Analyzing and Detecting Toxicities in Developer Online Chatrooms: A Fine-Grained Taxonomy and Automated Detection Approach

Junyi Tian*, Lingfeng Bao*§, Shengyi Pan*, Xing Hu*
*The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China
{junyt,lingfengbao,shengyi.pan,xinghu}@zju.edu.cn

***Abstract*—Developer online chatrooms serve as vital connections among developers in Open Source Software (OSS) projects. However, the presence of impatient and unfriendly users can disrupt the harmony within these chatrooms, leading to negative consequences such as reduced activity and attrition. To address this, we aim to first understand toxicity in developer chatrooms and further develop automated detection techniques. We collect chat messages from three active Gitter chatrooms, and conduct an in-depth analysis at the discussion thread level to examine intent and sentiments. Using a card-sorting method, we create a fine-grained taxonomy with seven toxicity categories and manually annotate a dataset of 5,158 threads. This enables us to gain insights into the nature of toxicity and identify shortcomings in existing detection methods. We further propose an automated binary toxicity detection approach that integrates textual features, non-textual features, and negative sentiment features derived from a Large Language Model (LLM). Our experimental results demonstrate an average F1-Score of 0.546, representing a significant 57.8% improvement over the best-performing baseline. We also validate the effectiveness of incorporating non-textual and negative sentiment features.**

***Index Terms*—toxicity detection, developer online chatrooms**

I. Introduction

With the growth of Open Source Software (OSS), communication channels like GitHub issues, Stack Overflow, and Gitter chatrooms [1] are vital for developers to share and learn. Constructive discussions drive progress, but there is also a small fraction of toxic remarks (e.g., harassment, conflict) within these communities. Toxicity refers to disruptive or inappropriate content [2], negatively impacting community members and projects [3], [4]. To foster a harmonious and constructive environment in OSS communities, research addressing toxic remarks is crucial.

Online chatrooms have been a prevalent platform for fostering connections within communities. They are becoming increasingly popular because of real-time and interactive discussions [5]. There are several popular online developer chat platforms, including Gitter, Slack, and Discord. Among them, Gitter stands out due to its direct connection with open-source projects on GitHub and ready access to its historical data [5]–[7]. However, the toxicity in online chatrooms has not received adequate attention.

Toxicity is a pervasive issue in online chatrooms, despite the presence of conduct guidelines. Some users choose to disregard these guidelines, leading to toxic discourse [4], [8]. This poses several challenges for administrators in monitoring chatrooms. Firstly, disentangling discussion threads from unstructured message streams and identifying toxic content is a labor-intensive task. Secondly, the rapid turnover of messages makes it difficult to promptly detect and remove problematic content. Lastly, a lack of timely intervention can disrupt the harmonious atmosphere and alienate users. Therefore, there is an urgent need for in-depth analysis and automated detection methods to alleviate the burden on administrators and improve the chat environment [2], [9]. Real-time toxicity detection can foster a positive atmosphere and encourage active participation from developers.

Existing sentence-level methods for toxicity detection [3], [9]–[12] have limitations in developer online chatrooms. These limitations stem from the focus on sentence-level analysis and the lack of customization for chatroom environments [6], [7], [13], [14]. Developer chatrooms primarily involve technical discussions, follow a Q&A format [13], [15], and exhibit tangled discussions with multiple rounds of information exchange [13], [14]. Given the high contextual coherence and technical relevance, thread-level toxicity detection would be more valuable than existing methods.

In this study, we explore toxicity in developer chatrooms and create a thread-level discussion dataset from representative chatrooms on Gitter. We develop a suitable toxicity taxonomy for developer chatrooms, define clear classification criteria, and analyze each toxicity category to understand their characteristics. To address the lack of thread-level datasets, we manually annotate a dataset reflecting real toxicity. Using the fine-grained taxonomy and dataset, we evaluate existing toxicity detection methods and identify reasons for false detection. We propose a thread-level toxicity detection approach that combines textual features from neural networks, handcrafted non-textual features, and negative sentiment features from a Large Language Model (LLM). It performs the binary classification task to determine whether a thread is toxit. We evaluate our approach, compare it with sentence-level methods, and demonstrate the effectiveness of incorporating non-textual and negative sentiment features in toxicity detection.

The contributions of our work are summarized as follows:

- We propose a thread-level toxicity taxonomy designed explicitly for online developer chatrooms and analyze the

§Corresponding author.

characteristics of different categories.

- We collect a comprehensive dataset from chatrooms on Gitter and manually annotate thread-level toxicity labels after disentangling the discussion threads.
- We conduct experiments to compare the performance of our approach with sentence-level baselines and evaluate the effectiveness of non-textual features and negative sentiment features in our method. The experiment results show that our approach significantly outperforms the baselines.
- To foster future works and adhere to good research practices, we provide a replication package [16] of our work.

II. Background & Motivation

A. Toxicity Detection

Toxicity detection identifies and mitigates harmful content in text for a harmonious online environment. Prior studies integrated sentiment analysis, such as VADER [17] in STRUDEL [9], as negative sentiments like anger and disgust may indicate toxicity. Our work also incorporates sentiment analysis. Various widely adopted methods for toxicity detection and sentiment analysis have emerged in recent years, covering general platforms and the software engineering domain.

Regular Expression (Regex) is a straightforward approach that uses certain keywords (e.g., *fuck*, *shit*) to detect toxicity. We use the regex list proposed by Cheriyan et al [3]. Moreover, we refine their regex list based on our dataset. Please refer to the complete regex list in our replication package [16].

Perspective API (PAPI) [11], developed by Jigsaw and Google's Counter Abuse Technology team, aims to address online toxicity and abusive behavior, promoting healthy conversations. Initially, multilingual BERT-based models were trained and then distilled into single-language Convolutional Neural Networks (CNNs) for each language. For this paper, we exclusively utilize the English model and focus specifically on the *Toxicity* attribute with the threshold at 0.5.

MAX Toxic Comment Classifier (MAX Classifier) [10] was created as a component of the IBM Developer Model Asset Exchange, utilizing a pre-trained BERT model for multi-label classification [18]. It was fine-tuned on the Toxic Comment Classification Dataset, which was sourced from the famous Toxic Comment Classification Challenge [19]. The MAX Classifier categorizes texts into six toxicity types, using a threshold of 0.5 for each type. If the value for any toxicity type exceeds 0.5, the text is classified as *Toxicity*.

STRUDEL, the first SE domain-specific toxicity detector, utilizes an SVM classifier incorporating pre-trained toxicity detectors like Perspective API [11] and VADER [17]. To mitigate the impact of SE technical terms, a verification step replaces them with neutral alternatives before re-evaluating the modified text for the final outcome.

BERT4SentiSE applies BERT [20] for sentiment analysis in SE domain [21], which shows promising performance in software-related texts. The BERT model is fine-tuned on an extended dataset sourced from Stack Overflow posts, which includes three labels: *Negative*, *Positive*, and *Neutral*. To identify potential instances of toxicity, we utilize the *Negative* sentiment labeled by BERT4SentiSE.

