# Sentiment Analysis for Software Engineering: How Far Can Pre-trained Transformer Models Go?

Ting Zhang, Bowen Xu*, Ferdian Thung, Stefanus Agus Haryono, David Lo, Lingxiao Jiang
School of Information Systems, Singapore Management University
Email: {tingzhang.2019, bowenxu.2017}@phdcs.smu.edu.sg, {ferdianthung, stefanusah, davidlo, lxjiang}@smu.edu.sg

***Abstract*—Extensive research has been conducted on sentiment analysis for software engineering (SA4SE). Researchers have invested much effort in developing customized tools (e.g., SentiStrength-SE, SentiCR) to classify the sentiment polarity for Software Engineering (SE) specific contents (e.g., discussions in Stack Overflow and code review comments). Even so, there is still much room for improvement. Recently, pre-trained Transformer-based models (e.g., BERT, XLNet) have brought considerable breakthroughs in the field of natural language processing (NLP). In this work, we conducted a systematic evaluation of five existing SA4SE tools and variants of four state-of-the-art pre-trained Transformer-based models on six SE datasets. Our work is the first to fine-tune pre-trained Transformer-based models for the SA4SE task. Empirically, across all six datasets, our fine-tuned pre-trained Transformer-based models outperform the existing SA4SE tools by 6.5-35.6% in terms of macro/micro-averaged F1 scores.**

***Index Terms*—Sentiment Analysis, Software Mining, Natural Language Processing, Pre-trained Models**

## I. Introduction

Sentiment analysis is a computational study of people's opinions, attitudes, and emotions toward an entity, which can be an individual, an event, or a topic [1]. Sentiment analysis for software engineering (SA4SE) has drawn much attention in recent years [2]–[10]. Most research considers sentiment analysis as a sentiment polarity classification task. For a given text unit, the goal is to determine its sentiment orientation, i.e., negative, neutral, or positive.

To understand the performance of SA4SE tools, three benchmarking studies have been conducted: Lin et al. [9] compared five sentiment analysis tools, i.e., SentiStrength, NLTK, Stanford CoreNLP, SentiStrength-SE, and Stanford CoreNLP SO on three datasets – mobile app reviews, Stack Overflow posts, and Jira issue comments. In terms of the number of correct predictions, SentiStrength-SE, Stanford CoreNLP, and SentiStrength perform the best for one of the datasets. Islam et al. [7] compared three SA4SE tools, i.e., SentiStrength-SE, Senti4SD, and EmoTxt, on three datasets – Jira issue comments, Stack Overflow posts, and code review comments. They found that SentiStrength-SE achieved the highest macro-averaged F1-score for Jira issue comment and code review comment datasets. At the same time, Senti4SD performed the best for the Stack Overflow post dataset. Novielli et al. [8] compared four tools, i.e., Senti4SD, SentiStrength-SE, SentiCR, and SentiStrength, on four datasets – Stack Overflow posts, Jira issue comments, code reviews comments, and Stack Overflow posts related to Java libraries.[1] They found that Senti4SD achieved the highest macro-averaged F1-score for the Stack Overflow dataset, while SentiCR was the highest for the other three datasets.

Inspired by these three studies, we raise the main research question that drives this study: *How well can pre-trained Transformer models perform for SA4SE task?* Transformer is a deep neural network architecture based solely on the attention mechanism. It replaces the most commonly used recurrent layers in the encoder-decoder architecture with multi-head self-attention [11]. Currently, Transformer has become the mainstream architecture of pre-trained models [12]. These pre-trained models are trained on large corpora to learn universal language representations and can further be beneficial to downstream natural language processing (NLP) tasks without the need to train new models from scratch.

To answer the aforementioned question, we conduct a large-scale exploratory study. Specifically, we (1) consider a diverse collection of six datasets (instead of three or four considered in prior work), (2) compare the effectiveness of the best performers in prior work [7]–[9] with state-of-the-art pre-trained Transformer models. This study investigates the following specific research questions:

- **RQ1**: *How accurate are Transformer models as compared to existing SA4SE tools?*
- **RQ2**: *How efficient are Transformer models as compared to existing SA4SE tools?*

To answer the above questions, we compare the accuracy and efficiency of the best performing SA4SE approaches in prior studies, against four pre-trained Transformer models. Prior studies [7]–[9] have highlighted that Stanford CoreNLP, SentiStrength, SentiStrength-SE, SentiCR, and SentiSD are the best performers on at least one dataset. For pre-trained Transformer models, we consider BERT [13], RoBERTa [14], XLNet [15], and ALBERT [16]. We fine-tune these models with labeled Software Engineering (SE) specific data for SA4SE tasks.[2] We evaluate the approaches on six datasets: API reviews (API), Stack Overflow posts (SO), Mobile app reviews (App), GitHub pull-request and commit comments (GitHub),

*Bowen Xu is the corresponding author.

[1]They referred to the dataset as 'Java Libraries' and it is the Stack Overflow dataset from Lin et al.'s work [9].

[2]For brevity, unless otherwise stated, we refer to these fine-tuned pre-trained models as Transformer models.

Jira issue comments (Jira), and Gerrit code review comments (CR).

The experimental results demonstrate that in all the six datasets, Transformer models, i.e., BERT, RoBERTa, XLNet, and ALBERT, can achieve better performance than the best performing SA4SE tools identified in prior studies [7]–[9]. Across these datasets, Transformer models consistently outperform previous SA4SE tools by 6.5% to 35.6% in terms of macro/micro-averaged F1-scores. This accuracy boost comes with some runtime costs: Transformer models are less efficient than existing SA4SE approaches (except Senti4SD and Stanford CoreNLP). Still, its runtime cost is not prohibitively high; it requires 15 seconds to 10 minutes to fine-tune, while it can predict sentiments of hundreds of text units (documents) in seconds.

The main contributions of this work are as follows:

1) We are the first to leverage various pre-trained Transformer-based models for the SA4SE task.
2) We provide a large-scale comparative analysis between five existing SA4SE tools and four Transformer models on six SE datasets and demonstrate that Transformer models perform better than prior specialized SA4SE tools.

The remainder of this paper is organized as follows. Section II introduces related work. Sections III and IV describe the best performing SA4SE tools identified in prior studies [7]–[9] and pre-trained Transformer-based models used in this paper, respectively. Section V elaborates on our methodology to answer the research questions. We present the results of our experiments for the two *RQ*s in Section VI. Then we analyze the threats to validity in Section VII. Finally, we conclude and present future work in Section VIII.

## II. Related Work

In this section, we introduce related work: the first group is about sentiment analysis for software engineering; the second group is on pre-trained models for NLP.

**Sentiment Analysis for Software Engineering.** Previous research has shown that emotions influence work outcomes and dynamics, such as task quality, productivity, creativity, group rapport, user focus, and job satisfaction (c.f. [17]). On the other side of the coin, work processes and outcomes influence developer emotions (c.f. [18]). Much research has been done to investigate aspects of this two-way relationship between developers' work and their emotions.

One line of work that has attracted much research interest is the sentiment analysis of software artifacts, such as bug reports and commit comments. For example, Guzman et al. [2] studied sentiments in commit comments in GitHub to analyze the social factors affecting software development. Villarroel et al. [4] mined emotional information from mobile apps reviews to support the release planning activity. To further analyze the impact of negative code review comments, Ahmed et al. [6] developed a code-review specific sentiment analysis tool. Fine-grained emotions have also been studied. Gachechiladze et al. [19] focus on automatic identification of anger direction (anger towards self, others, and object) in a collaborative software development environment. They found that all of the anger directions are present within the comments from Apache issue reports [19].

