Assessing Harmful Comments and Specificity in Code Review Feedback at Scale using Large Language Models

Audrey You
University of Auckland
New Zealand
ayou332@aucklanduni.ac.nz

Jingyi (Jenny) Wang
University of Auckland
New Zealand
jwan404@aucklanduni.ac.nz

Youxiang Lei
Multitudes
New Zealand
youxiang@multitudes.com

Lauren Peate
Multitudes
New Zealand
lauren@multitudes.com

Kelly Blincoe
University of Auckland
New Zealand
k.blincoe@auckland.ac.nz

Abstract

Code review is central to collaborative software development, yet feedback quality can vary widely, influencing code maintainability and developer interactions. This study investigates how large language models (LLMs) can assess code review feedback quality along two dimensions, sentiment (with a focus on harmful comments) and specificity, to support more constructive collaboration. Using over 204,000 feedback threads from 30 open-source software (OSS) repositories, we evaluate eleven LLMs, achieving F1-scores up to 0.83 for sentiment and 0.67 for specificity. Most OSS feedback is neutral or low in specificity, with highly detailed or overtly harmful comments comprising a small minority. Industry data from 45 organisations contains significantly more highly specific feedback and more minimal reviews, while harmful feedback remains rare. Deployment of our approach in commercial settings demonstrated practical value. Specificity classifications delivered immediate value, such as revealing mentorship gaps when senior developers provided more specific feedback than they received, while harmful comment classifications required careful UX framing to avoid user sensitivity. Our findings demonstrate the feasibility and practical utility of automated feedback-quality assessment in real-world environments.

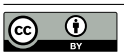
CCS Concepts

• **Software and its engineering → Collaboration in software development**.

Keywords

Code Review, Harmful Comments, Specificity, Feedback, LLM

ACM Reference Format:
Audrey You, Jingyi (Jenny) Wang, Youxiang Lei, Lauren Peate, and Kelly Blincoe. 2026. Assessing Harmful Comments and Specificity in Code Review Feedback at Scale using Large Language Models. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3803437.3805264



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805264

1 Introduction

In software engineering, collaboration among developers is essential for maintaining and improving software systems. A key practice underpinning this collaboration is code review, a systematic process in which developers evaluate proposed code changes before integration, promoting early defect identification and overall maintainability [3, 23]. Beyond technical benefits, code reviews serve important social and organizational functions, such as fostering knowledge sharing, mentoring, and trust [1, 7].

Despite its centrality in software engineering, the quality of feedback exchanged during open-source software (OSS) code reviews varies widely. Some comments provide clear and actionable guidance that improve code maintainability and developer learning, while others are vague, unhelpful, or even toxic or destructive [11, 12, 19]. This low quality feedback can reduce overall code quality, hinder collaboration, and negatively affect the inclusion of open-source software environments [5, 11, 19, 20].

Techniques have been proposed to identify harmful code review comments. Early techniques, like ToxiCR, require substantial domain-specific data and often struggle to generalize to other datasets [34]. Recent advances in LLMs have made it possible to address some of these limitations and enable large-scale analysis of feedback quality. Specifically, studies have employed LLMs such as GPT in zero-shot settings to detect toxicity in software engineering discussions, demonstrating promising results [25]. However, prior work has focused on general software engineering discussions rather than code reviews, leaving their performance in identifying qualities of feedback, such as harmful comments and specificity, in software code reviews largely unexplored.

This research evaluates the capability of LLMs in zero-shot settings to classify OSS code review feedback along two key dimensions: sentiment and specificity. The sentiment dimension captures the potential harm of feedback, where we use negative sentiment to capture toxic, destructive, or otherwise harmful communication. The specificity dimension assesses whether feedback provides concrete, actionable guidance versus vague or abstract comments. Together, these dimensions offer complementary perspectives on feedback quality: sentiment reflects the interpersonal and psychological impact of communication, while specificity addresses its practical utility for improving code. By classifying feedback along both dimensions, we can distinguish feedback that is both constructive and actionable from feedback that may be harmful or too vague

to be useful. Further, our study leverages these classifications to examine patterns in feedback quality across both OSS and commercial settings. We answer the following research questions:

- RQ1: How accurately do large language models classify OSS feedback compared to human-annotated ground truth, across sentiment and specificity dimensions?
- RQ2: What are the distributions of feedback quality (sentiment and specificity) in code review discussions in OSS and commercial settings?

This study advances understanding of code review practices, providing insights into feedback quality distribution, while demonstrating LLMs' effectiveness in automated code review quality assessment to support collaboration and learning in software engineering.

In addition to large-scale empirical analysis, we deployed our approach in commercial settings to evaluate its practical utility. The feature achieved high adoption, consistently ranking among the top three most-used capabilities, demonstrating strong practitioner engagement. The deployment also revealed important lessons: specificity classifications were perceived as more actionable and less sensitive than sentiment classifications, which required careful UX framing to support responsible use. Moreover, specificity metrics enabled one organization to identify a mentorship gap, where a senior developer provided significantly more specific feedback than they received, prompting targeted support discussions. These deployment insights highlight the real-world value of automated feedback-quality analysis and show that LLM-based classifications can support both aggregate-level monitoring and cautious, assistive use at the individual-comment level. In the discussion, we reflect on the lessons learned from the deployment of this feature.

2 Background and Related Work

2.1 Software Code Review and Feedback Quality

Code review has long been recognized as a key practice in software engineering, contributing significantly to code quality, maintainability, and defect prevention [3, 23]. Recent work found that comments on readability, defects, and maintainability are most likely to be resolved by developers, suggesting that certain types of feedback are inherently more actionable [17]. Beyond these technical considerations, code review also serves critical social and collaborative functions including knowledge sharing, learning, collective code ownership, and team cohesion [1, 3, 7]. However, the quality of feedback can vary widely. Some review comments provide precise, actionable insights, while others can be vague, unhelpful, or even detrimental to the development process [12]. Techniques have been devised to identify unclear code review comments [9]. Research has also identified bad practices that undermine the effectiveness of code reviews, termed code review smells [11]. Common smells include "Looks Good To Me" (LGTM) reviews which indicate superficial approvals, delayed responses, and lack of review [11, 18]. Such smells can lead to long-term inefficiencies, increased maintenance costs, and a decline in overall code quality [11]. Consequently, studies have proposed code review smell detection techniques to enable more effective code reviews [4, 11, 18].

2.2 Sentiment and Harmful Feedback

Complementing these process-focused smells and clarity / usefulness considerations of code review, researchers have examined the interpersonal and emotional dimensions of code reviews, recognizing that how feedback is delivered is also important. Sentiment within comments influences team dynamics, as well as how developers perceive and respond to reviews [14, 26]. Research found that positive sentiment in code reviews leads to faster acceptance of changes and enhances collaboration, while negative or harsh sentiment can prolong the review process and erode the intended benefits of constructive feedback and collaboration [14]. However, not all negative sentiment is harmful as it can be expressed for many reasons, including healthy disagreements.

