, i ju- F U C K." was classified as non-toxic (FN). In this case, the insertion of spaces between characters obfuscated the profanity, causing a tokenization failure that prevented accurate detection.

Key Finding 1: *Fine-tuned BERT-base-uncased demonstrated the best performance in detecting toxic and non-toxic comments for code review text, when trained on our new dataset.*

3 Module 2: Communication Coach

The Communication Coach is an automated pedagogical module designed to address the nuances of toxic behavior in software engineering. Because toxicity encompasses diverse forms like insults and entitlement [28, 44], binary classification often oversimplifies the problem and obscures specific harms that disproportionately affect underrepresented groups [11]. Moving beyond simple flagging upon detection (Section 2), Communication Coach classifies the comment into a subcategory and provides targeted explanatory feedback. This granular approach transforms standard moderation into a constructive learning opportunity, fostering transparency and long-term behavioral change. The following sections detail the design and evaluation of this multi-class classification module.

3.1 Dataset for Toxicity Subcategories

Prior SE studies have identified various subcategories of toxicity [9, 28, 44]. Among these, Sarker *et al.* [44] offer the most comprehensive taxonomy, identifying 11 distinct toxicity types in GitHub pull requests. Given the scarcity of fine-grained annotated data, we utilized their dataset of 600 samples, which contains 532 verified toxic instances. To create a balanced distribution for inference, we supplemented this core with 600 non-toxic comments drawn from our baseline dataset (Section 2). The resulting multiclass dataset comprises 1,200 code review comments (532 toxic samples and 668 non-toxic samples) spanning 12 distinct categories, including the non-toxic class. Among the 532 toxic samples, 140 instances are labeled with two distinct subcategories (e.g., *Profanity* and *Insult*), and three instances are labeled with three subcategories, consistent with the primary labeling schema of Sarker *et* al. [44]. We explicitly preserve these overlapping annotations, thereby formulating the problem as a multi-label, multi-class classification task [5].

3.2 Prompt-based Design For LLMs

To classify toxicity subcategories, we leveraged LLMs, which have demonstrated strong performance in prompt-based multi-class tasks [16, 48, 52]. While our detection module relies on MLMs for computational efficiency (Section 2), we transitioned to LLMs for this phase to exploit their generative reasoning capabilities. We evaluated state-of-the-art models from the OpenAI, Claude, and Llama families, employing iterative prompt optimization to maximize classification accuracy [18, 40, 48, 52].

A pivotal design decision in this phase was the adoption of *In-Context Learning* (prompting) over fine-tuning models like Llama 3.2. This choice was driven by the severe data scarcity inherent to fine-grained toxicity classification. Robust fine-tuning necessitates a substantial volume of diverse examples to ensure generalization. However, our subcategory dataset contains only 532 toxic instances distributed across 11 distinct categories, with long-tail classes such as *Arrogance* and *Obscenity* containing negligible samples. Such a limited volume is insufficient for fine-tuning multi-billion parameter models without inducing severe overfitting [20, 49]. Consequently, we leveraged prompt engineering, a strategy that has demonstrated efficacy in both general toxic text classification [18, 40] and recent SE-specific multiclass tasks [6]. This approach enables us to harness the sophisticated reasoning capabilities of LLMs while circumventing the high data requirements of parameter optimization.

Choice of LLMs: We conducted a comprehensive evaluation of LLMs in our dataset. Our model selection includes three families of models, covering both proprietary and open-source systems: GPT4o-mini, GPT 4o, Claude 3.5 Sonnet, Claude 3.7 Sonnet, Llama 3.1 70B, and Llama 3.3 70B.

Prompt Schema and Components: To operationalize the subcategory classification approach, we designed prompts capable of classifying comments into single or multiple toxicity categories or identifying them as non-toxic. The structure of our baseline prompt is defined as follows:

Elements of Basic Prompt for Multiclass Classification

(1) **Role Clarification:** You are a maintainer of an OSS project. Your task is to classify the subcategories of toxicity.
(2) **Task Assignment:** Please assign one or more toxicity categories to the input comment and to provide a brief rationale (step-by-step).
(3) **Contextualized Definitions & Few-Shot Examples:** To mitigate semantic drift, we provided concise, iteratively refined definitions for all 11 subcategories (e.g., *Identity Attack*, *Entitlement*). Each definition was paired with at least one domain-specific example to illustrate the boundary between technical feedback and toxicity.
(4) **Classification Guidelines:** If the comment is not toxic, respond with "Non-Toxic" and a supporting explanation.
(5) **Output Format Specification:** Respond in XML format: `<response> {response} </response> <category> {category} </category>`. Now classify the following comments.

This prompt is structured into five strategic components designed to optimize LLM performance: (i) role clarification, (ii) task assignment, (iii) category definitions supplemented by representative examples, (iv) explicit operational guidelines, and (v) a strict XML output specification. We designate this initial configuration as *Prompt 1*, which serves as the baseline for subsequent iterative refinements. Consistent with our binary classification framework (Section 2), we align our subcategory definitions directly with the taxonomy established by Sarker *et* al. [44]. Comprehensive details regarding the prompt architecture and its evolution are provided in our replication package [1].

Prompt Optimization and Iterative Refinement Prompt engineering has emerged as a critical discipline for maximizing LLM performance in domain-specific tasks [41], significantly reducing the cognitive load on developers [23]. Within ToxiShield, we implemented a unified prompting framework across two key modules: (i) The Communication Coach (Section 3), which assigns specific toxicity subcategories, and (ii) The Reframer (Section 4), which rewrites toxic comments into professional alternatives. Our approach is grounded in three core principles: structured reasoning, iterative validation against quantitative metrics, and schema-constrained outputs.