TABLE I: Sentiment Analysis for Software Engineering Tools

| Tool | Technique | Original Training/Test dataset |
|---|---|---|
| NLTK/VADER [25] | Lexicon and rule-based | Social media texts |
| Stanford CoreNLP [22] | Recursive neural tensor network | Movie reviews |
| SentiStrength [22] | Lexicon and rule-based | MySpace informal short texts |
| SentiStrength-SE [5] | Lexicon and rule-based | Jira issues comments |
| SentiCR [6] | Supervised learning | Code review comments |
| Senti4SD [26] | Supervised learning | Stack Overflow |
| Emotxt [23] | Supervised learning | Stack Overflow and Jira |
| DEVA [24] | Lexical approach | Jira issue comments |

The progress of sentiment analysis research in SE has promoted the development of corresponding tools. In the early stage, researchers mainly use NLTK [20], SentiStrength [21], Alchemy[3], and Stanford CoreNLP sentiment analyser [22] for sentiment polarity classification. By comparing the performance of general-purpose sentiment analysis tools in the SE field, Jongeling et al. [3] found that these tools produced inconsistent annotated labels, and they may not necessarily agree with each other. Therefore, they claim a need for SE domain-orientated sentiment analysis tools. Due to the non-optimal performances of these off-the-shelf sentiment analysis tools built from the general text, more SE domain-specific sentiment analysis tools have been introduced. Most of them focus on the classification of sentiment polarity. Senti4SD and SentiCR are two examples, and they are supervised learning-based sentiment analysis tools. The construction of emotion recognition toolkits for SE text has also drawn much attention. Emotxt [23], which is an open-source toolkit for detecting emotions, i.e., *love, joy, surprise, anger, sadness, fear*, is trained specifically on the datasets extracted from Stack Overflow and Jira. DEVA [24], which is specially built for SE text, is a dictionary-based approach to detect valence and arousal in text. It can capture individual emotional states (e.g., excitement, stress, depression, relaxation, and neutrality). Table I shows the current publicly available SA4SE tools.

Some exploratory studies [7]–[9] are similar to our work, and they compare the performance of general-purpose and SE-specific sentiment analysis tools in different datasets. We have discussed them in Section I. In this work, we want to extend these studies by comparing the best performing tools identified in their work against Transformer models.

**Pre-trained Models for Natural Language Processing.** We follow Qiu et al. [12] to categorize pre-trained models into two generations. The first generation of pre-trained models aims to learn word embeddings [12]. Typical examples are word2vec [27] and GloVe (Global Vectors for Word Representation) [28]. Many prior SE works, e.g., detection of incoherent comments [29] and identification of SE-relevant tweets [30], have utilized word embeddings. One apparent limitation of this kind of word

[3]This service from IBM was retired in 2017.

embeddings is that they are context-independent. Regardless of the context, the same word has the same embedding. As a consequence, these non-contextual embeddings fail to model polysemous words [12].

Recently, pre-trained models that learn contextual word representations and can be fine-tuned for downstream tasks have become popular [12]. They are the second-generation pre-trained models. One family is LSTM [31]-based. Among them, ULMFiT (Universal Language Model Fine-tuning) [32] and ELMo (Embeddings from Language Models) [33] are two front-runners. Since the introduction of Transformer architecture [11], a number of pre-trained Transformer-based models have been proposed (e.g., [13]–[16], [34]); these form another family, and many of its members have achieved state-of-the-art performance for a number of downstream NLP tasks.

## III. Prior SA4SE Tools

In this section, we briefly describe details about the best performing approaches identified by the prior benchmarking works [7]–[9]: Stanford CoreNLP, SentiStrength, SentiStrength-SE, SentiCR, and Senti4SD. We refer to them collectively as the PRIOR group.

**Stanford CoreNLP**, proposed by Socher et al. [35], is designed for single-sentence sentiment classification; it can return a sentiment value and polarity for a sentence. Socher et al. introduced the Stanford Sentiment Treebank, which includes fine-grained sentiment labels for 215,154 phrases in the parse trees. The parse trees consist of 11,855 sentences extracted from the movie review dataset, initially constructed by Pang and Lee [36]. Socher et al. also proposed a new model called Recursive Neural Tensor Network to capture the compositional effects with higher accuracy. Stanford CoreNLP is trained with this Recursive Neural Tensor Network on the Stanford Sentiment Treebank.

**SentiStrength** is a lexicon-based approach developed by Thelwall et al. [21]. As a lexicon-based approach, SentiStrength has several dictionaries, including both formal terms and informal texts (such as emoticon, idiom, slang). In these dictionaries, each term is labeled with a sentiment strength. Based on these dictionaries and linguistic analysis, given a sentence, SentiStrength will output two integers: one is for *positive* emotion, and the other is for *negative* emotion. It not only categorizes the emotional polarity but also gives the strength of the polarity. The scale for positive emotion is from 1 to 5, representing not positive to very strong positive; the range for negative emotion is from -1 to -5, representing not negative to very strong negative.

**SentiStrength-SE**, proposed by Islam and Zibran, is a customized version of SentiStrength, implemented by adding a domain-specific dictionary [5]. SentiStrength-SE is also the first sentiment analysis tool considering SE-specific context, and it is designed based on in-depth qualitative research. Specifically, Islam and Zibran first used SentiStrength to detect sentiment in Jira issue comments. They analyzed 151 Jira issue comments for which SentiStrength produced wrong outputs. This analysis was performed to identify the reasons/difficulties that affect the accuracy of SentiStrength. Finally, they identified 12 such difficulties. They also found that out of all the difficulties, the domain-specific meanings of words were the most prevalent. To build a domain dictionary, they first collected a large dataset of commit messages drawn from 50 open-source projects from GitHub provided by their earlier work [37]. Then they extracted the lemmatizations of all words in the dataset. Next, they kept the overlap between these word lemmatizations and the SentiStrength dictionary of sentiment words. A total of 716 words remain. Through manual assessments, they further eliminated words that carry neutral sentiments. Finally, the final word dictionary of SentiStrength-SE consists of 500 words, of which 167 are positive, and 293 are negative. They also extended the dictionary by adding new sentiment words and negations. Additionally, contextual information is considered in SentiStrength-SE.

**SentiCR** [6] is designed by Ahmed et al., particularly for code review comments. Based on the characteristics of code review comments, SentiCR has a suite of data preprocessing steps, including URL removal and code snippet removal. SentiCR includes a two-stage negation preprocessing approach. Ahmed et al. first build a chunk grammar (i.e., a set of rules indicating how sentences should be chunked) for *NLTK RegexpChunkParser* to identify negation phrases. Second, they modify all the verbs, adjectives, and adverbs in a negation phrase identified by the chunker by prepending *not* to it [6]. After generating feature vectors using TF-IDF, eight supervised classifiers are evaluated. They also use 10-fold cross-validation to validate each algorithm. GBT (Gradient Boosting Tree) [38] demonstrates the highest precision, recall, and accuracy among all the eight used algorithms. Thus, by default, the supervised classifier in SentiCR is GBT. The original SentiCR is trained to classify a code review comment as either negative or non-negative.

**Senti4SD** is another supervised learning-based tool. The largest difference between Senti4SD and previous SE-specific tools is how it generates feature vectors. Senti4SD [26] utilizes three different features based on (1) Generic sentiment lexicons. It uses SentiStrength lexicons; (2) Keywords (n-gram extracted from the dataset). It primarily uses uni-gram and bi-gram. The value of each keyword feature corresponds to its number of occurrences. In addition to uni-gram and bi-gram, it also includes other keyword features, e.g., total occurrences of uppercase words, and slang expressions for laughter; (3) Word representation in a distributional semantic model (DSM) specifically trained on Stack Overflow data. DSM uses the CBOW architecture implemented by word2vec [27]. Each Stack Overflow document (i.e., answers, questions, and comments) is represented as the vector sum of all the vectors of words in the document. Besides, it calculates four prototype vectors, namely, *p_pos, p_neg, p_neu*, and *p_subj*, respectively. *p_pos* is the sum of all the word vectors in each document, which have positive polarity in the SentiStrength lexicon dictionary. Similarly, by summing up all the nega-

tive/neutral word vectors in the document, we have *p_neg* and *p_neu*. *p_subj* is the sum of *p_pos* and *p_neg*. Using these four objective vectors for a document, Senti4SD calculates the similarity scores between document vectors to get the semantic features. Finally, based on the features mentioned above, Senti4SD is trained to distinguish sentiment polarities of text units by using Support Vector Machines (SVM).

