A Benchmark Study of the Contemporary Toxicity Detectors on Software Engineering Interactions

Jaydeb Sarker, Asif Kamal Turzo, Amiangshu Bosu
Department of Computer Science
Wayne State University
Detroit, MI, USA
{jaydebsarkar, asifkamal, amiangshu.bosu}@wayne.edu

2020 27th Asia-Pacific Software Engineering Conference (APSEC) | 978-1-7281-9553-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/APSEC51365.2020.00030

***Abstract*—Automated filtering of toxic conversations may help an Open-source software (OSS) community to maintain healthy interactions among the project participants. Although, several general purpose tools exist to identify toxic contents, those may incorrectly flag some words commonly used in the Software Engineering (SE) context as toxic (e.g., 'junk', 'kill', and 'dump') and vice versa. To encounter this challenge, an SE specific tool has been proposed by the CMU Strudel Lab (referred as the 'STRUDEL' hereinafter) by combining the output of the Perspective API with the output from a customized version of the Stanford's Politeness detector tool. However, since STRUDEL's evaluation was very limited with only 654 SE text, its practical applicability is unclear. Therefore, this study aims to empirically evaluate the Strudel tool as well as four state-of-the-art general purpose toxicity detectors on a large scale SE dataset. On this goal, we empirically developed a rubric to manually label toxic SE interactions. Using this rubric, we manually labeled a dataset of 6,533 code review comments and 4,140 Gitter messages. The results of our analyses suggest significant degradation of all tools' performances on our datasets. Those degradations were significantly higher on our dataset of formal SE communication such as code review than on our dataset of informal communication such as Gitter messages. Two of the models from our study showed significant performance improvements during 10-fold cross validations after we retrained those on our SE datasets. Based on our manual investigations of the incorrectly classified text, we have identified several recommendations for developing an SE specific toxicity detector.**

***Index Terms*—toxicity, chat, code review, developer communication, benchmark, rubric**

I. Introduction

Prior research have found multiple evidence of toxic interactions, such as: profanity, insult, hate speech, identity attack, misogynistic remarks, flirtations, or sexual innuendos, among several Free / Open Source Software (FOSS) projects [1]–[5]. Toxic interactions may have serious repercussions on a FOSS project. For example, a victim of toxic conversation may become afraid to express him/herself, therefore get demotivated, and may eventually leave the project.

The problem of online toxic conversations are more widespread than the FOSS projects. For example, a 2017 survey conducted by the Pew Research Center found that two out of five Americans have experienced online harassment [6]. Another study found 43% college students reporting being recipients of harassing messages [7]. More than one-third victims of those abusive online interactions reported feeling depressed [7]. Therefore, researchers have been focusing on automatic identification of toxic online conversations [8]–[12] to prevent such negative incidents. Jigsaw (a unit of Google)[1] is on the forefront of this research with building a public API named the Perspective API[2] to automatically score perceived toxicity of a text. Kaggle[3], in collaboration with the Jigsaw, started a competition called 'Toxic Comment Classification Challenge in 2018' [4] with the goal of building a classifier that can classify toxic contents better than the Perspective API. This challenge have produced high quality toxicity detectors with AUC [5] scores (i.e., 0.988).

Since software development is a collaborative activity, toxic conversations within a team may not only degrade relationships among team members but also have a great impact on the productivity of a developer [13]. Fear of bullying can refrain a developer from sharing his/her opinions or discourage a newcomer from seeking expert suggestions. Prior studies have found developers expressing frustrations over peers with 'prickly' personalities [14], [15]. Toxic conversations may not only demotivate developers but also waste valuable work hours [16]. Since software development communities, such as the FOSS projects, are professional communities, automated identifications of toxic conversations from software developer communications are crucial.

However, as prior research on building sentiment analysis tools for the Software Engineering (SE) domain has shown [17], toxicity detectors developed for other domains may not work well on SE conversations. An off-the-shelf toxicity detector may incorrectly flag some words commonly used in the SE context as toxic (e.g., 'junk', 'kill', and 'dump'). To encounter this challenge, an SE specific tool has been proposed by the CMU Strudel Lab (referred as the 'STRUDEL tool' hereinafter) by combining the output of the Perspective API [8] with the output from a customized version of the Stanford's Politeness detector tool [18]. Since the STRUDEL

[1]https://jigsaw.google.com
[2]https://www.perspectiveapi.com
[3]https://www.kaggle.com/
[4]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
[5]Area under receiver operating characteristic curve represents ability of of a binary classifier system as its discrimination threshold is varied.

2640-0715/20/$31.00 ©2020 IEEE
DOI 10.1109/APSEC51365.2020.00030

had a low F-score (0.57) during its limited evaluation with only 654 SE text, its practical applicability is uncertain. Therefore, this study aims *to empirically evaluate the STRUDEL tool as well as four other state-of-the-art general purpose toxicity detectors on a large scale SE dataset.* On this goal, we empirically developed a rubric to manually label each SE text as either 'toxic' or 'non-toxic' and using that rubric manually labeled two datasets of: i) 6,533 code review comments and ii) 4,140 Gitter messages. Using these datasets, we evaluated the five selected tools and answer the following three research questions:

(**RQ1**): **How do contemporary toxicity detectors perform on an SE dataset?**
Motivation: This evaluation will enable us to identify the practical applicability of contemporary toxicity detectors on SE conversations.
Results: None of the contemporary toxicity detectors achieved adequate performances to justify practical applications. The F-scores of the tools included in our study dropped more significantly on a formal SE communication dataset such as code review than on a informal communication such as Gitter messages. The significant disagreements between most of the tool pairs also suggest that results of an empirical study using one of these tools may differ significantly if we switch from one tool to another one.

(**RQ2**): **What are the categories of SE texts that contemporary toxicity detectors are more likely to misclassify?**
Motivation: This analysis will identify scenarios to consider when developing a customized toxicity detector for the SE domain.
Results: Most of the tools accurately identifies texts with expletives or swear words. However, those tools fails on words that have different meaning in the SE context. Moreover, contemporary tools also fail on sentences expressing humility, where an author uses demeaning words (e.g., 'stupid', 'dumb', and 'idiot') referring him/herself or his/her own work.