Developer ID	Session message
D1	How did you define a?
D1	Screenshot
D2	Sent above ^^
D1	Your re defining a with itself...
D1	If you change that to a = "word"
D1	Try that
D2	Worked now
D2	That was a stupid mistake
D2	Thanks a lot xD

Fig. 1. Instance misidentified as toxicity at the sentence level.

Developer ID	Session message
D1	how do start doing 100 days of coding for learning python 3 language. plis help
D2	@D1 google your question
D2	this is gitter not google

Fig. 2. Instance of toxicity that goes unrecognized at the sentence level.

B. Motivation

Numerous studies have been conducted to explore sentence-level toxicity detection in SE domain [2]–[4], [12]. However, few studies have been done on toxicity in developer chatrooms which typically involve multiple rounds of interaction. Previous studies of developer chatrooms show that thread is the smallest unit containing a complete intent [7], [14], [22], proving the strong necessity to incorporate contextual information. Therefore, our research focuses on the thread level, recognizing that enhancements in current toxicity detection methods can be pursued from two perspectives:

Reducing instances misidentified as toxicity at the sentence level. In Fig. 1, the highlighted part is a single sentence in the original conversation. The word *mistake* in this sentence, not attributed to anyone, might lead to misinterpretation at the sentence level, resulting in misclassification as *Toxicity*. Previous approaches (introduced in Section II-A) classified it as such. However, considering the context, it is evident that *stupid mistake* refers to the speaker's own error, reflecting self-deprecation or humility. Instances where questioners refer to themselves using words like *stupid*, *dumb*, or *idiot* are common in online chatrooms, leading to many misclassifications by traditional sentence-level methods. Thus, expanding detection to the thread level can effectively reduce such misclassifications caused by unclear intent due to context absence.

Increasing instances of toxicity that go unrecognized at the sentence level. Examples of non-sentential toxicity include seemingly harmless individual statements that reveal aggression when viewed in the context of the entire conversation. For instance, in Fig. 2, D2's response appears innocent but becomes toxic when considering the dismissive attitude towards a novice seeking help in a chatroom meant for advanced chats. Accurate judgment requires understanding the context, as some responders may recommend Google while politely explaining the chatroom's purpose, conveying a friendlier sentiment. Assessing off-topic discussions also requires clear comprehension of the conversation's intent. Thus, analyzing at the thread level improves the identification of such cases missed at the sentence level due to incomplete information.

III. Toxicity Detection at Thread-Level

A. Data Preparation

There are many datasets for sentiment analysis [21], [23]–[27] or toxicity detection in general platforms (e.g.,

TABLE I
THE DETAILED CHARACTERISTICS OF SELECTED CHATROOMS

Chatroom	Messages	Threads	Toxicity	Duration
Angular	1,135,913	1,011	21	2015/03 – 2022/07
Node.js	124,462	1,014	20	2014/12 – 2022/07
Python	28,399	3,133	60	2016/03 – 2022/07

Wikipedia [19], Twitter [28]). Among the limited datasets related to communications between developers, most of them are categorized based on the communication elements towards specific platforms, such as issues/comments of GitHub [2], [8], [9]. To the best of our knowledge, none of the existing works investigates toxicity in developer chatrooms. Thus, we build our own dataset by 1) collecting thread-level chat data, 2) building the thread-level taxonomy of toxicity in developer chatrooms using card sorting, and 3) manually labeling the chat data with the built toxicity taxonomy.

Gitter is a popular developer chatroom platform with numerous active users in open source projects [5]–[7], offering ample chat data for our research. However, toxic conversations are rare, making manual labeling costly. To gain insights into different types of toxicity, we conduct a preliminary study to select projects with a higher volume of toxic threads.

1) Crawling raw data: there are 24 categories of chatrooms on Gitter [29]. We randomly select 100 active chatrooms for each category and crawl all historical messages of them using the official API provided by Gitter.
2) Applying toxic detectors: we use existing toxicity detectors introduced in Section II-A to detect potential toxicities in the raw chat data. We obtain the union of positive (i.e., toxic) results from all the detectors and calculate the final proportion of potential toxicities for each chatroom.

After preliminary detection, we tend to select chatrooms with higher proportions of potential toxicity. To enhance the representativeness of the dataset, we take into account both the category and scale of the chatrooms. We categorize the scales into four levels, including small-scale ($< 10^4$ messages), medium-scale ($10^4 \sim 10^5$ messages), large-scale ($10^5 \sim 10^6$ messages), and extra-large-scale ($> 10^6$ messages). Due to the small number of potential toxicity in small-scale chatrooms, we focus our selection on the remaining three scale levels. Among these, we choose chatrooms with the highest proportion of potential toxicity, while ensuring they represent different development categories. Finally, the selected chatrooms are **Angular**, **Node.js**, and **Python**. The summarization of the selected chatrooms is shown in Table I.

B. Manual Labeling at Thread-Level

Our goal is to create a dataset at the thread level, where we first segment the historical messages from Gitter into discussion threads and further assign labels to each of these threads. We employ the disentanglement method proposed by Ehsan et al [15], which demonstrates promising performance in segmenting Gitter chatrooms. Considering the high cost of manual annotation, we conduct full manual annotation on the Python chatroom, which has a smaller number of messages. As for the larger Angular and Node.js chatrooms, we randomly select 1,000 threads as representative samples for manual annotation, respectively. Table I presents the number of messages and threads available in each chatroom.

To build a comprehensive thread-level taxonomy of toxicity for developer chats, we employ the open card sorting method [30], which has been widely utilized in various previous studies (e.g., identify the potential information types in developer chatrooms [14], identify the potential toxicity types in GitHub issues [2]). We assign a separate card to each discussion thread. The labeling of each card is a collaborative effort between the first two authors. We also provide the detection results at the sentence level as some sort of reference for annotators. The entire process involves two steps:

Step 1: We first use 30% of the card. The first two authors code these threads separately. Miller *et al.* [2] categorized GitHub issue threads into five toxicity classes based on the problems discussed and the project context. This is similar to our work on Gitter discussion threads. Therefore, we use their taxonomy as a foundation and refine it to create our own taxonomy. Firstly, the annotators are instructed to carefully review the content of each discussion thread and label them into toxicity according to the criteria in [2]. If the thread is labeled as *Toxicity*, it is further classified into toxicity categories proposed in [2], otherwise only labeled as *Non-toxicity*. In cases where the threads do not fit into the existing categories but are considered toxic to the chatroom by the annotators, they are allowed to propose a new category that they believe is most appropriate accordingly. After that, the two authors discuss the similarities and differences between GitHub issues and chatroom discussions. Based on these findings, they refine the taxonomy and resort to the cards. Finally, a new taxonomy for chatroom discussion threads is built as shown in Table. II, which has the description of each category and the rule to sort threads into each category. **Step 2:** According to the guidelines presented in Table. II, the remaining threads are marked independently by two authors. The agreement between the two authors is assessed using Cohen's Kappa coefficient [31], resulting in a Kappa value of 0.75, indicating a strong consensus. In cases where disagreements arise, the two authors engage in discussions to reach a mutual decision. The entire annotation process costs 672 person-hours.

