ToxiSpanSE: An Explainable Toxicity Detection in Code Review Comments

Jaydeb Sarker[♣], Sayma Sultana[♣], Steven R. Wilson[♢], Amiangshu Bosu[♣]
[♣]*Wayne State University, Detroit, Michigan, USA*
[♢]*Oakland University, Rochester, Michigan, USA*
jaydebsarker@wayne.edu, sayma@wayne.edu, stevenwilson@oakland.edu, amiangshu.bosu@wayne.edu

***Abstract*—Background: The existence of toxic conversations in open-source platforms can degrade relationships among software developers and may negatively impact software product quality. To help mitigate this, some initial work has been done to detect toxic comments in the Software Engineering (SE) domain.**

Aims: Since automatically classifying an entire text as toxic or non-toxic does not help human moderators to understand the specific reason(s) for toxicity, we worked to develop an explainable toxicity detector for the SE domain.

Method: Our explainable toxicity detector can detect specific spans of toxic content from SE texts, which can help human moderators by automatically highlighting those spans. This toxic span detection model, *ToxiSpanSE*, is trained with the 19,651 code review (CR) comments with labeled toxic spans. Our annotators labeled the toxic spans within 3,757 toxic CR samples. We explored several types of models, including one lexicon-based approach and five different transformer-based encoders.

Results: After an extensive evaluation of all models, we found that our fine-tuned RoBERTa model achieved the best score with 0.88 $F1$, 0.87 precision, and 0.93 recall for toxic class tokens, providing an explainable toxicity classifier for the SE domain.

Conclusion: Since *ToxiSpanSE* is the first tool to detect toxic spans in the SE domain, this tool will pave a path to combat toxicity in the SE community.

***Index Terms*—toxicity, span detection, software engineering, natural language processing, explainability**

I. Introduction

Toxicity, which is a large umbrella term comprising various antisocial behaviors such as offensive language, cyberbullying, hate speech, and sexually explicit content [1], is pervasive among various online platforms [2], [3]. As most of the Free and Open Source Software (FOSS) communities operate online, they are not immune from such toxic interactions [2], [4], [5]. As software development requires close collaboration and rapport among participants, toxicity can have severe repercussions for a FOSS community, which include decreased productivity, wastage of valuable time [4], negative feelings among the participants [6], barriers to newcomers' onboarding [7], [8], hostile environments towards minorities [9]. As proactive identification and mitigation of toxic interactions among FOSS developers are crucial, automated approaches can help FOSS moderators.

Prior studies [5], [10] found that off-the-shelf toxicity detectors do not perform well in the SE texts because some words ('die', 'kill', 'dead') in the SE context have a different meaning. Due to the unreliability of off-the-shelf natural language processing (NLP) tools on Software Engineering (SE) datasets [10], [11], recent works have proposed customized toxicity detectors trained on SE communications [4], [5]. While these tools boost reliable performances on SE datasets, we have identified a shortcoming of these two solutions. First, existing tools classify an entire paragraph on a binary scale, including hundreds of sentences. Even if only one of those sentences is toxic, it classifies the whole paragraph as toxic. A binary, paragraph-level classification of toxic texts may help the FOSS community to decide to remove a particular paragraph or establish a code of conduct for toxic comments. However, it becomes time-consuming for a moderator to identify the offending excerpt(s) from a large paragraph. Second, due to the lack of cultural differences, a moderator may fail to identify the offending sentences from a paragraph classified as toxic by these tools. Being motivated by recent advances in explainable machine learning (ML) models, this study aims to create a new SE domain-specific toxicity detector that overcomes this particular shortcoming. We aim to *develop an explainable toxicity detector for the Software Engineering domain, which can precisely identify toxic excerpts from a text to assist FOSS moderators.* 'Explainable' in the context of this study indicates the ability of the classifier to pinpoint the words/phrases responsible for a text's toxic classification [12].

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

A *toxic span* is defined as the fragment of a sentence or text that potentially causes the meaning of the text to be toxic [13]. A toxic span may contain a single word or a sequence of words. For example, "Yuck, this code is a crap" where the toxic spans are highlighted with red color. The SemEval-2021's Task 5 organizers provided 10K toxic posts from the Civil Comments dataset [14] with labeled

978-1-6654-5223-6/23/$31.00 ©2023 IEEE

span character offsets. An ensemble solution using BERT [15] achieved the best performance among the teams participating in this challenge. As prior research shows the necessity of SE domain-specific customization for NLP tools [10], [11], these toxic span detectors may not perform well on SE texts. Hence, we aim to build a customized solution.

On this goal, we select SE domain-specific toxicity dataset from Sarker *et* al. [5], which consists of a total of 19,651 Code Review (CR) comments with 3,757 ($\sim 19\%$) toxic samples. We manually label this dataset using two independent raters to develop ground-truth annotations for the toxic spans within the toxic samples. We measured inter-annotator agreement using Krippendorff's α [16], which was 0.81 (almost perfect agreement). We first developed a lexicon-based classifier using this dataset to establish a baseline model. We trained and evaluated five sequence-to-sequence transformer models. During our 10-fold cross-validation-based evaluations, we found a model based on a fine-tuned RoBERTa [17] achieving the best an $F1-$score of 0.88. Primary contributions of this work include:

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

Paper organization: The remainder of this paper is organized as follows. Section II provides the related works on toxicity and toxic span detection. We discuss the research methodology in SectionIII. Section IV presents the results. Section V discusses the lesson learned. Section VI addresses the threats to validity of this work. Finally, Section VII concludes the paper.

II. Background

A. Toxicity Phenomena

The term 'toxicity' represents the negative or antisocial interactions in online conversations [18]. A report from [18] showed that 47% Americans experienced harassment and abuse during online communication, and toxicity deters users from online engagement. Toxicity is a subjective phenomenon often subject to the opinions of beholders [19]. A broader view of toxicity is that of an umbrella of various antisocial behaviors such as hate speech, cyberbullying, trolling, and flaming [2]. The Conversational AI team from Google defined toxicity as "comments that are rude, disrespectful or otherwise likely to make someone leave a discussion" [20].