(**RQ3**): **Does retraining on a SE dataset improve the performances of contemporary toxicity detectors?**
Motivation: This analysis will show us the easeness or difficulty in building a customized toxicity detector for the SE domain.
Results: The results are highly promising with both of our retrained models outperforming contemporary models during 10-fold cross- validations on our SE datasets. A large scale labeled toxicity dataset of SE interactions may enable the development of reliable SE domain specific toxicity detectors.

The primary contributions of this paper are:

- An empirically developed rubric to manually label the toxicity of SE conversations.
- Two manually labeled toxicity datasets from the SE domain, with one dataset including 6,533 code review comments and the other dataset including 4,140 Gitter messages.
- An empirical evaluation of five contemporary toxicity detectors on two SE datasets.
- Empirical evidence depicting the possible development of a reliable toxicity detector for the SE domain by retraining exiting models on a SE dataset.
- A set of guidance for SE researchers on building customized toxicity detectors for the SE domain.
- **Enabling replication:** To enable future studies, we have made our dataset and results available online at: https://github.com/WSU-SEAL/toxicity-dataset

Paper organization: The remainder of this paper is organized as following. Section II provides a brief background on prior research in identifying online toxic contents. Section III details our research methodology. Section IV describes the results of our empirical evaluation. Section V discusses the considerations for building an SE domain specific toxicity detector. Section VI discusses the threats to validity of our findings. Finally, Section VII provides the future direction of our work and concludes this paper.

II. Background

Following subsections provide a definition of toxic contents in the context of the SE domain and briefly describe prior research in identifying online toxic contents.

A. Toxic Contents

Toxicity analysis is a natural language classification problem of identifying toxic contents. However, prior research had differing views on which contents should be considered as toxic. For example, the pew research center classifies offensive name calling, threats, and sexual harassment [6] as toxic interactions. Zaheri *et* al. includes insult, verbal sexual harassment, threats, obscene languages in their analyses of toxicity [19]. Georgakopoulos *et* al. included personal attacks, online harassment and bullying behaviors among toxic interactions [20]. Kurita *et* al. expanded their definition of toxic contents by including texts that can be harmful or offending to the recipient(s) [9]. The Perspective API [8] defines toxic contents as "texts that are rude, disrespectful, or unreasonable". Since the SE domain consists of professional communities, we adopt the following expansive definition of toxic contents in our analyses:

> *"An SE conversation will be considered as toxic, if it includes any of the following: i) offensive name calling, ii) insults, iii) threats, iv) personal attacks, v) flirtations, vi) reference to sexual activities, and vii) swearing or cursing."*

B. Prior Research on Toxicity Analysis

The Jigsaw team from Google developed the Perspective API to identify abusive online contents, which is considered one of the contemporary state-of-the-art tools. The models behind the Perspective API (PPA) are trained using manual annotations from crowd sourced human raters based on a published guidelines [21]. Besides the toxicity model, PPA

also provides experimental models to identify insults, profanities, identity_attacks, sexually_explicit contents, flirtations, and threats in a text. However, PPA is not free from flaws, as Hosseini *et* al. pointed out tricks to deceive the PPA [22].

Georgakopoulos et al. proposed a Convolutional Neural Network (CNN) based model trained on the Jigsaw dataset [20] and found that CNN based models performed better than bag-of-words based models in identifying toxic texts. Recently, researches have proposed several deep learning based toxicity classifiers [23]–[25] and provided guidelines on improving performances of these models.

Identifications of obfuscated or perturbed toxic words (e.g., 'fuc_k, and 'idoit') have been a major limitation of toxicity classifiers. Mishra et al. proposed a single embedding for unseen words to classify the toxic context, which is able to identify obfuscated and non-obfuscated words [26]. Kurita et al. proposed a model called Contextual Denoising Autoencoder (CDAE) to classify toxic contents [9]. They leveraged both character-level and contextual information in their models to overcome the limitations of contemporary toxicity classifiers in identifying perturbed toxic tokens (e.g., intentional typos to evade detection).

Due to the biases among the human raters against certain group of people (e.g., gay, lesbian, black, and muslim), toxicity classifiers based on several dataset are biased against those groups [27]. To overcome this challenge, Vaidya et al. proposed a multi-task learning model to reduced identity biases among toxicity classifiers [27].

Identifications of online bullying and hate speeches have also been focuses of several prior works. Waseem and Hovy proposed a set of 11 rules to annotate hateful tweets and labeled more than 16K public tweets using these criteria [28]. Using this dataset, they prepared a dictionary of the most hateful indicative words. Davidson et al. proposed a classifier to identify both hate speeches and offensive languages [29]. Chandrasekharan et al. proposed a model named Bag of Communities to identify abusive online interactions, where they showed that models trained based on labeled data obtained from one online community can be successfully reused to identify abusive contents from a different community [30].

Although, toxic contents have been found among SE interactions [1]–[5], most of the prior works in the SE domain focused on identifying software developers' sentiments [31]–[33]. Raman et al. proposed the first SE domain specific toxicity detector, and used that to study unhealthy interactions among FOSS developers [16].

III. Research Method

Currently, the STRUDEL dataset of 654 texts [16] is the only labeled toxicity dataset from the SE domain. Due to the small size of the STRUDEL dataset, we decided to create a new labeled toxicity dataset from SE interactions. Following subsections describe our approach to select text for our dataset, our manual labelling process, tool selection, and empirical evaluation of the selected tools.

A. Data Source Selection

Since toxic interactions are not very frequent among FOSS projects [16], [34], we looked into prior research and identified following two mediums, where toxic conversations are more likely to occur.