C. Taxonomy of Toxicity on Gitter

In this section, we provide detailed explanations for each toxicity category in our built taxonomy and the criteria used for labeling. As shown in Table II, we divide the toxic threads into eight categories:

Rudeness. Some users tend to use dirty words, such as *shit* and *fuck* in verbal expression, which can create a sense of discomfort or displeasure for other users and can thus be categorized as toxic. At the sentence level, any sentence containing dirty words is easily labeled as toxic. However, based on our findings at the thread level, the use of certain profanities without malicious intent may not necessarily impede the progress of a conversation. The overall tone of the conversation remains friendly. Therefore, we do not classify such conversations as toxic. We only label a thread as *Toxicity* when the proportion

TABLE II
THE TAXONOMY OF TOXICITY ON GITTER FOR DEVELOPER DISCUSSION THREADS

Category	Description	Rule	Support
Rudeness	Rudeness is messages that have dirty words but not too many and are less aggressive.	Includes dirty words and rude words that give people a bad feeling, but do not reach the level of severe verbal abuse or harassment.	28
Harassment	Harassment is messages that are harmful and offensive to other users.	Includes sexual language and imagery, racial discrimination, deliberate intimidation, stalking, name-calling, unwelcome attention, libel, and any malicious hacking.	14
Argument	Argument is messages that contain quarrel between different people.	Includes conflict, quarrel, and severe words between different users.	9
Criticism	Criticism is messages in which a user criticizes something or someone, or expresses strong dissatisfaction with something or someone.	Includes harsh criticism, disagreement, strong dissatisfaction, or sarcasm messages from users.	19
Complaint	Complaint is messages in which people express dissatisfaction with something or someone in a less intense way.	Includes complaints and dissatisfaction from users.	17
Command/Arrogance	Command/Arrogance is the commanding, indicative messages of what someone or something must do in a condescending tone.	Includes command, indicative messages, dictatorial messages, and some absolute messages.	29
Spamming	Spamming is messages that are off-topic or too many posted once to bother other users.	Includes posting off-topic messages to disrupt discussions, promoting a product, soliciting donations, advertising a job, or flooding discussions with files, text, or images.	26
Others	Others is toxicities that cannot be divided into the types above.	Includes messages which are indeed toxicities but hard to be sorted into types above.	2
Non-toxicity	Non-toxicity is threads that do not have any toxicities mentioned above.	Includes messages that are friendly, welcoming, respectful, technical, and general, and do not lead to bad feelings.	5,057

of explicit profanity (e.g., the dirty words) exceeds 20% in discussion or when it contains evident malicious intent, as illustrated in the dialogue example in Fig. 3.

Developer ID	Session message
D1	i'm not sure why my vsc is being an insufferable piece of shit rn
D2	every other IDE: *works completely fine*
D1	VSC: *Screw you, I'm out*

Fig. 3. Instance of toxicity that can be categorized as *Rudeness*.

Harassment. Harassment involves directing intense malice and offensive behavior towards a specific individual or group, going beyond mere rudeness. It encompasses actions such as discrimination, insults, and the use of sexual language, as demonstrated in the dialogue example Fig. 4, which includes toxic remarks implying gender discrimination. We define the scope of harassment based on our assessment with the specific details provided in Table II. It is important to note that harassment is considered a more severe category than rudeness. Therefore, The priority of *Harassment* is higher than that of *Rudeness* in our annotation.

Developer ID	Session message
D1	Look at all the masses in the stands
D1	No, Shady man, don't massacre the fans
D2	@D1 That's a very misogynistic song you are referencing there.

Fig. 4. Instance of toxicity that can be categorized as *Harassment*.

Argument. The strong interactive conversations between users also lead to friction and conflict. When inspecting the discussion threads, we do come across an instance where a heated argument between two users resulted in one person permanently leaving the chatroom. Thus, disharmonious conflicts could lead to the loss of participants. Furthermore, we also observe that not all conflicts necessarily involve the use of explicit profanity, highlighting the need to consider the overall conversation (i.e., with the comprehensive contexts) to assess the occurrence of conflicts. We only label discussion threads as toxic when extreme disputes arise due to differing views, as demonstrated in the dialogue example in Fig. 5.

Criticism. In open-source chatrooms, users often comment on open-source software. Most of these evaluations are impartial, constructive, and helpful, while a minority of individuals simply make derogatory and dismissive remarks like calling certain software *shit* or using harsh language that is not easily accepted by others. The latter, to some extent, constitutes malicious criticism, which is detrimental to the development of open-source chatrooms. Therefore, we categorize such instances as a form of toxicity, as shown in Fig. 6.

Developer ID	Session message
D1	I have developed a face recognition model. for local testing the model is loaded for each query
D1	but I need to serve multiple query at a same time withoout reloading the model
D1	how can that be possible
D2	Facial recognition that's BS @D1 no programmer would visit this place for help. They'd be willing to give help
D3	@D2 you are assuming he is a programmer by trade, there are many data scientists and others who have worked with the open source tools to create models but don't understand how to apply those in a programmatic fashion to other problem domains
D2	@D3 your Point? If he/she worked on machine learning they'd understand the math and programming involved with making a facial recognition system. Those aren't easy if he's trying to Build his own version of TRIPWIRE I wish him the best of luck
D2	@D3 this wasn't your argument to try and solve
D3	not sure it was an argument in the first place?
D3	you were being rather degrading and confrontational and I was simply trying to figure out why
D2	@D3 tripwire is based off a facial recognition algorithm that marks a person if they're eliciting suspicious behavior and what's the issue with being degrading, that's how I became good at programming. Someone criticizes my code I make it better

Fig. 5. Instance of toxicity that can be categorized as *Argument*.

Developer ID	Session message
D1	ok but tkinter is very complicated to use
D1	for gui
D2	I used Tkinter once and it was terrible indeed
D1	yeah i used it several times & it sucks & i can't find any other alternative
D1	i guess i have to work with tkinter somehow

Fig. 6. Instance of toxicity that can be categorized as *Criticism*.

Complaint. *Complaint* is chosen to complement *Criticism*. When checking the threads, we find that while some messages may not be harshly worded, they still exhibit a certain degree of dissatisfaction and other negative sentiment to somebody or something without any constructive content, as seen from the dialogue example in Fig. 7. The toxicity of this type of thread is much lower compared to *Criticism*, but it still probably brings a negative impact. After careful consideration, we decide to label such threads as a form of toxicity as well.

Developer ID	Session message
D1	don't you hate it when you try to help someone and they don't even tell you to eff off? Do I come across as rude or something? :smile:
D2	it happens
D1	Too frequently. By the way, Happy New Year.
D2	to you too
D3	sometimes people just get busy or they find the solution and off they go :P
D1	They come and get what they want and leave courtesy out the door. Damn kids :smile:

Fig. 7. Instance of toxicity that can be categorized as *Complaint*.