First, to enhance the model's inferential capabilities [24], we adopted Chain-of-Thought (CoT) prompting [54]. This technique decomposes the complex tasks of classification and detoxification into discrete, logical steps. Specifically for classification, CoT compels the model to articulate the rationale behind a specific label, thereby improving both accuracy and interpretability. Second, to mitigate hallucinations and reduce variability in toxicity subcategory classification, we employed a systematic, iterative prompt optimization strategy. Aligning with established methodologies for prompt-based multi-class classification [6, 18, 40, 48, 52], we refined our prompts through five distinct evolutionary stages (Figure 3). This evolution progressed from generic instructions to precise, persona-anchored directives. We explicitly conditioned the model to adopt the persona of an experienced OSS maintainer, ensuring it could distinguish constructive technical criticism

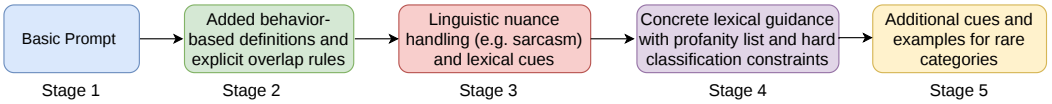


Fig. 3. Communication Coach: Iterative Prompt Evolution

from actual toxicity. At each stage, performance was rigorously assessed against a validation set using the Matthews Correlation Coefficient (MCC), a metric widely recognized for its robustness in multi-class classification tasks [6]. We specifically selected GPT-4o for the prompt optimization phase because its advanced instruction-following capabilities make it an ideal engine for structuring complex classification tasks. The five stages in our prompt optimization are as follows.

- *Stage 1 – The Baseline (Zero-Shot):* We commenced with a foundational zero-shot prompt that provided only class names and high-level classification instructions. This stage established a performance baseline, evaluating the model's inherent capabilities prior to the introduction of domain-specific guidance (Section 3.2).
- *Stage 2 – Operational Definitions (Observable Cues):* Manual analysis revealed that the model struggled with abstract definitions. Consequently, we re-framed the prompt to prioritize observable behavioral cues. Key iterations included: (i) replacing abstract definitions with actionable, behavior-based criteria for each subcategory; (ii) integrating canonical few-shot examples to ground the model's understanding of complex classes; (iii) explicitly distinguishing overlapping categories (e.g., *Insult* vs. *Trolling*) to resolve edge cases; and (iv) imposing strict output constraints to ensure parsing reliability on the full test set.
- *Stage 3 – Detecting Subtle Nuance:* To address the model's difficulty in detecting implicit toxicity, we refined the prompt to capture subtle nuances such as condescension and sarcasm. Iterations involved: (i) explicitly defining *Sarcasm* and *Irony* to prevent the literal interpretation of positive words in negative contexts; (ii) introducing initial lexical markers for strong sentiment; and (iii) expanding constraints to account for variations in offensive slang and non-standard profanity.
- *Stage 4 – Lexical and Rule-Based Constraints:* While Stage 3 improved tone detection, the model still missed toxicity driven by specific vocabulary. We addressed this by integrating lexical resources and logic rules: (i) embedding a curated list of profanity terms as high-probability toxicity indicators; (ii) incorporating an Anger List (e.g., capitalization patterns, aggressive punctuation) to detect heightened emotional states; (iii) applying negative constraints to reduce false positives arising from technical disagreements; and (iv) implementing critical logic rules (e.g., `IF profanity is used positively, DO NOT label as Insult`) to enforce consistency.
- *Stage 5 – Rare Category Refinement:* Evaluations of Stage 4 indicated persistent under-performance in rare categories, specifically *Arrogance* ($N = 5$) and *Identity-Attack* ($N = 17$). We performed targeted refinement by expanding these definitions with granular cues and additional representative examples to improve recall.

We selected the final prompt based on validation performance while preserving the basic prompt components. Table 3 details the iterative evolution of our prompt design with the GPT-4o model, highlighting the performance bottlenecks encountered, the specific refinements applied, and the resulting trajectory of the model's performance. From the prompt evolution, we observed that Stage 4 achieved the best macro MCC score, even though it missed one rare category, "Arrogance". We have used this best prompt for further validation of other models.

Table 3. Prompt Refinement with Macro MCC scores for Communication Coach

Stg.	Example	Misclassification Reason	MCC
1	Let's just remove this for now ... kill all the commented-out source.	Prompt has only abstract definitions and "kill" is misclassified as a threat.	0.31
2	Helluva test. I had to look at it for a while before it oozed through. Nice job!	Prompt does not contain explicit instructions to handle sarcasm.	0.30
3	@Imagebard feck off	Prompt does not identify "feck" as a profane word due to a misspelling, and it not having a list of commonly misspelled profanities.	0.31
4	Go ahead and merge this non-sense. I can work around it.	Prompt does not have cues in it that can help the model detect subtle hints of arrogance.	0.37
5	Oh boy this is sexy :smiley_cat:	Prompt misinterprets comment as obscene due to the presence of "sexy", a common marker for sexual content, and its definition of an obscene comment having any sexual content or references to sexual acts.	0.35

Table 4. Comparison of LLMs for Multi-Class Classification For Best Model in Each Family

		Precision		Recall		F1		MCC	
Model Name	**EM**	Avg	Macro	Avg	Macro	Avg	Macro	Avg	Macro
GPT 4o	**0.69**	0.72	0.41	**0.77**	**0.42**	0.74	0.40	0.65	0.37
Claude 3.5 Sonnet	0.63	**0.74**	0.44	0.75	**0.42**	**0.75**	**0.42**	**0.66**	**0.39**
Llama 3.3 70B	0.61	0.69	0.43	0.74	0.36	0.71	0.37	0.60	0.34

3.3 Evaluation of Communication Coach

Evaluation Metric We measured multiclass model performance using exact match (EM), precision, recall, F1, and MCC scores under two aggregations: (i) **Avg** (aggregate over all instances) and (ii) **Macro** (unweighted average across 12 caegories), as presented in Table 4.

Evaluation Results Table 4 presents the multi-class classification performance of the leading models from each LLM family (GPT-4o, Claude 3.5 Sonnet, and Llama 3.3 70B). For this final evaluation, we utilized the optimal prompt (Stage 4) identified in Table 3 and employed deterministic decoding (temperature = 0) to ensure reproducibility. Overall, Claude 3.5 Sonnet demonstrated the most balanced detection capability across the 12 categories, achieving the highest Macro F1-score (0.42) alongside a Macro MCC of 0.39. GPT-4o performed competitively (Macro F1: 0.40, MCC: 0.40), while Llama 3.3 70B exhibited lower efficacy (Macro F1: 0.37, MCC: 0.34). Interestingly, while Claude provided superior balance across minority classes, GPT-4o favored exact match accuracy 0.69. These findings highlight considerable variability in how contemporary LLMs handle imbalanced, multi-label toxicity categorization.

