An Empirical Study of Gradient-based Explainability Techniques for Self-admitted Technical Debt Detection

Guoqiang Zhuang[1,2], Yubin Qu[1,2*], Long Li[1], Xianzhen Dou[2], Mengao Li[3]

[1] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, China
[2] School of Information Engineering, Jiangsu College of Engineering and Technology, China
[3] Systems Engineering Research Institute, China State Shipbuilding Corporation, China
zgq@jcet.edu.cn, quyubin@hotmail.com, lilong@guet.edu.cn, douxianzhen@163.com, 18632154@qq.com

Abstract

Self-Admitted Technical Debt (SATD) is an intentionally introduced software code comment describing potential defects or other technical debt. Currently, deep learning is widely used in fields such as Natural Language Processing. Deep learning can be used for SATD detection, but there is a class imbalance problem and a large number of easily classified SATD instances that may potentially affect the loss value. As a result, we proposed a weighted focal loss function based on particle swarm to address the problem. Meanwhile, there is no empirical research based on local explanations for SATD detection. We have investigated local interpretation models such as Saliency Maps, Integrated Gradients, which are currently widely used in deep learning, and conducted empirical research for shared data sets. The research results show that our proposed weighted focal loss function can achieve the best performance for SATD detection; our model achieves 12.27%, 5.97%, and 5.62% improvement in Precision, Recall, and AUC compared to the baseline model, respectively; Local explanation models, including Saliency Maps and Integrated Gradients can cover nearly half of the manually labeled paradigms; these two interpretation models can also discover potential new paradigms.

Keywords: Self-Admitted Technical Debt, Deep learning, Explainability, Class imbalance, Focal loss

1 Introduction

Self-Admitted Technical Debt (SATD) is an intentionally introduced software code comment describing potential defects or other technical debt [1]. At present, software tests for software defects can be arranged by detecting SATD [2]. For example, the software comment, "*TODO (@author fdietz): show error dialog here*", shows that there is a potential bug. The lines of code related to this code comment should be the first to be paid attention to and test arranged. In the software development process, there is always a contradiction between rapid software development and the limited project budget and project delivery time. Technical debt is used to describe the sub-optimal solution in the software development process to meet the short-term needs of the project . Generally speaking, these technical debts allow the software development process to move forward quickly, at the cost of leaving a security risk for the longterm maintenance of the project. For technical debt, they should be paid rapidly in time. Otherwise, it will have negative effects and corresponding software errors. For example, architectural decay belongs to technical debt. Architectural decay incurs inadvertently. However, we focus more on deliberately introduced technical debt, called Selfadmitted technical debt (SATD). SATD was first proposed by Potdar and Shihab. Compared with traditional technical debt using source code analysis, SATD detection is more lightweight by only using source code comments. The SATD should be paid. Otherwise, an artificial software bug will be introduced into the software project. Therefore, in software engineering, detecting SATD is necessary for ensuring software quality.A binary classification technique is used to predict whether a comment shows SATD. There are many machine learning techniques, such as pattern-based SATD detection [3], traditional text-mining-based SATD classification [4], and Convolutional Neural Network (CNN) based approach [2].

The CNN-based approach achieved the best performance for within-project and cross-project in terms of F1-score, Recall, Precision metrics. Some key issues, including variant term frequency, key words project uniqueness, variable length of software comments, and semantic variation can be handled using CNN correctly [2, 5]. However, the CNN model is a black box. Why this CNN model can achieve the best performance is valuable for software engineering practitioners. Appropriate explanations can assist front-line developers to better locate potential software defects. But so far, there is no empirical research on the explainability of SATD models. Why do we need Explainable ML? Loan issuers are required by law to explain their models. The medical diagnosis model is responsible for human life. There should be no BlackBox-based diagnosis. If a self-driving car suddenly acts abnormally, we need to explain why. Explainable ML is important for decisions. The European Union's General Data Protection Regulation states that any algorithm with decision-making capability requires to be explained [6]. For SATD detection, we not only need a high-precision classification model, but we also need to be able to quickly locate problems based on the classification model. The SATD comments will be rapidly located using the CNN model and explained using gradient-based explainability techniques so that software testers can quickly locate potential defects and fix bugs.

The main contributions of this article are as follows:

*Corresponding Author: Yubin Qu; E-mail: quyubin@hotmail.com
DOI: 10.53106/160792642022052303021

(1) We proposed a weighted focal loss function based on particle swarm to address the class imbalance and the potential calculation error for a large number of easily classified instances when detecting SATD.

(2) Based on the gradient-based interpretability models, a small part of paradigms can be covered compared with 62 manually labeled paradigms;

(3) Based on the gradient-based interpretability models, there are almost the same paradigms that are discovered by different models;

(4) New paradigms are discovered based on gradient-based interpretability models.

The rest of the article is organized as follows. In Section 2, we illustrate SATD detection model using weighted Convolutional Neural Network and gradient-based explainability techniques. In Section 3, we then describe our experiment setup, including research questions, research methods, data collections, and commonly used evaluation metrics. In Section 4, we report and analyze our experimental results. In Section 5, we conclude this article and discuss key directions for future work.

2 Approach

The software development process requires the collaboration of different stakeholders (e.g., users, developers, and managers) and integrated development tools can provide intelligent support for software development, such as software defect prediction [7-8], SATD detection [2], bug location [9-10]. These development tools can not only provide intelligent decision-making for collaborative development but also should be able to provide a basis for decision-making. In recent years, some scholars have done interpretability research on software defect prediction to provide a detailed basis for decision-making [11].

In this section, we will describe our proposed deep learning model for SATD detection The deep learning flow for SATD detection is shown in Figure 1.