Toxicity in the SE domain is not uncommon. Recent studies from the SE domain analyzed toxicity and other antisocial behaviors. Sarker *et* al. defined a code review comment as toxic if it includes any antisocial behaviors such as offensive name-calling, insults, threats, personal attacks, flirtations, sexual reference, and profanities [10]. Miller *et* al. adjusted the meaning of toxicity from Conversational AI during analyzing GitHub issue discussions and mentioned that the toxicity umbrella covers trolling, flaming, hate speech, harassment, arrogance, entitlement, and cyberbullying [2]. Ferreira *et* al. defined the unnecessary disrespectful term toward discussion as incivility during analyzing the Linux Kernel Mailing list [21]. A similar term of toxicity is 'destructive criticism' during code review [9] that represents the negative feedback, including threats, poor task performance, or flaws of the individuals. In 2020, Egelman *et* al. defined 'pushback' in code review as a reviewer blocking a change request due to unnecessary interpersonal conflict [6]. In this work, we adapted the definition of toxicity from the study of Sarker *et* al. [10] because we used their comment-level toxicity labels, which were annotated using this definition, and they provided a rule book for marking a text as toxic or non-toxic which assisted us with our annotation process.

B. Toxicity in SE

FOSS communities have reported toxic contents in developers' communications in blog posts [22], [23], podcasts [24], and talks [25]. Large open-source foundations face an increase in toxicity in their groups. For example, the Linux Community experiences toxicity [23], and the founder apologized for using toxicity in Linux Kernel Mailing lists [26]. By conducting a survey, the Perl Foundation found that several members stepped down due to receiving abusive messages [27].

Researchers from the SE community also conducted empirical studies to understand toxicity in FOSS domain [28], [29]. Another line of research focused on analyzing the toxic behavior of open source developers' interactions such as GitHub issue discussions [2], [4], code review comments [5], [9], [10], [30], and Gitter messages [10]. Toxic communication impacts developers' mental health like a 'poison' [28] and can cause burnout [4]. To combat toxicity in open source, Raman *et* al. developed a toxicity classifier trained with a small-scale GitHub issue discussion dataset [4]. However, several studies showed that their tool [4] performed poorly on the large-scale SE texts [2], [10], [30], [31]. The number of annotated toxic texts in open-source communication is relatively small, which makes it challenging to build a reliable toxicity classifier. However, the existence of toxicity has severe repercussions among the open-source software developers such as newcomers onboarding [4], disproportionate impacts on underrepresented groups [4], [9], [32], and pushbacks [6]. To detect the toxicity from code review comments, Sarker *et* al. developed a machine learning-based toxicity classifier (referred to as 'ToxiCR') trained with 19,651 labeled code review comments from Gerrit projects [5]. In 2022, Qiu *et* al. developed a classifier for identifying interpersonal conflicts during code reviews [30]. Cheriyan *et* al. developed a classifier to detect swearing and profanity [33] for four different SE platforms. Our work differs from the prior works by focusing

on finer-grained identification of which phrases make a text toxic in the SE context.

Recent studies also focused on understanding antisocial behaviors in open-source communities. Miller *et* al. conducted a study for a better understanding the toxicity with a qualitative analysis of 100 toxic issue comments on GitHub [2]. They manually investigate the nature, context, participants, and after impacts of toxicity on GitHub. After analyzing the 1,545 emails from Linux Kernel Mailing lists, Ferreira *et* al. found that the common forms of incivility are frustration, name-calling, and impatience [21]. Gunawardena *et* al. defined 'destructive criticism' as another antisocial behavior, which includes negative feedback which is nonspecific and is delivered in a harsh or sarcastic tone. Their survey of 93 developers suggests destructive criticism as a barrier to promoting diversity and inclusion [9].

C. Toxic Span Detection

Although the detection of toxicity [34]–[36], hate speech [37]–[39], and offensive language [40], [41] are common in online platforms, the idea of span detection of toxicity has only more recently gained attention with the SemEval-2021 toxic span detection task [42]. Toxic spans represent a part of the text that is responsible for the toxicity of the posts [13]. This direction is inspired by prior NLP studies on aspect-based sentiment analysis [43], [44], which aims to detect the sentiment of a text and find the specific region of a text that expresses the sentiment using attention-based deep neural network models [43]. While earlier studies focused on the attention mechanism championed by Vaswani *et* al. [45], Sen *et* al. found that machine attention does not reliably overlap with human attention maps [46]. To improve explainability using attention-based mechanisms, recent works have proposed transformer-based sequence-to-sequence models [47], [48].

In the SemEval-2021 task, Pavlopoulos *et al.* provided a labeled dataset [42] of toxic spans with 10,000 samples [13] curated from the Civil Comments dataset. The raters marked the span that corresponds to the toxicity of a text. The task is a binary classification because it contains toxic and non-toxic tokens. Moreover, they fixed the ground truth of the dataset if the majority of the raters labeled the span as toxic. Ninety-one teams made submissions with different methods in the SemEval-2021 competition to detect toxic spans [42]. One of the teams proposed a BERT-based ensemble method toxic span detection approach where they achieved 70.83% $F1$ score and secured first place in SemEval-2021 task [49]. A RoBERTa-based method performed only slightly worse, with a 70.77% $F1$ score, and other approaches based on fine-tuning of pre-trained transformer models ([50], [51]) also performed well for that toxic span detection tasks. Since several studies worked on toxic span detection for online civil comments, Pavlopoulos *et* al. annotated a new dataset for toxic to civil transfer [52]. Although several studies have proposed toxic span detectors for online comments, no such tool exists for the SE domain. Since NLP tools may not work reliably on a cross-domain dataset [11], the development and evaluation of a SE domain-specific toxic span detector are essential. Such a tool will not only enable a finer-grained analysis of toxicity but also enable proactive notification to authors.

III. Research Method

After selecting a dataset from a prior work [5], two of the authors independently annotated toxic spans in each text. Using this annotated dataset, we train and evaluate sequence-to-sequence transformer models that output the probability of each word belonging to a toxic span in the current text context. Finally, we use postprocessing steps to identify toxic spans from the output probabilities based on empirically determined thresholds. The following subsections detail our research methodology.

A. Dataset

1) Dataset Selection: The number of datasets for toxicity detection in Software Engineering communication is small [4], [10]. We explored previous studies on toxicity and antisocial behaviors in open-source interactions and found four studies that provided manually labeled datasets for toxicity [4], [5], [10] and incivility [21] detection. Raman *et* al. labeled only 611 texts from GitHub issue discussions as toxic or non-toxic [4]. In 2020, Sarker *et* al. provided a dataset of 6,533 CR comments and 4,140 Gitter messages labeled as toxic or non-toxic [10]. They also provided a rubric to identify a text as toxic or non-toxic. In a subsequent study of building a toxicity detection tool, they annotated 19,651 CR comments with binary, comment-level toxicity scores [5]. Moreover, Ferreira *et* al. provided an annotated 1,545 emails from the Linux Kernel Mailing List where they labeled each message as civil or uncivil [21]. Given that Sarker *et* al.'s dataset [5] is the largest one in the Software Engineering domain for toxicity detection, we select their dataset for our study. Moreover, their detailed rubric also guides our annotators on how to label toxic spans.