- **Code review** is a software development practice, where a developer sends his/her changes to peers for manual reviews. Although, code review is a formal process, incidents of profanity or insults are not uncommon during code reviews. [5], [34]. We selected three popular FOSS projects (i.e., Android, Chromium OS, and LibreOffice) as our data sources, since a recent research suggests toxicity among code reviews of those projects [5].
- **Gitter** is an open-source instant messaging and chat room system for software developers. Although Gitter is similar to Instant Relay Chat (IRC), Gitter's integration with Github repositories has made it popular among recent FOSS projects. Since prior research found toxic interactions among FOSS IRC channels [1], [35], we considered chat messages as one of our sources. We selected the gitter channel of the Ethereum project [6], since it is one of the most active channels on Gitter.

B. Data Mining

The code review repositories of the three selected projects are managed by Gerrit[7]. We wrote a Python script to access Gerrit's REST API to mine all the publicly available code reviews for the three projects and store the data in a MySQL database. Using an approach similar to Paul et al. [5], we identified the bot accounts to exclude the comments not written by humans. We used the GitterPy[8] library to connect to Gitter's REST API and download all the messages to our MySQL database. Table I shows an overview of the messages mined by our scripts from the four FOSS projects.

C. Dataset Generation

Due to the rarity of toxic interactions, a fully-randomized selection of text from our data sources would create a highly unbalanced dataset of less than 1% toxic texts. To overcome this challenge, we adopted a customized stratified sampling strategy [36] by leveraging the Google's Perspective API. First, we use the PPA to compute the toxicity score of each text. The PPA score for a text varies between 0 to 1, which indicates the probability of that text being toxic. Since a PPA score of 0.5 or above suggests a text as more likely to be 'toxic' than to be 'non-toxic', we included all the texts with PPA scores above 0.5. Based on this selection, we obtained 3,213 code review texts and 1,950 Gitter messages.

Second, we divided the texts with PPA scores less than 0.5 from our two datasets into five equally spaced PPA score groups with each group spanning an interval of 0.1. From the code review dataset, we randomly selected 664 text from each

[6]https://gitter.im/ethereum/go-ethereum
[7]https://www.gerritcodereview.com/
[8]https://github.com/myuz/GitterPy

TABLE I
AN OVERVIEW OF THE MESSAGES MINED BY OUR MINING SCRIPTS

Project	Data Source	Time period	Total messages	Toxic messages*
Android	Code review: https://android-review.googlesource.com/	December 2008 to June 2019	152,065	647
Chromium OS	Code review: https://chromium-review.googlesource.com/	April 2011 to March 2020	1,176,642	2,485
LibreOffice	Code review: https://gerrit.libreoffice.org/	March 2012 to June 2019	12,273	81
Ethereum	Gitter: https://gitter.im/ethereum/go-ethereum	June 2014 to March 2020	122,355	1,950

*As classified by the Perspective API

group. For example, we randomly selected 664 code review comments with a PPA score between 0 to 0.1, 664 texts with a PPA score between 0.1 to 0.2 and so on. Using this stratified sampling, we selected additional 3,320 code review comments that are classified as 'non-toxic' by PPA. Similarly, we selected additional 2,190 gitter messages (i.e., 438 messages from each group) with PPA scores less than 0.5.

In addition to these two datasets, we randomly sampled 2,000 'non-toxic' and '1,000' toxic texts from the labeled Jigsaw test dataset [37]. We use this dataset of 3,000 texts to evaluate baseline performances of the selected tools on a non-SE dataset.

D. Manual Labeling

During the first stage of our manual labeling, we focused on developing a rubric to manually label the toxicity class of the selected texts. Our initial rubric was based on the guidelines published by the Conversation AI team [21]. Two of the authors independently went through 1,000 texts to prepare a set of rules. Then, we had a discussion session to create a unified set of rules for labeling. Using this set of rules, two of the authors independently labeled all the selected texts. Table II shows the set of rules with examples taken from our dataset.

After the independent manual labeling, we compared the labels from the two raters to identify conflicts. Out of the 6,533 comments from the code review dataset, the two raters agreed on 5,950 comments (i.e., 91.1%). On the Gitter dataset, out of the 4,140 messages, the raters agreed on 3,730 messages (i.e., 90.1%). We also measured the level of agreement between the two raters using Cohen's Kappa (κ) [38], which was estimated as 0.727 for our code review dataset and as 0.781 for our Gitter dataset. Kappa (κ) values are commonly interpreted as follows: values $\leq$ 0 as indicating 'no agreement' and 0.01–0.20 as 'none to slight', 0.21–0.40 as 'fair', 0.41– 0.60 as 'moderate', 0.61–0.80 as 'substantial', and 0.81–1.00 as 'almost perfect agreement'. Therefore, the level of agreement between the two raters during our manual labeling can be considered as 'substantial'.

Finally, we had discussion sessions to review the disagreements and come up with an agreed upon rating for each of the texts with a conflicted rating. After the conflict resolution process, we found 20% 'toxic' texts in our code review dataset, while the Gitter dataset had 35.4%. Since real-time chats are bit more informal than code reviews, a higher ratio of toxic texts in the Gitter dataset may not be surprising. Table III provides a brief overview of the Jigsaw sample dataset as well as the two SE datasets after the completion of our manual labeling.

E. Tool Selection

We selected total five tools for evaluation. We selected the Perspective API [8], since it has been widely used, and is considered one of state-of-the-art tool for toxic text classification. The STRUDEL tool [16] was selected, since it is the only SE domain specific toxicity detector. The remaining three tools for evaluation were selected based on following two criteria.

1) The design and evaluation of the tool was published as a research paper.
2) The source code of the tool is publicly available for download and evaluation.

Although, we noticed an influx of toxicity detector publicly available on the Github (https://github.com/topics/toxic-comment-classification) as a part of the 2018 Kaggle classification challenge [39], most of those tools fail our first inclusion criteria. In the following subsections, we provide a brief overview of the five tools selected for our analyses.