Command/Arrogance. The rationale for this category is generally from [2]. They proposed *Entitled* and *Arrogant* comments are common forms of toxicity. Our research also

validates this conclusion. It often occurs among users who are prone to answer questions. Certain individuals, considering themselves experienced, tend to express authoritarian or directive statements rather than offering advisory opinions, as illustrated in the dialogue example in Fig. 8. This can lead to a sense of arrogance among other users and make the content difficult to accept. Generally, this type of toxicity is subtle and does not involve specific characteristic language. It requires the comprehensive context in the thread to determine whether the speaker's tone exhibits arrogance.

Developer ID	Session message
D1	Anybody online?
D2	I think there's more people interested in webdev than programming so that's why python is empty

Fig. 8. Instance of toxicity that can be categorized as *Command/Arrogance*.

Spamming. Spamming is a frequent source of toxicity in online chatrooms. It has a variety of forms, but the key to it is off-topic. Amount of off-topic messages can impede meaningful and effective conversations, and hide valid information. One of the typical examples is advertisements, which can evoke a sense of aversion among users, as presented in the dialogue segment in Fig. 9. In developer chatrooms, another type of spamming can also cause users to feel uncomfortable. Instead of making regular inquiries, some users inundate the chatroom with an excessive number of images or text to describe the issues they encountered, sometimes resulting in a flooding effect. Most importantly, spamming is difficult to discern from individual sentences and often requires considering the thread context and topic to make an accurate judgment.

Developer ID	Session message
D1	Silence
D1	I silence you]
D1	I silence you all
D2	@D3 checkout @D1
D3	@D2 hey
D3	I don't know what's up with that
D1	@D2 just kidding
D3	@D1 Please stop spamming the room. Please review our code of conduct is available at https://code-of-conduct.freecodecamp.org/.
D1	@D3 , I silence you. Silence. I silence you all. ok ok thatwas the last time, i won't insert nonsense messages. promise.

Fig. 9. Instance of toxicity that can be categorized as *Spamming*.

D. Analysis of Toxicity

When building the taxonomy of toxicity, we find that there are some intersection relationships between different categories, which aligns with the study of [2]. Table II illustrates that the total support for all categories exceeds the number of samples, indicating overlaps. There are some findings: Firstly, we observe overlaps between *Rudeness* and *Criticism* or *Complaint*, where *Criticism* is defined as severe *Complaint*. Notably, *Rudeness* has a higher proportion within *Criticism* (73.68%) than within *Complaint* (less than 50%). This aligns with our expectations. Additionally, minimal overlaps (only one respectively) are found among *Rudeness*, *Harassment*, and *Argument*, suggesting that not all arguments necessarily involve extreme rudeness. Finally, minor overlaps are observed between categories like *Criticism* and *Command/Arrogance*, or *Argument* and *Command/Arrogance*. While our goal is to classify toxicity across different attributes, real-world instances can be complex, often exhibiting multiple features that meet criteria for various categories, leading to multi-label scenarios.

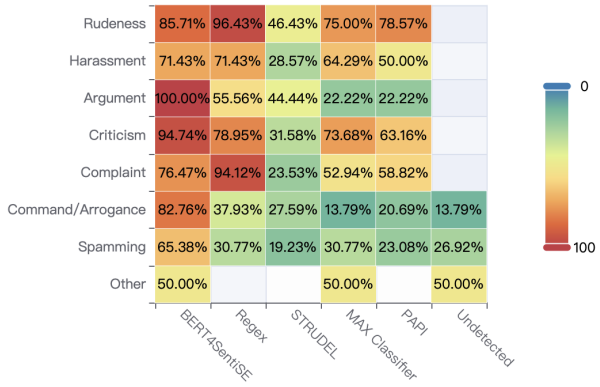

Fig. 10. Percentage of toxicity per type per detector.

Based on manual thread labeling, we further analyze the performance of existing detectors across different toxicity types (as shown in Fig. 10) to understand their shortcomings. Overall, *Command/Arrogance* and *Spamming* exhibit lower coverage rates due to their less apparent characteristics at the sentence level. Conversely, *Rudeness*, *Harassment*, and *Criticism* demonstrate higher coverage rates, likely due to their frequent use of explicit language. However, not all threads in the *Argument* category contain explicit language, highlighting the need to consider user interactions at the thread level. Detector BERT4SentiSE shows high coverage across toxicities, supporting our assumption that toxicity often involves negative emotions. Additionally, All the detectors follow the pattern that higher levels of profanity are more likely to be detected.

E. Trigger of Misidentification

For these sentence-level detectors, we carry out a thread-level analysis to investigate the triggers behind their misclassifications. In Table. III, we calculate the ratio of false positive (FP) to positive (P) of each detector. All detectors show a high rate of misclassification in *Toxicity*, with the lowest one reaching 0.672, while the highest one reaches as high as 0.970. To investigate the triggers of misclassification and potential solutions, we conduct an analysis on samples that are wrongly detected as FP by the existing sentence-level detectors.

TABLE III
THE LABELING DETAILS AND THE RATIO OF FP TO P OF EACH DETECTOR

Detector	Toxicity	Labeled	Ratio of FP to P
BERT4SentiSE	77	2,594	0.970
Regex	57	262	0.782
STRUDEL	28	169	0.834
MAX Classifier	43	179	0.760
PAPI	38	116	0.672

We utilize the same card sorting method [30] as used in establishing the toxicity taxonomy to analyze the triggers, and the process is consistent. Finally, we obtain five types of triggers and the main toxicity detectors influenced by them, as shown in Table. IV:

Abbreviation. This trigger only appears in the MAX Classifier, suggesting a unique case where users commonly use abbreviations during chat, leading to detector confusion.

TABLE IV
TRIGGERS OF MISCLASSIFICATION AND MAIN INFLUENCED DETECTORS

Type	Main Detector	Example
Abbreviation	MAX Classifier	@A ok chcking. @B FYI.
Negativity	BERT4SentiSE, STRUDEL	how do we avoid a file from being edited by user or deleted by mistake ?
Tone	ALL	I know going straight from front end to generator-angular-fullstack was confusing as hell for me at first and I think going straight to django might be the same for newbies
Humility	ALL	Thanks Lucas:). I know this looks silly to you, your probably a master but this stuff is nerve racking in the beginning lol
Technical Terms	MAX Classifier, Regex	hi all, looking for help with POST for JSON API on python. loads and dumps

Negativity within Sentence. In sentence-level sentiment analysis task, *confusion* and *frustration* is always labeled as negative sentiment, which we believe is the reason why BERT4SentiSE and STRUDEL (merging sentiment analysis methods in it) misclassify messages containing negative word as toxicity. It can be solved by incorporating thread-level context information.

Tone. While annotating toxicity, we find that messages containing profanity like "shit" or "suck" may not necessarily be toxic. In some cases, these words are used to express strong emotions rather than intentionally attacking someone or something. Thus, we attribute this misclassification to *tone*. We believe that considering more context, particularly at the thread level, can help resolve this issue.