## IV. Pre-trained Transformer-based models

In this section, we briefly introduce the four pre-trained Transformer-based models, i.e., BERT, RoBERTa, XLNet, and ALBERT. We refer to these models collectively as the PTM (Pre-trained Transformer Model) group.

**BERT** has been designed to learn pre-trained contextual word representations from unlabeled texts [13]. Its architecture is a multi-layer bidirectional Transformer encoder. It learns the contextual word representations by optimizing for two tasks. The first task is *masked language model* (MLM); MLM randomly masks some words from the input text, and the task is to predict the masked words based on their contexts (i.e., words appearing before and after each of the masked words). The second task is *next sentence prediction* (NSP); NSP predicts if one sentence follows another. The original paper provides two implementation versions: basic-size ($BERT_{BASE}$) and large-size ($BERT_{LARGE}$). $BERT_{BASE}$ has 12 layers, a hidden layer size of 768, 12 self-attention heads, and 110M parameters. In comparison, $BERT_{LARGE}$ has 24 layers, a hidden layer size of 1,024, 16 self-attention heads, and 340M parameters. Given the large number of parameters, fine-tuning $BERT_{LARGE}$ needs more time and consumes more memory than $BERT_{BASE}$. Thus, in this work, we use $BERT_{BASE}$. Based on the experimental results reported in the original paper, $BERT_{LARGE}$ usually outperforms $BERT_{BASE}$.

**RoBERTa (Robustly optimized BERT approach)** is reported to achieve state-of-the-art results on the benchmarks GLUE, RACE, and SQuAD when Liu et al. [14] released it. In their paper, Liu et al. presented a replication study of BERT pre-training, which measures the impact of critical hyper-parameters and training data sizes. By modifying the pre-training steps of BERT, RoBERTa can achieve substantially better performance than BERT. Regarding the modifications, firstly, RoBERTa uses larger mini-batch sizes to train the model for a longer time over more data. Secondly, RoBERTa removes the NSP loss in BERT, and it trains on longer sequences. Moreover, RoBERTa is trained with dynamic masking: the masking pattern will be generated every time a sequence is fed to the model.

**XLNet** [15] combines the strengths of autoregressive (AR) language modeling and autoencoding (AE) to deal with their individual limitations. XLNet is capable of learning contextual information by maximizing the expected log-likelihood of a sequence of words considering all permutations. By integrating the segment recurrence mechanism and relative encoding scheme of Transformer-XL [34], XLNet can produce a better performance, especially for long texts. It achieves the lowest error rates for the IMDB dataset, and it outperforms BERT on 20 tasks, including sentiment analysis.

TABLE II: Datasets

| dataset | # doc | # (%) positive | # (%) neutral | # (%) negative |
|---|---|---|---|---|
| API | 4,522 | 890 (19.7) | 3,136 (69.3) | 496 (11) |
| SO | 1,500 | 131 (8.7) | 1,191 (79.4) | 178 (11.9) |
| App | 341 | 186 (54.5) | 25 (7.3) | 130 (38.1) |
| GitHub | 7,122 | 2,013 (28.3) | 3,022 (42.4) | 2,087 (29.3) |
| Jira | 926 | 290 (31.3) | - | 636 (68.7) |
| | **# doc** | **# (%) non-negative** | | **# (%) negative** |
| CR | 1,600 | 1,202 (75.1) | | 398 (24.9) |

**ALBERT** [16] is a lite version of BERT, and it is proposed to address the GPU/TPU memory limit and long training time issues of BERT. It applies two parameter-reduction techniques, i.e., a factorized embedding parameterization, and cross-layer parameter sharing. The application of these two techniques can improve efficiency by decreasing the number of BERT parameters to a great extent without seriously affecting performance. One thing worth mentioning is that although ALBERT has fewer parameters than BERT, it has a larger architecture; thus, it is computationally more expensive than $BERT_{LARGE}$. ALBERT has been shown to outperform $BERT_{LARGE}$ on the GLUE, RACE, and SQuAD benchmarks.

## V. Methodology

This section first describes the six datasets used in this work, and defines the sentiment analysis task based on the polarity labels in the datasets. Then, we elaborate on the implementations of all the considered approaches. Lastly, we describe the relevant evaluation metrics and settings.

### A. Datasets

In this comparative study, we make use of six publicly available datasets with annotated sentiment polarities. Table II shows the detailed statistics of the six datasets, including the total number of documents in a dataset (# doc) and the number (and percentage) of documents with one of the sentiment polarities (e.g., # (%) positive, # (%) neutral, # (%) negative).

- **API reviews (API) [39].** It includes 4,522 sentences from 1,338 Stack Overflow posts. This dataset contains both API aspects and the polarities of provided opinions, i.e., positive, negative, and neutral), curated by Uddin and Khomh.
- **Stack Overflow posts (SO) [9].** It consists of 1,500 sentences. Lin et al. [9] obtained this dataset from the Stack Overflow dump dated July 2017. They pick discussion threads that (i) are tagged with Java, and (ii) contain one of the following words: library, libraries, or API(s). Then, they randomly selected 1,500 sentences and manually labeled their sentiment polarities. This dataset is similar to the API dataset. However, they are significantly different on sentiment polarity distribution: SO dataset has similar distribution about the positive and negative polarity. However, in the API dataset, the sentences with a positive sentiment are nearly twice those with a negative sentiment.

- **Mobile app review dataset (App) [9].** This dataset has 341 reviews randomly chosen by Lin et al. from the dataset of 3k reviews formerly provided by Villarroel et al. [4]. Considering a 95% confidence level and 5% confidence interval, these 341 reviews is a statistically significant sample of the 3k reviews. When performing random selection, the proportions of reviews belonging to these four categories, namely *bug reporting, a suggestion for new features, request for improving non-functional requirements (e.g., performance of the app), and other*, have been retained in the new dataset.
- **GitHub pull-request and commit comments (GitHub) [10].** It consists of 7,122 sentences from GitHub pull-request and commit comments. Novielli et al. conducted an iterative extraction to obtain the annotated text units from the dataset provided by Pletea et al. [40].
- **Jira issue comments (Jira) [9].** This dataset has been used in several prior studies (e.g., [23], [41]) and it is previously provided by Ortu et al. [42] with four emotions labelled: *love*, *joy*, *anger*, and *sadness*. Lin et al. [9] assign a positive polarity to the sentences labelled with *love* and *joy*, a negative polarity to the sentences labelled with *anger* and *sadness*. It does not contain any neutral polarity and is a binary-class dataset.
- **Code review comments (CR) [6].** It is released with SentiCR and is a binary-class dataset, including negative and non-negative comments. The comments are collected from 20 popular open-source projects that practice tool-based code reviews supported by the same tool (i.e., Gerrit) [6]. Unlike the other datasets, the unit of text in this dataset is not a sentence, but rather a document (whereas each document can include multiple sentences). Therefore, for SentiStrength, SentiStrength-SE, and Senti4SD, we concatenate these sentences into one long sentence to be able to use these tools.

We considered all the three datasets used in Lin et al.'s benchmarking work [9] – mobile app reviews (App), Stack Overflow posts (SO), and Jira issue comments (Jira). We include another three datasets (i.e., GitHub, API, and CR) which have diverse characteristics in at least four aspects. First, the added three datasets were constructed from various repositories: pull-request and issue comments from GitHub [10]; API reviews from Stack Overflow [39]; and Gerrit code review comments from open-source projects [6]. Second, the sizes of the three datasets differ significantly. For example, the GitHub dataset is more than 20 times larger than the mobile app reviews. Third, among the three datasets, the GitHub dataset is balanced in the number of positive, neutral, and negative text units, while the other two are imbalanced. Fourth, different from the other two, the code review comment dataset only has two sentiment polarities: non-negative and negative.