2) Dataset Annotation: We got each of our toxic samples manually annotated by two independent annotators. To diversify the annotators, we chose one woman and one man for the annotation task. As manual labeling toxic text is a subjective task, we sought to reduce subjectivity bias during manual annotation by asking our annotators to carefully read and follow the rubric for toxicity developed by [5]. Although Sarker *et* al.'s dataset [5] includes 19,651 CR comments, only 3,757 are labeled as toxic. Therefore, our annotators only labeled the 3,757 toxic ones, assuming that the non-toxic samples do not include any toxic spans (empty span offsets).

For annotation, we use the Label Studio platform [53]. Figure 1 shows an example of our annotation interface. We exported the labeled data from Label Studio, which returns the code review text and corresponding character span offsets of the toxicity annotations for each sample. Table I shows two example annotations. The first example shows a toxic sample where the word 'sucked' makes text toxic, and this span occurs in character offsets 10-15. The third example is non-toxic and

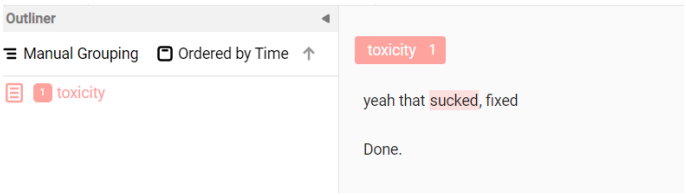

Fig. 1. Manual Labeling using Label Studio, toxic span is highlighted

TABLE I
RAW DATASET WITH CHARACTER SPANS. RED MARKED REPRESENTS SELECTED TOXIC WORDS

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

therefore has no span selected, resulting in an empty list ([]) of character offsets.

3) Inter-annotator Agreement: We wrote a Python script to compare the spans produced by the two annotators. Unsurprisingly, we have found conflicts between the labeling samples. Previous studies have suggested several *chance-corrected* agreement measures to compute the inter-annotator agreement (IAA) [54]. *Chance-corrected* measures such as Cohen's κ [55], Fleiss' κ [56], and Scott's π [57] distinguish the observed disagreements (D_o) from expected disagreements (D_e). Therefore, these IAA measures are unsuitable for sequential tagging with potential partial overlaps [58].

Hence, similar to prior studies developing sequence tagging datasets [54], [58]–[60], we chose Krippendorff's α [16] as the IAA measure. Krippendorff's α is more robust as it can handle multiple annotators and missing values and considers partial agreement/disagreements among the labelers. Krippendorff's α allows the distance-based formulation and it is designed for context-specific tasks. The formula of Krippendorff's is: $\alpha = 1 - \frac{\hat{D}_o}{\hat{D}_e}$, for a given distance function of $D(a, b)$ where $\hat{D}_o$ represents the observed average distance and $\hat{D}_e$ is expected average distance [54]. Since Krippendorff's α calculates several distance functions such as nominal, interval, and ordinal [16], we chose the nominal distance function for our measurement. To calculate Krippendorff's α score, we wrote our script using the existing implementation [61].

Our labeled dataset has 3757 toxic code review samples labeled by two raters for toxic spans. For calculating Krippendorff's α with nominal distance, created two arrays of labels. We split each sample (s) to a set of tokens $s = t_0, t_1, ..., t_j$ where t_j is a token inside the sample s. As our primary dataset contains the character level span offsets, we preprocessed it for token-level offsets. Table II shows an example of defining the token array for a sample. There is a total of 15 tokens after excluding the comma (,) from the input text. Therefore, we generate an array of 15 elements (same as the length of tokens) in which each position corresponds to a token from the CR text. We have a same-length array for Rater1 and Rater2 where we set 1 if the token is inside the span selection, otherwise 0. Following this process, we generated 3,757 arrays for all toxic samples of Rater1 and Rater2. For computing agreement, we merge all the token-level annotations for each rater into a single array where each array contains a total of 84,951 ratings. We calculated Krippendorff's α using the nominal distance between these two arrays and found the α value as 0.81 (almost perfect agreement). This agreement score is significantly higher than a prior work [58] where the agreement score α is 0.46.

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

B. Tool Design

We compared two different approaches to design *ToxiSpanSE*. First, we used a lexicon-based naive approach, where words belonging to a predefined list are marked as toxic spans. Second, we used a supervised learning-based approach with five different transformer-based encoders. Figure 2 depicts our model architecture for the transformer-based models with an example prediction.

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

1) Preprocessing: The model takes the CR text and the target spans (labeled spans) with character offsets as input. Further, we split each text into sentences using en_core_web_sm from the spacy library [62] and keep corresponding character span offsets for each sentence. We have 39,438 sentences after splitting 19,651 CR texts; among those, 5,465 sentences have toxic spans. Therefore, around 13.85% samples in our dataset have at least one toxic span. We chose sentence-level evaluation for two reasons: i) a sentence may itself have toxic spans, ii) prior work also did sentence splitting for toxic span detection [50]. Further, we apply a tokenizer to convert each sentence to corresponding tokens. In this study, we use tokenizers that are appropriate for each model. For the lexicon-based model, we chose *NLTK word_tokenize [63]* from Python. On the other hand, we use transformer-based encoder models' corresponding tokenizer from huggingface [64]. We use *AutoTokenizer* function and select: i) *bert-base-uncased*, ii) *roberta-base*, iii) *distilbert-baseuncased*, iv) *albert-base-v2*, and v) *xlnet-base-cased* tokenizers for their corresponding encoder model. Moreover, we set the maximum length to 70 during the tokenization of each sentence, as using a Python script we empirically found that 98.5% of our sentence samples have less than 70 tokens. This pruning was essential as