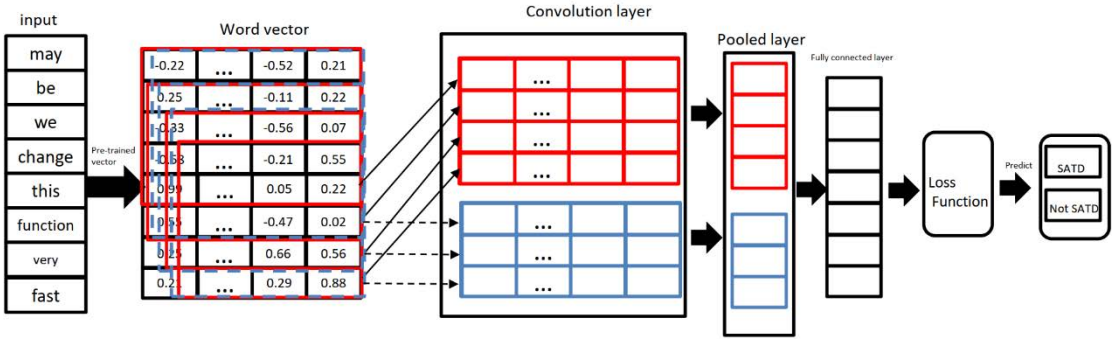


Figure 1. Deep learning flow of SATD detection

In this paper, we will describe Gradient-based Explainability Techniques for SATD detection. The Weighted TextCNN model achieves the best performance for SATD detection [2, 12]. Global explanation [13] and local explanation [14-18] can be used for the CNN model. The global interpretation model pays more attention to the overall characteristics of the input examples, while the local interpretation pays more attention to the impact of different input features on the classification results. For the SATD detection, we use a gradient-based local explanation model. For local explanation, we pay more attention to which component is critical for the CNN model. Below we present the formal definition of a gradient-based local explanation model. The explainability analysis process is shown in Figure 2.

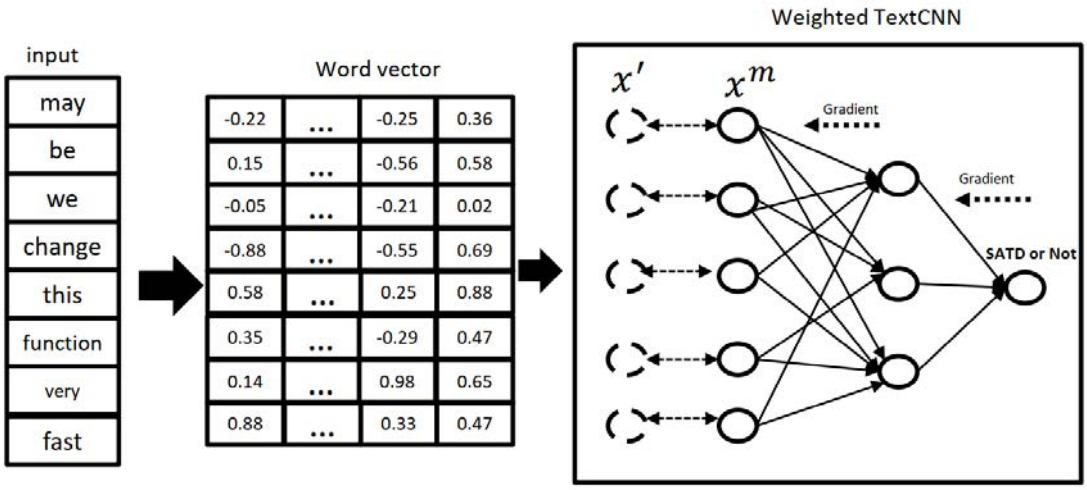


Figure 2. Explainability analysis process

2.1 SATD Detection Model using Weighted Convolutional Neural Network

In this section, we will describe the SATD detection model used in our experiment. The model takes the programmer's code comments as input. The words are inputted to the CNN using word-embedding technology, and the neural network is trained by the weighted focal loss function. Finally, the input code comments are classified as SATD or not.

First, we provide a formal definition of the SATD detection model.

Definition 2.1: a SATD detection model. A SATD detection model is a function:

$$(x_1, x_2, x_3 \dots \dots x_m) \to y \quad \textbf{(1)}$$

x is the software comment word sequences with the length of word sequences m and y is the outcome space (e.g., SATD or non-SATD). Typically training data $Data_{train}$ is used to train the CNN model and test data $Data_{test}$ is used to evaluate the CNN model.

2.1.1 The Architecture of Convolutional Neural Network

Our CNN is inspired by Ren [2]. This architecture is originated from TextCNN [12], which can handle sentence classification tasks using a shallow neural network. As shown in Figure 1, our deep learning framework is stacked by the following layers, including the word embedding layer of code comments, the convolutional network layer, the pooling layer, the fully connected layer, and the final loss function calculation layer.

First we will show the word embedding layer of code comments. The pre-training vector used in the framework is based on Global Vectors for Word Representation (Glove). It is a word representation tool based on count-based and overall statistics. During the training process, the output word vector dimension is 300 dimensions, the length of n-grams is set to 5, the CBOW (continuous bag of word) model is used in the training process, the low-frequency threshold is 5, and the number of iterations is 5. The output layer adopts the negative sampling method, that is, each time 5 labels other than the current label are selected as negative samples, they are added to the loss function as the probability of negative samples. In the word vector embedding layer, the code comment is converted into a word vector matrix, and the output value is used as the input of the convolutional neural layer.

Next, we will describe the convolutional layer. The deep semantic understanding of words is filtered through different convolution kernels. The semantic learning framework for SATD detection originated from the TextCNN framework proposed by Kim et al. [12]. The framework uses a shallow neural network to analyze the semantics of the text. The network structure is simple and the processing speed is faster. The code comments are *V*, and the word is represented as *W*, so each word comment can be represented as a sequence of words. These words are represented as a vector matrix. According to the different requirements, we set the size of each filter in the convolutional layer of TextCNN. Later we will describe the specific method of setting the hyperparameter of different filter sizes. The filter window represents the number of co-occurring words that the network architecture focuses on at the same time in TextCNN. This parameter is a hyperparameter and needs to be optimized. Generally speaking, the filter window size $\text{filter\_size} \in [1,2,3,4,5,6]$. In the English corpus, simple English word segmentation can be achieved by using spaces. The different sizes of multiple filters enable CNN to find different co-occurring words based on semantics. As shown in Figure 1, in the convolutional layer, the red output represents the semantic filtering result with a convolution kernel of 5, and the blue output represents the semantic filtering result with a convolution kernel of 6.