For a given text unit, each approach will predict its sentiment polarity label. According to the number of sentiment polarities in a dataset, we formulate the problem as a binary or ternary text classification task. Specifically, the classification tasks for the datasets Jira and CR correspond to binary classification tasks. Both datasets have two polarity labels: positive and negative for Jira; negative and non-negative for CR. For the other four datasets, the classification problems are formulated as ternary-class classification tasks as they have three polarity labels, i.e., positive, neutral, and negative.

### *B. Implementations*

**SA4SE Tools:** For *Stanford CoreNLP*[4], we used its Python wrapper[5]. Given a sentence, Stanford CoreNLP returns the sentiment polarity with its corresponding sentiment value (*Very negative*=0, *Negative*=1, *Neutral*=2, *Positive*=3, *Very positive*=4). As Stanford CoreNLP gives a sentiment value and polarity to individual sentences, when a text unit in some datasets has more than one sentence, we calculate the average sentiment value of all sentences for the text unit. If the average sentiment value of a text unit is greater than 2, we assign it a *positive* polarity; if the value is less than 2, we assign it a *negative* polarity; otherwise, we assign it a *neutral* polarity (c.f. [9]).

For *SentiStrength*[6] and *SentiStrength-SE*[7], following Lin et al., we sum up the two sentiment strength scores returned by the tool to get the overall polarity for a sentence. If the total score is greater than 0, we assign a *positive* polarity to the whole text unit; if the total score is less than 0, we assign a *negative* polarity, and a *neutral* polarity is for the text unit that has a total score of 0. For the code review comment dataset, we need to distinguish between *non-negative* and *negative*; hence, if the overall score is less than 0, we assign *negative* to the text unit; otherwise, we assign *non-negative* to it.

*SentiCR*[8] only classifies two polarities, i.e., negative and non-negative. We re-train it on each dataset to classify three polarities, i.e., positive, neutral, and negative. We only changed the training dataset and kept all the parameters as default.

For *Senti4SD*, we use the classifier pre-trained on Stack Overflow dataset[9]. Senti4SD can classify three polarities, i.e., positive, neutral, and negative. As the code review comment dataset only has *negative* and *non-negative* polarities, we assign both the *positive* and *neutral* as *non-negative*.

**Transformer Models.** Many existing Transformer models are pre-trained for general domains. For instance, a combination of BooksCorpus [43] and English Wikipedia is used as all or part of the BERT and XLNet pre-training corpus. To build a sentiment classification model, we add a feed-forward dense layer and softmax activation function on top of each model. A certain pre-trained model's parameters have been reused as a starting point. We feed our SE training data to a pre-trained Transformer model's tokenizer and get the required formatted data; Then, we use the formatted data to train the pre-trained model further to get a fine-tuned model. Finally, we test it on the held-out test data. As found in BERT paper [13],

[4]http://stanfordnlp.github.io/CoreNLP/
[5]https://github.com/smilli/py-corenlp/
[6]http://sentistrength.wlv.ac.uk/download.html
[7]https://laser.cs.uno.edu/Projects/Projects.html
[8]https://github.com/senticr/SentiCR
[9]https://github.com/collab-uniba/Senti4SD

TABLE III: Models

| Architecture | Used Model | Parameters | Layers | Hidden | Heads |
|---|---|---|---|---|---|
| BERT | bert-base-cased | 110M | 12 | 768 | 12 |
| RoBERTa | roberta-base | 125M | 12 | 768 | 12 |
| XLNet | xlnet-base-cased | 110M | 12 | 768 | 12 |
| ALBERT | albert-base-v1 | 11M | 12 | 768 | 12 |

the following values of hyper-parameter for BERT fine-tuning procedure work well across all tasks: (1) Batch size: 16, 32; (2) Number of epochs: 2, 3, 4; (3) Learning rate (Adam): 5e-5, 3e-5, 2e-5. For all these models, we run them in 4 epochs with a batch size of 16. Moreover, we set the learning rate to 2e-5. We used AdamW optimizer.

Table III lists the models with their names in the Huggingface Transformers library [44] and default configurations.

### C. Evaluation Metrics

Following the previous work [8], we report the precision, recall, and F1-score of each approach for each polarity. We also report the macro- and micro-averaged metrics to show overall multi-classification performances. The formula to calculate $P$ (precision), $R$ (recall) and $F1$ (F1-score) are as follows: $P = \frac{TP}{TP+FP}$,$R = \frac{TP}{TP+FN}$, and $F1 = 2 \cdot \frac{P \cdot R}{P+R}$. $TP$ refers to the number of true positives (text units correctly classified as positive), $FP$ refers to the number of false positives (text units mistakenly classified as positive), and $FN$ refers to false negatives (text units mistakenly classified as negative).

The macro-averaged metric regards the measurement of each sentiment class equally. It takes precision, recall, and F1-score of each class and then averages them. The micro-averaged metric calculates measurement over all data points in all classes, and tends to be mainly influenced by the performance of the majority class [8]. The formulas for macro- and micro-averaged precision (P) are shown below:

$$P_{macro} = \frac{\sum_{i=1}^{k} P_i}{k} \quad (1)$$

$$P_{micro} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} TP_i + \sum_{i=1}^{k} FP_i} \quad (2)$$

$P_{macro}$ and $P_{micro}$ represent macro- and micro-averaged precision respectively. $P_i$, $TP_i$ and $FP_i$ represent the precision, number of true positives, number of false positives for the $i$th class respectively. $k$ denotes the number of sentiment polarity classes. We can calculate macro- and micro-averaged recall and F1, denoted as $R_{macro}$, $R_{micro}$, $F1_{macro}$, $F1_{micro}$, similarly. We consider a model is better than another only when it achieves higher values of both $F1_{macro}$ and $F1_{micro}$.

### D. Experimental Setting

Following Novielli et al. [8], we split each dataset into a training set (70%) and a test set (30%). Since SentiCR is originally designed for binary classification, we re-train it using the training set and test it on the test set for three classes. For Senti4SD, SentiStrength, and SentiStrength-SE, they do not need re-training. Concerning the four pre-trained Transformer models, we fine-tune them with the training set and then evaluate them on the test set.

## VI. Evaluation

In this section, we report the performance of the nine sentiment analysis approaches on the six datasets described in Section V-A. For each dataset, we highlight the best performance in terms of the two main metrics (i.e., macro- and micro-averaged F1-scores) in **bold**. We answer the research questions based on the experimental results as follows.

### A. RQ1: How accurate are Transformer models as compared to existing SA4SE tools?

To answer RQ1, we compare all the nine approaches in both the PRIOR and PTM groups. Tables IV and V present the performance of the nine approaches on the six datasets.

**API and SO Datasets:** Similar to the SO dataset, the API dataset is constructed from Stack Overflow posts. Thus, we look at the results of both datasets together. In terms of both macro- and micro-averaged F1, the approaches in PTM group outperform those in the PRIOR group. For the API dataset, The best performing PTM approach (ALBERT) can achieve macro- and micro-averaged F1-scores of 0.82 and 0.89, respectively. On the other hand, the best performing PRIOR approach (SentiCR) can only achieve macro- and micro-averaged F1-scores of 0.66 and 0.82, respectively. We observe a similar finding for the SO dataset.

**App Dataset:** We find that all the approaches perform relatively poorly on the App dataset. One potential reason is that this dataset is highly imbalanced and quite small; there are only a few text units with a neutral sentiment. Due to this limited number of text units for training, the approaches that have been trained on this dataset (i.e., all PTM approaches and SentiCR) have worse performance than the lexicon-based approaches (i.e., SentiStrength and SentiStrength-SE) for the neutral sentiment. Still, overall, we observe that approaches in the PTM group outperform those in the PRIOR group.