TABLE II
EXAMPLE OF INTER-RATER AGREEMENT AND CONFLICT RESOLUTION

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

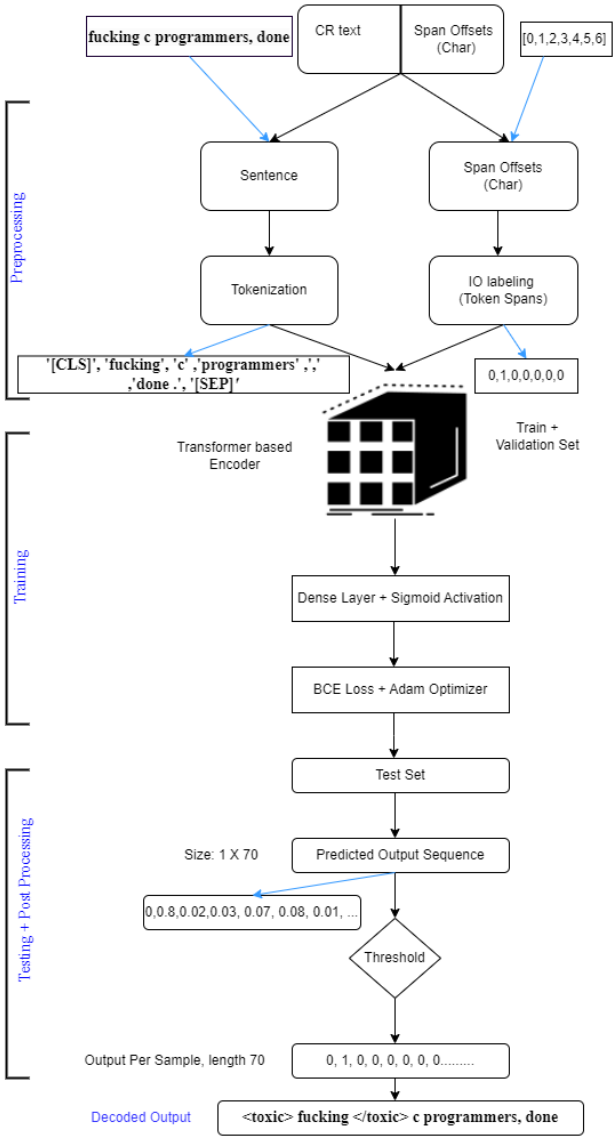

Fig. 2. Model Architecture of ToxiSpanSE. Optimal threshold for each model was empirically selected (detailed in IV-B2), Blue arrow → shows an example

taking a large token length significantly increases both required memory and training time. Each transformer-based pre-trained tokenizer splits the sentences into sub-word token strings and adds an unknown token to its dictionary if it finds them. For each sentence, each transformer-based tokenizer generates a special token at the start of the sentence at the end of the sentence (i.e., the bert-base tokenizer puts the [CLS] token at first and the [SEP] token at the end of each text). To pass the tokens of each sentence into the encoder model, we take three inputs for each sample from the tokenizers that are input_ids, token_type_ids, and attention_mask. We can decode the vector to the original string using input_ids.

2) IO Encoding: Prior NLP works with sequence labeling datasets followed BIO [50] or IO [65] tagging to encode the spans. *BIO* stands for Beginning, Inside, and Outside, where *B*-indicates the beginning token of a toxic span, *I*- indicates that the token is inside the toxic span, and *O* indicates a token outside the toxic span. BIO is suitable for NLP tasks to divide a span of text into multiple chunks. As we aim to identify which text spans contain something toxic, a simpler one, i.e., the IO -encoding, is sufficient for our goal. Moreover, IO simplifies our processing steps. In our IO encoding, every *I* tag corresponds to a token inside a toxic span, and *O* indicates outside.

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

3) Lexicon Based Model: We also designed a naive model referred to as the 'lexicon-based' model for detecting toxic spans from our dataset. In general, toxic spans contain many common words, including profanity, sexually explicit, and swear words. The purpose of developing this model is to evaluate whether a simple lexicon search-based approach compares against state-of-the-art transformer-based models. Our lexicon-based model matches each token in a text against a list of common toxic tokens with our ground truth tokens. We curated a list of toxic tokens ($\Sigma TOK = tok_0, tok_1, ..., tok_i$) from two prior studies, which include 85 profane words from Sarker *et* al. [5] and the top 100 toxic tokens from Kurita *et* al. [66]. In total, our lexicon list contains 167 tokens since there are overlapping tokens between those two lists.

Ground Truth For Lexicon Based Classification: To tokenize each sentence, we use *NLTK word_tokenize* from Python. Further, we use *textspan* library [67] to get the exact location of selected tokens from the human labeling spans. We use a similar IO encoding approach for this model (70-length vector for each input) where if that is inside the labeled span, we put

the token position as 1, otherwise 0.

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

4) Transformer based model: The Transformer deep learning architecture that emerged in 2017 [45] is based on multi-head self-attention and has shown to perform significantly better than Recurrent Neural Network (RNN)-based models for sequence-to-sequence tasks. Transformers use a self-attention mechanism for computing the internal representation of input and outputs. Moreover, the transformer-based model does not require any pre-computed context-free embedding vectors. Instead, it can generate context-based embeddings by pre-training the entire model as an encoder for sequence-to-sequence tasks.

From the preprocessing steps, we have inputs (input_ids, token_type_ids, attention_mask) for each sentence and we have generated ground truth (target labels) using IO encoding. Inputs and targeted IO encoding are passed to the encoder layer to generate context-based embeddings. Since there are several Transformer based encoders available for sequence classification tasks, we consider the following transformers which performed well in a prior token-level classification task [42]. In this work, we used Transformer based encoders from the HuggingFace library [64], selecting the following pre-trained encoders:

- BERT: Devlin *et* al. proposed the pretraining of the Deep Bidirectional Transformers for Language Understanding (BERT) model in 2018 that was trained with masked language modeling (MLM) and next sentence prediction (NSP) [15]. We use the BERT-base model, which has 12 transformer layers with 768 hidden states and 12 attention heads with 110 M parameters. BERT can be fine-tuned with domain-specific datasets for sentence and sequence classification tasks.
- DistilBERT: Sanh *et* al. proposed a lighter and faster version of the bert-base model, using modeling distillation, referred to as "DistilBERT" [68]. It has around 66 M parameters (40%
- RoBERTa: An optimized version of BERT is RoBERTa, which achieved better performance than BERT-base in some NLP tasks by pretraining the model for a longer time and on more data than the original BERT [17]. The base model has the same architecture as the BERT-base model.
- ALBERT: ALBERT has a similar architecture to the BERT-base but has only 128 hidden embedding layers that reduced the total parameters to 12 M [69]. We chose to use the ALBERT-base model for this study.
- XLNet: Unlike the autoencoder (AE) language models (i.e., BERT), Yang *et* al. proposed XLNet, which is based on autoregressive language modeling [70]. XLNet sought to overcome the limitations of the BERT model by maximizing the expected likelihood over all permutations of the factorization order. Moreover, its performance does not rely on data corruption. We use *xlnet-base-cased* model from the transformer library, which has a similar size as *BERT-base* model.