3.4 Error Analysis of Communication Coach

Confusion Matrix To assess the detection performance of the best model (Claude 3.5 Sonnet) with the most optimized prompt on multiclass classification, we conducted an evaluation in Figure 4. To rigorously resolve the ambiguity in the multi-label ground truth for this analysis, a human rater

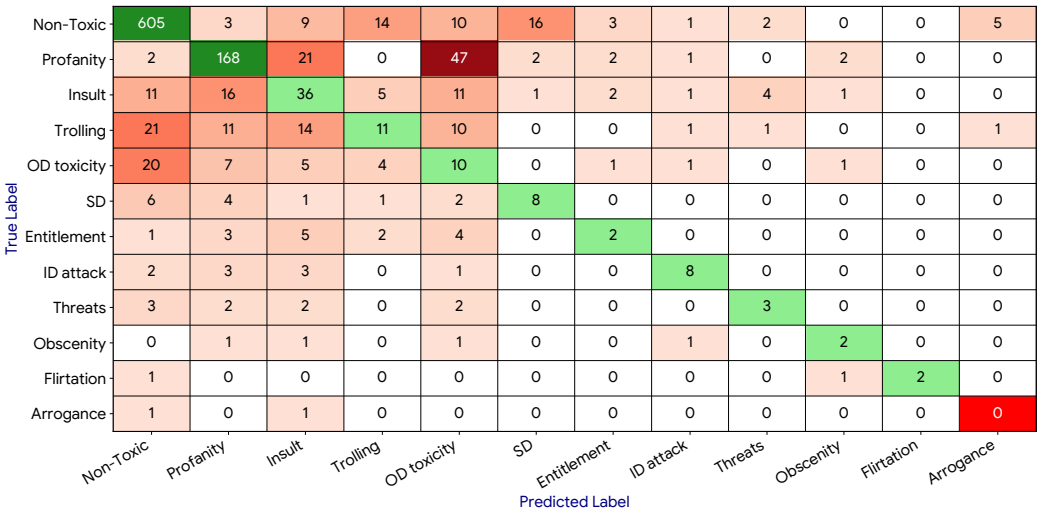


Fig. 4. Confusion Matrix for Communication Coach, where SD represents Self Deprecation, Identity Attack represents ID Attack. Values in the diagonal (greener shades) are correct classifications.

manually assigned the single most appropriate label to each sample (i.e., total sample size: 1200) from the best model's output. 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 (OD Toxicity), which was frequently misclassified as Non-Toxic, Profanity, Insult, 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.

Manual Evaluation: To complement our quantitative metrics and real-world performance, we conducted a manual evaluation of our best-performing multiclass model, Claude 3.5 Sonnet. Two authors independently reviewed a random sample of 100 instances (44 non toxic and 56 toxic samples) drawn from the full dataset of 1,200, categorizing each prediction into one of three outcomes: correct, where the model correctly identified a comment as non-toxic or its exact subcategory, alternate, where the either partially predicted or misclassified the ground-truth labels, and incorrect, where a model misclassified a toxic comment as non-toxic or vice versa. Inter-rater reliability, measured via Cohen's κ, was 0.66, indicating substantial agreement [26]. Of the 100 evaluated instances (44 are non-toxic and 56 are toxic), 58 were correctly classified, 25 were strictly misclassified, and 17 were assigned an alternate toxic label. Our annotators also evaluated the model's generated explanations for its classifications. The model performed particularly well at identifying and justifying overt toxicity. For instance, when correctly classifying *"and that FUCKIN BRANCH NAME"* as Profanity, the model accurately explained it as *"Explicit use of profanity ("FUCKIN") to emphasize a point about a branch name"*. However, the model struggled with nuanced or overlapping subcategories. For example, *"Frigging love you Niki. Seriously"* was misclassified as non-toxic despite its original Flirtation label. Interestingly, the model's explanation accurately recognized the text as "Highly informal and affectionate language" yet it still applied a non-toxic label. Similarly, *"how dare u accuse me of trying to freeload off other ppl"* was flagged as Insult and Trolling, though it was originally labeled as Threat. These instances demonstrate that while the

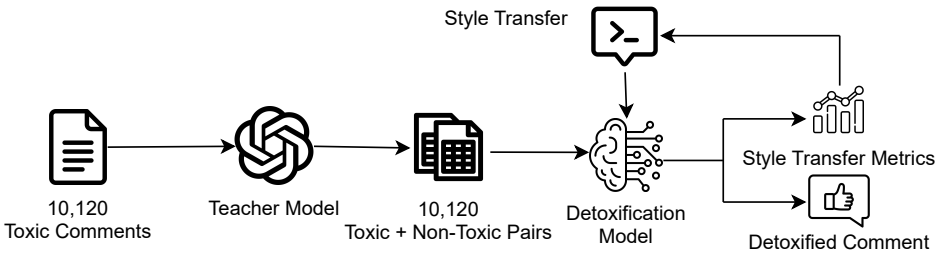


Fig. 5. The Reframer: A teacher model is used to create a parallel dataset, which is used to train a student detoxification model with iterative prompting.

model reliably detects general toxicity and generates logical rationales, it occasionally struggles to resolve the blurred boundaries between closely related interpersonal toxicities.

Key Finding 2: *Claude 3.5 Sonnet achieves best performance on multiclass toxicity classification with* 0.39 *MCC and* 0.42 *F1 score. However, for nuanced or data-sparse toxicity types, models fail to capture subcategories efficiently, which requires a more balanced dataset.*

4 Module 3: The Reframer

While flagging toxicity is a prerequisite for mitigation [2, 44], interventions that focus solely on detection risk stigmatize contributors without offering a path to improvement. In collaborative developer workflows, it is far more effective to guide users toward constructive alternatives than to simply reject their contributions. However, achieving effective real-time detoxification is non-trivial; it requires a precise balance between neutralizing hostility and preserving semantic fidelity, ensuring that the technical substance of the feedback remains intact. To address this challenge, we developed a specialized detoxification module that automatically rephrases toxic communication in OSS environments.

4.1 Evaluating Detoxification

Following the methodology established by Ostheimer *et al.* [34], we employed four complementary metrics to evaluate the efficacy of our detoxification models in capturing the critical trade-off among neutralizing toxicity, preserving technical intent, and maintaining linguistic fluency.

Style Transfer Accuracy (DETOX): This metric quantifies the model's success in stripping toxicity from the text. We utilized ToxiCR [45] to compute toxicity probability scores for both the original and detoxified samples, calculating the net percentage reduction. As the primary objective of our tool is mitigation, high style accuracy serves as the critical indicator of success.