**GitHub Dataset:** Among all the six datasets, GitHub is the largest and most balanced one. The four approaches from the PTM group achieved similar performance: BERT, RoBERTa, and XLNet produce the same macro- and micro-averaged F1-scores; ALBERT performs slightly worse, 0.03 lower than the other three approaches. Their performance is better than that for approaches in the PRIOR group. In the PRIOR group, SentiCR is the best performer, with SentiStrength-SE being a close second. The other three approaches produce substantially lower macro- and micro-averaged F1-scores. Stanford CoreNLP has the worst performance, which shows that Stanford CoreNLP has poor generalization of SE data across different repositories.

**Jira Dataset:** For the Jira dataset, we found that all the PTM approaches perform well with high macro- and micro-averaged F1-scores ($\geq$ 0.96). However, in the PRIOR group, only SentiCR achieves macro- and micro-averaged F1-scores greater than 0.90. The other approaches in the PRIOR group have

TABLE IV: Results for API, SO, App, and GitHub Datasets

| Dataset | Approach | Positive | | | Neutral | | | Negative | | | Macro-avg | | | Micro-avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| API | **Stanford CoreNLP** | 0.47 | 0.41 | 0.44 | 0.85 | 0.60 | 0.71 | 0.22 | 0.66 | 0.33 | 0.51 | 0.56 | 0.49 | 0.57 | 0.57 | 0.57 |
| | **SentiStrength** | 0.44 | 0.45 | 0.45 | 0.81 | 0.77 | 0.79 | 0.44 | 0.45 | 0.45 | 0.55 | 0.57 | 0.56 | 0.68 | 0.68 | 0.68 |
| | **SentiStrength-SE** | 0.59 | 0.33 | 0.42 | 0.77 | 0.91 | 0.83 | 0.47 | 0.26 | 0.33 | 0.61 | 0.50 | 0.53 | 0.73 | 0.73 | 0.73 |
| | **SentiCR** | 0.85 | 0.52 | 0.65 | 0.82 | 0.98 | 0.89 | 0.81 | 0.31 | 0.45 | 0.83 | 0.61 | **0.66** | 0.82 | 0.82 | **0.82** |
| | **Senti4SD** | 0.56 | 0.33 | 0.41 | 0.76 | 0.93 | 0.84 | 0.44 | 0.10 | 0.17 | 0.59 | 0.45 | 0.47 | 0.73 | 0.73 | 0.73 |
| | **BERT** | 0.85 | 0.72 | 0.78 | 0.92 | 0.95 | 0.93 | 0.73 | 0.73 | 0.73 | 0.83 | 0.80 | 0.81 | 0.89 | 0.89 | **0.89** |
| | **RoBERTa** | 0.78 | 0.79 | 0.78 | 0.93 | 0.93 | 0.93 | 0.72 | 0.70 | 0.71 | 0.81 | 0.81 | 0.81 | 0.88 | 0.88 | 0.88 |
| | **XLNet** | 0.75 | 0.75 | 0.75 | 0.91 | 0.91 | 0.91 | 0.63 | 0.59 | 0.61 | 0.76 | 0.75 | 0.76 | 0.85 | 0.85 | 0.85 |
| | **ALBERT** | 0.88 | 0.77 | 0.82 | 0.92 | 0.96 | 0.94 | 0.71 | 0.68 | 0.70 | 0.84 | 0.80 | **0.82** | 0.89 | 0.89 | **0.89** |
| SO | **Stanford CoreNLP** | 0.23 | 0.42 | 0.30 | 0.92 | 0.69 | 0.79 | 0.34 | 0.82 | 0.48 | 0.50 | 0.64 | 0.52 | 0.68 | 0.68 | 0.68 |
| | **SentiStrength** | 0.25 | 0.42 | 0.32 | 0.89 | 0.80 | 0.84 | 0.40 | 0.52 | 0.46 | 0.52 | 0.58 | 0.54 | 0.74 | 0.74 | 0.74 |
| | **SentiStrength-SE** | 0.31 | 0.13 | 0.19 | 0.83 | 0.94 | 0.89 | 0.44 | 0.18 | 0.26 | 0.53 | 0.42 | 0.44 | 0.80 | 0.80 | 0.80 |
| | **SentiCR** | 0.48 | 0.32 | 0.38 | 0.90 | 0.90 | 0.90 | 0.45 | 0.55 | 0.49 | 0.61 | 0.59 | **0.59** | 0.82 | 0.82 | 0.82 |
| | **Senti4SD** | 0.50 | 0.34 | 0.41 | 0.85 | 0.96 | 0.90 | 0.75 | 0.14 | 0.23 | 0.70 | 0.48 | 0.51 | 0.83 | 0.83 | **0.83** |
| | **BERT** | 0.65 | 0.63 | 0.64 | 0.94 | 0.95 | 0.94 | 0.73 | 0.68 | 0.71 | 0.77 | 0.75 | 0.76 | 0.90 | 0.90 | **0.90** |
| | **RoBERTa** | 0.57 | 0.76 | 0.65 | 0.96 | 0.92 | 0.94 | 0.78 | 0.82 | 0.80 | 0.77 | 0.83 | **0.80** | 0.90 | 0.90 | **0.90** |
| | **XLNet** | 0.50 | 0.76 | 0.60 | 0.96 | 0.90 | 0.93 | 0.74 | 0.84 | 0.79 | 0.73 | 0.83 | 0.77 | 0.88 | 0.88 | 0.88 |
| | **ALBERT** | 0.71 | 0.32 | 0.44 | 0.90 | 0.95 | 0.92 | 0.61 | 0.61 | 0.61 | 0.74 | 0.63 | 0.66 | 0.86 | 0.86 | 0.86 |
| App | **Stanford CoreNLP** | 0.77 | 0.68 | 0.72 | 0.14 | 0.43 | 0.21 | 0.69 | 0.54 | 0.61 | 0.53 | 0.55 | 0.51 | 0.61 | 0.61 | 0.61 |
| | **SentiStrength** | 0.75 | 0.90 | 0.82 | 0.12 | 0.29 | 0.17 | 0.73 | 0.30 | 0.42 | 0.53 | 0.49 | 0.47 | 0.64 | 0.64 | 0.64 |
| | **SentiStrength-SE** | 0.73 | 0.81 | 0.77 | 0.15 | 0.57 | 0.24 | 0.91 | 0.27 | 0.42 | 0.60 | 0.55 | 0.48 | 0.60 | 0.60 | 0.60 |
| | **SentiCR** | 0.86 | 0.83 | 0.84 | 0.00 | 0.00 | 0.00 | 0.68 | 0.81 | 0.74 | 0.51 | 0.55 | **0.53** | 0.77 | 0.77 | **0.77** |
| | **Senti4SD** | 0.72 | 0.85 | 0.78 | 0.12 | 0.29 | 0.17 | 0.65 | 0.30 | 0.41 | 0.50 | 0.48 | 0.45 | 0.61 | 0.61 | 0.61 |
| | **BERT** | 0.86 | 0.95 | 0.90 | 0.00 | 0.00 | 0.00 | 0.87 | 0.89 | 0.88 | 0.58 | 0.61 | 0.59 | 0.86 | 0.86 | 0.86 |
| | **RoBERTa** | 0.95 | 0.92 | 0.93 | 0.00 | 0.00 | 0.00 | 0.84 | 1.00 | 0.91 | 0.60 | 0.64 | **0.61** | 0.88 | 0.88 | **0.88** |
| | **XLNet** | 0.87 | 0.98 | 0.92 | 0.00 | 0.00 | 0.00 | 0.86 | 0.81 | 0.83 | 0.57 | 0.60 | 0.58 | 0.85 | 0.85 | 0.85 |
| | **ALBERT** | 0.91 | 0.86 | 0.89 | 0.00 | 0.00 | 0.00 | 0.72 | 0.92 | 0.81 | 0.54 | 0.59 | 0.57 | 0.83 | 0.83 | 0.83 |
| GitHub | **Stanford CoreNLP** | 0.61 | 0.36 | 0.45 | 0.44 | 0.40 | 0.42 | 0.40 | 0.61 | 0.48 | 0.48 | 0.46 | 0.45 | 0.45 | 0.45 | 0.45 |
| | **SentiStrength** | 0.65 | 0.66 | 0.66 | 0.60 | 0.58 | 0.59 | 0.63 | 0.66 | 0.65 | 0.63 | 0.63 | 0.63 | 0.63 | 0.63 | 0.63 |
| | **SentiStrength-SE** | 0.87 | 0.85 | 0.86 | 0.77 | 0.86 | 0.81 | 0.82 | 0.71 | 0.76 | 0.82 | 0.81 | 0.81 | 0.81 | 0.81 | 0.81 |
| | **SentiCR** | 0.88 | 0.86 | 0.87 | 0.78 | 0.91 | 0.84 | 0.86 | 0.68 | 0.76 | 0.84 | 0.82 | **0.82** | 0.83 | 0.83 | **0.83** |
| | **Senti4SD** | 0.79 | 0.84 | 0.82 | 0.69 | 0.86 | 0.76 | 0.82 | 0.47 | 0.60 | 0.77 | 0.73 | 0.73 | 0.74 | 0.74 | 0.74 |
| | **BERT** | 0.92 | 0.95 | 0.93 | 0.90 | 0.92 | 0.91 | 0.93 | 0.87 | 0.90 | 0.92 | 0.91 | **0.92** | 0.92 | 0.92 | **0.92** |
| | **RoBERTa** | 0.93 | 0.96 | 0.94 | 0.91 | 0.92 | 0.92 | 0.93 | 0.89 | 0.91 | 0.93 | 0.92 | **0.92** | 0.92 | 0.92 | **0.92** |
| | **XLNet** | 0.90 | 0.97 | 0.94 | 0.94 | 0.89 | 0.91 | 0.91 | 0.92 | 0.91 | 0.92 | 0.93 | **0.92** | 0.92 | 0.92 | **0.92** |
| | **ALBERT** | 0.91 | 0.93 | 0.92 | 0.85 | 0.94 | 0.89 | 0.94 | 0.78 | 0.85 | 0.90 | 0.88 | 0.89 | 0.89 | 0.89 | 0.89 |