In this experiment, we select those pre-trained encoders from the HuggingFace library [64]. After the embeddings with size (1 X 70), we set a Dense layer to set the required final output size. Moreover, since we are doing a binary sequence classification task, the 'sigmoid' activation function is added to this Dense layer to generate the final output's probability. Therefore, our final output vector is a sequence of floating point values (from 0 to 1 due to the sigmoid function) with a length of 70.

5) Post Processing: After fine-tuning the model (details in next section), we predict the probability score with the test samples. The model provides a probability score from 0 to 1 for each token (70 per sample (s)). Using an empirically determined threshold (Section IV-B2) parameter, we decide whether a token is in the toxic class (1) or non-toxic (0). Further, from the prediction vector, we generate a set ($Pred_s$) of indexes for the output tokens in the toxic class. Our ground truth has already been preprocessed as toxic and non-toxic tokens. We also generate a set of the indexes of toxic tokens from the ground truth (G_s) of the test set. Finally, we wrote a Python script to decode each token from the sample and show the output like figure 2. We have input *"fucking c programmers, done"*, and the model provides the output *"<toxic>fucking </toxic>c programmers, done"*. To make the tool user-friendly, we use a tag (<toxic>) at the start and (</toxic>) at the end for predicted tokens inside toxic spans.

IV. Evaluation

A. Evaluation Metrics

Since our task is based on a sequence tagging approach for toxic spans, we adjusted our evaluation metric from Martino *et* al. [71] that is based on Potthast *et* al.'s plagiarism detection work [72]. Recently, Pavlopoulos *et* al. also used the same metric for toxic span detection in online discussions [52]. We decided to use this metric because it provides partial credit for matching the toxic spans inside a sequence. Unlike prior studies [52], [71], [72], we have chosen token-level comparison instead of character level for measuring the precision, recall, and F_1 score. The token-level comparison is taken because token-to-token comparison provides more explainability (comparing the ground truth toxic word to predicted toxic word) than the character label comparison. For example, a token(s) can represent the toxicity of the whole text, whether a character inside a token does not represent that meaning.

Let a code review sample(s) represent a sequence of tokens $tok_0, ...tok_j \subseteq s$. After IO encoding, the ground truth vector is a sequence of 1's and 0's with 70 values. We calculate the ground truth token offset as $G_s = pos_{tok_m}, ..., pos_{tok_n}$. So, for each sample($s$), G_s contains the position of all toxic tokens (pos_{tok_m}). When no toxic token exists in the sample, the $G_s = empty$. Similarly, a predictor model predicts the

tokens with a floating value. Further, we use a threshold (our experimental evaluation to identify optimal thresholds for each setup is detailed in Section IV-B2) to decide whether that token is toxic (1) or non-toxic (0). We generate the predicted token offsets $Pred_s$ for each sample from that vector. For better understanding, we put five examples in table III with ground truth (GT) and predicted (Pred) token offsets. Since we processed our text into tokens in preprocessing steps, the first token offset (tok_0) is for the special token (such as [CLS] for bert tokenizer). Therefore, our first token (i.e., 'it') position count starts from 1.

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

$$P^s(Pred_s, G_s) = \frac{|Pred_s \cap G_s|}{|Pred_s|} \tag{1}$$

$$R^s(Pred_s, G_s) = \frac{|Pred_s \cap G_s|}{|G_s|} \tag{2}$$

$$F1^s(Pred_s, G_s) = \frac{2 * P^s(Pred_s, G_s) * R^s(Pred_s, G_s)}{P^s(Pred_s, G_s) + R^s(Pred_s, G_s)} \tag{3}$$

In the equation 1, we define the precision P^s for each sample. We define the numerator as the length of the intersection of the set of predicted offsets ($Pred_s$) and ground truth token offsets (G_s). The denominator is the length of predicted offsets ($Pred_s$). Similarly, we calculate the recall (R^s) by using equation 2. Finally, we combined equation 1 and 2 to calculate the $F1$ in equation 3.

However, these equations can fail due to 0 in denominators. For example, if a model predicts none of the tokens from a sentence belonging to toxic spans, precision is undefined for that sentence. Similarly, for a correctly marked non-toxic instance, recall is undefined. We used the same approach as both Pavlopoulos *et* al. [52] and the SemEval-2021 Task 5 [13] to measure a variation of precision, recall, and F-score for span detection tasks. In this variation, if the number of predicted toxic tokens is 0 (i.e., $|Pred_s| = 0$), we check the number of toxic tokens in the ground truth set ($|G_s|$). If both sets are empty, the prediction is correct, and we assign this prediction a $precision = 1$; otherwise, we assign $precision = 0$. On the other hand, if the ground truth set is empty (i.e., $|G_s| = 0$), we assign $recall = 1$ only if the predicted set is also empty (i.e., $|Pred_s| = 0$), and $recall = 0$ otherwise. We would also like to mention that these custom precision/recall measures do not follow traditional precision/recall curve characteristics due to this variation.

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

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

B. Experimental Setup

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

1) Hyperparameters: We set the following hyperparameters during the training of our model:

- *Loss Function:* We chose a variant of a binary cross-entropy loss function for our task. Since we have multiple tokens (length = 70) with a range of fractional values from 0 to 1, we have added a too-small value (epsilon from Keras) with each prediction. This procedure will help our model to be more stable and prevent the prediction from 0 that can cause ($\log 0 = undefined$) problems. Moreover, we added a clipping between 1 and 0 inside the binary cross-entropy loss to avoid exploding gradients.
- *Optimizer and Learning Rate:* We use *Adam* optimizer with a learning rate $1e-5$.
- *Number of Epochs:* We set the number of epochs as 30 in each fold.
- *Early Stopping Monitor:* To prevent the model from overfitting during the training, we set the *EarlyStopping* function from the Keras library [73] where with the monitor with 'val loss'. During training for each epoch, the model is trained with the training dataset and tested with the validation dataset. While the validation loss does not decrease for four consecutive epochs, the model stops

TABLE III
EXAMPLE OF MODEL PREDICTIONS. RED REPRESENTS THE TOXIC TOKENS

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

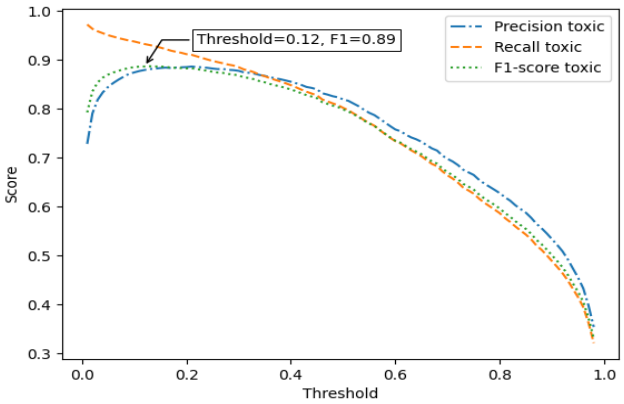

Fig. 3. Threshold variation for RoBERTa model (Using Validation Set)

training and saves the best model. We also empirically monitored that the optimal 'val_loss' provided the best $F1_1$ score for the validation set.