In the pooling layer, the maximum pooling operation is performed for the feature map obtained by each convolution kernel to complete feature extraction, achieve dimensionality reduction, and reduce potential overfitting. Finally, a fully connected linear classifier in the output layer is trained.

The loss function of our CNN framework will be described in the next section in detail.

2.1.2 Our Weighted Focal Loss Function

Assuming that an input comment is marked as SATD labeled as $y = 1$, we use cross entropy Loss to calculate the loss between the classification result *p* of the classification model and the true label *y*.

$$Loss(p, y) = \begin{cases} -\log(p), & y = 1 \\ 0, & y = 0 \end{cases} \quad \textbf{(2)}$$

However, this Loss function does not work well in our SATD detection model. Because the code comments with SATD will be much smaller than the code comments without SATD in actual work, this is a natural class imbalance problem [2]. In Ren's experiment, a weighted cross-entropy loss was proposed to their CNN. Assuming there are *n* SATD comments and *m* non-SATD comments, then they defined the weights α.

$$\alpha = \frac{m}{m+n} \quad \textbf{(3)}$$

The *Loss* can be rewritten in the following format with the tuning parameter α. The wrong classification of SATD instances is penalized more based on Equation (4).

$$Loss(p, y) = -\alpha * \log(p) \quad \textbf{(4)}$$

However, there is another problem. In our dataset, the highest percentage of SATD comments is only 5.57%. A large number of non-SATD examples have obvious characteristics, and the accumulation of their loss will most likely exceed the weighted rare SATD instances [19]. That is to say, although the adjustment factor α is used to solve the problem of insufficient punishment for the minority class, in the face of the majority class which is far more than the minority class, the adjustment factor is lacking to deal with easy-to-classify examples. We need to add more parameters to adjust the loss value of easy-to-classify examples. To deal with this problem, *FocalLoss* was proposed [19].

$$FocalLoss(p, y) = -(1-p)^{\gamma} * \log(p) \quad \textbf{(5)}$$

By combining with *FocalLoss*, the loss function (4) is transformed.

$$Loss(p, y) = -\alpha(1-p)^{\gamma} * \log(p) \quad \textbf{(6)}$$

In Lin's experiment,$\gamma \in [0,5]$. γ is a hyperparameter that was chosen using empirical study. In our experiment, we will describe the specific method of setting the hyperparameter γ.

From Equation (6), we can see that: (1) When a SATD instance is misclassified, $Loss(p, y)$ will be tuned by $\alpha(1-p)^{\gamma}$. The parameter $(1-p)^{\gamma}$ increases the loss contribution value to the samples that are not easy to classify in the SATD class. The parameter α increases the penalty for minor classification errors. (2) When a SATD instance is correctly classified, $Loss(p, y)$ will not be tuned by the parameter $(1-p)^{\gamma}$. In addition, the parameter α only increases the penalty for SATD instances.

2.1.3 Hyperparameter Optimization Method Based on Particle Swarm

The SATD detection model based on TextCNN uses a shallow neural network instead of a multi-layer deep neural network. The hyperparameter optimization process does not require special optimization of the structure of the neural network. The parameters to be optimized include the dimension of the vector, the learning rate, the size of the filter, the size of the filter window, etc. The word vector is 300 dimensions for Glove so the dimension of the word embedding vector is set to 300; The initial filter window size set in the frame is (1,2,3,4,5), different from the six sizes of filter window size (1,2,3,4,5,6) in the TextCNN network architecture of Ren et al [2]. For the optimization of parameters such as the number of network layers of the convolutional neural network, the genetic algorithm is more effective. The hyperparameter optimization method based on the particle swarm optimization algorithm is suitable for the hyperparameters of the convolutional neural network.

Set the initial value of the learning rate, the size of the filter, the size of the filter window, and use the particle swarm optimization algorithm to optimize the hyperparameters, which is based on the social sharing of information to find the best position of the particles. The training data set $Data_{train}$ is repeated ten times and ten-fold cross-validation is performed. The average value of the performance results on the test data sets is used as the result of the particle swarm optimization algorithm. Define a particle p in particle swarm optimization algorithm as:

$$p = (\gamma, \mathrm{lr}, \mathrm{filter\_size}, \mathrm{window\_size},) \quad \textbf{(7)}$$

The dimension of each particle is 4. Other main parameters in the particle swarm algorithm, including the number of particle swarms, inertia weights, stopping criteria, etc., are the same as the values in Bratton's experiments [20]. In the process of particle search, each particle finds a position that satisfies the best fitness value through information sharing and ends the algorithm. The hyperparameter optimization algorithm based on the particle swarm algorithm is shown in the following steps.

Step 1: Set the performance function in the training set of the convolutional neural network as the objective function of the particle swarm algorithm;

Step 2: Encode each parameter in Equation (7) to form a particle swarm;

Step 3: Set the main parameters in the particle swarm algorithm and limit the search space of each parameter;

Step 4: Run the algorithm until the performance criteria requirements are met;

Step 5: If the performance cannot meet the requirements, re-run step 3 of the algorithm.

2.2 Gradient-based Explainability Techniques

Below we introduce gradient-based local explanation models for SATD detection.

2.2.1 Saliency Maps

Zeiler et. al analyzed critical pixels for classification by using occlusion sensitive [13]. Karen et. al used Saliency Maps to visualize the image classification model [14]. Considering this sharp fluctuation, the direct gradient at any given pixel is not as meaningful as the local average gradient value. By adding random noise to the inputs many times, the transformed image is combined and averaged to achieve the effect of "introducing noise" to "eliminate noise" [15, 21]. For SATD detection, Saliency Maps is used to visualize the model. There are 4 key steps for instance explanation.

Firstly, the weighted TextCNN model is trained based on the training data $Data_{train}$. Word embedding is used for inputs and the class imbalance problem is resolved using the weighted focal loss function.