macro-averaged F1-scores lower than 0.6, and micro-averaged F1-scores lower than 0.79. Two noteworthy facts are that SentiStrength outperforms SentiStrength-SE by 20% in terms of the micro-averaged F1-score. Also, Stanford CoreNLP outperforms Senti4SD by 31.8%. This shows that SE-specific sentiment analysis tools do *not* always outperform general-purpose ones in SE datasets. The performance of different approaches from the PRIOR group gives us another insight: SentiStrength and SentiStrength-SE are both lexicon-based and do not need training. They outperform Stanford CoreNLP and Senti4SD, which have been trained in other datasets. SentiCR, which has been re-trained in this Jira dataset, achieves the best result in the PRIOR group. This highlights that the lexicon-based approaches may be better than supervised ones (if training is not done on a suitable dataset).

**CR Dataset:** For the dataset CR, all the approaches perform better in detecting non-negative polarity than negative polarity, with each PTM approach outperforming all PRIOR approaches. Also, all the approaches trained on the CR dataset, including all PTM approaches and SentiCR outperforms the other four non-CR specific tools (i.e., their training and construction did not involve CR-datasets).

**Overall:** We found that the best and worst-performing approaches differ for different datasets. Also, no approach can achieve the best performance on all datasets. For example, RoBERTa achieves the highest micro-averaged F1-score while SentiStrength-SE has the lowest score on the App dataset. On API dataset, BERT and ALBERT achieve the highest micro-averaged F1-score, while Stanford CoreNLP has the lowest score. Also, the performance gap between the best and worst performance on different datasets varies. The difference between micro-averaged F1-scores ranges from 32.4% (on SO) to 122.7% (on Jira). For macro-averaged F1-scores, the difference is from 35.6% (on App) to 139% (on Jira).

TABLE V: Results for Jira and CR Datasets

| Dataset | Approach | Positive | | | Negative | | | Macro-avg | | | Micro-avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| Jira | **Stanford CoreNLP** | 0.83 | 0.50 | 0.62 | 0.96 | 0.63 | 0.76 | 0.60 | 0.38 | 0.46 | 0.58 | 0.58 | 0.58 |
| | **SentiStrength** | 0.95 | 0.91 | 0.93 | 0.99 | 0.72 | 0.83 | 0.65 | 0.54 | 0.59 | 0.78 | 0.78 | 0.78 |
| | **SentiStrength-SE** | 0.98 | 0.85 | 0.91 | 0.99 | 0.54 | 0.70 | 0.66 | 0.46 | 0.54 | 0.65 | 0.65 | 0.65 |
| | **SentiCR** | 0.96 | 0.81 | 0.88 | 0.90 | 0.98 | 0.94 | 0.93 | 0.89 | **0.91** | 0.92 | 0.92 | **0.92** |
| | **Senti4SD** | 0.90 | 0.86 | 0.88 | 1.00 | 0.21 | 0.34 | 0.63 | 0.35 | 0.41 | 0.44 | 0.44 | 0.44 |
| | **BERT** | 0.99 | 0.96 | 0.97 | 0.98 | 0.99 | 0.99 | 0.98 | 0.98 | **0.98** | 0.98 | 0.98 | **0.98** |
| | **RoBERTa** | 0.98 | 0.96 | 0.97 | 0.98 | 0.99 | 0.98 | 0.98 | 0.97 | **0.98** | 0.97 | 0.97 | 0.97 |
| | **XLNet** | 0.98 | 0.96 | 0.97 | 0.98 | 0.99 | 0.98 | 0.98 | 0.97 | **0.98** | 0.98 | 0.98 | **0.98** |
| | **ALBERT** | 0.97 | 0.94 | 0.95 | 0.97 | 0.98 | 0.98 | 0.97 | 0.96 | 0.96 | 0.97 | 0.97 | 0.97 |
| **Dataset** | **Approach** | **Non-negative** | | | **Negative** | | | **Macro-avg** | | | **Micro-avg** | | |
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| CR | **Stanford CoreNLP** | 0.91 | 0.55 | 0.69 | 0.37 | 0.83 | 0.51 | 0.64 | 0.69 | 0.60 | 0.62 | 0.62 | 0.62 |
| | **SentiStrength** | 0.81 | 0.82 | 0.82 | 0.41 | 0.40 | 0.41 | 0.61 | 0.61 | 0.61 | 0.72 | 0.72 | 0.72 |
| | **SentiStrength-SE** | 0.80 | 0.94 | 0.86 | 0.57 | 0.25 | 0.35 | 0.68 | 0.59 | 0.60 | 0.77 | 0.77 | 0.77 |
| | **SentiCR** | 0.87 | 0.83 | 0.85 | 0.54 | 0.62 | 0.58 | 0.71 | 0.73 | **0.72** | 0.78 | 0.78 | **0.78** |
| | **Senti4SD** | 0.78 | 0.97 | 0.86 | 0.60 | 0.16 | 0.25 | 0.69 | 0.56 | 0.56 | 0.77 | 0.77 | 0.77 |
| | **BERT** | 0.94 | 0.87 | 0.90 | 0.67 | 0.83 | 0.74 | 0.80 | 0.85 | 0.82 | 0.86 | 0.86 | 0.86 |
| | **RoBERTa** | 0.92 | 0.93 | 0.92 | 0.76 | 0.74 | 0.75 | 0.84 | 0.83 | **0.84** | 0.88 | 0.88 | **0.88** |
| | **XLNet** | 0.87 | 0.95 | 0.91 | 0.78 | 0.54 | 0.64 | 0.82 | 0.75 | 0.77 | 0.85 | 0.85 | 0.85 |
| | **ALBERT** | 0.90 | 0.84 | 0.87 | 0.59 | 0.72 | 0.65 | 0.75 | 0.78 | 0.76 | 0.81 | 0.81 | 0.81 |