2) Threshold Selection: Since the models from each fold generate a sequence of 70 length vectors, with a floating value for each token, we need to set a threshold to convert to binary (0 or 1) values. In text classification tasks, a threshold of 0.5 is a common choice [5], [52]. To identify optimum thresholds, we evaluated our validation set by varying the threshold from 0.01 to 0.99 with a 0.01 increment. We aim to find the threshold resulting in the best $F1_1$ score for the validation set. We empirically evaluated each model with this threshold variation for the validation set in 10-fold. With the mean from 10-folds, we found the optimal threshold value for each model to maximize $F1_1$ score. For example, the RoBERTa model achieved the best $F1_1$ of 0.89 with a threshold of 0.12 with validation data set. Figure 3 shows performance (precision, recall, and F1) variations for the RoBERTa model against threshold variations using the validation set. We also noticed that the $F1-score$ for the toxic class remains the same from threshold 0.08 to 0.18 for the RoBERTa model. As we take a *variation* of precision and recall measurement, the plot does not behave like the general characteristics of precision and recall. Figure 3 also depicts that by increasing the threshold value, both precision and recall decreased. After calculating the optimal threshold from each model using the validation set, we use that optimal threshold for the corresponding model to predict the test set. During the test set prediction, we also did similar 10-fold cross-validation and got the mean of each metric. Finally, we report the results of each model's performance with the optimal threshold in Table IV.

C. Results with optimal threshold

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

Since our dataset is highly imbalanced, having a large number of empty spans, all of the five transformer models achieved similar scores for P_0, R_0, and $F1_0$ in the range of $0.90 \sim 0.95$. For toxic tokens, RoBERTa outperformed the other four models and achieved 0.87 precision, 0.93 recall, and 0.88 $F1_1$ score. BERT-base model achieved the second best performance with $F_1 = 0.86$. DistilBERT and ALBERT models achieved similar performance with 0.85 $F1_1$ score. However, DistilBERT has fewer parameters than other transformer models in our study, and this model is faster during fine-tuning than others. On the other hand, XLNet lacks the performance for both toxic and non-toxic classes than other transformer-based models.

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

D. Error Analysis from the best model

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

TABLE IV
EXPERIMENTAL RESULTS WITH THE OPTIMAL THRESHOLD. THE RUNTIME OF EACH MODEL AND PERFORMANCES DURING EACH FOLD IS INCLUDED IN THE REPLICATION PACKAGE [74]

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

the first column shows the error types, the second column is for ground truth (GT) token span offsets, and the third column is for predicted token span offsets.

1) Partial Disagreement: Since we are giving partial credit for the sequence classification metric, we decide to formulate a new error category as Partial Disagreement (PD). We consider an error as PD where the ground truth span offset has some values and the predicted span has some value with some disagreements. We found a total of 945 sentences (2.4% of the total sample and 27.75% of the error sample) in this category. The first three examples of Table V represent the PD category. In the first example, we can see that our rater labeled the 'what the hell' phrase inside the toxic span whereas the model predict 'the hell' as toxic.

2) False Positives: We consider False Positives (FP) when a sentence has no spans in its ground truth label but some of its portion is predicted as toxic. In our evaluation, a total of 2226 FPs (5.64% of total samples, and 65.35% of the error sample) occurred. We can see three examples of FPs in Table V. In those examples, the tokens (FC, stupid, and WTF) do not represent toxic meanings for those texts. In the third example of FP, the 'WTF' represents a library of Linux, not a 'what the fuck' phrase. Similarly, the 'stupid' word has been used by the reviewer to him/herself. For that reason, that sentence does not have any toxic meaning.

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

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

V. DISCUSSION

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

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

Lesson #3: Transformer-based models are reliable and explainable for the FOSS community: In our extensive evaluation, we found that the RoBERTa model outperformed others by achieving 0.88 $F1_1$ score while the other three transformer models also performed well for the toxic class. Since the sequence tagging approach is a challenging task for a new domain, our model can be used by the project maintainers to flag the toxic portion of a text. Moreover, since our best model has fewer false negative cases, FOSS maintainers can use this tool to detect the actual toxic segment from a toxic comment. Apart from that, we have used friendly post-processing, which provides an output with tagging: "you're not talking about neutron, (<toxic>) shut up (</ toxic>)".

Lesson #4: Proactive toxic prevention tool development: Since *ToxiSpanSE* is highly precise in identifying toxic ex-

TABLE V
EXAMPLE OF SOME ERRORS

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

cerpts, it is possible to leverage this model to proactively discourage toxic texts. For example, a Gerrit code review plug-in can be developed that highlights toxic excerpts similar to grammatical mistakes or typos, while a review is being written. Such highlights will make an author aware of potential toxic interpretations and may initiate a self-reflection.

Although the project maintainers would decide on content moderation, they can use our work to develop a tool to rephrase the toxic content to civil comments. Prior work introduced this concept for online communication text [52]. The research community from the SE domain and FOSS maintainers may think of this step to reduce the toxic comments from developers' communication.

VI. Threats to Validity

A. Internal Validity: Our selection of code review dataset from a prior work [5] remains a threat to validity. Biases in the curation of this dataset propagate to our study as well. However, Sarker *et* al. [5]'s dataset remains the largest labeled toxicity dataset for the SE domain, and it was curated using stratified sampling criteria to span various toxic instances. Since this selected dataset contains only code review comments, it may not adequately represent various other categories of developer communications such as issue discussions or technical question answering. However, that threat may be minimal as we focus on toxic phrases separate from a text's technical contents.