Negative sentiment can become harmful when it is toxic or destructive. Toxic feedback includes "offensive name calling, insults, threats, personal attacks, flirtations, reference to sexual activities, and swearing or cursing" [32], while destructive feedback is both non-specific and inconsiderate [5, 19]. Many other terms have been used to describe harmful interactions in software projects, including lack of kindness [36], incivility, defined as "personal attacks and unnecessary disrespectful comments" [15], and pushback, defined as "negative interpersonal interactions" during code review [13]. While each of these terms have slightly different definitions, they all imply harmful interactions. Given our focus on code review, we use the general term of harmful feedback in this paper.

Harmful feedback in software code review can have detrimental effects on recipients and impact the overall dynamic of the team [19, 20]. Harsh feedback can lead to decreased productivity and higher turnover rates [6]. Pushback has been found to provoke strong emotional reactions and lead to long-term behavior changes, including avoiding working with the reviewer again [13]. Destructive criticism has been found to reduce motivation to continue working [19]. These effects can be greater for women and other marginalized groups, impacting inclusion [13, 19].

2.3 Detection of Harmful Feedback

Given the implication of harmful feedback, detecting and addressing it has become a priority. Specifically, studies have focused on toxicity detection. However, studies have found that accurately classifying toxic code reviews presents challenges due to its subjectivity and contextual nuances [32]. Existing "off-the-shelf" toxicity detectors, while generally effective in online environments, struggle with software engineering datasets due to their distinctive language and context, including code snippets and technical terminology [34]. Thus, several studies have focused on developing domain-specific toxicity detection tools tailored for software engineering contexts.

ToxiCR, a supervised classifier, was specifically designed to detect toxic comments in code reviews and demonstrated superior accuracy (95.8%) and F1-score (0.889) compared to general-purpose toxicity detectors [30, 34]. Similarly, ToxiSpanSE introduced an explainable toxicity detection approach by identifying specific toxic spans within comments, improving transparency and helping human moderators better understand toxic interactions beyond a simple toxic or non-toxic classification of the entire comment [31]. However, ambiguous or mixed-sentiment comments such as describing one's own code as "ugly", and domain-specific language

such as "kill" or "bug", can result in false positives, and they struggle to generalize to other datasets [34]. As such, these tools still face limitations in fully understanding context and nuances [30, 31].

In recent years, more advanced techniques like transformer-based models, e.g. BERT (Bidirectional Encoder Representations from Transformers), and Large Language Models (LLMs) have emerged. They have been applied to automate a variety of software engineering tasks, including the detection of harmful discussions and feedback. Recent studies found that BERT outperformed classical machine learning models for unkind and incivil interactions with F1-scores of 0.82 and 0.95, respectively [16, 36]. Similarly, another study obtained promising results using BERT-based models to identify offensive language on multiple software engineering platforms [10]. Another study used GPT-4o for toxicity annotation, and LLaMA-3.1 70B to summarize and predict derailment (the point at which toxicity emerges), achieving an F1-score of 0.69 for derailment prediction [21]. Given the limited availability and imbalance of software-specific toxicity datasets, researchers have also explored zero-shot learning approaches, with one study demonstrating ChatGPT's capability to identify toxic language in GitHub discussions without fine-tuning, achieving promising results [25].

While these approaches have demonstrated promise in using LLMs to classify software engineering discussions, only one specifically focused on software code review [36]. They also each examine only one specific type of harmful interaction (e.g., incivility, toxicity). Our study considers harmful interactions more broadly. In addition, research has not investigated classifying specificity of code review comments, which is another important dimension of useful feedback [5, 19]. Yang et al. did consider four quality attributes in code review comments: emotion, question, evaluation, and suggestion [36]. Of these, emotion captures kindness, which is one type of harmful interaction, and the other three capture sub-elements of specificity. We take a broader view of both concepts. By simultaneously analyzing both harmful comments and specificity in code review feedback across OSS and industrial contexts, this research provides a more comprehensive understanding of feedback quality and enables practitioners to identify and address multiple dimensions of problematic communication.

3 Methodology

To answer our research questions, three datasets were collected: a targeted OSS dataset with a balanced amount of harmful code reviews to evaluate the accuracy of LLMs (RQ1), and two general large-scale datasets to investigate the distribution of feedback quality in OSS and commercial settings (RQ2). The OSS datasets were obtained through the GitHub API to ensure the use of the most up-to-date information available. The commercial dataset was collected through a research partnership with our industry partner, Multitudes, a software startup that provides an engineering insights platform for developer productivity and experience.

3.1 Datasets

3.1.1 Potential Harmful OSS Feedback Dataset. To evaluate the accuracy of the LLMs, we needed a dataset with a balanced amount of harmful feedback. Preliminary analysis and existing literature suggest that overtly toxic or destructive feedback is relatively uncommon in popular OSS repositories, which are often moderated [30, 33]. Therefore, a targeted dataset was created.

We collected the top 500 most-starred GitHub repositories with active development in the last 90 days and retrieved all moderator-locked pull requests (PRs) from these repositories. Data was collected on June 4, 2025. Adapting Miller et al.'s methodology for heated issue discussions [24], we retained only PRs explicitly locked as "Too Heated" and extracted all associated comments. This approach yields discussions likely to contain contentious or negative interactions. We excluded bot-generated comments and PR author comments to focus on genuine peer review interactions. This dataset consisted of 573 feedback threads.

3.1.2 General OSS Feedback Dataset. To examine the distribution of feedback quality, a more general feedback dataset was needed. We selected repositories with at least 10,000 stars and evidence of active development within the last 90 days, ensuring both popularity and relevance. From this set, following the guidance of Kalliamvakou et al. [22], the top repositories in terms of stars were manually reviewed to confirm they were software development projects. Non-software development projects, such as learning materials or datasets, were excluded and replaced with the next most starred repository until 30 repositories were selected (see Table 1).

From these 30 repositories, PR comments within the last three years and related meta-data were collected. Again, comments from automated tools and the PR author were excluded. The resulting general dataset comprised 204,333 feedback threads, with data collection starting on June 6, 2025.

3.1.3 Industry Dataset. The industry dataset comprises 105,468 feedback threads drawn from repositories across 45 randomly sampled organizations in Multitude's customer database. The sample provides broad geographic coverage, spanning organizations based in North America, Europe, Latin America, and Australasia, and with varying sizes and industry sectors. For confidentiality reasons, the full list of repositories cannot be provided. All feedback threads from these sampled organizations for the period 1 January 2025 to 17 January 2026 was used to construct the industry dataset. The count of feedback threads collected from each organization ranged from 36 to 13,322 observations (median of 1323). Bot-authored comments were filtered out during collection by our industry partner.

3.1.4 Data Schema and Preprocessing. All datasets were structured using a unified schema, including the textual content and relevant metadata such as timestamps, repository identifiers, and author information. All comments made by a single reviewer on a single PR are grouped into one continuous text block, preserving the full conversational context of the reviewer's feedback. This block is referred to as a feedback thread. Feedback threads serve as the primary input for LLMs used in this study and enable contextual understanding of reviewer behavior. The full schema is provided in our online replication package [37].