Humility. Another major cause of misclassification in toxicity involves words like "stupid" or "fool", which may be considered insults when directed towards others but can also be used self-deprecatingly by individuals when asking questions. Incorrectly identifying the intended target of these expressions is a significant factor contributing to misclassification. Detection methods require comprehensive context and understanding of the speaker's role in the conversation (i.e., the asker or the respondents) to accurately determine the intended subject of such words and resolve this issue.

Technical Terms. SE-specific technical terms having a negative impact on the performance of toxicity detectors have been raised in previous studies [9]. We regard this trigger as one type of out-of-vocabulary (OOV) problem [32], [33]. This trigger mainly appears in general detectors, which motivates the need for SE-specific toxicity detectors.

All the above analysis results have provided us with insights on how to establish a more effective toxicity detector.

IV. AUTOMATED THREAD-LEVEL TOXICITY DETECTION

In this study, we propose a thread-level toxicity detector to automate the detection of toxicity in developer chatrooms. The detector is composed of three main components: a text semantic encoder, a large language model (LLM) that facilitates the examination of negative sentiments within texts, and an encoder for processing non-textual features (see Fig. 11). We follow Pan *et al.* [14] to preprocess each thread by replacing special tokens (e.g., code snippet) and merging consecutive messages from the same user. Due to the extremely limited number of available toxic samples, we focus on identifying toxicity without categorizing it into specific types. The fine-grained taxonomy built in Section III is primarily utilized to gain a comprehensive understanding of the characteristics of toxicities in developer chatrooms and the shortcomings of the existing sentence-level detection techniques [3], [9], [12].

A. Textual Features

To extract textual features at the thread level, the $Encoder_{text}$ first encodes each message within the thread and then combines the embeddings of messages to get the embedding representing the entire thread. At the sentence/message level, we leverage BERT [20], one of the most widely used pre-trained models (PTMs) to encode each message within the thread. BERT is pre-trained and can be adapted to various downstream tasks through finetuning. In our case, considering the limited data available, BERT enables us to reduce training efforts and achieve more satisfying results. At the thread level, we choose Bidirectional Long Short Term Memory Network (Bi-LSTM) [34] to capture the contextual information of the entire discussion thread from both forward and backward directions. We concatenate the features from both directions and obtain the final textual feature through `MAX` pooling.

Another challenge we face is the severe class imbalance issue of the dataset, where **Toxicity** samples, serving as positive instances, account for less than 0.2% of the overall data. To address this problem, we adopt the online negative sampling (ONS) [35], [36] strategy during training. Specifically, it involves dynamically re-undersampling negative samples in each training epoch while retaining all positive data. Compared with the traditional undersampling technique (which only undersamples the negatives once before the training starts), online negative sampling allows us to better utilize the abundant negative samples, enabling the model to learn from a larger pool of unseen negative instances while maintaining stable positive instance learning.

B. Negative Sentiment Features from LLM

LLMs like LLaMA [37] and GPT [38] have shown promising performance regarding generation-based tasks. However, their direct applicability to specific classification tasks remains somewhat constrained. Transferring new classification criteria to existing LLMs poses a challenge, making them less suitable for direct task completion [39], [40]. Nonetheless, extant studies have highlighted the exceptional capabilities of LLMs in sentiment analysis [39], and the annotation of negative sentiments proves instrumental in unearthing potential toxicity. Thus, our approach does not entail directly applying LLMs to detect the presence of toxic threads. Instead, we utilize them to analyze the discussion texts and provide insights into potential negative sentiments. To obtain better responses from LLM, we design a prompt.

Prompt Design. Prompt has to include both the instructions of the toxicity detection task and the input thread. We design the prompt (see Fig. 12) following the handbook from OpenAI [41]. Specially, we adopt two prompts in total. The first prompt, as shown in Prompt 1 of Fig. 12, is used to capture the potential negative sentiments contained in the thread. The second prompt, depicted in Prompt 2 of Fig. 12, is used to extract the number of SE-related words in the threads, serving

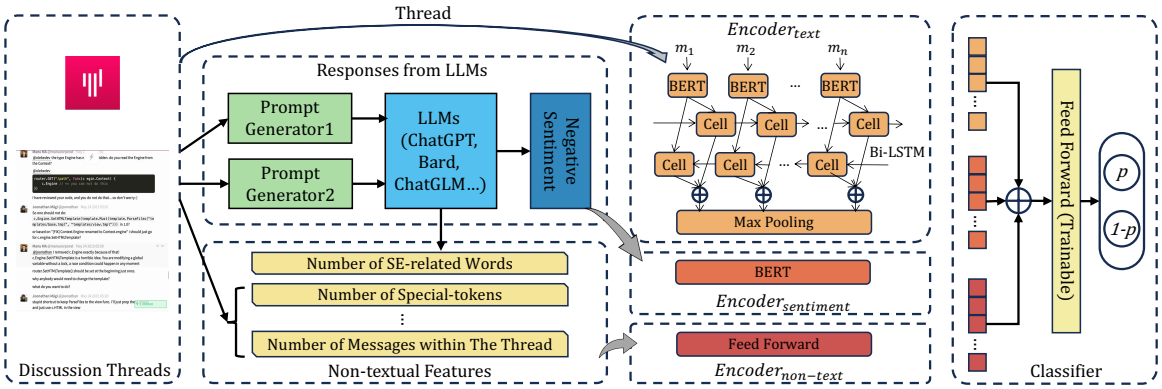

Fig. 11. Overview of automated toxicity detection.

Prompt 1:
System template: Can you tell me what type of negative sentiment the following dialogue from developer chatroom related to {project} has? Please answer with just one word.
Human template: {thread}

Prompt 2:
System template: Can you tell me how many words related to Software Engineering domain in the following dialogue? Please answer with just one Arabic numerals.
Human template: {thread}

Thread example:
D1: I have also applied for a CS degree . fingers crossed EMOJITAG
D2: *rejected* EMOJITAG
D1: ?. MENTIONTAG go to your cave troll
D2: lol

Response from ChatGPT:
Prompt 1: sarcastic
Prompt 2: 1

merging consecutive messages
Replacement of special tokens

Fig. 12. Prompt template for the LLM.

as one of the non-textual features (see Section IV-C). The placeholders {project} and {thread} can be replaced with specific project and discussion thread, respectively. The input format of the thread follows the example illustrated in Fig. 12.

As the outputs obtained from LLM are in text format, we utilize BERT to transfer the responses of the negative sentiment query into embeddings (which will be further combined with embeddings of other types of features), annotated as $Encoder_{sentiment}$.