Fluency (FL): We evaluated the grammatical correctness and naturalness of the detoxified output using a pre-trained CoLA (Corpus of Linguistic Acceptability) classifier [53]. High fluency scores indicate that the rephrased comments are syntactically sound and readable, a prerequisite for their acceptance in professional developer workflows.

Content Preservation (PRESERVE): To assess semantic fidelity, we utilized the embedding-based similarity metric proposed by Wieting *et* al. [55]. By encoding both the original and detoxified comments into paraphrastic sentence embeddings and computing their cosine similarity, this method captures conceptual equivalence rather than mere word overlap. Unlike rigid lexical metrics such as BLEU [35], this approach effectively credits valid paraphrases, ensuring that the technical meaning of the code review remains intact after detoxification.

J-Score: To provide a holistic performance benchmark, we calculated J-Score, defined as the harmonic mean of Content Preservation, Fluency, and Style Accuracy. J-Score ensures high performance in one area (e.g., detoxification) is not achieved at the expense of others (e.g., semantic loss).

4.2 Dataset Construction via Knowledge Distillation

We formulate the detoxification of code reviews as a Text Style Transfer (TST) task, specifically aiming to shift the stylistic attribute from toxic to 'neutral' or 'professional' while rigorously preserving technical semantics. Effective TST requires a parallel corpus that pairs each toxic comment with a faithful, non-toxic rewrite, enabling models to learn content preservation alongside toxicity reduction [15, 36]. However, such a corpus does not exist for the software engineering domain. To address this scarcity, we employed a teacher-student knowledge distillation framework [19] to synthesize a high-quality parallel dataset.

Teacher-Generated Parallel Data: We utilized the 10,120 toxic comments identified in our filtering phase (Table 1) as the source data. To generate their non-toxic counterparts, we leveraged multiple state-of-the-art LLMs acting as "teacher" models. These teachers were prompted to rewrite each toxic comment into a respectful alternative, strictly adhering to the original technical intent. This process yielded several candidate parallel datasets, one per teacher model. To select the optimal training basis for our downstream (student) models, we evaluated each teacher-generated dataset using the quantitative metrics defined in Section 4.1. The teacher model that maximized the aggregate J-Score was selected to construct the final parallel corpus.

Prompt Engineering for Data Synthesis: The quality of the parallel dataset is directly dependent on the efficacy of the teacher's prompt. We iteratively refined these templates against our evaluation metrics until achieving stable performance. Detailed prompts are provided in our replication package [1]. We optimized our prompts using several key strategies to ensure consistency and semantic fidelity:

- *Chain-of-Thought (CoT) and Few-Shot Learning:* To mitigate hallucination and improve reasoning, we employed CoT prompting combined with few-shot examples. This guided the teacher models to first analyze the toxic elements and then systematically reconstruct the sentence, ensuring the technical core remained intact.
- *Structured Output Constraints:* To facilitate automated parsing and integration, we enforced a strict output schema. The teacher models were instructed to generate responses in a standardized format, such as: "`Detoxified: <rewritten comment>; Rationale: <explanation of changes>`". This structure minimized output variability and ensured every rewritten comment was accompanied by an explanation, a critical feature of ToxiShield's pedagogical approach.

Teacher Model Evaluation: To assess the quality of datasets generated by the candidate teacher models, we sampled 500 detoxified pairs from each dataset. We evaluated them using the metrics introduced in Section 4.1. For each dataset, we report the average scores across these metrics along with the overall J-score, which summarizes their trade-offs. Table 5 shows the evaluation metrics of the four candidate teacher models used in our study. While all teacher models demonstrated strong detoxification capabilities, their performance varied in striking a balance between meaning preservation and fluency. Among them, *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.

4.3 Detoxification Model Training

Following the identification of *GPT-4o-2024-05-13* as the superior teacher model, we utilized its generated parallel corpus to fine-tune a lightweight, open-source student LLM. This student

Table 5. Translation metrics for teacher models used to generate parallel datasets

Model	DETOX (%)	FL (%)	PRESERVE (%)	J-Score (%)
OpenAI GPT 3.5 Turbo	91.74	98.81	71.01	85.46
OpenAI GPT 4o-05-13	96.17	**99.01**	**73.86**	**88.14**
Claude 3	**97.50**	98.73	57.41	79.36
Llama 3 - 70B	96.90	95.08	65.51	83.10

Table 6. Performance comparison of detoxification models

Model	DETOX (%)	FL (%)	PRESERVE (%)	J-Score (%)
GPT-4o mini (off-the-shelf)	**96.35**	98.36	65.14	83.57
Llama 3.1 8B	91.95	98.03	66.42	83.03
Phi 3.5	94.16	96.48	64.77	82.36
Gemma 2 2B	87.59	94.49	68.47	81.96
Qwen 2.5 Instruct 7B	94.88	98.66	65.98	83.73
GPT-3.5 FT ([38] off-the-shelf)	88.50	**99.31**	59.38	78.51
Llama 3.2 3B	95.27	97.03	**67.07**	**84.00**

model serves as the core inference engine for ToxiShield's detoxification module. To optimize computational efficiency without compromising model performance, we employed Low-Rank Adaptation (LoRA) [20]. The fine-tuning process consisted of 10 epochs using a Stratified 8:2 train: test split of the 10,120 samples, employing 10 epochs with a learning rate of 2×10^{-4}, a weight decay of 0.01, and 5 warm-up steps. All training procedures were executed on a cloud instance equipped with a single 24 GB NVIDIA RTX 4090 GPU. To ensure reproducibility and minimize stochasticity, we enforced deterministic generation parameters across all experiments (temperature = 0.0, maximum output length = 256 tokens). This strict control allows us to attribute performance variations directly to model architecture and prompt design rather than random sampling noise.

4.4 Detoxification Model Evaluation

We evaluated the performance of our distilled student models using the identical metric set applied in the teacher phase, ensuring direct comparability. Table 6 summarizes the results across seven distinct LLM architectures. While the student models generally yielded lower raw scores than their teacher counterparts, several demonstrated exceptional trade-offs between toxicity reduction and semantic preservation. Most notably, *Llama 3.2 3B* emerged as the top-performing candidate, achieving the highest aggregate J-Score of 84.00%. This model effectively balanced strong style transfer accuracy (95.27%) and fluency (97.03%) with respectable content preservation (67.07%). *Qwen 2.5 Instruct 7B* followed closely with a J-Score of 83.73%, while the proprietary *GPT-4o mini* achieved a comparable 83.57%. In terms of specific component metrics, *Gemma 2 2B* achieved the highest content preservation (68.47%) but suffered from lower style transfer accuracy (87.59%), whereas *Phi 3.5* excelled in toxicity reduction (94.16%) but lagged in content preservation (64.77%). Given that the J-Score, the harmonic mean of these metrics, is the standard benchmark for TST tasks, these results confirm *Llama 3.2 3B* as the optimal choice. Crucially, this finding highlights that a specialized, small-parameter open-source model can outperform larger proprietary models like *GPT-4o mini* when fine-tuned for domain-specific detoxification.