3.2 Data Sampling

Analysis of the OSS and industry datasets shows substantial variation in feedback activity across the 30 selected repositories and 45 organizations, as illustrated in Figure 1. We adopted a hybrid

Table 1: List of Selected GitHub Repositories

AutoGPT	Stable Diffusion Web UI	Visual Studio Code	yt-dlp	scrcpy	PowerToys
Linux Kernel	Awesome Self-Hosted	Microsoft Activation Scripts	youtube-dl	TensorFlow	Kubernetes
Bootstrap	Next.js	D3.js	LangChain	Axios	Python Algorithms
Node.js	Developer Roadmap	Oh My Zsh	TypeScript	Flutter	Three.js
Electron	Ollama	Rust Programming Language	Transformers	React	React Native

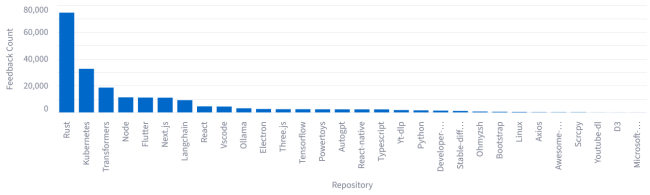
stratified sampling approach to balance representation across repositories. Each repository or organization was treated as a stratum, ensuring that all selected sources contributed feedback to the final sample. Within each stratum, feedback threads were sampled in proportion to the repository's (organization's) overall activity, while applying minimum and maximum caps to maintain balanced representation. The sampling procedure consisted of four steps:

(1) Exclude very low-activity repositories. Repositories with fewer than 100 feedback threads were excluded to avoid under-representation in the evaluation. This step removed three repositories (Microsoft-activation-scripts, D3, and Youtube-dl), leaving 27 repositories with a combined total of 204,244 feedback threads. This step was not applied to the industry dataset, as some organizations onboarded with Multitudes during the collection period, which naturally contributed to variation in activity levels. Consequently, lower-activity organizations do not inherently misrepresent the broader industry sample and were retained.
(2) Define sampling caps. The target sample size was set at 6,000 feedback threads for both the OSS and Industry sample datasets to ensure statistical reliability against practical constraints. We used a minimum cap of 100 to ensure low-activity sources remain represented and a maximum cap of 400 to prevent high-activity sources from dominating. This aligns with the statistical principle that a minimum sample size of 384 items achieves a 95% confidence level with a ±5% margin of error, rounded to 400 for simplicity [27].
(3) Calculate proportional sample sizes. For each stratum, we computed the sample size proportional to its feedback volume relative to the total dataset using the target size of 6,000, subject to the minimum (100) and maximum (400) caps.
(4) Random selection of feedback threads. Within each stratum of the OSS and Industry dataset, feedback threads were randomly selected according to the allocated sample sizes.

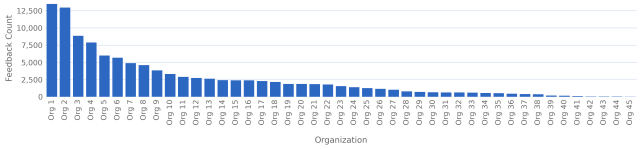
The final sampled datasets comprised 5,288 feedback threads for OSS and 6,573 feedback threads for industry.

3.3 Data Labeling

Subsets of the OSS datasets were manually labeled by two student researchers to establish a ground truth for evaluating the LLMs on sentiment and specificity: 300 feedback threads from the potential harmful OSS dataset (out of 573 total), and 500 feedback threads from the sampled general OSS dataset (out of 5,288 total). These sample sizes were chosen to balance feasibility with coverage across both datasets. Sentiment was used to capture harmful feedback threads. Personal attacks, vague criticism, judgmental comments, terseness, repeated nitpicking, and harsh language were



(a) OSS dataset



(b) Industry dataset

Figure 1: Distribution of feedback threads.

labeled as negative according to the definition of harmful feedback below which includes toxic and destructive feedback [13, 19, 32]. Non-harmful feedback was labeled as positive or neutral based on the sentiment of the feedback. For specificity, comments that included links or code snippets with reasoning or suggestions/ideas with reasoning were labeled as highly specific; those that contained similar information but without associated reasoning and those that contained thought-provoking questions were labeled as neutral, and comments without specific suggestions for improvement were labeled highly unspecific. Labels were reviewed and discussed jointly with the two students and their supervisors.

Additionally, a minimal label was applied to all feedback threads, indicating whether the feedback is very short and contains little substantive content. Comments with fewer than 25 characters were labeled True, while longer comments were labeled False.

To ensure consistency of manual labeling, both researchers independently labeled a subset of 30 feedback threads, focusing on the main sentiment labels (Positive, Neutral, Negative). Inter-coder reliability was checked using Cohen's Kappa (κ) coefficient, resulting in a score of 0.834, which indicates a strong level of agreement. Therefore, the remaining manual labeling was split between the two coders after discussing the reasons for the differences in the initial set and coming to a common understanding.

Table 2 shows the final distribution of the sentiment and specificity labels for the harmful and general subsets.

In this paper, "harmful feedback" is used as an umbrella analytical term encompassing toxic, destructive, incivil or otherwise inconsiderate communication during code reviews. "Negative sentiment" refers to the model output label for harmful feedback.

Table 2: Distribution of sentiment and specificity labels in the manually labeled subsets

	Sentiment			Specificity		
Dataset	Positive	Neutral	Negative	Highly Unspecific	Neutral Specificity	Highly Specific
Potential Harmful OSS dataset (n = 300)	18	245	37	–	–	–
Sampled General OSS dataset (n = 500)	78	395	27	154	185	161

3.4 Experiment Design

We evaluated LLMs in zero-shot settings using two automated pipelines implemented in BoundaryML (BAML): one for sentiment classification and one for specificity classification

A standardized prompt template was used for each pipeline. For the sentiment pipeline, models were instructed to classify feedback threads as positive, neutral, or negative, assess confidence, provide an overall reasoning for the predicted sentiment, and, if the sentiment was negative, return one or more specific negative reasons. For the specificity pipeline, models were instructed to classify feedback as highly specific, neutral specificity, or highly unspecific, assess confidence, provide overall reasoning for the predicted specificity, and, if applicable, return one or more specificity reasons.

To reduce reliance on natural-language labels and prevent models from memorizing lexical associations, we adopted a symbol tuning strategy, replacing conventional labels (e.g., "negative", "neutral", "positive") with arbitrary symbols (e.g., apple, banana, cherry). This encourages models to infer the task structure from input-output relationships, improving generalization, supporting algorithmic reasoning, and overriding prior label knowledge [35].

The complete prompt templates used, symbol mappings, and model output field definitions are in our replication package [37].