1) Perspective API (PPA): The developers from the Google's Conversational AI and Jigsaw developed the Perspective API to identify abusive online contents [8]. The primary goal of the PPA is to develop a online platform to reduce the toxic comments and create spaces for healthy conversations[9]. The Perspective API currently provides six different models to rate the level of toxicity, profanity, insult, identity attack, threat, sexual explicit from a text. The PPA scores for a text from the six models vary from 0 to 1, which indicate the the probability of that text belonging to a particular category (e.g., toxic, threat, or insult) [40]. We use the publicly available REST API to compute PPA scores for each text in our datasets. Using the score from the PPA 'toxicity model', we classify a text as toxic if it has a PPA score of 0.5 or higher.

2) STRUDEL Toxicity Detector (STRUDEL): Raman et al. proposed the first toxicity detector for the SE domain recently [16]. They observed that many texts were incorrectly classified as 'toxic' by the PPA due to the occurrences of words that are considered 'toxic' in non-SE context but are included in the technical SE vocabulary (e.g., kill, abort, and die). They developed an automated pre-processing model to identify words that are significantly over-represented in a SE dataset compared to general English and replace those words with more neutral filler words. They used a modified version of the Stanford's politeness detector tool [18] to identify the

[9]https://jigsaw.google.com/

TABLE II
SET OF RULES FOR CLASSIFYING A TEXT AS EITHER 'TOXIC' OR 'NON-TOXIC' WITH EXAMPLES TAKEN FROM OUR DATASET

#	Rule	Rationale	Example*
Rule 1:	Inclusion of a profane or curse words in a sentence would be marked as 'toxic'.	Profanities are the most common sources of online toxicities.	"we don't want to fuck 64-bit bit up like 32-bit was fucked."
Rule 2:	Inclusion of an acronym, that refers to expletive or swearing, in a sentence would be marked as 'toxic'.	Sometimes people use acronyms of profanities, which are equally toxic as its' expanded form.	"wtf is going on with this nonstop?"
Rule 3:	Insult to another person or person's work would be marked as 'toxic'.	Insulting another developer may create a toxic environment and should not be encouraged.	"YOU MUST BE A BIG FOOL"
Rule 4:	Attacking a person's identity (e.g., race, religion, nationality, gender or sexual orientation) would be marked as 'toxic'.	Identity attacks are considered toxic among all categories of online conversations.	"you are twice as smart as a typical stupid American consumer, you get to have an unlimited number of children"
Rule 5:	Threatening another person or a community would be marked as 'toxic'.	Threats may stir hostility between two developers and force the the recipients leave the community.	"One of these days I'm going to slap you @****"
Rule 6:	Both implicit or explicit References to sexual activities would be marked as 'toxic'.	Implict or explicit references to sexual activities may make some developers, particularly females, uncomfortable and make them leave a conversation.	"i know but...well its like masturbating vs sex you see what i mean "
Rule 7:	Flirtations would be marked as 'toxic'.	Flirtations may also make a developer uncomfortable and make a recipient avoid the other person during future collaborations	"Just like how I told that woman I thought she looked pretty. "
Rule 8:	If a demeaning word (e.g., 'dumb', 'stupid', 'idiot', 'ignorant') refers to either the writer him/herself or his/her work, the sentence would not be marked as 'toxic', if it does not fit any of the first seven rules.	It is common in SE community to use those word for expressing their own mistakes. In those cases, the use of those toxic words to himself/herself does not make toxic meaning.	"stupid me, my editor shows them the same color and tricks me every time."
Rule 9:	A sentence, that does not fit rules 1 through 8, would be marked as 'non-toxic'.	General non-toxic comments.	"you can delete this main logic as the wrapper.py handles it for you."

** Examples are provided verbatim, to accurately represent the context. We did not censor any text, except omitting the reference to a person's name.*

TABLE III
AN OVERVIEW OF THE THREE DATASETS

Dataset	# total texts	# toxic	# non-toxic
Jigsaw Sample	3,000	1,000	2,000
Code Review	6,533	1,310	5,223
Gitter Ethereum	4,140	1,468	2,672

politeness score of the pre-processed text. Finally, they used an SVM classifier to classify the text as either 'toxic' or 'non-toxic' by combining the PPA score of the unmodified text with the politeness score of the pre-processed text.

The STRUDEL tool was evaluated using a dataset of 654 issue comments mined from Github, where only 167 comments were labeled as 'toxic'. During their evaluation it achieved a precision of 0.91 and a recall of 0.42 of their test dataset. Moreover, on a dataset of 100,000 randomly sampled GitHub issues it achieved 50% precision in identifying 'toxic' comments. In our evaluation, we use the pretrained STRUDEL model available on Github.

3) Deep Pyramid Convolutional Neural Networks (DPCNN): Johnson and Zhang proposed a deep neural network based model for toxic text classification, which they named as Deep Pyramid Convolutional Neural Networks (DPCNN) [41]. DPCNN outperformed prior state-of-the-art models on six benchmark datasets in both sentiment and toxicity classification. A DPCNN implementation also performed well in the 2018 Kaggle classification challenge [39] by achieving an AUC score of 0.98. Similar to the PPA, DPCNN also provides the probability of a text being toxic from 0 to 1. In our evaluation, we use a DPCNN model trained using the Jigsaw dataset. Similar to the PPA scores, we used the threshold of 0.5 to consider the DPCNN classification as either toxic or non-toxic.

4) BERT with fast.ai (BFS): Devlin et al. proposed a pre-trained deep bidirectional representations from unlabeled text language model called Bidirectional Encoder Representations from Transformers (BERT) [42]. BERT based models are currently considered as the state-of-arts in natural language processing (NLP) classification tasks . Kurita et al. proposed a toxicity classification model [9] by fine tuning a BERT model for fast.ai library[10] for the [11]. A BFS model trained using the Jigsaw dataset achieved one of the highest public AUC scores

[10]https://www.fast.ai/

[11]https://www.kaggle.com/keitakurita/bert-with-fastai-example

in the Kaggle competition [39].