C. Non-textual Features

We mainly follow the non-textual features proposed by Pan *et al.* [14], which include four categories: **length features, structural features, participant features, and special-token features**. For each category of non-textual features, three perspectives are considered: **the whole thread, the first message, and the activities of the asker** (please refer to [14] for more details). However, we have made specific adaptations to align better with our task. Specifically, we introduce two newly designed special-token features. One of these new features quantifies the number of approvals, while the other assesses the usage of emojis. We believe these two special tokens offer valuable insights for our task. Furthermore, the number of SE-related words obtained from LLM in Section IV-B is also included as a non-textual feature. This non-textual feature primarily serves toxicity detection targeting the *Spamming* category, which helps to determine if the discussion is closely related to the technical topics. For encoding non-textual features, we have adopted a concise $Encoder_{non-text}$ (i.e., a fully connected feed-forward network with one hidden layer) due to the computability of numerical features themselves.

D. Incorporating All Features

Once we have obtained three types of features, we combine them all together for the final toxicity detection. Denote the embedding from the text encoder as e_{text}, the embedding representing negative sentiment as $e_{sentiment}$, and the embedding of non-textual features as $e_{non-text}$. These three embedding features are first concatenated to obtain a comprehensive representation of the thread $e = [e_{text} \bigoplus e_{sentiment} \bigoplus e_{non-text}]$ and further passed through a nonlinear projection header for classification. Finally, a softmax layer is applied to obtain the binary prediction results of the model (i.e., Toxicity or Non-toxicity). We employ cross-entropy as the loss function and adjust the class weights (i.e., with a ratio of pos:neg=3:1) to address the data imbalance issue.

V. Experiments and Results

In this section, we present detailed experimental settings, introduce the research questions, and provide answers to each question based on the experimental results.

A. Experiment Setting

Testing Scenarios. We use two scenarios as follows:

- 5-fold cross-validation. The dataset containing all samples is divided into five equally sized subsets, maintaining the original class distribution, wherein each of the five rounds, four of them are used as the training set, and the remaining one is used as the testing set.
- Cross-chatroom-validation. For each chatroom in the dataset, we utilize all samples from it as the testing set, while using all samples from the remaining chatrooms as the training set, in order to evaluate the model's generalization performance when faced with previously unseen chatrooms.

Implementation Details. We select ChatGPT with *gpt-3.5-turbo* from the available large language models (LLMs) due to its proven superior performance. Using the chat completion API [41], we provide prompts and receive responses. To ensure stability, we configure the parameters as follows: $temperature = 0$ and $n = 1$. This prioritizes accurate replies over creative ones, generating a single result each time. We

TABLE V
THE DETAILED PERFORMANCE OF OUR APPROACH AND BASELINES ON TOXICITY UNDER CROSS-VALIDATION SCENARIO

Detector	Toxicity			Non-toxicity			Overall
Name	*P*	*R*	*F1*	*P*	*R*	*F1*	*Acc*
BERT4SentiSE	0.030	**0.761**	0.057	**0.991**	0.502	0.667	0.507
Regex	0.218	0.565	0.314	**0.991**	0.959	0.975	0.952
STRUDEL	0.164	0.275	0.203	0.985	0.972	0.979	0.959
MAX Classifier	0.242	0.424	0.306	0.988	0.973	0.981	0.962
PAPI	0.328	0.374	0.346	0.988	0.985	0.986	0.973
Our work	**0.618**	0.503	**0.546**	0.990	**0.993**	**0.992**	**0.983**

TABLE VI
THE DETAILED PERFORMANCE OF OUR APPROACH AND BASELINES ON TOXICITY UNDER CROSS-CHATROOM SCENARIO

Detector	Angular			Node.js			Python		
Name	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
BERT4SentiSE	0.030	**0.952**	0.058	0.030	**0.750**	0.057	0.030	**0.700**	0.057
Regex	0.213	0.619	0.317	0.190	0.550	0.282	0.231	0.550	0.325
STRUDEL	0.119	0.238	0.159	0.147	0.250	0.185	0.194	0.300	0.235
MAX Classifier	0.176	0.429	0.250	0.312	0.500	0.385	0.250	0.400	0.308
PAPI	0.184	0.333	0.237	0.370	0.500	0.426	0.412	0.350	0.378
Our work	**0.476**	0.476	**0.476**	**0.647**	0.550	**0.595**	**0.419**	0.433	**0.426**

use pre-trained BERT [20] base model from HuggingFace's Transformers library [42] and follow the suggestion by Devlin *et al.* [20] to set the learning rate (lr) as 2e-5. To mitigate data imbalance, we apply the Online Negative Sampling (ONS) strategy with a sampling ratio of *neg:pos=3:1*, which has been regarded as optimal for training network [35], [36]. During the training process, we set dropout to 0.1 to avoid over-fitting. We use AdamW as the optimizer and have lr of 1e-4 for modules besides BERT. The learning rate linearly warm-ups over the first 150 steps and then decays in the remaining steps. Finally, An early stopping with patience of 30 epochs is set.

Baseline. Since thread-level toxicity detection methods haven't been explored before, we use sentence-level methods as baselines for comparison. These baselines include Regular Expression, Perspective API, MAX Toxic Comment Classifier, STRUDEL, and BERT4SentiSE (as discussed in Section II-A). To ensure a fair comparison with thread-level approaches and preserve the characteristics of sentence-level methods, we refrain from modifying them to operate at the thread level. Instead, we maintain the strategy of labeling a thread as toxic if any sentence within it is marked as toxic.

Metrics. We evaluate the performance of all approaches using commonly used classification evaluation metrics, including precision, recall, and F1-Score. While toxic samples are our primary detection target, metrics related to non-toxic samples can also reflect the performance of the approaches to some extent. Therefore, we evaluate the approaches in both categories.

B. Research Questions & Experiment Results

RQ1: Is our proposed approach effective in accurately detecting toxicity at the thread level?

Motivation. Existing toxicity detection methods mainly operate at the sentence level (Section II-A). As the first to focus on thread-level detection, we compare our approach with baselines to assess the accuracy of coarser granularity detection. Additionally, the generalization capability of models in unknown chatrooms presents a challenge, so we evaluate performance in cross-chatroom scenarios.

Results. Tab. V presents the results of the 5-fold cross-validation scenario with the best result for each metric highlighted in bold. The BERT4SentiSE method achieves an extremely high recall value because it detects a broader range of negative sentiment than toxicity, albeit at very low precision. The second highest recall value is attained by the Regex method, which, however, lacks an understanding of the overall sentence semantics, leading to a lower precision value. The performance of the SE-specific method STRUDEL is also unsatisfactory. Conversely, the general toxicity detection method PAPI performs best in baselines, achieving an F1-score (F1) of 0.346. However, our approach demonstrates outstanding performance compared to all the baselines. Compared to the best-performing baseline, our method shows an improvement of 88.4% in precision, 34.5% in recall, and 57.8% in F1-score (F1). The surge in precision showcases our method's improved ability to differentiate toxic and non-toxic instances, minimizing false positives. The significant increase in recall indicates our method's broader coverage of toxicity compared to existing approaches.

In the cross-chatroom scenario, we select Angular, Node.js, and Python chatrooms as target chatrooms respectively. Due to the limit of the dataset, when using the Python chatroom as the test set, the number of samples in the test set is more than in the training set, leading to a lack of training samples. However, experimental results demonstrate strong performance across all three chatrooms, consistently outperforming the baselines. Notably, the detection efficacy for the Node.js chatroom surpasses even that of the cross-validation scenario. While the performance in the Python chatroom is comparatively lower, there is a noticeable improvement over the baseline.