3.4.1 Harmful feedback classification validation. An experiment was conducted on the sentiment pipeline to evaluate LLM performance in harmful feedback classification and validate the prompt. Eleven models (Claude Sonnet 3.5, Claude Sonnet 3.7, Claude Sonnet 4.0, LLaMA 3.3 70B, LLaMA 3.1 70B, Mistral Large, Mixtral 8x7B Instruct, Command R+, DeepSeek-R1, Nova Pro, and Jamba 1.5 Large) were tested on the 300 ground truth-labeled feedback threads from the harmful OSS dataset. Models were selected based on capability, provider diversity, specialization, and recency. Evaluation metrics included precision, recall, and F1-score, and negative precision. Feedback threads with negative ground truth labels that were predicted as neutral or positive were analyzed. A fine-tuned prompt was devised but did not result in improved classification, so the original prompt was maintained.

3.4.2 OSS Sentiment and Specificity Experiments. Following the harmful feedback classification validation, the eleven models were evaluated on both sentiment and specificity classification against the 500 ground truth labeled items from the sampled general OSS dataset. Evaluation metrics included precision, recall, F1-score, negative precision (for sentiment model only), and highly specific precision, defined as the proportion of threads predicted as highly specific that were correctly labeled (for specificity model only).

From this, nine models were selected to classify the full OSS dataset. Seven models were selected for sentiment (Claude Sonnet 3.5 / 3.7 / 4.0, Command R+, LLaMA 3.3 70B, Mistral Large, and Nova Pro) and seven for specificity (Claude Sonnet 3.5 / 4.0, DeepSeek-R1, LLaMA 3.3 70B, Mistral Large, Mixtral 8x7B Instruct, and Nova Pro) with partial overlap between the two sets. Models were selected based on strong performance in preliminary experiments, alongside recency and diversity across providers.

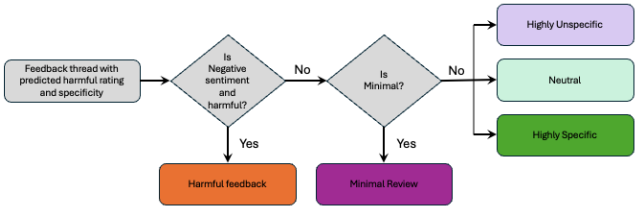


Figure 2: Hierarchical prioritisation of feedback categories

3.4.3 Feedback Categories. To allow joint sentiment-specificity analysis, predicted feedback threads were placed into a single category according to a hierarchical prioritization: negative sentiment predictions were classified as Harmful feedback, taking precedence over specificity labels; feedback flagged as minimal was classified as Minimal Review; and all remaining feedback received a specificity label (Highly Unspecific, Neutral, or Highly Specific). This ensured that every feedback fell into exactly one category, even though a comment could theoretically satisfy multiple conditions (e.g., both highly unspecific and harmful). The prioritization reflects our analytic focus on harmful communication, which takes precedence over specificity signals when presenting overall feedback quality distributions. Figure 2 maps this hierarchical process.

3.4.4 Model Evaluation and Selection. All 49 possible sentiment–specificity model pairs (i.e., the Cartesian product of seven sentiment and seven specificity models) were evaluated using the 500-thread ground truth general dataset. For each pair, the predicted distribution across the five hierarchical categories was compared to the ground truth distribution using the χ^2 (Chi-square) distance.

All seven sentiment models and seven specificity models were applied to the full general OSS dataset, generating predictions for 204,333 feedback threads. The sentiment/specificity combination identified as best-performing, the pair with the smallest chi-square distance (χ^2) against the ground truth, was used for primary analysis. This same pair was applied to the sampled industry dataset to generate sentiment and specificity labels. Bot-generated threads were removed from the OSS data through a three-step filtering procedure. First, we excluded threads authored by accounts on a list of known bots in the dataset. Second, we excluded threads whose author name matched common bot-naming conventions, ending in bot or beginning with bot-. Lastly, we applied content-based

Table 3: Evaluation metrics for harmful feedback classification validation

Model	Recall	F1	Prec.	Neg. Prec.
Claude Sonnet 3.5	0.733	0.767	0.860	0.524
Claude Sonnet 3.7	0.713	0.751	0.845	0.517
Claude Sonnet 4.0	0.607	0.653	0.850	0.354
Command R+	0.610	0.653	0.852	0.374
DeepSeek R1	0.522	0.568	0.853	0.356
Jamba 1.5 Large	0.595	0.653	0.823	0.247
LLaMA 3.1 70B Instruct	0.635	0.683	0.848	0.397
LLaMA 3.3 70B Instruct	0.633	0.678	0.850	0.398
Mistral Large (24.02)	0.717	0.748	0.849	0.422
Mixtral 8x7B Instruct	0.617	0.661	0.846	0.362
Nova Pro	0.695	0.730	0.846	0.415

Table 4: Evaluation metrics for sampled general OSS dataset (sentiment)

Model	Recall	F1	Prec.	Neg. Prec.
Claude Sonnet 3.5	0.714	0.835	0.744	0.710
Claude Sonnet 3.7	0.754	0.853	0.778	0.690
Claude Sonnet 4.0	0.694	0.723	0.875	0.370
Command R+	0.670	0.857	0.700	0.348
DeepSeek R1	0.549	0.838	0.578	0.342
Jamba 1.5 Large	0.720	0.840	0.756	0.194
LLaMA 3.1 70B Instruct	0.732	0.870	0.757	0.477
LLaMA 3.3 70B Instruct	0.726	0.873	0.752	0.500
Mistral Large (24.02)	0.824	0.851	0.832	0.542
Mixtral 8x7B Instruct	0.666	0.867	0.698	0.290
Nova Pro	0.823	0.868	0.834	0.521

heuristics to the remaining threads: those containing templated bot message patterns, and those consisting entirely of bot commands were also removed. The filtered feedback threads were manually reviewed, and accounts identified as human were retained.

4 Results

4.1 RQ1: LLM Accuracy in Classifying OSS Feedback

4.1.1 Sentiment. **Potential Harmful OSS Feedback Dataset:** Table 3 summarizes the performance of all evaluated LLMs against the 300 labeled feedback threads from the harmful OSS dataset. While all models achieved relatively high precision overall, negative precision was notably lower, indicating less effective classification performance for negative feedback. Figure 3 shows the confusion matrices for the best- and worst-performing models by F1-score.

General OSS Feedback Dataset: Table 4 summarizes the performance of all LLMs on sentiment classification, measured against the 500 labeled feedback threads from the sampled general OSS dataset. Negative precision varies significantly across models. Figure 4 show the confusion matrices for the best- and worst-performing models by F1-score.

Across models, certain feedback threads with a negative ground truth were frequently misclassified as neutral or positive. In particular, many models misclassified these comment types:

Excessive Nitpicks (8 models misclassified). Nitpicking refers to comments in a feedback thread that focus on minor issues or formatting details. For example, one thread had many nitpicks, including: "Unneeded initialization" (Comment #3), "Please move the comment after line 71 and remove blank comment lines" (Comment #15), and "Please add final dot, Below too" (Comment #17). Although each comment is brief and focuses on minor issues, repeated nitpicking is considered a reason for negative sentiment in our annotation scheme.