In our evaluation, we use the publicly available python implementation of the BFS kernel [12]. We retrain a BFS model with the Jigsaw dataset and were able to achieve the an AUC score of 0.9853 (i.e., same as the listed public score). As BFS also outputs the probability of a text being toxic from 0 to 1, we use the threshold of 0.5 to classify each text as either 'toxic' or 'non-toxic' based on its BFS score.

5) Hate Speech Detection (HSD): Davidson et al. proposed an automated multi-class classifier to classify a text as either a hate speech, or an offensive language or neither [29]. The HSD model was classified using a dataset of 25K manually labeled tweets and achieved a precision 0.91, recall of 0.90, and F1 score of 0.90 during five-fold cross validations. During our evaluation, we use the pretrained HSD models and classify a text as 'toxic' if it was classified as either 'a hate speech' or 'an offensive language' by the HSD model.

F. Evaluation Metrics

In our evaluation, we use the following five measures to compare the performances of the tools on our SE datasets.

- **Accuracy:** Accuracy is the ratio of texts that were correctly classified by a tool.
- **Precision:** The ratio between the number of toxic texts that are correctly classified by a tool and the number of of texts that are marked as toxic by the same tool.
- **Recall:** The ratio between the number of toxic texts that are correctly classified by a tool and the number of toxic texts in the dataset.
- **F-score:** The harmonic mean of precision and recall.
- **Cohen's Kappa (κ):** The level of agreement between a tool's classification and our manual labeling as measured using Cohen's Kappa (κ) [38].

We also calculate the level of agreement between each pair of tools using the Cohen's Kappa (κ) [43] to determine how obtained results would vary if a different tool was selected for analyses.

IV. Results

The following subsections present the results of the three research questions introduced in the Section I.

A. RQ1: How do Contemporary Toxicity Detectors Perform on an SE Dataset?

Table IV shows the precision, recall, F-score, and accuracy of the five tools, when evaluated on our three datasets. On the baseline Jigsaw sample test dataset, DPCNN achieves the best precision, F-score, and accuracy, while STRUDEL achieved the best recall. Four out of the five tools (i.e., except HSD) achieved 'substantial' agreement with the human raters.

On the code review dataset, the STRUDEL tool, which is customized for the SE domain, achieves the best recall. However, due to lower precision than the PPA, STRUDEL falls behind in terms of both F-score and accuracy. Although, HSD had the the second highest precision among the five tools on the code review dataset, it also had the lowest F-score due to its failure to identify (i.e., false negatives) toxic texts that do not express hate speeches. DPCNN had the highest precision, accuracy and kappa on the code review dataset. The kappa values suggest that the five tools achieved at best 'Fair' agreements with the human raters and therefore may not suitable to identify toxic texts from code review interactions.

All the five tools performed better on the Gitter dataset than each performed on the code review dataset, since the Gitter dataset has higher ratios of toxic comments as well as higher number of messages with profanities. Similar to the code review dataset, the STRUDEL achieved the best recall on the Gitter dataset but falls behind the PPA in terms of both Accuracy and F-Score. Both the BFS and the HSD achieved high precisions, but failed to achieve high F-scores due to large number of false negatives. The kappa values suggest that four out of the five tools achieved 'moderate' agreements with the human raters, which can be considered as improvements over the performances achieved by those tools for the code review dataset.

By comparing each tools performance on the two SE datasets against its performance on the Jigsaw sample, we noticed significant degradations of F-scores. Both precisions and recalls of each tool dropped by more than 0.10 on the two SE datasets. Among the five tools, PPA provides the best F-scores on both of our SE datasets and may be considered as the baseline for building SE domain specific toxicity detectors.

Table V shows the agreements between each pair of tools measured using the Cohen's Kappa (κ). Since we have five different tools, there are ten possible pairings. The highest level of agreements were seen between the PPA and STRUDEL pairs on the both SE datasets. Based on the κ values, agreements between these two tools can be considered as 'substantial'. Since STRUDEL uses PPA scores as an input for classification, 'substantial' agreements between these two tools are not surprising. The only other pair that showed 'substantial' agreement is the DPCNN-BFS pair on the Gitter dataset, since both DPCNN and BFS models are trained using the same dataset (i.e., Jigsaw toxicity dataset), their agreements on the Gitter dataset may not be surprising, while their 'moderate' level of agreement on the code review dataset deserves more investigation. The lowest level of agreement was obseved between the STRUDEL-HSD pair. These results suggest that if an empirical investigation of toxic texts are conducted using one of these tools, the results may be different if we select one of the different tools, except for switching between STRUDEL and PPA may still yield the same results.

[12]https://www.kaggle.com/keitakurita/bert-with-fastai-example/

TABLE IV
PERFORMANCE OF THE FIVE TOXICITY ANALYSIS TOOLS ON THE THREE DATASETS

Dataset	Tools	Precision (p)	Recall(r)	F-Score (f)	Accuracy (A)	Kappa (κ)
Jigsaw Sample *(baseline)*	PPA	0.762	0.986	0.858	0.893	0.775
	STRUDEL	0.734	**0.990**	0.843	0.877	0.746
	DPCNN	**0.896**	0.829	**0.861**	**0.911**	**0.796**
	BFS	0.887	0.834	0.859	0.909	0.793
	HSD	0.889	0.427	0.577	0.791	0.461
Code Review	PPA	0.397	0.762	**0.522**	0.720	**0.351**
	STRUDEL	0.347	**0.861**	0.495	0.648	0.294
	DPCNN	**0.708**	0.285	0.406	**0.833**	**0.33**
	BFS	0.663	0.253	0.366	0.824	0.287
	HSD	0.705	0.051	0.095	0.805	0.071
Gitter Ethereum	PPA	0.707	0.806	**0.753**	**0.813**	**0.604**
	STRUDEL	0.626	**0.880**	0.732	0.771	0.542
	DPCNN	0.901	0.511	0.652	0.806	0.532
	BFS	0.892	0.524	0.660	0.809	0.539
	HSD	**0.978**	0.238	0.382	0.728	0.283