RQ2: Do the handcrafted non-textual features and negative sentiment analysis results from LLM enhance the performance of the thread-level toxicity detection method?

Motivation. In toxicity detection, we include non-textual and negative sentiment features from LLM. To assess their effectiveness, we conduct ablation experiments, selectively removing either non-textual or negative sentiment features. Other experimental settings align with those described in Section V-A.

Results. Firstly, we conduct an ablation study on non-textual features under the cross-validation scenario, and the results are shown in the *Ours-no-nontext* row of Tab. VII. Removing non-textual features results in a significant decrease in all *Toxicity* metrics, with precision, recall, and F1-score (F1) dropping by 17.3%, 21.5%, and 22.9%, respectively. This decline is evident even in the *Non-toxicity* category, which initially has high metric values. Next, we conduct ablation experiments on non-textual features under the cross-chatroom scenario and the results are presented in the *Ours-no-nontext* row of Tab. VIII. The results vary from different target chatrooms. When the target chatroom is Angular, the absence of non-textual features leads to a significant decline of over 30% in all performance metrics. When the target chatroom is Node.js, precision decreases while recall increases. Even considering the slight decrease in the F1-score (F1), the absence of non-textual features still results in an overall reduction in detection performance. When the target chatroom is Python, the metrics

TABLE VII
THE DETAILED PERFORMANCE OF OUR APPROACH AND ABLATION STUDIES ON TOXICITY UNDER CROSS-VALIDATION SCENARIO

Detector Name	Toxicity			Non-toxicity			Overall
	P	*R*	*F1*	*P*	*R*	*F1*	*Acc*
Our work	**0.618**	**0.503**	**0.546**	**0.990**	**0.993**	**0.992**	**0.983**
Ours-no-nontext	0.511	0.395	0.421	0.988	0.991	0.990	0.979
ours-no-sentiment	0.534	0.359	0.422	0.987	**0.993**	0.990	0.981

TABLE VIII
THE DETAILED PERFORMANCE OF OUR APPROACH AND ABLATION STUDIES ON TOXICITY UNDER CROSS-CHATROOM SCENARIO

Detector Name	Angular			Node.js			Python		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
Our work	0.476	0.476	0.476	**0.647**	0.550	**0.595**	**0.419**	0.433	0.426
Ours-no-nontext	0.333	0.286	0.308	0.520	**0.650**	0.578	0.409	**0.450**	**0.429**
ours-no-sentiment	**0.500**	**0.571**	**0.533**	0.458	0.550	0.500	0.104	0.217	0.141

with or without non-textual features are very close, indicating minimal impact on the detection results.

When dealing with chatrooms where relevant information has been learned, the non-textual features show a more pronounced impact. When facing entirely new chatrooms, the impact is slightly less pronounced. Nevertheless, they still manage to maintain a stable level of detection effectiveness. Therefore, the inclusion of non-textual features proves to be effective.

Secondly, we conduct an ablation study on the negative sentiment features from LLM under the cross-validation scenario, and the results are presented in the *ours-no-sentiment* row of Tab. VII. The experimental results closely mirror those of the ablation study on non-textual features. However, in comparison, there is a more substantial decrease in recall, indicating that these features are more instrumental in identifying toxicity. However, under the cross-chatroom scenario as shown in Tab. VIII, the ablation results of negative sentiment features from LLM show a stark contrast to non-textual features. It leads to a decrease in detection performance in the Angular chatroom but triples the results in the Python chatroom.

VI. RELATED WORK

Toxicity detection is crucial for maintaining effective and harmonious online discussions. Existing studies primarily focus on platforms like Twitter [43], Facebook [44], and Wikipedia [11], employing rule-based, traditional machine learning, and deep learning approaches. Early methods by Burnap et al. [28] used Bayesian Logistic Regression, Random Forest Decision Tree, and Support Vector Machine on Twitter. Del Vigna et al. [44] explored hate speech on Facebook using neural networks and LSTM. Perspective API demonstrated the best detection performance by leveraging pre-trained models [11]. However, these methods struggle with understanding professional expressions in developer communication, leading to reduced accuracy in capturing intentions.

Toxicity detection research in the Software Engineering (SE) domain is relatively limited, with a focus on the GitHub platform. Raman et al. [9] introduced STRUDEL, the first SE-specific toxicity detector, using domain adaptation to enhance classification accuracy. Sarker et al. [45] demonstrated improved performance of existing detectors after retraining on SE domain datasets. Ferreira et al. [4], [8], [12] conducted in-depth analysis on GitHub Issues/Comments and Code Reviews, proposing SE-specific toxicity detection methods based on BERT, which outperformed traditional methods and validated the effectiveness of pre-trained models in this domain.

Research on real-time Chat platforms like Gitter is limited, with a focus on sentence-level analysis using general criteria [3], [45]. Gitter chatrooms foster free expression, often involving multi-turn Q&A interactions, providing complete context and intentions for toxicity detection. Adopting the thread as the research granularity, we develop a toxicity taxonomy tailored to conversational patterns in developer chatrooms. Our approach incorporates thread structure information features into toxicity detection, validated on our self-constructed dataset, demonstrating its effectiveness.

VII. THREATS TO VALIDITY

Threats to Internal Validity. Due to the high cost of manual annotation, building a large-scale labeled dataset is challenging. To ensure dataset representativeness, we select diverse chatrooms with different topics and scales. During annotation, we manually correct potential inaccuracies in thread disentanglement and employ two annotators who achieve a Cohen's Kappa coefficient of 0.75 for high agreement. To mitigate biases from a small dataset, we use 5-fold cross-validation in our experiments. To improve detection results, our method relies on negative sentiment from LLMs. We choose ChatGPT, known for superior sentiment analysis performance [46]. To enhance response stability, we follow official guidance [41], experiment with prompt formats, and identify the most effective template (Fig. 12).

Threats to External Validity. We address two threats to the generalizability of our approach. The first is to predict toxicity in unknown developer chatrooms. We conduct experiments and prove the strong performance of our approach, outperforming the baselines by a significant margin. The other is that we only involve Gitter as the data source. However, Gitter represents a significant platform among developer online chatroom platforms with its highly real-time and interactive characteristics typical of such communication modes. So the method proposed based on Gitter demonstrates applicability across various developer online chatroom platforms.

VIII. CONCLUSION

In this paper, we thoroughly investigate toxic discussions in developer chatrooms, establishing a thread-level toxicity taxonomy and creating a dataset of 5,158 samples. We analyze misclassification triggers in sentence-level methods and propose an automated toxicity detection approach combining textual, non-textual, and negative sentiment features from LLM. Our method outperforms baselines with precision, recall, and F1-Score values of 0.618, 0.503, and 0.546 respectively, performing well in cross-validation. Ablation experiments confirm the effectiveness of non-textual and negative sentiment features, enhancing robustness and performance. Future work includes collecting more threads from different chatrooms for generalizability and classifying toxicities into specific categories for better user insights.