Harmful Comment in Long Thread. Models frequently misclassified negative feedback threads when the harmful comment is embedded in an extensive discussion. An example is this comment which was comment 18 out of 19 in a thread: "What?!? That is a horrible suggestion. Seriously, just horrible. I'm just a contributor here, but I've put a lot of time into this." (9 models misclassified)

Feedback with Neutral Ground Truth Misclassified as Negative. In contrast, models also misclassified feedback with a firm but neutral tone as negative, often when comments contained corrective or directive phrasing typical in review contexts. For example: "Thanks for contributing. However, your PR doesn't follow our contribution guidelines correctly (e.g. title doesn't match file name, indentation is incorrect) and/or it doesn't provide enough value for the average reader. Closing due to that." (11 models misclassified)

4.1.2 Specificity. Similarly, Table 5 summarizes the performance of all LLMs on specificity classification, measured against the 500

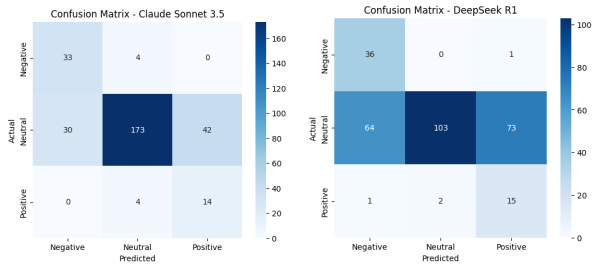


Figure 3: Confusion matrices for the best (Claude 3.5 Sonnet) and worst (DeepSeek R1) models for harmful feedback classification validation

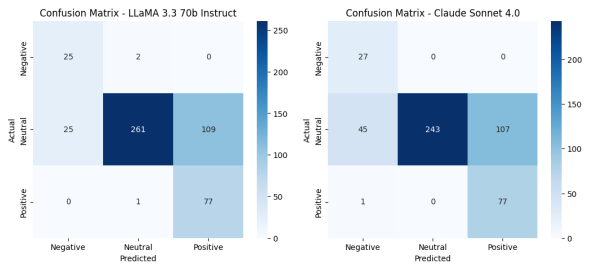


Figure 4: Confusion matrices for the best (LLaMA 3.3 70B Instruct) and worst (Claude Sonnet 4.0) models for sampled general OSS dataset (sentiment)

Table 5: Evaluation metrics for sampled general OSS dataset (specificity)

Model	Recall	F1	Prec.	Highly Spec. Prec.
Claude Sonnet 3.5	0.592	0.580	0.584	0.561
Claude Sonnet 3.7	0.553	0.527	0.534	0.515
Claude Sonnet 4.0	0.614	0.591	0.590	0.593
Command R+	0.576	0.562	0.552	0.498
DeepSeek R1	0.588	0.554	0.545	0.540
Jamba 1.5 Large	0.448	0.421	0.393	0.520
LLaMA 3.1 70B Instruct	0.594	0.568	0.573	0.554
LLaMA 3.3 70B Instruct	0.586	0.571	0.576	0.529
Mistral Large (24.02)	0.570	0.585	0.575	0.504
Mixtral 8x7B Instruct	0.631	0.674	0.636	0.621
Nova Pro	0.526	0.500	0.499	0.500

labeled feedback threads from the sampled general dataset. Figure 5 shows the confusion matrices for the best- and worst-performing models by F1-score. As seen in the confusion matrices, a large share of misclassifications involved confusion with the neutral class, where highly specific or highly unspecific instances were predicted as neutral, and conversely, neutral instances were predicted as highly specific or highly unspecific. By contrast, extreme misclassifications between highly specific and highly unspecific were less frequent.

4.1.3 Best Sentiment–Specificity Model Combination. All 49 sentiment/specificity model pairs were evaluated on the sampled general OSS dataset of 500 feedback threads. The top ten pairs with the smallest χ^2 distances are shown in Table 6. Taking both predictive alignment and output validity into account, the Claude Sonnet 3.5 / Claude Sonnet 3.5 pair emerged as the optimal combination. It produced a distribution closely aligned with the ground truth while maintaining fully valid outputs across all feedback threads with a χ^2 distance of 3.43. While some pairs, such as Claude Sonnet 3.7 / LLaMA 3.3 70B, achieved slightly smaller χ^2 distances (2.45), their predictions included non-negligible output errors, including invalid sentiment classifications (1%), making them unsuitable for large-scale deployment based on the needs of Multitudes.

Figure 6 compares the predicted distribution of the 500 feedback threads generated by the best-performing model pair to the ground truth distribution for the sampled general dataset.

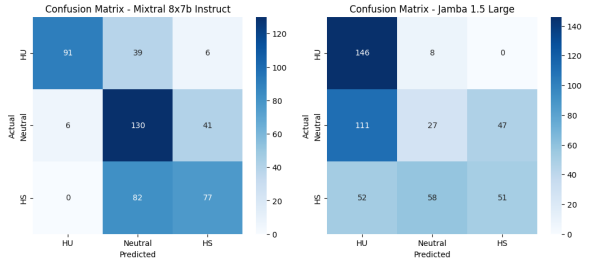


Figure 5: Confusion matrices for the best (Mixtral 8x7B Instruct) and worst (Jamba 1.5 Large) models (specificity)

4.2 RQ2: Distribution of Feedback Quality

The Claude Sonnet 3.5 sentiment/specificity combination, identified as the best-performing pair, was applied to the full large-scale general OSS dataset of 204,333 feedback threads and the sampled industry dataset of 6,573 threads. For the OSS data, the model produced valid classifications for 203,314 feedback threads, with 1019 threads (0.5%) excluded due to invalid or incomplete outputs. 121,922 feedback threads remained after filtering for bot-generated and bot-command-only threads. For the industry data sample, the model produced valid classifications for 6,506 feedback threads, with 67 invalid outputs (1.02%).

For the filtered OSS dataset, the large-scale distribution has notable differences in category proportions when compared to the ground truth. In the ground truth dataset, Highly Specific (32.0%) and Neutral (33.6%) comments were most common, followed by Minimal Review (15.0%), Highly Unspecific (14.0%), and Harmful Feedback (5.4%). The large-scale results therefore indicate an increase in both Neutral (+13.1 percentage points) and Highly Unspecific feedback (+2.0 points), and a decrease in Highly Specific Feedback (-8.0 points). Meanwhile, Harmful Feedback and Minimal Reviews declined slightly (-3.5 points and -3.6 points respectively).

We also observed differences in feedback quality distribution between the OSS and industry datasets. In the OSS feedback distribution, Neutral Specificity comprises almost half (46.7%) of all feedback, but less than a quarter of industry feedback (24.1%). Industry feedback has a slightly higher share of Highly Specific feedback (29.3%; +5.3 points), but also a substantially higher amount of low-informational feedback, with 45.9% of feedback as either Highly Unspecific or Minimal, compared to 27.4% for the OSS feedback. A review of the feedback threads showed these were mostly rubber-stamping reviews e.g, "Nice Job", "LGTM" or "Looks good to me!"