B. Construct Validity: Annotator bias during manual labeling is a potential threat to validity. To mitigate this threat, we reused an already established rubric [5], used a gender-diverse group of annotators including one woman and one man and arranged a discussion with the annotators to build a shared understanding of the rubric before starting the annotation process. Moreover, we followed recommended practices of independent labeling and conflict resolution through discussions. A high value of Krippendorff's α (i.e., – 0.81, 'almost perfect agreement') indicates the reliability of our labeling process.

We followed the definition and rubric of toxicity established by Sarker *et* al. [5]. While Sarker *et* al's conceptualization of toxicity is similar to the ones proposed by Raman *et* al. [4] and Miller *et* al. [2], there are subtle differences between their rubrics and ours. Therefore, models trained using our dataset may have degraded performance on datasets released by other studies. Similarly, our models may encounter degraded performance on SE datasets of other anti-social communication, such as incivility [21] and destructive criticism [9]. However, this limitation does not apply to our tool pipeline, and it can be retrained to fit other conceptualizations.

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

D. Conclusion Validity: Using the position-based metric threatens conclusion validity. To mitigate this threat, we adopted our metrics from prior studies with span detection [52], [71]. Moreover, since most of our labeled instances are non-toxic, we separately report the performance measures (i.e., P, R, and $F1$) for both toxic and non-toxic classes.

VII. Conclusion and Future Work

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

REFERENCES

[1] B. Barbarestani, I. Maks, and P. Vossen, "Annotating targets of toxic language at the span level," in *Proceedings of the Third Workshop on Threat, Aggression and Cyberbullying (TRAC 2022)*, 2022, pp. 43–51.

[2] C. Miller, S. Cohen, D. Klug, B. Vasilescu, and C. Kästner, ""did you miss my comment or what?" understanding toxicity in open source discussions," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022, pp. 710–722.

[3] A. A. Anderson, S. K. Yeo, D. Brossard, D. A. Scheufele, and M. A. Xenos, "Toxic talk: How online incivility can undermine perceptions of media," *International Journal of Public Opinion Research*, vol. 30, no. 1, pp. 156–168, 2018.

[4] 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 *International Conference on Software Engineering, New Ideas and Emerging Results*, ser. ICSE. ACM, 2020, p. TBD.

[5] J. Sarker, A. K. Turzo, M. Dong, and A. Bosu, "Automated identification of toxic code reviews using toxicr," *ACM Transactions on Software Engineering and Methodology*, 2023.

[6] C. D. Egelman, E. Murphy-Hill, E. Kammer, M. M. Hodges, C. Green, C. Jaspan, and J. Lin, "Predicting developers' negative feelings about code review," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 174–185.

[7] I. Steinmacher and M. A. Gerosa, "How to support newcomers onboarding to open source software projects," in *Open Source Software: Mobile Open Source Technologies: 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Proceedings 10*. Springer, 2014, pp. 199–201.

[8] C. Jensen, S. King, and V. Kuechler, "Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists," in *2011 44th Hawaii international conference on system sciences*. IEEE, 2011, pp. 1–10.

[9] S. D. Gunawardena, P. Devine, I. Beaumont, L. P. Garden, E. Murphy-Hill, and K. Blincoe, "Destructive criticism in software code review impacts inclusion," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–29, 2022.

[10] J. Sarker, A. K. Turzo, and A. Bosu, "A benchmark study of the contemporary toxicity detectors on software engineering interactions," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2020, pp. 218–227.

[11] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Software Engineering*, vol. 22, pp. 2543–2584, 2017.

[12] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[13] J. Pavlopoulos, J. Sorensen, L. Laugier, and I. Androutsopoulos, "Semeval-2021 task 5: Toxic spans detection," in *Proceedings of the 15th international workshop on semantic evaluation (SemEval-2021)*, 2021, pp. 59–69.

[14] D. Borkan, L. Dixon, J. Sorensen, N. Thain, and L. Vasserman, "Nuanced metrics for measuring unintended bias with real data for text classification," in *Companion proceedings of the 2019 world wide web conference*, 2019, pp. 491–500.

[15] 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.

[16] K. Krippendorff, "Reliability in content analysis: Some common misconceptions and recommendations," *Human communication research*, vol. 30, no. 3, pp. 411–433, 2004.

[17] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[18] L. Aroyo, L. Dixon, N. Thain, O. Redfield, and R. Rosen, "Crowdsourcing subjective tasks: the case study of understanding toxicity in online discussions," in *Companion proceedings of the 2019 world wide web conference*, 2019, pp. 1100–1105.

[19] A. Lenhart, M. Ybarra, K. Zickuhr, and M. Price-Feeney, "Online harassment, digital abuse, and cyberstalking in america. data & society research institute," *Center for Innovative Public Health Research. Retrieved from: https://datasociety. net/pubs/oh/Online_Harassment_2016. pdf*, 2016.

[20] C. AI, "What if technology could help improve conversations online?" [Online]. Available: https://www.perspectiveapi.com/

[21] 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.

[22] H. Barnes, "Toxicity in open source," 2020. [Online]. Available: https://boxofcables.dev/toxicity-in-linux-and-open-source/

[23] A. Diggs, "Windows is sh*t:' linux users and the technical superiority problem," 2021. [Online]. Available: https://medium.com/linuxforeveryone/windows-is-sh-t-linux-users-and-the-technical-superiority-problem-196a597aa860/

[24] ——, "Linux + coffee #4: Tribalism and toxicity," October 22nd, 2020. [Online]. Available: https://www.linux4everyone.com/linux-plus-coffee-3-community-tribalism-toxicity-gatekeeping

[25] L. Reinhard, "This is bigger than us: Building a future for open source," 2014. [Online]. Available: https://2014.jsconf.eu/speakers/lena-reinhard-this-is-bigger-than-us-building-a-future-for-open-source.html

[26] S. Vaughan-Nichols, "Linus torvalds takes a break from linux," 2018. [Online]. Available: https://www.zdnet.com/article/linus-torvalds-takes-a-break-from-linux/

[27] J. SALTER, "The perl foundation is fragmenting over code of conduct enforcement," 2021. [Online]. Available: https://arstechnica.com/gadgets/2021/08/the-perl-foundation-is-fragmenting-over-code-of-conduct-enforcement/

[28] K. D. A. Carillo, J. Marsan, and B. Negoita, "Towards developing a theory of toxicity in the context of free/open source software & peer production communities," *SIGOPEN 2016*, 2016.