Secondly, the instance of interest is inputted. Loss is calculated using forward computing.

Thirdly, backpropagation is executed based on the calculated loss. For the SATD detection mapped function *f*, when the element x_m changes to $x_m + \Delta x$, the output changes from y to $y + \Delta y$.

Finally, the gradient for the output of the input's word embedding is gathered. The gradient is used to indicate the ratio of the change in y to the change in $(x_1, x_2, x_3 \ldots \ldots x_m)$.

$$\frac{\partial y}{\partial x_m} = \frac{\Delta y}{\Delta x} \quad \textbf{(8)}$$

2.2.2 Integrated Gradients

There may be gradient saturation so that gradient cannot always reflect importance. This is a limitation for gradient-based explainable techniques. By integrating the gradient along different paths, it is easy to calculate the contribution of the non-zero gradient in the unsaturated zone to the importance of decision-making [18, 22].

Compared with the previously used Saliency Maps algorithm, Integrated Gradients focuses on the gradient saturation phenomenon in a special area, where key features affect the classification model. When the key feature is in the saturated area, increasing the amount of change of the key feature does not bring about a significant change in the gradient value. The reason for the problem is that the key feature has entered the saturation interval, and the gradient cannot reflect the key feature. The Integrated Gradients algorithm can be represented as:

$$\emptyset^{IG}(f,x,x') = (x - x') \times \int_{\theta=0}^{1} \frac{\delta f(x'+\theta(x-x'))}{\delta x} d\theta \quad \textbf{(9)}$$

x' represents the baseline. For SATD detection, the baseline is *Zero vector* which is the value of *PAD*. $\emptyset^{IG}(f,x,x')$ is the Integrated Gradients value. When $\theta = 0$, $\emptyset^{IG}(f,x,x')$ represents the baseline. When $\theta = 1$, $\emptyset^{IG}(f,x,x')$ represents the input instance, for example, 'probably buggy code'. When $\theta \in (0,1)$, $\emptyset^{IG}(f,x,x')$ represents the integral value from x to x'.

Compared with DeepLift [17] and Layer-wise relevance propagation (LRP) [23] algorithm, Integrated Gradients can satisfy three important properties, such as sensitivity, completeness, and symmetry-preserving. There are 4 key steps for instance explanation.

Firstly, train the weighted TextCNN model and choose the baseline, *Zero vector*.

Secondly, calculate the predicted output of the instance using forward computing.

Thirdly, backpropagation is executed based on the predicted output. Based on (9), the integral gradient of each input feature is calculated by the linear interpolation method.

Finally, the gradient for the output of the input's word embedding is gathered. The gradient is used to indicate the importance for every feature.

3 Experimental Design

In this section, three research questions are proposed. Next, we describe the baseline paradigms and two gradient-based methods. Then, we discuss our selected datasets, the evaluation metrics, and the experimental environment.

3.1 Research Questions

We propose three research questions we investigate in our experiments to conduct the empirical study. The first question is to determine whether our proposed detection model can achieve the best performance. The other two research questions investigate different aspects of the gradient-based interpretability model for SATD detection.

(1) How effective is our weighted focal loss function for imbalance software comments compared with the cross entropy loss function?

(2) Can the gradient-based interpretability model cover 62 manually labeled paradigms?

(3) Can different interpretability models discover new SATD paradigms?

3.2 Research Methods

Two methods are compared in the experiment. The baseline paradigms are based on manually labeled data [1].

(1) Baseline Paradigms. There are 62 manually labeled paradigms from fore large open-source projects – namely Eclipse, Chromium OS, ArgoUML, and Apache httpd [1]. The statistics of manually labeled paradigms are shown in Table 1. From Table 1, we can see that 95.16% of paradigms contain words ranging from 1 to 5. In particular, paradigms with three words account for the highest proportion, 29.03%.

(2) Saliency Maps. As shown in Equation (2), the gradient for software comment is calculated.

(3) Integrated Gradients. As shown in Equation (3), the Integrated Gradients for software comment is calculated along the linear path.

Saliency Maps and Integrated Gradients are used for the weighted TextCNN model so that the gradient of inputs is gathered. These gradients can reflect the importance of different features. We first investigate whether these two methods can discover 62 manually labeled paradigms. Then, we investigate whether these can discover new paradigms.

Table 1. Statistics of manually labeled paradigms

Length of paradigms	Manually labeled paradigms	Count of paradigms
One word	hack; retarded; stupid; ugly; nuke; hacky; silly; kludge; fixme; barf; yuck; crap; inconsistency; kaboom;	14
Two words	take care; is problematic; give up; temporary solution; causes issue; temporary crutch; toss it; certainly buggy; bail out;	9
Three words	at a loss; remove this code; may cause problem; trial and error; this is wrong; cause for issue; some fatal error; this is bs; just abandon it; probably a bug; something bad happened; fix this crap; this is uncool; abandon all hope; prolly a bug; don't use this; something's gone wrong; workaround for bug;	18
Four words	there is a problem; something serious is wrong; get rid of this; hope everything will work; remove me before production; it doesn't work yet; this isn't quite right; this doesn't look right; this isn't very solid;	9
Five words	hang our heads in shame; this can be a mess; give up and go away; is this next line safe; is this line really safe; doubt that this would work; risk of this blowing up; you can be unhappy now; something bad is going on;	9
Six words	unknown why we ever experience this; treat this as a soft error;	2
Seven words	this is temporary and will go away;	1

3.3 Data Collection

To perform our research, we use open source datasets. These datasets are derived from ten open source projects, including ArgoUML, Columba, Hibernate, JMeter, Apache Ant, EMF, JEdit, JFreeChart, JRuby, and SQuirrel. These ten open source projects have different application areas, and the complexity of the projects is not the same. After the source codes of these open source projects have been collected, Eclipse plug-in, called JDeodorant [24], is used to perform data analysis on these source codes to obtain the start and end positions of the corresponding code comments and the type of code comments.