The proportion of harmful feedback remains a small category for both industry and OSS repositories. In the industry dataset, this made up 0.6% of the total dataset, while in the OSS repositories it made up 1.9% of the total dataset. This aligns with prior research that found that harmful feedback was relatively rare, but caused significant issues when it did occur [13]. To assess whether feedback quality distributions differed between settings, we conducted chi-square tests examining the association between feedback quality categories and repository type. Results, shown in Table 7 reveal that differences are statistically significant but the effect sizes (as measured by Cramer's V) are negligible.

5 Discussion

5.1 Implications

Our findings address the accuracy of LLMs in classifying code review feedback within sentiment and specificity dimensions, and reveal insights into the landscape of feedback quality across popular Github OSS repositories as well as industry repositories.

In both of the sentiment experiments, Claude Sonnet 3.5 consistently performed the best among all evaluated models. It achieved the highest F1-score 0.767 in the potential harmful OSS feedback dataset and one of the highest F1-scores (0.835) in the sampled general OSS feedback dataset. While specialized, fine-tuned SE-domain models still achieve higher peak accuracy with BERT-based models reaching F1-scores as high as 0.95 for incivility detection [16] and

Table 6: Top 10 Sentiment/Specificity Model Pairs on Sampled Dataset

Sent. Model	Spec. Model	Sent. Inv. %	Spec. Inv. %	High Spec.	Neu. Spec.	High Unspec.	Minimal	Harmful	χ^2	p
Ground Truth	**Ground Truth**	**0**	**0**	**160**	**168**	**70**	**75**	**27**	**0.00**	**1.00**
Claude Son. 3.7	LLaMA 3.3 70B	0.01	0	151	165	81	74	29	2.45	0.784
Claude Son. 3.7	Claude Son. 3.5	0.01	0	146	170	81	74	29	3.14	0.678
Claude Son. 3.5	Claude Son. 3.5	0	0	145	169	80	75	31	3.43	0.634
Claude Son. 3.5	LLaMA 3.3 70B	0	0	150	161	83	75	31	3.92	0.561
Claude Son. 3.5	Mistral Large	0	0	137	205	52	75	31	16.68	0.005
Claude Son. 3.7	Mistral Large	0.01	0	139	208	50	74	29	18.16	0.003
Nova Pro	Claude Son. 3.5	0.042	0	137	165	77	73	48	20.45	0.001
Nova Pro	LLaMA 3.3 70B	0.042	0	143	156	80	73	48	20.48	0.001
Mistral Large	Claude Son. 3.5	0.042	0	137	161	78	76	48	20.86	0.001
Mistral Large	LLaMA 3.3 70B	0	0	141	154	81	76	48	21.50	0.001

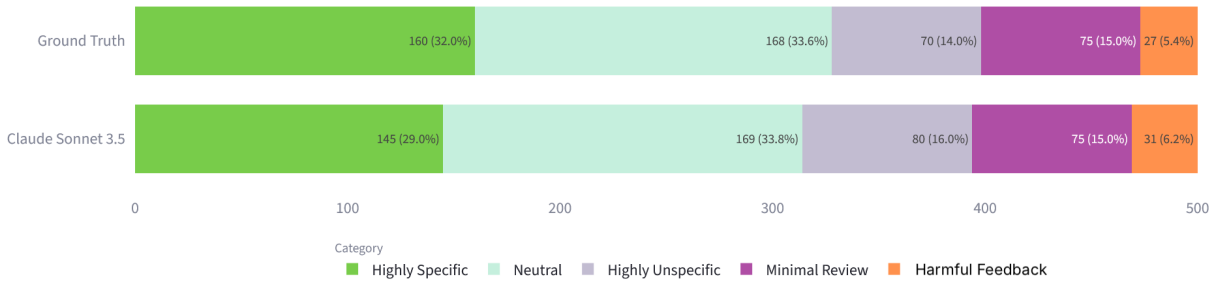


Figure 6: Predicted distribution on the 500-feedback sampled dataset compared with ground truth

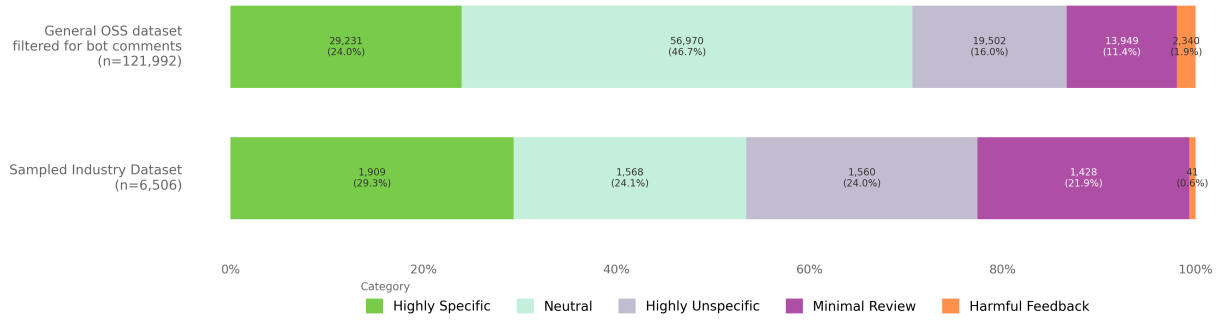


Figure 7: Predicted distribution on the general OSS dataset and sampled Industry dataset

ToxiCR achieving 0.889 for toxicity detection [34], the LLMs demonstrated robust performance across a diverse range of 30 repositories without any task-specific training. This generalizability represents a significant advantage, as domain-specific models are typically optimized on curated datasets with well-defined toxicity labels that may not fully capture the variability of real-world code review comments. Our analysis of the incorrect predictions highlighted a key limitation of LLMs in zero-shot settings, that they struggle with subtle, context-dependent cues in technical critique, often over-flagging firm but professional feedback as harmful.

Unlike detection of toxic or harmful comments, which has been widely examined in software engineering literature, the classification of feedback specificity remains largely unexplored. To the best of our knowledge, no prior work has explicitly modeled or

Table 7: Chi-Square Test of Independence Between OSS and Industry feedback categories

Feedback Category	χ^2	p-value	Cramer's V
Highly Specific	97.1	<0.001	0.0275
Neutral	1270.9	<0.001	0.0995
Highly Unspecific	287.3	<0.001	0.0473
Minimal Review	647.2	<0.001	0.0710
Harmful Feedback	55.6	<0.001	0.0209

measured the specificity of code review comments. One study proposed a technique to classify certain sub-elements of specificity: asking questions, identifying defects, and providing suggestions [36]. Prior studies have mostly focused on classifying related constructs such as usefulness [8, 28], often associating usefulness to code change-triggering capability. However, specificity is an important dimension of constructive feedback that directly influences how actionable and valuable review comments are to developers [5]. By formalizing specificity as a measurable construct and demonstrating both the potential and limitations of LLMs in operationalizing it, this study opens new avenues for automated feedback quality assessment in the code review process. This contribution provides a foundation for future tools that could help maintainers and contributors improve communication effectiveness, ultimately enhancing collaboration quality in software teams.