Error Analysis of Best Performing Model: To validate our model performance, we manually validate the misclassification of our model. Specifically, we focus on where our model failed to rephrase the toxic comments. We labeled a detoxified comment as misclassified if it is scored 0.5 or

above by ToxiCR. The best performing model, Llama 3.2 3B produced 78 misclassifications out of 2018 detoxified comments in the test set. We classify these instances into four categories.

- *Name Error:* This type of error occurred when ToxiCR flagged otherwise benign comments as toxic due to the presence of profane substrings within usernames. Specifically, six such instances were observed, accounting for 7.69% of the total misclassifications. For example, the comment "@warunalakshitha, do we need to escape the dots?" was incorrectly labeled as toxic because the username contained a substring that matched a profane word.
- *Context Error:* The most prevalent category of errors is Context error, with 29 occurrences. Comments often had toxic text as names of variables, files, or folders. Our model did not rephrase them, as rephrasing them would change the context entirely. For example, *"I'd change it to 'is_disgusting_for', as the current name implies that the item itself is disgusted by someone picking it up."* was identified as toxic due to the variable name containing the word *disgusting*.
- *General Error:* General errors are classified as those instances where our model failed to remove toxicity. For instance, *"Resonators won't be shitty pickaxes now? That's great!"* where our model failed to detoxify it. There are 20 samples of this category in our sample.
- *ToxiCR Error:* These were solely due to the Toxicity Filter module misclassifying a detoxified comment as toxic. There were 23 such cases. For example, *'::' is C++ code. Another example is 'Device.enumerat-ed_found'?* was identified as toxic by ToxiCR.

Manual Validation 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. [38]. According to our evaluation with this 100 samples, the average score of minimal change is 4.35, Context Preservation is 4.16, and Communication Style is 4.48, which indicates a higher performance of our Reframer model for our dataset. 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 [26], these correspond to almost perfect agreement for Minimal Change and substantial agreement for Context Preservation and Communication Style, indicating strong consistency in our manual detoxification evaluation.

Key Finding 3: *Llama 3.2 3B yielded the highest overall $\mathcal{J}$-Score (84%), effectively outperforming the other evaluated models in preserving fluency and meaning while detoxifying text.*

5 User Evaluation of ToxiShield with Integrated Browser Extension

We developed a browser extension (Figure 6) designed to detect and mitigate toxic comments within GitHub pull requests in real-time. Leveraging a fully on-device architecture, the tool operates via a two-stage pipeline: first, a fine-tuned detection model monitors input as it is typed; second, upon identifying toxicity, a localized LLM generates a non-toxic alternative alongside an explanatory rationale. When a comment is classified as non-toxic, the tool takes no action and the comment proceeds to submission unchanged. This local-first design ensures user privacy and eliminates reliance on costly external APIs. To enable this lightweight deployment, we exported our trained models to the ONNX format, ensuring cross-platform portability. Furthermore, we applied quantization techniques to significantly reduce model size and accelerate inference. These optimizations are critical for achieving the low latency required for a real-time user experience without compromising detection accuracy.

5.1 Participant Recruitment and Survey

Following Institutional Review Board approval, we recruited 10 professional software developers from the United States and Bangladesh to evaluate the tool's usability in a real-world setting. To

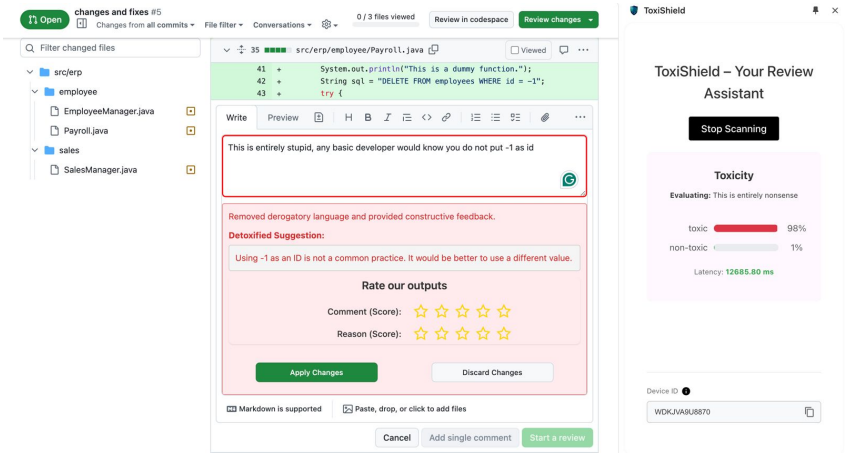


Fig. 6. Browser Extension to integrate ToxiShield with GitHub-based pull request reviews

Table 7. 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

ensure proficiency, 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. Crucially, to strictly prioritize user privacy, we deliberately refrained from collecting telemetry or usage logs. At the conclusion of the study, participants completed a comprehensive survey consisting of both Likert-scale items and open-ended questions. Regarding usage frequency, three participants reported using the tool frequently, five occasionally, and two rarely. All survey instruments and anonymized response data are available in our replication package [1].