This data set was collected and labeled by Maldonado et al. . The process of data collection is shown three main steps.Firstly, an eclipse plug-in tool, named JDeodorant, was used to parse the source code and extract the comments information, including the line that each comment starts, finishes and the different type of comments. A total of 259,229 lines of code comments were obtained from these 10 open source projects, with an average of 25,923 lines per project. Only a small part of the code contains SATD, and manual labeling of these source codes will be time-consuming and labor-intensive.

Secondly, as only a small ratio of the source code comments was described as SATD in the raw dataset, five filtering heuristics are developed to identify comments with SATD and eliminate comments that are unlikely to be classified as SATD. After automatically cleaning, a large number of unrelated machine-generated comments were eliminated. The number of code comments that need to be manually annotated has become 62,566. The use of heuristic strategies greatly reduces the workload of manual labeling, and at the same time can effectively improve the classification accuracy of the machine learning model.

After cleaning the raw dataset, finally, each comment was labeled with SATD or non-SATD by Maldonado and Shihab using manually examining each comment. These manually labeled data sets may have personal biases of users, and inaccurate data sets will affect the subsequent training of machine learning models. Therefore, stratified sampling was performed on the full data set, and the stratified sampling results of Maldonado et al. showed that a confidence level of 99% and a confidence interval of 5% were achieved [3]. To further confirm the validity of this data set, another independent individual was invited to sample the data hierarchically and label the data set. Comparative experiments show that the hierarchical sampling results of different user groups have extremely high consistency, with Cohen's Kappa coefficient of 0.81. The experimental results of sampling and labeling of different user groups prove that the data set has strong statistical reliability.

3.4 Evaluation Metrics

To evaluate the classification performance of the SATD detection model we proposed, we define TP (True Positive) to represent the number of SATD instances that are predicted as SATD instances; we define TN(True Negative) to represent the number of non-SATD instances which are predicted as non-SATD instances; we define FP (False Positive) to represent the number of non-SATD instances which are predicted as SATD instances; we define FN(False Negative) to represent the number of SATD instances which are predicted as non-SATD instances.

The *Precision* indicator is used to represent the proportion of the correctly classified SATD instances among the instances classified as SATD.

$$Precision = \frac{TP}{TP+FP} \quad \textbf{(10)}$$

The *Recall* indicator is used to represent the proportion of the correctly classified SATD instances among the instances which are labeled as SATD.

$$Recall = \frac{TP}{TP+FN} \quad \textbf{(11)}$$

Recall also is called *TPR*. In contrast, The *FPR* indicator is used to represent the proportion of the correctly classified non-SATD instances among the instances which are labeled as non-SATD.

$$FPR = \frac{TN}{TN+FP} \quad \textbf{(12)}$$

For SATD detection, because of the class imbalance problem, *AUC* (Area Under Curve) is used to measure the performance of the CNN model. Given a binary classification model and its threshold, a coordinate point (X=*FPR*, Y=*TPR*) can be calculated from the true value and predicted value of all samples (positive/negative). When the value is closer to 1, it means that the classification performance of the model is better; if the value is close to 0.5, it means that the performance of the model is roughly the same as the random classification effect.

The evaluation metrics for the local explainability of the model are divided into two categories. One is to evaluate the result of interpretation manually, and the other is to evaluate the model by using algorithm indicators. Yeh et al. [26] proposed that the Infidelity and Sensitivity indicators can be used to quantitatively evaluate the algorithm model. In this experiment, our research goal is to compare the relationship between the local explainability model and the code review paradigms. We use the method of manual verification to evaluate the explainability model.

Given the same training dataset, the same test dataset, and the CNN model for SATD detection, the scores of critical factors are ranked. The Top-N phases with Top-N highest scores are gathered. These phrases are used to evaluate the effectiveness of the interpretability model.

3.5 Experiment Environment

The experiment equipment is a desktop workstation, equipped with a Intel Core i5 CPU, 24G RAM, and Nvidia RTX 2070 GPU, running on Windows 10 Operating System.

4 Experimental Results

We present the results of our case study to our two research questions.

4.1 How Effective is Our Weighted Focal Loss Function for Imbalance Software Comments Compared with The Cross Entropy Loss Function?

Motivation: The SATD detection model can be formalized as a binary classification problem using natural language processing technology. Our goal is to accurately divide the software comments into whether they are SATD or not. Based on the consideration of the naturalness of software code comments, we merged the datasets. Kim's TextCNN model is used to perform in-depth semantic analysis. For the class imbalance problem that naturally exists in the data set, we propose to use the weighted focal loss function to calculate the loss of instances. The SATD detection model with the best classification performance is the basis for interpretability analysis. Therefore, we need to study whether the performance of our proposed detection model can exceed the baseline model.

Approach: To address RQ1, we use the loss function in Equation (6) to calculate the loss value for the examples in the training data set; the hyperparameter optimization method based on particle swarm is used to determine the best hyperparameter in Equation (6). The classification model used as the baseline is a classification model using the cross-entropy information loss function in Equation (2). We did not compare with the model in reference [2]. The reason is that we preprocessed the dataset and obtained classification models with different concerns. The indicators used to evaluate the classification model include *Precision, Recall*, and *AUC*. These evaluation metrics are often used in machine learning to compare the classification performance of different classification models. We repeat the process 10 times on the training data set. We perform random stratified sampling every time and take the average of the results on the test dataset as the result of model evaluation.

Results: The statistics of the performance of different SATD detection models are shown in Table 2. We can see that our model using weighted focal loss function outperforms the baseline model only using cross entropy loss function. On average, our model achieves 12.27%, 5.97%, and 5.62% improvement in *Precision, Recall*, and *AUC* compared to the baseline model, respectively. Therefore, we can say that our proposed model achieves the best SATD detection performance and can be used as a classification model for subsequent interpretability analysis.