For specificity classification, the models achieved moderate overall performance, with Mixtral 8x7B obtaining the highest F1-score of 0.674. This level of accuracy reflects a reasonable but imperfect ability to differentiate between levels of feedback specificity. For comparison, prior work has reported that comment usefulness can be predicted with approximately 66% accuracy [28]. Most errors occurred along the borderline categories: neutral versus highly specific and neutral versus highly unspecific, where distinctions often hinge on the reviewer's reasoning or the availability of contextual information. Extreme misclassifications between the highly specific and highly unspecific were less common. Although our labeling guidelines define clear specificity categories and definitions, what constitutes "vital" or high-value information in a review comment remains inherently subjective and can vary between reviewers, projects, and organizations [29]. This subjectivity likely contributes to borderline misclassifications by the models.

Collectively, these findings underscore both the potential and limitations of using LLMs to evaluate social and communicative aspects of software engineering at scale. While the models show promising accuracy across sentiment and specificity, they are constrained by context and the nuanced nature of human feedback. Analyses also reveal that open-source feedback is predominantly neutral or low in specificity. These insights suggest opportunities to enhance the quality of OSS feedback and to guide the design of LLM-based feedback analysis tools, potentially supporting more constructive and inclusive review practices.

As LLMs take on a larger share of coding, testing, and reviewing tasks, code review practices will inevitably evolve. Recent work found that LLM and human code review comments tend to focus on different types of issues and generally complement each other [17]. Humans remain central to the code review process and are likely to continue evaluating aspects that require contextual understanding, architectural reasoning, and strategic alignment. The LLM-based classifications studied here can complement, not replace, human reviewers by revealing patterns, highlighting potential issues, and supporting more constructive interactions during the development process.

5.2 Deployment Insights

Alongside our experiments, we gathered observations from a deployed implementation of our LLM-based code review classification feature in Multitudes's developer productivity tool. This provided early insights into the practical use of our research. Initial feedback from practitioners and usage data showed strong engagement. The feature became one of the top three most-used features and maintained this position for months. User feedback revealed a key insight: while sentiment classifications required careful framing and raised user sensitivity concerns, specificity classifications proved more practically valuable with minimal friction.

Industry practitioners expressed frustration when they believed that their feedback had been misclassified as harmful. Even though the language in the deployed feature used the softer classification of "negative" (instead of "harmful" or "destructive"), it was presented in red, and some users pointed out that there's less confidence in an LLM classification versus a human one. This suggests that while LLMs show promise for aggregate-level analysis of feedback trends, the use case of evaluating individual comments warrants caution.

To address this challenge, the "negative" label was reframed as "needs attention" and paired with a yellow visual indicator instead of red. This framing explicitly acknowledges the uncertainty in LLM classifications and encourages users to review the feedback themselves rather than treating the model's judgment as definitive. We also provided users with the ability to relabel reviews when they disagreed with the model's assessment.

In contrast, specificity labels demonstrated meaningful practical value with fewer user sensitivity concerns. For example, one organization reviewed feedback specificity distributions across its developers and discovered that a senior developer was providing more specific feedback than they were receiving, which showed a gap in support for their own learning. This insight prompted discussions about mentorship for them, illustrating how specificity metrics can drive actionable improvements in code review culture.

Our key learnings from this are:

- For data/AI features: Do user research using actual user data, rather than dummy data, since the user reactions came from seeing their own data in the "negative" bucket.
- Ensure that users have the ability to relabel/give feedback on classifications they disagree with.

5.3 Threats to Validity and Limitations

While this research provides large-scale, data-driven insights into the quality and characteristics of code review feedback, several threats to validity and limitations should be acknowledged.

The OSS data was sourced exclusively from GitHub, with repository selection based on popularity and recent activity, which may not fully reflect the diversity of OSS development across platforms or less active projects.

For RQ1, ground truth sentiment labels were independently annotated by two researchers, achieving high inter-coder reliability (Cohen's Kappa = 0.834). In contrast, specificity labels were not dual-coded, which may have introduced subjectivity and inconsistency despite agreed guidelines. Future work should use independent annotation and report reliability to strengthen the validity of specificity labels.

All LLM experiments used zero-shot prompting without fine-tuning, which may underestimate performance and create sensitivity to prompt variations. Performance differences across models

may reflect training data biases rather than intrinsic capabilities. While precision, recall, and F1-score offer standardized performance measures, they don't capture all aspects of feedback quality relevant in practice, such as tone, constructive phrasing, or pedagogical value. Future work should incorporate human evaluation or qualitative analysis to complement these quantitative metrics.

Although Claude Sonnet 3.5 was identified as providing the best sentiment/specificity performance and was therefore used in the subsequent OSS and industry feedback quality distribution analyses, it was retired on 5 January 2026. Consequently, replication or extension of this work using the same model will not be feasible.

Bot-comment filtering was applied to both datasets, including manual validation. However, some bot comments may remain in either set and could impact the results.

5.4 Future Work

Building on the limitations and insights identified in this study, several avenues for future research and practical improvements emerge. First, incorporating contextual information into LLM evaluations could improve accuracy, particularly for subtle, context-dependent forms of negativity and borderline specificity. This could include linking review comments to the corresponding PR metadata, code changes, and prior discussion.

Additionally, to address the limitations of zero-shot LLMs in SE feedback, future work could investigate few-shot learning, calibration methods, or reinforcement learning approaches on a corpus of human-annotated code review comments. Such a study could compare the trade-offs between zero-shot models, which offer scalability and generalizability for large-scale deployment, and few-shot or task-adapted models, which may better capture nuanced or context-dependent feedback.

The sentiment classification could be extended to examine the more granular categories of harmful feedback, such as judgmental, vague, or harsh language, which were labeled in the current ground truth and also predicted by the LLMs, but not systematically evaluated. Applying this finer-grained analysis could provide deeper insights into patterns of harmful feedback in software engineering, informing both LLM development and the design of tools for promoting inclusive and effective code review practices.

Furthermore, we saw clear differences in the distribution of feedback between OSS and industry, with closed repositories showing significantly more highly specific feedback but also more minimal reviews. In contrast, both datasets had a similar distribution of harmful feedback. The higher prevalence of specific feedback in industry repositories could be due to differences in organizational norms and workplace culture or the more episodic contributions in OSS repositories, which may make feedback for the purpose of learning less prevalent. Potential reasons for the greater number of minimal reviews in industry settings could be perfunctory sign-offs to meet compliance requirements or a result of more detailed review conversations that occurred outside of GitHub. Future work could investigate the reasons for these differences.

Although the deprecation of Claude Sonnet 3.5 limits direct reproducibility, it reflects the rapid evolution of LLMs toward stronger reasoning capabilities. While the successive Claude generations evaluated in our study (3.7 and 4.0) did not show consistent improvement on our classification tasks, more recent models such as Claude Sonnet 4.6 and Opus 4.6 have demonstrated substantial gains across reasoning benchmarks [2]. Such advances may prove particularly beneficial for the more challenging aspects of feedback classification, such as distinguishing borderline specificity categories. We therefore consider it valuable to replicate this study with newer-generation models. Our industry partner has already transitioned to a more recently-released LLM model in the deployed setting, observing marginal improvements in classification accuracy.