5.2 Evaluation Instrument

To rigorously evaluate developer perceptions of ToxiShield, we employed the Technology Acceptance Model (TAM) [27], a widely adopted framework for assessing user adoption of new technologies. Our survey instrument incorporated nine items adapted from TAM, categorized into four distinct dimensions as detailed in Table 7. For the items belonging to PU, PEOU, and BI, users

rated each item on a a scale of 1 to 5, where 1 means 'Strongly Disagree' and 5 indicates 'Strongly Agree'. The analysis of these dimensions is as follows.

i) Perceived Usefulness (PU): This dimension measures the degree to which a user believes that using a specific system will enhance their job performance. Our results indicate that developers found ToxiShield moderately useful, particularly regarding feature relevance and task support. However, variability in time-saving metrics suggests that while the tool is helpful, further optimization is required to maximize workflow efficiency.

ii) Perceived Ease of Use (PEOU): PEOU assesses the degree to which using the system is free of effort. ToxiShield received high ratings in this category, confirming that its design is accessible and intuitive. While the overall user experience was positive, lower mean scores related to error handling and task completion speed highlight specific usability bottlenecks that warrant attention in future iterations.

iii) Attitude Toward Use (ATU): This metric captures the user's affective reaction to the system. We measured general satisfaction on a 10-point scale. The high aggregate score demonstrates that the majority of developers hold a positive attitude toward ToxiShield, validating its role in improving the code review experience.

iv) Behavioral Intention (BI): BI predicts the likelihood of a user continuing to use the technology or recommending it to others. Although the reported intention was positive, the standard deviation reveals variability in user advocacy. This suggests that while the core value proposition is sound, improvements in usability and feature robustness are necessary to solidify user commitment.

5.3 Evaluation Results

The TAM analysis confirms that ToxiShield is generally well-received by professional developers, with particular strength in ease of use and overall satisfaction. While the tool's utility is acknowledged, enhancing task efficiency and error management mechanisms will be critical for driving broader adoption. We have a total of 8 open ended questions in our survey. Two of our authors conducted a thematic analysis of the open-ended responses following Braun and Clarke [3], 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 build,"* 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 a *"Perhaps a chat-bot interface like Co-pilot would be more effective"* to support constructive decision-making during the writing process.

6 Implications and Lessons Learned

Socio Technical Implications of ToxiShield in OSS Development: By introducing real-time, rationale-aware intervention, ToxiShield carries distinct socio-technical implications for the health of OSS communities. First, ToxiShield fundamentally alters the economics of moderation. By automating the detection process and providing immediate, explanatory feedback, the tool significantly reduces the emotional and cognitive labor demanded of OSS maintainers, who currently shoulder the burden of manual policing. Second, the tool serves a vital pedagogical function. As ToxiShield does not merely flag content, it educates contributors by articulating why specific language is harmful. This transforms moderation from a punitive act into a learning opportunity, fostering long-term behavioral change. Furthermore, by offering detoxified alternatives that preserve the original technical intent, ToxiShield directly supports diversity and inclusion efforts. Given that toxicity disproportionately impacts minority contributors [28], a tool that neutralizes hostility without silencing technical critique is essential for retaining a diverse contributor base. Finally, recognizing that Codes of Conduct (CoC) are often insufficient on their own [2], ToxiShield acts as an operational enforcement mechanism, bridging the gap between community ideals and daily practice. By embedding these values directly into the communication pipeline, ToxiShield can help cultivate a more resilient and inclusive OSS ecosystem.

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 6). 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 Llama 3.2 3B can offer a strong balance of effectiveness, efficiency, and deployability for real-time developer tooling.

Open-Source LLMs vs Proprietary Models: Our evaluation shows fine-tuned open-source models can match or surpass proprietary systems. While GPT-4o mini achieved the highest style accuracy, Llama-3.2 3B secured the best overall score by better preserving content. This demonstrates that small, open-source models are ideal for real-time deployment: they are cost-effective, transparent, and locally hosted. By eliminating reliance on external APIs, these lightweight checkpoints bypass common privacy and compliance hurdles, making ToxiShield practical for widespread enterprise adoption. Future production deployments will incorporate telemetry to provide longitudinal insights into adoption and ToxiShield's long-term impact on developer communication norms.

7 Threats to Validity

Internal Validity 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. Moreover, 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 [44]. Future studies should evaluate larger, more diverse corpora to mitigate the limited statistical power and sampling bias of our dataset.

External Validity First, our dataset's focus on code reviews, with its specific jargon and code snippets, may limit the applicability of our findings to other domains. However, since the data is from GitHub, our results should remain valid within the broader software engineering community.

Table 8. Feature comparison highlighting ToxiShield's comprehensive capabilities. Checkmarks (✓) indicate support, while crosses (✗) indicate unsupported features. MC-> Multi Class, DT-> Detoxification

Tool	Binary	MC	DT	Explanation	Real Time
Raman *et* al. [39]	✓	✗	✗	✗	✗
Egelman *et* al.	✗	✗	✗	✗	✗
ToxiCR [45]	✓	✗	✗	✗	✗
ToxiSpanSE [42]	✓	✗	✗	✗	✗
Ferreira *et* al. [14]	✗	✗	✗	✗	✗
Rahman *et* al. [38]	✓	✗	✓	✗	✗
Mishra & Chatterjee [29]	✗	✗	✗	✗	✗
ToxiShield	✓	✓	✓	✓	✓

Second, our models were optimized for a single task (toxicity classification) and may not transfer effectively to other NLP applications or software contexts. Finally, a potential threat to the generalizability of our findings is the sample size of our user study ($N = 10$). Our recruitment was constrained by strict IRB limitations on geographic eligibility and by a high barrier to entry (a two-week commitment and installation of a browser plugin). However, we mitigated this threat by ensuring deep engagement; all participants provided extensive feedback over the two-week period. Furthermore, within the context of usability evaluation, prior research posits that 5 participants are sufficient to identify 80% of usability issues [51], and 10 participants can uncover up to 92% of issues [12]. Thus, while our sample size limits broad demographic generalization, we believe it is sufficient to capture the primary usability and workflow challenges of the tool.

Construct Validity First, our definition of toxicity [28, 45] may not encompass subtle forms of toxic behavior, such as passive-aggressive comments. Second, our use of prompt tuning could lead to overly dependent results on specific prompts, thereby affecting reproducibility. To mitigate the related risk of LLM non-determinism, we fixed generation parameters (e.g., temperature, top-p) and conducted a structured error analysis, which increased the consistency and trustworthiness of the detoxification process.

Ecological Validity While our real-time toxicity tool has proven effective in controlled tests, its practical success depends on user adoption and seamless workflow integration. As the first real-time detoxification tool built for the SE domain, it's a key step toward healthier developer communication. We will use feedback from real-world deployment to address challenges like alert fatigue, continuously improving the tool's adoption and impact.