TABLE V
LEVEL OF AGREEMENTS BETWEEN THE TOOL PAIRS ON OUR SE DATASETS

Dataset	Level of agreement between tool pairs (κ)				
		STRUDEL	DPCNN	BFS	HSD
Code Review	PPA	**0.777**	0.225	0.221	0.033
	STRUDEL		0.156	0.151	0.023
	DPCNN			0.595	0.18
	BFS				0.142
Gitter Ethereum	PPA	**0.812**	0.497	0.504	0.236
	STRUDEL		0.383	0.39	0.168
	DPCNN			0.759	0.49
	BFS				0.481

Finding 1: *While contemporary toxicity detectors have moderate agreements with human raters on identifying toxic texts from informal conversations such as chat messages, they perform poorly on a more formal SE conversation such as code reviews. Since only one out of the ten possible pairs had substantial agreements with each other, the results of an empirical study may significantly differ, if we switch from one tool to another from those nine low agreement pairs.*

B. RQ2: What are the Categories of SE Texts that Contemporary toxicity Detectors are More Likely to Misclassify?

We conducted secondary investigations to identify cases where most of the tools misclassified to identify the challenges in developing an SE domain specific toxicity detector.

All the tools used in our study are based on supervised models, pretrained with a large labeled datasets. However, many of the words has different meanings in the SE context than in general English. Most of the tools failed for such words. In the following we list such words with examples.

- **kill:** is frequently used during code reviews and developer chats to suggest killing a process or simply removing a code snippet. For example, "*yeah, they don't seem to be needed, so let's kill them.*", suggests removing some unnecessary code snippet, which was misclassified as toxic by most of the tools.
- **execute:** refers to running a process or application in the SE domain. For example," *Any program executed by any kernel thread, including usermodehelper, from rootfs will switch to init?*", refers to running a program, but was incorrectly classified as toxic.
- **die, dead:** Both 'die' and 'dead' refers to state of program execution or code snippets and are often misclassified by off-the-shelf toxicity detectors. For example, *"Remove the old, dead code.*"
- **garbage:** is another word that can be used both in toxic and non-toxic ways. For example, in "*initialize init_pid_ to -1 here so it doesn't have garbage in it*", garbage cannot be classified as toxic. However, a developer referring another developer's code as 'garbage' would be toxic. We noticed 'garbage' used in non-toxic contexts for most of the cases and were misclassified by the tools.
- **dummy:** is often used to refer to placeholder files or objects. "*What is it used for? An empty dummy file should work.*", is an example of misclassification of a text with this word.
- **junk:** Under slang terms, 'junk' refers to male privates. However, in the SE domain 'junk' often refers to useless objects and can be misclassified. For example, "*I'd like to have that here too, since input may have junk data after a valid CBOR.*"
- **dirty:** In the SE domain, 'dirty' often refers to a modified file or memory location. A misclassification with the word dirty:, "*why not place this in the dirty bits iteration? (with a comment on why we need it for D3D11)*".
- **trash:** refers to removing file, code snippet or objects. A misclassified example is, "*You really don't need to derive from std::less¡¿. If anything, you should be deriving from std::binary_function, but it is really not needed for std::set to work correctly, so I would just trash that base class.*"
- **daemon:** refers to a computer program that runs as a background process in the SE domain. However, in a non-SE domain, it may refer to something supernatural and therefore, was classified as toxic. An example of such occurrence is : "*Based on the old version, it looks like lxc should be built even if USE=daemon is not sent. ...*"

- **naked:** is usually considered as a toxic word with a sexually explicit reference. However, in the C programming language a 'naked pointer' refers to pointers that can be used to point to another object. Texts with the word 'naked' was frequently misclassified during code reviews. For example, *"For now, let's keep it like this, there's a discussion going on what to do with the naked C++ pointers.*"
- **dump:** can be used as a slang to indicate 'the act of defecation'. However, in the SE context dump often refers to storing data. For example, an example of misclassification with dump is: "*Use json.dump, json.load instead of doing your own string parsing. ...*"
- **stupid, dumb, idiot, fool, ignorant:** Most of the classifiers marked all the texts with these words as toxic. However, during both code reviews as well as Gitter chats, developers frequently used those words to express humility. For example, "*Maybe a stupid question: where's this variable defined?*".
- **CAPITALIZED ACRONYMS:** In C or C++ constant variables are often declared in all caps. References to code segments are often included in code reviews. For example, "*Make this another DCHECK.*" Some of the tools incorrectly marked unknown capitalized acronyms as toxic.

Finding 2: *Many of the words, that are used under toxic contents in non-SE domains, have different meanings in the SE context, are more frequently misclassified by the toxicity detectors.*

C. RQ3: Does Retraining on a SE Dataset Improve the Performances of Contemporary Toxicity Detectors?

While we intended to to reevaluate all the five tools after retraining those on a SE dataset, it was feasible for us to retrain only two models (i.e., DPCNN and BFS) with our dataset. We could not retrain the PPA, since its source code is proprietary. STRUDEL uses the PPA model and the dataset to customize the Stanford politeness detector for STRUDEL is not publicly available. HSD is a multiclass model with three classes ('hate speech', 'offensive', and 'neither'). Since our datasets are not labeled accordingly, we excluded the HSD model.

We evaluated the the models using 10-fold cross-validations. Table VI shows the average performances of the two models after retraining on our datasets. Both of the models achieves significant performance improvements after retraining on our SE dataset. The DPCNN based model achieved an F-Score of 0.88 on the code review dataset, which is better than its baseline performance (i.e., F-Score of 0.86 on the Jigsaw sample). But it under-performed on the Gitter dataset with an F-Score of 0.731. On the other hand, the BFS based model's F-Score of 0.860 on the Gitter dataset was almost similar as its baseline performance (F-Score of 0.859). However, BFS under-performed on the code review dataset with an F-Score of 0.731.