Table 2. Performance of different SATD detection model

Model	Precision	Recall	AUC
Baseline model using cross entropy loss function	0.717	0.904	0.902
Our model using weighted focal loss function	0.805	0.958	0.953

4.2 Can the Gradient-based Interpretability Model Cover 62 Manually Labeled Paradigms?

Motivation: Although there have been preliminary studies on the interpretability analysis of SATD detection, the TextCNN model we used has not been studied from the perspective of the gradient of the model. In image processing, gradient-based methods can be used to analyze the key factors in the picture. What we need to study is whether we can get the same result when using different gradients to interpret the model for the TextCNN model. If the same results can be obtained, it shows the validity of the interpretability model, and it also shows that the follow-up research on the SATD detection problem can adopt a gradient-based model.

Approach: To address RQ2, we use two gradient-based interpretability models. The specific calculation process is shown in Equation (8) and (9). Both models are based on the TextCNN model. After the model training is completed, interpretability analysis is performed on all code comments, and keywords that affect the model classification are extracted. The explanation process is to perform word embedding for each code comment and input the trained model. Perform backpropagation on the neural network to obtain the gradient of the word embedding output. Therefore, it is necessary to perform a summation calculation in the length dimension of the output to obtain the attribution of each word. If the attributable importance value of the word is greater than the threshold, the word is stored.

Results: The statistics of paradigms using Saliency Maps are shown in Table 3. The statistics of paradigms using Saliency Maps are shown in Table 4. In the tables, the verified phrases are shown in bold italics. When using Saliency Maps, 27 paradigms are verified with a percentage of 43.5%. When using Integrated Gradients, 28 paradigms are verified with a percentage of 45.2%. In the results of the two gradient-based interpretability models, less than half of the paradigms were verified. The two algorithms showed consistency in the ratio of paradigms. However, we still need to analyze how many same paradigms are verified. A total of 24 paradigms in the analysis results of the two algorithms are the same. This result can show that the two algorithms have strong consistency when interpreting the model based on the gradient. However, from another point of view, only less than half of the paradigms have been verified, indicating that there is still room for improvement in gradient-based interpretable models.

Table 3. Statistics of paradigms using Saliency Maps

Length of paradigms	Paradigms	Count of paradigms
One word	***hack***; ***retarded***; stupid; ***ugly***; nuke; ***hacky***; ***silly***; ***kludge***; ***fixme***; barf; ***yuck***; ***crap***; inconsistency; kaboom;	14/9
Two words	***take care***; ***is problematic***; give up; ***temporary solution***; causes issue; ***temporary crutch***; toss it; certainly buggy; ***bail out***;	9/5
Three words	at a loss; ***remove this code***; ***may cause problem***; trial and error; ***this is wrong***; cause for issue; some fatal error; this is bs; just abandon it; probably a bug; something bad happened; fix this crap; this is uncool; abandon all hope; prolly a bug; don't use this; something's gone wrong; workaround for bug;	18/2
Four words	***there is a problem***; something serious is wrong; ***get rid of this***; hope everything will work; remove me before production; it doesn't work yet; this isn't quite right; this doesn't look right; this isn't very solid;	9/2
Five words	***hang our heads in shame***; ***this can be a mess***; give up and go away; ***is this next line safe***; is this line really safe; doubt that this would work; risk of this blowing up; you can be unhappy now; ***something bad is going on***;	9/4
Six words	u***nknown why we ever experience this***; ***treat this as a soft error***;	2/2
Seven words	***this is temporary and will go away***;	1/1

Table 4. Statistics of paradigms using Integrated Gradients

Length of paradigms	Manually labeled paradigms	Count of paradigms
One word	***hack***; ***retarded***; ***stupid***; ***ugly***; nuke; ***hacky***; ***silly***; ***kludge***; ***fixme***; ***barf***; ***yuck***; ***crap***; inconsistency; kaboom;	14/11
Two words	***take care***; ***is problematic***; give up; ***temporary solution***; causes issue; **temporary crutch**; toss it; certainly buggy; **bail out**;	9/6
Three words	at a loss; ***remove this code***; ***may cause problem***; trial and error; ***this is wrong***; cause for issue; some fatal error; this is bs; just abandon it; probably a bug; something bad happened; fix this crap; this is uncool; abandon all hope; prolly a bug; don't use this; something's gone wrong; workaround for bug;	18/3
Four words	***there is a problem***; something serious is wrong; ***get rid of this***; hope everything will work; remove me before production; it doesn't work yet; this isn't quite right; this doesn't look right; this isn't very solid;	9/2
Five words	***hang our heads in shame***; ***this can be a mess***; give up and go away; ***is this next line safe***; is this line really safe; doubt that this would work; risk of this blowing up; you can be unhappy now; ***something bad is going on***;	9/4
Six words	***unknown why we ever experience this***; treat this as a soft error;	2/1
Seven words	***this is temporary and will go away***;	1/1

4.3 Can Different Interpretability Models Discover New SATD Paradigms?

Motivation: Code comments are technical debts that programmers use to actively introduce in the process of program development. These technical debts have variable lengths and uncertain technical specification definitions, etc., which belong to the category of natural language. Therefore, there may be missed technical debts in the paradigm of manual labeling. Based on the CNN model, these SATD can be uniformly coded and modeled to use the local interpretability model to discover new potential paradigms.

Approach: To address RQ3, we use the interpretability model to backpropagate all SATDs and find the corresponding gradient of each word. Then, the gradients corresponding to each of these words are summed to obtain all the contributions values of each word in the entire project. Based on the contribution value, we manually judge the importance of the word.

Results: The top-100 importance value of the one-word paradigm using Saliency Maps and Integrated Gradients are shown in Table 5 and Table 6. Firstly, compared with 62 manually labeled paradigms, "fixme", "hack", "workaround" et.al are validated when using Saliency Maps and Integrated Gradients. Moreover, "todo", "needed et.al are proposed.

Through an in-depth study of code comments, it is found that these words have a greater possibility of being used as iconic words indicating whether software comments show SATD.