While we shared some deployment insights, future work could more systematically study the impact of this feature in real world environments. Such studies could also investigate changes in feedback quality over time with continued use of the tool.

6 Conclusion

We investigated the feasibility of using LLMs to assess code review feedback quality at scale, focusing on two complementary dimensions: sentiment and specificity. Across more than 300,000 feedback threads from both open-source and commercial software projects, we demonstrated that modern LLMs can classify feedback with reasonable accuracy in zero-shot settings, enabling large-scale analysis of communication patterns that had previously been impractical.

Our findings reveal that most OSS code review feedback is neutral or low in specificity, with highly detailed and overtly harmful feedback comprising only a small proportion of interactions. In contrast, industrial repositories exhibit markedly higher levels of highly specific feedback alongside a greater prevalence of minimal reviews, highlighting fundamentally different collaboration dynamics between open and closed development contexts. These differences suggest that OSS data should be used with caution when drawing inferences about industrial software practices, as reliance on OSS alone risks obscuring key organizational behaviors.

Beyond empirical insights, this work provides evidence that LLM-based feedback quality analysis can be deployed in production settings, but only when paired with careful design choices that respect uncertainty and developer trust. Our deployment experience shows that aggregate-level analysis is more readily accepted than individual-level judgments, and that framing model outputs as indicators for reflection, rather than definitive evaluations, is essential for practitioner adoption.

Taken together, this paper contributes (1) a scalable methodology for analyzing feedback quality across sentiment and specificity, (2) the first large-scale empirical comparison of code review feedback quality in OSS and industry settings, and (3) practical lessons from deploying LLM-based feedback analysis in a real-world tool. While challenges remain in handling nuanced and context-dependent feedback, our results demonstrate that LLMs already offer meaningful value for understanding and improving software code review.

Acknowledgments

Thanks to Vivek Katial and Divya Roshinkumar for contribution to the study design and analysis. The study was partially supported by a Rutherford Discovery Fellowship, administered by the Royal Society Te Apārangi, New Zealand.

References

[1] Sharif Ahmed and Nasir U Eisty. 2023. Exploring the advances in identifying useful code review comments. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–7.

[2] Artificial Analysis. [n. d.]. Independent analysis of AI. https://artificialanalysis.ai/ Accessed April 1, 2026.

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[4] Krzysztof Baciejowski, Damian Garbala, Szymon Zmijewski, and Lech Madeyski. 2023. Are code review smells and metrics useful in pull request-level software defect prediction? In *Developments in Information and Knowledge Management Systems for Business Applications: Volume 6*. Springer, 27–52.

[5] Robert A Baron. 1988. Negative effects of destructive criticism: Impact on conflict, self-efficacy, and task performance. *Journal of Applied Psychology* 73, 2 (1988), 199.

[6] Frank D Belschak and Deanne N Den Hartog. 2009. Consequences of positive and negative feedback: The impact on emotions and extra-role behaviors. *Applied Psychology* 58, 2 (2009), 274–303.

[7] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2016. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering* 43, 1 (2016), 56–75.

[8] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*. IEEE, 146–156.

[9] Junkai Chen, Zhenhao Li, Qiheng Mao, Xing Hu, Kui Liu, and Xin Xia. 2025. Understanding practitioners' expectations on clear code review comments. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1257–1279.

[10] Jithin Cheriyan, Bastin Tony Roy Savarimuthu, and Stephen Cranefield. 2021. Towards offensive language detection and reduction in four software engineering communities. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. 254–259.

[11] Emre Doğan and Eray Tüzün. 2022. Towards a taxonomy of code review smells. *Information and Software Technology* 142 (2022), 106737.

[12] Eduardo Witter dos Santos and Ingrid Nunes. 2018. Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. *Journal of Software Engineering Research and Development* 6, 1 (2018), 14.

[13] 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 (ICSE)*. 174–185.

[14] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and MA Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Information and Software Technology* 114 (2019), 37–54.

[15] 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.

[16] 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.

[17] Saul Goldman, Hong Yi Lin, Jirat Pasuksmit, Patanamon Thongtanunam, Kla Tantithamthavorn, Zhe Wang, Ray Zhang, Ali Behnaz, Fan Jiang, Michael Siers, Ryan Jiang, Mike Buller, Minwoo Jeong, and Ming Wu. 2025. What Types of Code Review Comments Do Developers Most Frequently Resolve?. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 3760–3765.

[18] Mahmut Furkan Gön, Burak Yetiştiren, and Eray Tüzün. 2024. Towards unmasking lgtm smells in code reviews: A comparative study of comment-free and commented reviews. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 163–174.

[19] 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.

[20] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. 352–355.

[21] Mia Mohammad Imran, Robert Zita, Rebekah Copeland, Preetha Chatterjee, Rahat Rizvi Rahman, and Kostadin Damevski. 2025. Understanding and predicting derailment in toxic conversations on GitHub. *arXiv preprint arXiv:2503.02191* (2025).

[22] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071.

[23] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2014. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. 192–201.

[24] 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.

[25] Shyamal Mishra and Preetha Chatterjee. 2024. Exploring ChatGPT for toxicity detection in GitHub. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering (ICSE): New Ideas and Emerging Results*. 6–10.

[26] Rajshakhar Paul, Amiangshu Bosu, and Kazi Zakia Sultana. 2019. Expressions of sentiments during code reviews: Male vs. female. In *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 26–37.

[27] S Pirani. 2024. Navigating the complexity of sample size determination for Robust and Reliable Results. *International Journal of Multidisciplinary Research & Reviews* 3, 02 (2024), 73–86.

[28] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 215–226.

[29] Pooja Rani, Arianna Blasi, Nataliia Stulova, Sebastiano Panichella, Alessandra Gorla, and Oscar Nierstrasz. 2023. A decade of code comment quality assessment: A systematic literature review. *Journal of Systems and Software* 195 (2023), 111515.

[30] Jaydeb Sarker. 2022. Identification and mitigation of toxic communications among open source software developers. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–5.

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

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

[33] 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.

[34] 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 (2023), 1–32.

[35] Jerry Wei, Le Hou, Andrew Lampinen, Xiangning Chen, Da Huang, Yi Tay, Xinyun Chen, Yifeng Lu, Denny Zhou, Tengyu Ma, et al. 2023. Symbol tuning improves in-context learning in language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 968–979.

[36] Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. Evacrc: Evaluating code review comments. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 275–287.

[37] Audrey You, Jingyi (Jenny) Wang, Youxiang Lei, Lauren Peate, and Kelly Blincoe. 2026. Replication package for "Assessing Harmful Comments and Specificity in Code Review Feedback at Scale using Large Language Models". https://doi.org/10.5281/zenodo.19382051