Finding 3: *Both models achieved significant performance boosts after retraining on our SE datasets. Two out of the four models beat its' baseline F-Scores achieved on a non-SE dataset. A large scale labeled toxicity dataset of SE interactions may enable developing SE domain specific toxicity detectors that can be used for identifying toxic texts from real-world SE interactions.*

V. Implications

In this paper, we evaluated five contemporary toxicity detectors on two SE datasets. Following are the key lessons obtained from this study.

1) **Off-the-shelf tools are reliable in identifying profanities.** Profanities are the most common sources of online toxicities. We found most of the tools highly reliable in flagging texts with profanities. Therefore, if an SE community only wants to flag profane languages, off-the-shelf tools such as PPA can be useful with its profanity detection model.

2) **Off-the-shelf tools are not reliable on SE datasets.** Although, PPA and STRUDEL show moderate performance on identifying toxic texts from Gitter messages, their performances are not reliable on formal conversations such as code reviews. Therefore, off-the-shelf tools must be evaluated for reliability on a dataset drawn from the study context before their application.

3) **Retraining off-the-shelf tools on a SE dataset significantly improves performance.** While we conducted a preliminary investigation by retraining two of the off-the-shelf tools on our SE dataset, the results are highly promising. We believe, if we retrain contemporary models using a larger and more robust SE dataset than the one used in this study and add SE domain specific preprocessing, we can develop a reliable toxicity classifier for the SE domain.

4) **SE domain specific preprocessing may improve performances.** We noticed that several misclassifications from the existing tools were due to code snippets included in developer communications. Since SE domain specific sentiment analysis tools also recommend filtering out code snippets [31], we believe that preprocessing steps to identify and remove code snippets may improve the performances of SE domain specific toxicity detectors.

5) **Excluding SE domain specific words may cause false negatives.** The results of RQ2(Section IV-B) illustrated several words that may have different meaning in an SE context. While using the approach adopted by Raman et al. [16], we can replace these words with a more neutral words, and reduce those misclassifications, this approach may also generate false negatives if these words are truly used to express a toxic opinion. For example, following lists shows toxic usage of those words from our dataset:

TABLE VI
PERFORMANCE OF THE TOXICITY DETECTORS AFTER RETRAINING ON OUR SE DATASET

Dataset	Tools	Precision (p)	Recall(r)	F-Score (f)	Accuracy (A)	Kappa (κ)
Code Review	DPCNN	**0.880**	**0.890**	**0.880**	**0.920**	**0.817**
	BFS	0.780	0.688	0.731	0.898	0.669
Gitter Ethereum	DPCNN	**0.840**	0.670	0.740	**0.910**	0.692
	BFS	0.838	**0.884**	**0.860**	0.892	**0.773**

- garbage: "*Why you changed this to %ecx? it is garbage here.*"
- kill: "*go kill yourself*"
- junk: "*so I don't have to clean up my junk after myself* "
- dirty: "*Only if you promise to talk dirty to me*"
- dump: "*I wouldn't recommend telling girls they are pretty, I mean if it's the first or second thing you say your intentions are clear, you just want sex, and then they dump you.*"
- die: "*well the US can go and die*"
- dead: "*... should just move over to ethereum immediately... no point in flogging a dead horse*"

6) **A reliable toxicity detector must identify the target of words to identify expressions of humility.** Sentences using the words: 'idiot', 'stupid', 'dumb', 'ignorant', and 'fool' to express humility were often misclassified by the contemporary toxicity detectors. We found both toxic and non-toxic usages of those words. Since during expressions of humility, these words refer to the author him/her self or his/her works, a reliable toxicity classifier must identify the target of those words to identify toxic contexts from non-toxic ones.

VI. THREATS TO VALIDITY

The first threat to validity for this study is our selection of data sources which come from four FOSS projects. While these projects represent four different domains, many domains are not represented in our dataset. Moreover, our projects represent some of the top OSS projects with organized governance. Therefore, several categories of highly offensive texts are underrepresented in our datasets.

Second, our stratified sampling strategy was based on the scores provided by the PPA. Although, we manually verified all the texts classified as 'toxic' by the PPA, we randomly selected only 5,510 texts that had PPA scores of less than 0.5. Among those texts, we identified 513 toxic texts (9.3%). Therefore, if the PPA misclassified some categories of 'toxic' comments and also our random selections missed those, instances of such texts may be missing in our datasets.

Third, we accepted the default parameters for the selected tools and did not use parameter tuning to improve performances. Therefore, some of the tools may have achieved better performances on our datasets through parameter tuning.

Finally, although we have selected a diverse set of tools trained on different datasets, we may be missing some tools that could have achieved a better performance on our datasets. To enable evaluations of more tools on our datasets, we had made those publicly available on a Github repository.

VII. CONCLUSION

In this paper, we empirically evaluated STRUDEL, the only SE domain specific toxicity detector, as well as four other state-of-the-art general purpose toxicity detectors on two labeled SE datasets. We empirically developed a rubric to manually label toxic SE interactions, and using this rubric, we manually labeled a dataset of 6,533 code review comments and 4,120 Gitter messages.

The results of our analyses suggest that none of the contemporary toxicity detectors could achieve adequate performances to justify practical applications. The performances of the tools included in our study dropped more significantly on a formal SE communication dataset such as code review than on a dataset of informal communication such as Gitter messages. The significant disagreements between most of the tool pairs also suggest that results of an empirical study using one of these tools may differ significantly if we switch from one tool to another one. One of the primary limitations of existing tools are their failures to identify non-toxic contexts of certain words that are commonly used in toxic contexts in a non SE-domain but may have different meanings in the SE domain. Sentences with source code snippets and with the words, such as: 'idiot', 'stupid', 'dumb', 'ignorant', and 'fool' to express humility were also frequently misclassified. We retrained two of the models from our study on our SE dataset and obtained highly promising results with two out of the four models beating its' baseline F-Scores obtained on a non-SE dataset. These results suggest that the development of a highly reliable SE domain specific toxicity detector is feasible by retraining existing models on a large-scale and robust labeled dataset of SE interactions.