Table 5. Top-100 importance value of one-word paradigm using Saliency Maps

Key word	Importance	Key word	Importance	Key word	Importance	Key word	Importance	Key word	Importance
pad	2829.1	do	37.1	i	21.6	that	16.6	so	13.1
todo	1515.9	!	35.1	i18n	21.4	handle	16.5	note	13.1
this	222.7	>	34.0	get	20.8	from	16.2	fix	13.0
fixme	191.8	value	33.8	use	20.8	by	16.1	have	12.6
the	148.2	it	30.3	check	20.4	make	16.0	probably	12.6
we	137.6	checking	28.7	argouml	20.0	really	15.8	as	12.3
to	117.6	param	27.7	workaround	19.6	null	15.4	needs	12.0
a	75.0	tfm	27.4	add	19.1	{	15.1	string	12.0
be	67.8	of	27.2	user	19.0	or	15.0	out	11.8
if	62.9	<	26.6	method	18.8	but	15.0	2	11.7
for	59.1	used	25.6	what	18.7	can	14.8	does	11.4
n't	57.7	xxx	25.2	1	18.7	better	14.7	p	11.3
hack	54.9	an	24.9	why	18.3	implement	14.2	only	11.2
;	54.1	all	24.7	defer	18.2	more	14.0	need	11.1
should	53.4	and	24.5	sss	18.0	$	13.7	on	10.8
not	50.5	default	24.4	way	18.0	file	13.6	set	10.7
here	50.2	work	24.4	are	17.7	pop	13.6	model	10.7
is	44.9	into	23.7	fdietz	17.6	’s	13.4	now	10.4
in	40.5	no	23.1	uml	17.6	]	13.2	namespace	10.3
needed	38.3	with	22.5	argument	17.5	code	13.2	create	10.1

Table 6. Top-100 importance value of one-word paradigm using Integrated Gradients

Key word	Importance	Key word	Importance	Key word	Importance	Key word	Importance	Key word	Importance
todo	2136.9	in	32.2	remove	17.7	implemented	10.6	how	7.4
fixme	289.8	implement	31.6	handle	17.4	yuck	9.7	broken	7.3
this	240.5	i	31.5	does	16.6	kludge	9.6	uml2	7.3
should	132.8	why	29.2	i18n	16.6	very	9.2	good	7.3
not	105.9	defer	28.5	require	16.2	check	9.2	from	7.2
hack	102.8	it	27.3	author	15.6	tfm	9.0	note	7.2
to	89.1	argument	26.6	know	14.7	bad	8.9	thing	7.2
needed	65.2	way	26.6	ugly	14.6	!	8.8	efficient	7.1
we	62.9	really	25.7	maybe	14.4	more	8.8	correct	7.1
do	58.1	used	25.6	renederer	14.4	of	8.8	when	6.9
be	51.4	fdietz	24.9	could	14.3	support	8.7	addtrigger	6.9
is	48.4	needs	24.0	use	14.1	temporary	8.5	think	6.7
a	48.2	don’t	22.3	with	13.9	these	8.5	out	6.6
workaround	46.5	probably	21.3	around	13.7	dms	8.2	unused	6.5
xxx	44.7	into	21.0	make	13.5	tweak	8.0	nice	6.5
here	41.8	fix	20.3	would	11.8	that	7.9	necessary	6.3
need	41.3	bug	19.9	some	11.7	stupid	7.8	can	6.2
checking	41.0	perhaps	19.7	on	11.6	but	7.8	an	5.9
better	40.3	for	19.5	and	11.2	code	7.6	replace	5.7
work	39.4	move	18.3	may	11.0	implementation	7.4	bit	5.7

5 Conclusion

SATD is an intentionally introduced software code comment describing potential defects. Although it is possible to process code comments based on methods such as pattern recognition and natural language text processing, deep learning is better at embedding and categorizing variable-length code comments. However, there is no empirical research on explanatory aspects of CNN-based gradients. We have investigated local interpretation models such as Saliency Maps and Integrated Gradients, which are currently widely used in image processing, and conducted empirical research based on open data sets. The research results show that these models can cover nearly half of the manually labeled paradigms; at the same time, these models can also fully analyze the code comments and discover potential new paradigms. In the future, it is necessary to study new interpretable algorithms for SATD so that more paradigms can be discovered and the quality of software development can be improved. The class overlap problem in the dataset is also an important factor that may affect the SATD detection model. In subsequent research, we should also pay attention to the impact of the class overlap problem on the detection model [26]. The effect of term weighting on the SATD classifier is also an issue to be considered later [27].

Acknowledgment

This work was supported by National Natural Science Foundation of China （62172350）, Philosophy and Social

Science Research Projects in Jiangsu (2020SJB0836), Nantong Science and Technology Project (JC2021124), Guangxi Key Laboratory of Trusted Software(kx202046, kx202013), Scientific Research Projects of Jiangsu College of Engineering and Technology (GYKY/2020/4), Research Project of Modern Educational Technology in Jiangsu Province (2021-R-94735), Special Project of China Higher Education Association(21SZYB23) . Sponsored by Special Foundation for Excellent Young Teachers and Principals Program of Jiangsu Province and Qing Lan Project of Jiangsu province.

References

[1] A. Potdar, E. Shihab, An Exploratory Study on Self-admitted Technical Deb, *2014 IEEE International Conference on Software Maintenance and Evolution. IEEE*, Victoria, British Columbia, Canada, 2014, pp. 91-100.

[2] X. X. Ren, Z. C. Xing, X. Xia, D. Lo, X. Y. Wang, J. Grundy, Neural Network-based Detection of Self-admitted Technical Debt: From Performance to Explainability, *ACM transactions on software engineering and methodology*, Vol. 28, No. 3, pp. 1-45, August, 2019.

[3] E. S. Maldonado, E. Shihab, Detecting and Quantifying Different Types of Self-admitted Technical Debt, 2*015 IEEE 7Th international workshop on managing technical debt*, Bremen, Germany, 2015, pp. 9-15.

[4] Q. Huang, E. Shihab, X. Xia, D. Lo, S. P. Li, Identifying Self-admitted Technical Debt in Open Source Projects using Text Mining, *Empirical Software Engineering*, Vol. 23, No. 1, pp. 418-451, February, 2018.