8 Related Work

Antisocial Behavior in Open Source Software Development Environments: Antisocial behavior in the SE domain is broadly defined as actions that harm community participation [28, 37, 43, 44]. Although OSS environments exhibit lower toxicity than general social media (i.e., Reddit or Twitter), their professional and technical dynamics make these interactions qualitatively distinct [28]. Toxicity often manifests subtly through personal attacks ('destructive criticism' [17]), aggressive rejections ('pushback' [8]), or dismissive tones ('incivility' [13]). Additionally, a recent large-scale study [44] revealed that toxicity is unevenly distributed, peaking in gaming projects and strongly correlates with technical factors such as code churn and review intervals. Building on this extensive foundation, our study moves beyond characterization to develop adaptive, real-time strategies to mitigate toxic interactions in OSS environments.

Automated Detection of Toxicity in Software Engineering: General-purpose Natural Language Processing (NLP) tools often fail to accurately detect toxicity within Software Engineering (SE)

contexts due to their inability to parse technical jargon and developer-specific communication styles [33, 43]. Recognizing this gap, Raman *et al.* [39] pioneered the field by developing the first toxicity detector specifically tailored for OSS communities. However, subsequent evaluations revealed significant limitations in this early model's generalizability and accuracy [28, 37, 43]. Addressing these shortcomings, later studies sought to refine detection capabilities. Qiu *et al.* [37] designed a classifier that integrated metrics for 'pushback' [8] and 'toxicity' [39], while Sarker *et al.* [45] established a new benchmark with ToxiCR, a tool trained on 19,651 manually labeled code review comments that achieved an 88.9% F1-score. Most recently, Rahman *et al.* [38] shifted the paradigm toward generative AI, utilizing both vanilla and fine-tuned Large Language Models (LLMs) to not only identify uncivil comments but also propose civil alternatives. Our work builds upon these foundational efforts, advancing the state of the art by integrating accurate detection with real-time, explanatory mitigation strategies.

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 [4], serving as a primary driver of significant stress and burnout [39]. 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 [8, 32]. Most critically, toxicity disproportionately harms minority developers, acting as a systemic barrier that obstructs Diversity, Equity, and Inclusion efforts [17]. Building upon this evidence of harm, our study proposes actionable strategies to mitigate these adverse effects and foster healthier OSS communities.

Novelty: ToxiShield distinguishes itself from existing toxicity detection approaches through three advancements, as summarized in Table 8. First, it moves beyond binary classification by employing multi-class analysis to identify toxicity subcategories. Second, it provides explanatory feedback detailing *why* a comment is toxic to foster behavioral change. Finally, ToxiShield represents the first empirically evaluated, real-time toxicity mitigation tool integrated into developer workflows.

9 Conclusion

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 pipeline enables real-time detection, multiclass 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 research in automated toxicity detection and inclusive developer collaboration.

ACKNOWLEDGEMENT

This research is partially supported by the US National Science Foundation under Grant No. 2340389, and a start-up grant from the University of Nebraska at Omaha, College of Information Science and Technology. The findings of this research do not necessarily reflect the views of the National Science Foundation. We are also grateful to the Applied Machine Learning Lab in the Department of CSE at Bangladesh University of Engineering and Technology for logistical support.

DATA AVAILABILITY

The dataset, source code, and experimental results of ToxiShield are available here [1] and DOI: https://doi.org/10.5281/zenodo.19490337.

References

[1] Md Awsaf Alam Anindya, Showvik Biswas, Anindya Iqbal, Jaydeb Sarker, and Amiangshu Bosu. 2026. *ToxiShield: Replication Package.* https://github.com/WSU-SEAL/ToxiShield

[2] Sarthak Bharadwaj, Fabio Santos, and Bianca Trinkenreich. 2025. 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)*. 1–11. doi:10.1109/ESEM64174.2025.00016

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

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

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

[6] Tanni Dev, Sayma Sultana, and Amiangshu Bosu. 2025. 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)* (Honolulu, Hawai, USA, USA). 1–12. doi:10.1109/ESEM64174.2025.00059

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186. doi:10.18653/v1/N19-1423

[8] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting developers' negative feelings about code review. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 174–185. doi:10.1145/3377811.3380414

[9] Ramtin Ehsani, Mia Mohammad Imran, Robert Zita, Kostadin Damevski, and Preetha Chatterjee. 2024. 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*. 515–519. doi:10.1145/3643991.3644887

[10] Ramtin Ehsani, Rezvaneh Rezapour, and Preetha Chatterjee. 2023. 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*. 2092–2096.

[11] Ramtin Ehsani, Rezvaneh Rezapour, and Preetha Chatterjee. 2025. Analyzing Toxicity in Open Source Software Communications Using Psycholinguistics and Moral Foundations Theory. (2025), 1–8. doi:10.1109/NLBSE66842.2025.00006

[12] Laura Faulkner. 2003. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers* 35, 3 (2003), 379–383. doi:10.3758/BF03195514

[13] Isabella Ferreira, Jinghui Cheng, and Bram Adams. 2021. 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 (2021), 1–35. doi:10.1145/3479497

[14] Isabella Ferreira, Ahlaam Rafiq, and Jinghui Cheng. 2024. Incivility detection in open source code review and issue discussions. *Journal of Systems and Software* 209 (2024), 111935. doi:10.1016/j.jss.2023.111935

[15] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. 2018. Style transfer in text: Exploration and evaluation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32. doi:10.1609/aaai.v32i1.11330

[16] 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. 2023. Zero-shot Topical Text Classification with LLMs - an Experimental Study. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 9647–9676. doi:10.18653/v1/2023.findings-emnlp.647

[17] Sanuri Dananja Gunawardena, Peter Devine, Isabelle Beaumont, Lola Piper Garden, Emerson Murphy-Hill, and Kelly Blincoe. 2022. Destructive criticism in software code review impacts inclusion. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–29. doi:10.1145/3555183

[18] Keyan Guo, Alexander Hu, Jaden Mu, Ziheng Shi, Ziming Zhao, Nishant Vishwamitra, and Hongxin Hu. 2023. An Investigation of Large Language Models for Real-World Hate Speech Detection. In *2023 International Conference on Machine Learning and Applications (ICMLA)*. 1568–1573. doi:10.1109/ICMLA58977.2023.00237

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

[20] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nZeVKeeFYf9

[21] Mia Mohammad Imran, Robert Zita, Rahat Rizvi Rahman, Preetha Chatterjee, and Kostadin Damevski. 2026. Toxicity Ahead: Forecasting Conversational Derailment on GitHub. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26)* (Rio de Janeiro, Brazil). ACM, Rio de Janeiro, Brazil. doi:10.1145/3744916.3787839