Based on our investigations, we have identified several key lessons that may help researchers in developing an SE domain specific toxicity detector. The rubrics developed in this study to manually label toxic SE interactions as well as the two labeled datasets, which are publicly available, will be also helpful. The future direction for this research include: i) the development of a large-scale and robust labeled dataset, ii) the development and evaluation of SE domain specific text preprocessing steps to improve the performances of toxicity classifiers, and iii) the development of a reliable toxicity classifier for the SE domain.

REFERENCES

[1] 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.

[2] Anonymous, "Leaving toxic open source communities," 2014. [Online]. Available: https://modelviewculture.com/pieces/leaving-toxic-open-source-communities

[3] D. Nafus, J. Leach, and B. Krieger, "Gender: Integrated report of findings," *FLOSSPOLS, Deliverable D*, vol. 16, 2006.

[4] N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai, and E. Murphy-Hill, "Investigating the effects of gender bias on github," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 700–711.

[5] R. Paul, A. Bosu, and K. Z. Sultana, "Expressions of sentiments during code reviews: Male vs. female," in *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering*, ser. SANER '19. IEEE, 2019, conference.

[6] M. Duggan, "Online harassment 2017," 2017.

[7] M. Lindsay, J. M. Booth, J. T. Messing, and J. Thaller, "Experiences of online harassment among emerging adults: Emotional reactions and the mediating role of fear," *Journal of interpersonal violence*, vol. 31, no. 19, pp. 3174–3195, 2016.

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

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

[10] I. Gunasekara and I. Nejadgholi, "A review of standard text classification practices for multi-label toxicity identification of online content," in *Proceedings of the 2nd workshop on abusive language online (ALW2)*, 2018, pp. 21–25.

[11] A. G. d'Sa, I. Illina, and D. Fohr, "Bert and fasttext embeddings for automatic detection of toxic speech," in *SIIE 2020-Information Systems and Economic Intelligence*, 2020.

[12] B. Alshemali and J. Kalita, "Improving the reliability of deep neural networks in nlp: A review," *Knowledge-Based Systems*, vol. 191, p. 105210, 2020.

[13] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 133–142.

[14] A. Bosu, A. Iqbal, R. Shahriyar, and P. Chakroborty, "Understanding the motivations, challenges and needs of blockchain software developers: A survey," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2636–2673, 2019.

[15] A. Filippova and H. Cho, "The effects and antecedents of conflict in free and open source software development," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 705–716.

[16] 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.

[17] 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, no. 5, pp. 2543–2584, 2017.

[18] C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts, "A computational approach to politeness with application to social factors," *arXiv preprint arXiv:1306.6078*, 2013.

[19] S. Zaheri, J. Leath, and D. Stroud, "Toxic comment classification," *SMU Data Science Review*, vol. 3, no. 1, p. 13, 2020.

[20] 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.

[21] C. AI, "Annotation instructions for toxicity with sub-attributes," https://github.com/conversationai/conversationai.github.io/blob/master/crowdsourcing_annotation_schemes/toxicity_with_subattributes.md, 2018.

[22] H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran, "Deceiving google's perspective api built for detecting toxic comments," *arXiv preprint arXiv:1702.08138*, 2017.

[23] H. Chen, S. McKeever, and S. J. Delany, "The use of deep learning distributed representations in the identification of abusive text," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 13, no. 01, 2019, pp. 125–133.

[24] A. Elnaggar, B. Waltl, I. Glaser, J. Landthaler, E. Scepankova, and F. Matthes, "Stop illegal comments: A multi-task deep learning approach," in *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference*, 2018, pp. 41–47.

[25] S. Srivastava, P. Khurana, and V. Tewari, "Identifying aggression and toxicity in comments using capsule network," in *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, 2018, pp. 98–105.

[26] P. Mishra, H. Yannakoudakis, and E. Shutova, "Neural character-based composition models for abuse detection," *arXiv preprint arXiv:1809.00378*, 2018.

[27] A. Vaidya, F. Mai, and Y. Ning, "Empirical analysis of multi-task learning for reducing model bias in toxic comment detection," *arXiv preprint arXiv:1909.09758*, 2019.

[28] Z. Waseem and D. Hovy, "Hateful symbols or hateful people? predictive features for hate speech detection on twitter," in *Proceedings of the NAACL student research workshop*, 2016, pp. 88–93.

[29] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Eleventh international aaai conference on web and social media*, 2017.

[30] E. Chandrasekharan, M. Samory, A. Srinivasan, and E. Gilbert, "The bag of communities: Identifying abusive behavior online with preexisting internet data," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 3175–3187.

[31] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: a customized sentiment analysis tool for code review interactions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 106–111.

[32] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018.

[33] M. R. Islam and M. F. Zibran, "Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text," *Journal of Systems and Software*, vol. 145, pp. 125–146, 2018.

[34] C. D. Egelman, E. Murphy-Hill, E. Kammer, M. M. Hodges, C. Green, C. Jaspan, and J. Lin, "Pushback: Characterizing and detecting negative interpersonal interactions in code review," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, ser. ICSE '20, 2020.

[35] K. Crowston and M. Squire, "Lessons learned from a decade of floss data collection," in *Big Data Factories*. Springer, 2017, pp. 79–100.

[36] C.-E. Särndal, B. Swensson, and J. Wretman, *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[37] N. Thain, L. Dixon, and E. Wulczyn, "Wikipedia Talk Labels: Toxicity," 2 2017. [Online]. Available: https://figshare.com/articles/dataset/Wikipedia Talk Labels Toxicity/4563973

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

[39] "Toxic comment classification challenge," Mar 2018. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[40] E. Wulczyn, N. Thain, and L. Dixon, "Ex machina: Personal attacks seen at scale," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1391–1399.

[41] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 562–570.

[42] 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.

[43] J. Cohen, "Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit." *Psychological bulletin*, vol. 70, no. 4, p. 213, 1968.