IX. ACKNOWLEDGMENTS

This research/project is supported by the Fundamental Research Funds for the Central Universities (No. 226-2022-00064), and the National Science Foundation of China (No.62372398, No.72342025, and U20A20173)

REFERENCES

[1] "Gitter," Online. [Online]. Available: https://gitter.im

[2] C. Miller, S. Cohen, D. Klug, B. Vasilescu, and C. KaUstner, "" did you miss my comment or what?" understanding toxicity in open source discussions," in *Proc. of ICSE*, 2022, pp. 710–722.

[3] J. Cheriyan, B. T. R. Savarimuthu, and S. Cranefield, "Towards offensive language detection and reduction in four software engineering communities," in *Evaluation and Assessment in Software Engineering*, 2021, pp. 254–259.

[4] I. Ferreira, J. Cheng, and B. Adams, "The" shut the f** k up" phenomenon: Characterizing incivility in open source code review discussions," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–35, 2021.

[5] E. Parra, M. Alahmadi, A. Ellis, and S. Haiduc, "A comparative study and analysis of developer communications on slack and gitter," *Empirical Software Engineering*, vol. 27, no. 2, p. 40, 2022.

[6] E. Parra, A. Ellis, and S. Haiduc, "Gittercom: A dataset of open source developer communications in gitter," in *Proc. of MSR*, 2020, pp. 563–567.

[7] L. Shi, X. Chen, Y. Yang, H. Jiang, Z. Jiang, N. Niu, and Q. Wang, "A first look at developers' live chat on gitter," in *Proc. of ESEC/FSE*, 2021, pp. 391–403.

[8] I. Ferreira, B. Adams, and J. Cheng, "How heated is it? understanding github locked issues," in *Proc. of MSR*, 2022, pp. 309–320.

[9] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *Proc. of ICSE: New Ideas and Emerging Results*, 2020, pp. 57–60.

[10] N. P. Simon Plovyt, Patrick Titzler, "Ibm developer model asset exchange: Toxic comment classifier," Online, 2020. [Online]. Available: https://github.com/IBM/MAX-Toxic-Comment-Classifier

[11] G. Jigsaw, "Perspective api," Online, 2017. [Online]. Available: https://www.perspectiveapi.com

[12] I. Ferreira, A. Rafiq, and J. Cheng, "Incivility detection in open source code review and issue discussions," *arXiv preprint arXiv:2206.13429*, 2022.

[13] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "Exploratory study of slack q&a chats as a mining source for software engineering tools," in *Proc. of MSR*. IEEE, 2019, pp. 490–501.

[14] S. Pan, L. Bao, X. Ren, X. Xia, D. Lo, and S. Li, "Automating developer chat mining," in *Proc. of ASE*. IEEE, 2021, pp. 854–866.

[15] O. Ehsan, S. Hassan, M. E. Mezouar, and Y. Zou, "An empirical study of developer discussions in the gitter platform," *TOSEM*, vol. 30, no. 1, pp. 1–39, 2020.

[16] "Replication package link of this paper." [Online]. Available: https://figshare.com/s/7c4c697e4b523aed437e

[17] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Proceedings of the international AAAI conference on web and social media*, vol. 8, no. 1, 2014, pp. 216–225.

[18] K. Trivedi, "Multi-label text classification using bert – the mighty transformer," 2019. [Online]. Available: https://medium.com/huggingface/multi-label-text-classification-using-bert-the-mighty-transformer-69714fa3fb3d

[19] J. AI, "Toxic comment classification challenge," Online, 2018.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[21] E. Biswas, M. E. Karabulut, L. Pollock, and K. Vijay-Shanker, "Achieving reliable sentiment analysis in the software engineering domain using bert," in *Proc. of ICSME*. IEEE, 2020, pp. 162–173.

[22] L. Shi, M. Xing, M. Li, Y. Wang, S. Li, and Q. Wang, "Detection of hidden feature requests from massive chat messages via deep siamese network," in *Proc. of ICSE*, 2020, pp. 641–653.

[23] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: a customized sentiment analysis tool for code review interactions," in *Proc. of ASE*. IEEE, 2017, pp. 106–111.

[24] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," in *Proc. of ICSE*, 2018, pp. 128–128.

[25] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proc. of ICSE*, 2018, pp. 94–104.

[26] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams, "The emotional side of software developers in jira," in *Proc. of MSR*, 2016, pp. 480–483.

[27] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 522–559, 2019.

[28] P. Burnap and M. L. Williams, "Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making," *Policy & internet*, vol. 7, no. 2, pp. 223–242, 2015.

[29] "Categories of chatrooms on gitter," Online. [Online]. Available: https://gitter.im/home/explore

[30] D. Spencer, *Card Sorting: Designing Usable Categories*. Rosenfeld Media, 2009.

[31] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[32] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-specific named entity recognition in software engineering social content," in *Proc. of SANER*, vol. 1. IEEE, 2016, pp. 90–101.

[33] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th conference on program comprehension*, 2018, pp. 200–210.

[34] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[35] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," in *Proc. of ICSE*. IEEE, 2021, pp. 324–335.

[36] S. Pan, J. Zhou, F. R. Cogo, X. Xia, L. Bao, X. Hu, S. Li, and A. E. Hassan, "Automated unearthing of dangerous issue reports," in *Proc. of ESEC/FSE*, 2022, pp. 834–846.

[37] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[38] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[39] J. Kocoń, I. Cichecki, O. Kaszyca, M. Kochanek, D. Szydło, J. Baran, J. Bielaniewicz, M. Gruza, A. Janz, K. Kanclerz *et al.*, "Chatgpt: Jack of all trades, master of none," *Information Fusion*, p. 101861, 2023.

[40] M. V. Reiss, "Testing the reliability of chatgpt for text annotation and classification: A cautionary remark," *arXiv preprint arXiv:2304.11085*, 2023.

[41] "Guides of chatgpt," Online. [Online]. Available: https://platform.openai.com/docs/guides/gpt/chat-completions-api

[42] "Bert on huggingface," Online. [Online]. Available: https://huggingface.co/bert-base-uncased

[43] E. W. Pamungkas, V. Basile, and V. Patti, "Do you really want to hurt me? predicting abusive swearing in social media," in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 2020, pp. 6237–6246.

[44] F. Del Vigna, A. Cimino, F. Dell'Orletta, M. Petrocchi, and M. Tesconi, "Hate me, hate me not: Hate speech detection on facebook," in *Proceedings of the first Italian conference on cybersecurity (ITASEC17)*, 2017, pp. 86–95.

[45] J. Sarker, A. K. Turzo, and A. Bosu, "A benchmark study of the contemporary toxicity detectors on software engineering interactions," in *Proc. of APSEC*. IEEE, 2020, pp. 218–227.

[46] M. Belal, J. She, and S. Wong, "Leveraging chatgpt as text annotation tool for sentiment analysis," 2023.