[22] Satyanarayana Chowdary Kadiyala, Jaydeb Sarker, and Bianca Trinkenreich. 2026. How should self-deprecation comments be classified? A toxicity analysis study on Zephyr. In *Proceedings of the 5th ACM/IEEE International Workshop on NL-based Software Engineering (NLBSE)*. doi:10.1145/3786164.3788447

[23] Anjali Khurana, Hariharan Subramonyam, and Parmit K Chilana. 2024. 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*. 288–303. doi:10.1145/3640543.3645200

[24] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[25] Deepak Kumar, Patrick Gage Kelley, Sunny Consolvo, Joshua Mason, Elie Bursztein, Zakir Durumeric, Kurt Thomas, and Michael Bailey. 2021. Designing Toxic Content Classification for a Diversity of Perspectives. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. USENIX Association, 299–318. https://www.usenix.org/conference/soups2021/presentation/kumar

[26] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (March 1977), 159. doi:10.2307/2529310

[27] Qingxiong Ma and Liping Liu. 2005. *The Technology Acceptance Model*. doi:10.4018/9781591404743.ch006.ch000

[28] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian Kästner. 2022. " Did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering*. 710–722. doi:10.1145/3510003.3510111

[29] Shyamal Mishra and Preetha Chatterjee. 2024. Exploring ChatGPT for Toxicity Detection in GitHub *(ICSE-NIER'24)*. Association for Computing Machinery, New York, NY, USA, 6–10. doi:10.1145/3639476.3639777

[30] Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. 2021. XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation. doi:10.48550/ARXIV.2106.04563

[31] Emerson Murphy-Hill, Jill Dicker, Delphine Carlson, Marian Harbach, Ambar Murillo, and Tao Zhou. 2024. Did Gerrit's Respectful Code Review Reminders Reduce Comment Toxicity? In *Equity, Diversity, and Inclusion in Software Engineering: Best Practices and Insights*. Apress Berkeley, CA, 309–321. doi:10.1007/978-1-4842-9651-6_18

[32] Emerson Murphy-Hill, Ciera Jaspan, Carolyn Egelman, and Lan Cheng. 2022. The pushback effects of race, ethnicity, gender, and age in code review. *Commun. ACM* 65, 3 (2022), 52–57. doi:10.1145/3474097

[33] Nicole Novielli, Fabio Calefato, Filippo Lanubile, and Alexander Serebrenik. 2021. Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. *Empirical Software Engineering* 26, 4 (2021), 77. doi:10.1007/s10664-021-09960-w

[34] Phil Sidney Ostheimer, Mayank Kumar Nagda, Marius Kloft, and Sophie Fellenz. 2024. 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)*. ELRA and ICCL, Torino, Italia, 15802–15822. https://aclanthology.org/2024.lrec-main.1373/

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[36] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. 2018. Style Transfer Through Back-Translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 866–876. doi:10.18653/v1/P18-1080

[37] Huilian Sophie Qiu, Bogdan Vasilescu, Christian Kästner, Carolyn Egelman, Ciera Jaspan, and Emerson Murphy-Hill. 2022. 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* (Pittsburgh, Pennsylvania) *(ICSE-SEIS '22)*. Association for Computing Machinery, New York, NY, USA, 41–55. doi:10.1145/3510458.3513019

[38] Md Shamimur Rahman, Zadia Codabux, and Chanchal K Roy. 2024. Do Words Have Power? Understanding and Fostering Civility in Code Review Discussion. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1632–1655. doi:10.1145/3660780

[39] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. 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*. 57–60. doi:10.1145/3377816.3381732

[40] Sarthak Roy, Ashish Harshvardhan, Animesh Mukherjee, and Punyajoy Saha. 2023. Probing LLMs for hate speech detection: strengths and vulnerabilities. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Association for Computational Linguistics, 6116–6128. doi:10.18653/v1/2023.findings-emnlp.407

[41] Elvis Saravia. 2022. Prompt Engineering Guide. *https://github.com/dair-ai/Prompt-Engineering-Guide* (12 2022).

[42] Jaydeb Sarker, Sayma Sultana, Steven R Wilson, and Amiangshu Bosu. 2023. ToxiSpanSE: An explainable toxicity detection in code review comments. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12. doi:10.1109/ESEM56168.2023.10304855

[43] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2020. A benchmark study of the contemporary toxicity detectors on software engineering interactions. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 218–227. doi:10.1109/APSEC51365.2020.00030

[44] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2025. The Landscape of Toxicity: An Empirical Investigation of Toxicity on GitHub. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 623–646. doi:10.1145/3715744

[45] Jaydeb Sarker, Asif Kamal Turzo, Ming Dong, and Amiangshu Bosu. 2023. Automated Identification of Toxic Code Reviews Using ToxiCR. *ACM Transactions on Software Engineering and Methodology* 32, 5 (July 2023), 1–32. doi:10.1145/3583562

[46] Sulayman K Sowe, Ioannis Stamelos, and Lefteris Angelis. 2008. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software* 81, 3 (2008), 431–446. doi:10.1016/j.jss.2007.03.086

[47] Sayma Sultana, Jaydeb Sarker, Farzana Israt, Rajshakhar Paul, and Amiangshu Bosu. 2025. Automated Identification of Sexual Orientation and Gender Identity Discriminatory Texts from Issue Comments. *ACM Transactions on Software Engineering Methodology (TOSEM)* 34 (2025). doi:10.1145/3757739

[48] Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023. Text Classification via Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 8990–9005. doi:10.18653/v1/2023.findings-emnlp.603

[49] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[50] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Tayana Conte, Marco Gerosa, Anita Sarma, and Igor Steinmacher. 2021. Pots of gold at the end of the rainbow: what is success for open source contributors? *IEEE Transactions on Software Engineering* 48, 10 (2021), 3940–3953. doi:10.1109/TSE.2021.3108032

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

[52] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. 2023. Large Language Models Are Zero-Shot Text Classifiers. doi:10.48550/ARXIV.2312.01044

[53] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural Network Acceptability Judgments. *Transactions of the Association for Computational Linguistics* 7 (09 2019), 625–641. doi:10.1162/tacl_a_00290

[54] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[55] John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. 2019. Beyond BLEU: Training Neural Machine Translation with Semantic Similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4344–4355. doi:10.18653/v1/P19-1427

[56] Frances Zlotnick. 2017. GitHub Open Source Survey 2017. http://opensourcesurvey.org/2017/. doi:10.5281/zenodo.806811

Received 2025-09-10; accepted 2026-03-24