[29] M. Squire and R. Gazda, "Floss as a source for profanity and insults: Collecting the data," in *2015 48th Hawaii International Conference on System Sciences*. IEEE, 2015, pp. 5290–5298.

[30] H. S. Qiu, B. Vasilescu, C. Kästner, C. Egelman, C. Jaspan, and E. Murphy-Hill, "Detecting interpersonal conflict in issues and code review: cross pollinating open-and closed-source approaches," in *Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society*, 2022, pp. 41–55.

[31] J. Sarker, "'who built this crap?' developing a software engineering domain specific toxicity detector," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3559508

[32] ——, "Identification and mitigation of toxic communications among open source software developers," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[33] 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.

[34] J. Pavlopoulos, P. Malakasiotis, and I. Androutsopoulos, "Deeper attention to abusive user content moderation," in *Proceedings of the 2017 conference on empirical methods in natural language processing*, 2017, pp. 1125–1135.

[35] M. M. Bhat, S. Hosseini, A. Hassan, P. Bennett, and W. Li, "Say 'yes'to positivity: Detecting toxic language in workplace communications," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2017–2029.

[36] S. V. Georgakopoulos, S. K. Tasoulis, A. G. Vrahatis, and V. P. Plagianakos, "Convolutional neural networks for toxic comment classification," in *Proceedings of the 10th hellenic conference on artificial intelligence*, 2018, pp. 1–6.

[37] P. Burnap and M. L. Williams, "Hate speech, machine classification and statistical modelling of information flows on twitter: Interpretation and communication for policy decision making," 2014.

[38] ——, "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.

[39] N. D. Gitari, Z. Zuping, H. Damien, and J. Long, "A lexicon-based approach for hate speech detection," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 10, no. 4, pp. 215–230, 2015.

[40] Y. Chen, Y. Zhou, S. Zhu, and H. Xu, "Detecting offensive language in social media to protect adolescent online safety," in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. IEEE, 2012, pp. 71–80.

[41] V. Isaksen and B. Gambäck, "Using transfer-based language models to detect hateful and offensive language online," in *Proceedings of the Fourth Workshop on Online Abuse and Harms*, 2020, pp. 16–27.

[42] J. Pavlopoulos, "Semeval 2021 task 5: Toxic spans detection," 2020. [Online]. Available: https://competitions.codalab.org/competitions/25623

[43] L. Xu, L. Bing, W. Lu, and F. Huang, "Aspect sentiment classification with aspect-specific opinion spans," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 3561–3567.

[44] Y. Qiang, X. Li, and D. Zhu, "Toward tag-free aspect based sentiment analysis: A multiple attention network approach," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[46] C. Sen, T. Hartvigsen, B. Yin, X. Kong, and E. Rundensteiner, "Human attention maps for text classification: Do humans and neural networks focus on the same words?" in *Proceedings of the 58th annual meeting of the association for computational linguistics*, 2020, pp. 4596–4608.

[47] H. Chefer, S. Gur, and L. Wolf, "Transformer interpretability beyond attention visualization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 782–791.

[48] Y. Qiang, D. Pan, C. Li, X. Li, R. Jang, and D. Zhu, "Attcat: Explaining transformers via attentive class activation tokens," in *Advances in Neural Information Processing Systems*, 2022.

[49] Q. Zhu, Z. Lin, Y. Zhang, J. Sun, X. Li, Q. Lin, Y. Dang, and R. Xu, "Hitsz-hlt at semeval-2021 task 5: Ensemble sequence labeling and span boundary detection for toxic span detection," in *Proceedings of the 15th international workshop on semantic evaluation (SemEval-2021)*, 2021, pp. 521–526.

[50] T. A. Suman and A. Jain, "Astartwice at semeval-2021 task 5: Toxic span detection using roberta-crf, domain specific pre-training and self-training," in *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, 2021, pp. 875–880.

[51] G. Chhablani, A. Sharma, H. Pandey, Y. Bhartia, and S. Suthaharan, "Nlrg at semeval-2021 task 5: toxic spans detection leveraging bert-based token classification and span prediction techniques," *arXiv preprint arXiv:2102.12254*, 2021.

[52] J. Pavlopoulos, L. Laugier, A. Xenos, J. Sorensen, and I. Androutsopoulos, "From the detection of toxic spans in online discussions to the analysis of toxic-to-civil transfer," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 3721–3734.

[53] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2022, open source software available from https://github.com/heartexlabs/label-studio. [Online]. Available: https://github.com/heartexlabs/label-studio

[54] A. Braylan, O. Alonso, and M. Lease, "Measuring annotator agreement generally across complex structured, multi-object, and free-text annotation tasks," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1720–1730.

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

[56] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[57] W. A. Scott, "Reliability of content analysis: The case of nominal scale coding," *Public opinion quarterly*, pp. 321–325, 1955.

[58] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee, "Hatexplain: A benchmark dataset for explainable hate speech detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 17, 2021, pp. 14 867–14 875.

[59] N. Ousidhoum, Z. Lin, H. Zhang, Y. Song, and D.-Y. Yeung, "Multilingual and multi-aspect hate speech analysis," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 4675–4684.

[60] F. Del Vigna12, A. Cimino23, 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.

[61] T. Grill and S. Castro, "Python implementation of krippendorff's alpha – inter-rater reliability," Github. https://github.com/grrrr/krippendorff-alpha, 2017.

[62] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[63] E. Loper and S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, 2002, pp. 63–70.

[64] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[65] R. Chen, J. Wang, and X. Zhang, "Ynu-hpcc at semeval-2021 task 5: Using a transformer-based model with auxiliary information for toxic span detection," in *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, 2021, pp. 841–845.

[66] K. Kurita, A. Belova, and A. Anastasopoulos, "Towards robust toxic content classification," *arXiv preprint arXiv:1912.06872*, 2019.

[67] Y. Tamura, "Text span utilities for rust and python," Github. https://github.com/tamuhey/textspan, 2020.

[68] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[69] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[70] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[71] G. Da San Martino, S. Yu, A. Barrón-Cedeno, R. Petrov, and P. Nakov, "Fine-grained analysis of propaganda in news article," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 5636–5646.

[72] M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso, "An evaluation framework for plagiarism detection," in *Coling 2010: Posters*, 2010, pp. 997–1005.

[73] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[74] J. Sarker, S. Sultana, S. Wilson, and A. Bosu, "Toxispanse: Replication package," Github. https://github.com/WSU-SEAL/ToxiSpanSE, 2023.