TABLE VI: Comparison between the Best Performers in the PRIOR and PTM Groups

| Metric | Group | API | SO | App | GitHub | Jira | CR |
|---|---|---|---|---|---|---|---|
| **Macro-avg F1** | Best **PRIOR** | 0.66 | 0.59 | 0.53 | 0.82 | 0.91 | 0.72 |
| | Best **PTM** | 0.82 | 0.80 | 0.61 | 0.92 | 0.98 | 0.84 |
| | Improvement | 24.2% | 35.6% | 15.1% | 12.2% | 7.7% | 16.7% |
| **Micro-avg F1** | Best **PRIOR** | 0.82 | 0.83 | 0.77 | 0.83 | 0.92 | 0.78 |
| | Best **PTM** | 0.89 | 0.90 | 0.88 | 0.92 | 0.98 | 0.88 |
| | Improvement | 8.5% | 8.4% | 14.3% | 10.8% | 6.5% | 12.8% |

Among all the approaches in the PRIOR group, we found that SentiCR achieves the best performance on five out of six datasets except SO. Also, Stanford CoreNLP performs the worst on five out of six datasets except on Jira. Among the approaches in the PTM group, we found that RoBERTa achieves the best performance on four datasets, i.e., App, GitHub, SO, and CR. ALBERT performs the worst on App, GitHub, SO, and CR, but it is the best performer on API.

We also observed that all the PTM approaches outperform PRIOR approaches up to 35.6% in terms of macro- and micro-averaged F1-scores (see Table VI). This demonstrates the effectiveness of the PTM approaches.

**RQ1 Main Findings:** The Transformer models outperform the prior SA4SE tools consistently across the six datasets, although the best performing model differs across different datasets. The improvements achieved by the Transformer models range from 6.5% to 35.6% in terms of macro- and micro-averaged F1-scores.

### *B. RQ2: How efficient are Transformer models as compared to existing SA4SE tools?*

The time efficiency of SA4SE approaches can be a concern in practice. Thus, we report the training (fine-tuning, for PTM approaches) and prediction time of all the approaches. Prediction time covers the time from processing the data to output the predicted label. Here, we provide a manual estimation of the exact prediction time of SentiStrength and SentiStrength-SE as they use a graphical user interface.

We run all the approaches on a desktop computer with Nvidia GeForce RTX 2080 Ti and Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz. The PTM group runs with both GPU and CPU, which the PRIOR group only uses CPU. All approaches, except for SentiStrength, are running with Ubuntu 18.04.4 LTS. SentiStrength runs on a Windows 10 virtual machine on the Ubuntu system, because only its .exe is available online.

In Table VII, we report: the training (fine-tuning) time (in seconds) for each training set; and the prediction time (in seconds) on each test set. For approaches in the PTM group, in terms of both fine-tuning and prediction time, XLNet takes

TABLE VII: Training (or fine-tuning) and prediction time (seconds)

| Approach | API | | SO | | App | | GitHub | | Jira | | CR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Train** | **Pred** | **Train** | **Pred** | **Train** | **Pred** | **Train** | **Pred** | **Train** | **Pred** | **Train** | **Pred** |
| BERT | 212.61 | 8.17 | 68.18 | 2.69 | 15.80 | 0.63 | 328.79 | 13.01 | 43.39 | 1.64 | 73.65 | 3.06 |
| RoBERTa | 215.02 | 7.87 | 71.77 | 2.62 | 15.91 | 0.59 | 338.64 | 12.42 | 43.78 | 1.61 | 76.35 | 2.91 |
| XLNet | 375.28 | 18.40 | 126.15 | 6.14 | 28.01 | 1.39 | 590.44 | 29.04 | 26.82 | 3.77 | 154.89 | 9.22 |
| ALBERT | 199.76 | 7.95 | 66.74 | 2.64 | 14.91 | 0.61 | 315.45 | 12.55 | 40.81 | 1.62 | 72.10 | 2.91 |
| SentiCR | 74.85 | 1.79 | 5.31 | 0.51 | 0.96 | 0.17 | 137.32 | 3.44 | 0.72 | 0.33 | 3.80 | 0.86 |
| Senti4SD | - | 48.81 | - | 31.11 | - | 23.59 | - | 63.95 | - | 27.88 | - | 31.62 |
| Stanford CoreNLP | - | 283.39 | - | 28.63 | - | 11.29 | - | 418.65 | - | 11.89 | - | 280.59 |
| SentiStrength | - | <1 | - | <1 | - | <1 | - | <1 | - | <1 | - | <1 |
| SentiStrength-SE | - | 1.69 | - | <1 | - | <1 | - | 3.22 | - | <1 | - | <1 |

the most time (approximately double the time used by the other three approaches). For the approaches in the PRIOR group, in terms of prediction time, Stanford CoreNLP is the most expensive and SentiCR runs the fastest. Generally, the prediction time used by Transformer models is two times more than that of SentiStrength, SentiStrength-SE, and SentiCR. However, it is less than 50% of the time used by Senti4SD and Stanford CoreNLP.

**RQ2 Main Findings:** In general, training (fine tuning) is more expensive than prediction. The time cost for fine-tuning the Transformer models ranges from 15 seconds to 10 minutes, depending on the datasets used. In terms of prediction time, all approaches make predictions for up to hundreds of text units (documents) within seconds. The Transformer models cost less than 50% of Senti4SD and Stanford CoreNLP to make predictions, but cost two times more than the time needed by SentiCR, SentiStrength and SentiStrength-SE.

## VII. Discussion

This section presents the lessons learned from our experiments and discusses threats to validity.

### A. Lessons Learned

**Fine-tuning pre-trained Transformer-based models is promising for SA4SE**. Lin et al. [9] mentioned that no prior SA4SE tool is ready for real usage of identifying sentiment expressed in SE data yet. We get similar results when applying the same approach (i.e., Stanford CoreNLP) to Stack Overflow posts (i.e., SO dataset). On the other hand, we found that even the worst-performing Transformer model (i.e., ALBERT) achieves 0.66 in terms of macro-averaged F1-score, which outperforms Stanford CoreNLP by 27%. The micro-averaged F1-scores produced by the Transformer models range from 0.86 to 0.90. The promising effectiveness of Transformer-based approaches has also been observed on the other five datasets. Although there is no gold standard or concrete thresholds of various evaluation metrics to decide whether a SA4SE tool can be put into *real use*, our experiment results show that the Transformer-based approach is more ready than the existing techniques for sentiment analysis in SE. Thus, we encourage researchers to consider the simply fine-tuning pre-trained Transformer-based approaches as the baseline in future work. Moreover, we advocate inventing more advanced Transformer-based models to make SE-specific sentiment analysis tools more practical.

**Specific training (or fine-tuning) can boost performance**. The approaches can be divided into two groups based on whether they are trained (or fine-tuned) on specific datasets or not. We fine-tuned all the pre-trained Transformer models and trained SentiCR for each dataset. Based on our results, we found that all the approaches that have been trained (or fine-tuned) on SE datasets outperform those that have not been trained (or fine-tuned) across all the six datasets. Moreover, we find that Senti4SD, which is designed based on Stack Overflow data [26], performs the best for API and SO datasets. These indicate that a tool trained on the same data source can perform better for the same or similar data sources.

**Challenges in assigning sentiment labels**. Previous work [9] shows that even human raters have more than 18% disagreements on the same sentences as sentiment identification may be subjective. We also observed that it is hard to determine the sentiment labels of some sentences. For example, the sentence "*It's always sad to see a reference like that go, but it was probably a good move.*" is labeled as *negative*. However, part of it (i.e., "it was probably a good move") should be considered as positive. Thus, there may be a need to introduce additional labels, e.g., mixed sentiment, or to go more fine-grained (i.e., attaching sentiment labels to phrases instead of sentences). Customized solutions may boost performance further. We also found that no approach can always achieve the best performance on all six datasets. It indicates that customization of the technical design is also required in future work. It would be interesting to extend the current Transformer-based models to consider the specific properties of the SE datasets that we have.

**Composition of different solutions may boost performance further**. We find that some sentences can only be assigned correct sentiment labels by the (generally) under-performing SA4SE solution. To illustrate this, we conducted a brief error

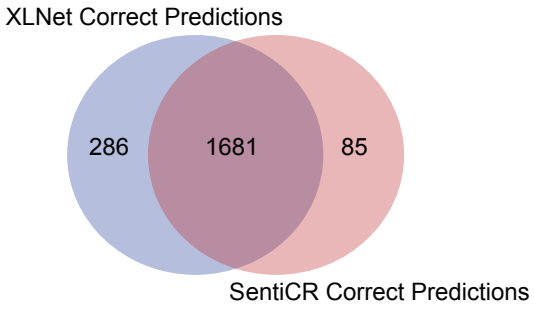


Fig. 1: Venn Diagram

TABLE VIII: Prediction Examples

| Sentence | Label | XLNet | SentiCR |
|---|---|---|---|
| *Yes, it would be really cool if you could update the wiki. But don't forget to say it will only work from version 1.4.0 forward!"* | Positive | Positive | Negative |
| *Thanks for your comments and tests! (Aptana is driving me nuts, I'm currently searching for another IDE)* | Positive | Negative | Positive |
| *Looks good. Mind if I add a CityHash implementation in here?* | Neutral | Positive | Neutral |
| *Strange indentation here* | Neutral | Negative | Neutral |
| *Pretty simple script for a TBC boss I say. I wonder who did it originally...* | Negative | Neutral | Negative |
| *If you mean #any_instance, you were better off not knowing. It's a nasty code smell.* | Negative | Negative | Neutral |

analysis on the largest GitHub dataset to help understand the different performances of different approaches. In total, we have 2,137 sentences in the test set. We focus on comparing the best performing approaches from the PRIOR and PTM groups, i.e., SentiCR and XLNet. Figure 1 depicts the number of the correct predictions produced by the two approaches. Among all the 2,137 sentences, XLNet and SentiCR correctly predict 1,967 (92%) and 1,766 (82%) sentences; among them, 1,681 (78%) are in common. From the Venn diagram, we find that although SentiCR performs worse (in general) than XLNet, for 85 sentences, it outperforms SentiCR. Table VIII shows some examples where one of the approaches fails, but the other is successful. Thus, by composing many different tools, we can boost the performance further. As future work, we want to explore the possibility of combining all the existing solutions for a higher accuracy. Table VIII shows some examples of sentences from the GitHub dataset with the prediction results produced by XLNet and SentiCR. For each example, one of the two approaches makes a wrong prediction.

### B. Threats to Validity

One potential threat to internal validity relates to errors that we may have made in our experiments. We have also released a replication package[10] for others to check and extend.

[10]https://github.com/soarsmu/SA4SE

Threats to external validity are related to the generalizability of our research and experiments. We consider six sentiment classification datasets, larger than the datasets considered in a closely related work [9]. These six datasets are diverse from several aspects, e.g., scale, type of software artifacts, class distribution, etc. Our experimental setting follows Novielli et al. [8], for each dataset, uses 70% for training (or fine-tuning) and 30% for testing. In the future, we plan to employ k-fold cross validation, which is a more rigorous evaluation method.

Threats to construct validity are related to the suitability of our evaluation metrics and quality of manually-labeled datasets that we use. Precision, recall, and F1-score are widely used to evaluate SA4SE solutions [8], [10], [39]. As we make use of the publicly available datasets used in prior works, we inherit the latter threat from the prior studies.

## VIII. Conclusion and Future Work

In this work, we have conducted an extensive comparative study on the performance of prior SA4SE tools and pre-trained Transformer models. We are the first to investigate the effectiveness of various pre-trained Transformer-based models for the SA4SE task. Our comparative study includes six datasets: GitHub pull-request and commit comments, API reviews from Stack Overflow, mobile app reviews, Stack Overflow posts, Jira issue comments, and code review comments. Our experimental results reveal that the best performing fine-tuned Transformer model outperforms the best performing prior SA4SE tool by 6.5% to 35.6% in terms of the macro- and micro-averaged F1-scores.

Overall, Transformer-based approaches are more ready to be applied in the real world for sentiment analysis of SE data than the existing SA4SE tools. In the future, we are interested in a few directions: (1) applying Transformer-based SA4SE models for further downstream tasks (e.g., API recommendation), and (2) investigating the effectiveness of Transformer-based models for other SE tasks.

## Acknowledgment

We thank the anonymous reviewers for their insightful comments. This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-1-193).

## References

[1] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[2] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: an empirical study," in *the 11th Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 352–355.

[3] R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 531–535.

[4] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *38th International Conference on Software Engineering (ICSE)*, 2016, pp. 14–24.

[5] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 203–214.

[6] 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.

[7] M. R. Islam and M. F. Zibran, "A comparison of software engineering domain specific sentiment analysis tools," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 487–491.

[8] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 364–375.

[9] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 94–104.

[10] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile, "Can we use se-specific sentiment analysis tools in a cross-platform setting?" *arXiv preprint arXiv:2004.00300*, 2020.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[12] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *arXiv preprint arXiv:2003.08271*, 2020.

[13] 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.

[14] 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.

[15] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5753–5763.

[16] 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.

[17] M. De Choudhury and S. Counts, "Understanding affect in the workplace via social media," in *Proceedings of the 2013 conference on Computer supported cooperative work*, 2013, pp. 303–316.

[18] S. Romano, D. Fucci, M. T. Baldassarre, D. Caivano, and G. Scanniello, "An empirical assessment on affective reactions of novice developers when applying test-driven development," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2019, pp. 3–19.

[19] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, "Anger and its direction in collaborative software development," in *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. IEEE, 2017, pp. 11–14.

[20] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[21] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the American society for information science and technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[22] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[23] F. Calefato, F. Lanubile, and N. Novielli, "Emotxt: a toolkit for emotion recognition from text," in *2017 seventh international conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 2017, pp. 79–80.

[24] M. R. Islam and M. F. Zibran, "Deva: sensing emotions in the valence arousal space in software engineering text," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1536–1543.

[25] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth international AAAI conference on weblogs and social media*, 2014.

[26] 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.

[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[28] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[29] A. Cimasa, A. Corazza, C. Coviello, and c, "Word embeddings for comment coherence," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 244–251.

[30] A. Sulistya, G. A. A. Prana, A. Sharma, D. Lo, and C. Treude, "Sieve: Helping developers sift wheat from chaff via cross-platform analysis," *Empirical Software Engineering*, vol. 25, no. 1, pp. 996–1030, 2020.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[33] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[34] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[35] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[36] R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur, "Opinion mining and sentiment analysis," in *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 452–455.

[37] M. R. Islam and M. F. Zibran, "Towards understanding and exploiting developers' emotional variations in software engineering," in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 2016, pp. 185–192.

[38] M. Pennacchiotti and A.-M. Popescu, "Democrats, republicans and starbucks afficionados: user classification in twitter," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 430–438.

[39] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *IEEE TSE*, 2019.

[40] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: sentiment analysis of security discussions on github," in *the 11th working conference on mining software repositories (MSR)*, 2014, pp. 348–351.

[41] 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.

[42] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 303–313.

[43] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 19–27.

[44] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.