[5] Z. Zhang, X. Cui, P. Li, J. Jiang, X. Ji, Hyperspectral Data Analysis based on Integrated Deep Learning, *International Journal of Performability Engineering*, Vol. 16, No. 8, pp. 1225-1234, August, 2020.

[6] General Data Protection Regulation, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016. *Official Journal of the European Union (OJ)*, Vol. 59, No. L119, pp. 1-88, May, 2016.

[7] Y. B. Qu, X. Chen, Y. Q. Zhao, X. L. Ju, Impact of Hyper Parameter Optimization for Cross-project Software Defect Prediction, *International Journal of Performability Engineering*, Vol. 14, No. 6, pp. 1291-1299, June, 2018.

[8] S. Wang, Y. Li, W. Mi, Y. Liu, Software Defect Prediction Incremental Model using Ensemble Learning, *International Journal of Performability Engineering*, Vol. 16, No. 11, pp. 1771-1780, November, 2020.

[9] R. Gao, W. E. Wong, MSeer-An Advanced Technique for Locating Multiple Bugs in Parallel, *IEEE Transactions on Software Engineering*, Vol. 45, No. 3, pp. 301-318, March, 2019.

[10] V. Debroy, W. E. Wong, A Consensus-based Strategy to Improve the Quality of Fault Localization, *Software: Practice and Experience*, Vol. 43, No. 8, pp. 989-1011, August, 2013.

[11] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, J. Grundy, An Empirical Study of Model-agnostic Techniques for Defect Prediction Models, *IEEE Transactions on Software Engineering*, https://doi.org/10.1109/TSE.2020.2982385, March, 2020.

[12] Y. Kim, Convolutional Neural Networks for Sentence Classification, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014, pp. 1746-1751.

[13] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, H. Lipson, Understanding Neural Networks Through Deep Visualization, *arXiv preprint arXiv:1506.06579*, June, 2015.

[14] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), *European conference on computer vision*, Springer, Cham, 2014, pp. 818-833.

[15] K. Simonyan, A. Vedaldi, A. Zisserman, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, *arXiv preprint arXiv:1312.6034*, December, 2013.

[16] D. Smilkov, N. Thorat, B. Kim, F. Viégas, M. Wattenberg, SmoothGrad: Removing Noise by Adding Noise, *arXiv preprint arXiv:1706.03825*, June, 2017.

[17] A. Shrikumar, P. Greenside, A. Kundaje, Learning Important Features through Propagating Activation Differences, *Thirty-fourth International Conference on Machine Learning*, Sydney, Australia, 2017, pp. 3145-3153.

[18] M. Sundararajan, A. Taly, Q. Yan, Axiomatic Attribution for Deep Networks, *Thirty-fourth International Conference on Machine Learning*, Sydney, Australia, 2017, pp. 3319-3328.

[19] T. Lin, P. Goyal, R. Girshick, K. He, P. Dollar, Focal Loss for Dense Object Detection, *Proceedings of the IEEE international conference on computer vision*, Venice, Italy, 2017, pp. 2999-3007.

[20] D. Bratton, J. Kennedy, Defining a Standard for Particle Swarm Optimization, *2007 IEEE swarm intelligence symposium*, Honolulu, Hawaii, USA, 2007, pp. 120-127.

[21] T. Wei, X. Zhao, L. Pei, L. Li, A Co-Saliency Object Detection Model for Video Sequences, *International Journal of Performability Engineering*, Vol. 16, No. 11, pp. 1793-1802, November, 2020.

[22] M. Sundararajan, A. Taly, Q. Yan, Gradients of Counterfactuals, *arXiv preprint arXiv:1611.02639*, November, 2016.

[23] A. Binder, S. Bach,G. Montavon,K. Müller,W. Samek, Layer-wise Relevance Propagation for Deep Neural Network Architectures, in: K. Kim, N. Joukov (Eds.), *Information science and applications*, Springer, Singapore, 2016, pp. 913-922.

[24] N. Tsantalis, T. Chaikalis, A. Chatzigeorgiou, JDeodorant: Identification and Removal of Type-checking Bad Smells, *12th European Conference on Software Maintenance and Reengineering*, Athens, Greece, 2008, pp. 329-331.

[25] C. Yeh, C. Hsieh, A. Suggala, D. Inouye, P. Ravikumar, On the (In)fidelity and Sensitivity of Explanations, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Vancouver, Canada, 2019, pp. 10965-10976.

[26] Y. Qu, X. Chen, L. Li, Cross-Version Software Defect Prediction Method for Relieving Class Overlap Problem, *Journal of Jilin University (Science Edition)*, Vol. 59, No. 2, pp. 372-378, March, 2021.

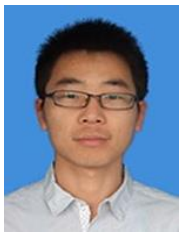
[27] K.-H. Tseng, C.-H. R. Lin, J.-S. Liu, C.-M. A. Huang, Y.-H. Wang, A Study on Text Classification: Term Weighting Algorithm Analysis, *Journal of Internet Technology*, Vol. 22, No. 2, pp. 311-325, March, 2021.

Biographies

Guoqiang Zhuang received the M.S. degree in Soochow University. His research interests include software engineering, and software defect prediction.

Yubin Qu was born in Nanyang, China in 1981. He received the B.S. and M.S. degrees in Computer Science and Technology from Henan Polytechnic University in China in 2004 and 2008. Since 2009, he has been a lecture with Information Engineering Institute, Jiangsu College of Engineering and Technology. He is the author of more than 10 articles. His research interests include software maintenance, software testing, and machine learning.

Long Li received his Ph.D. degree from Guilin University of Electronic Technology, Guilin, China in 2018. He is now a lecturer at the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His research interests include cryptographic protocols, privacy-preserving technologies in big data and IoT.



Xianzhen Dou was born in Xuzhou, China in 1987. He received the M.S. degree in School of Electronics and Information from Nantong University in China in 2013. Since 2019, he has been a lecture with Information Engineering Institute, Jiangsu College of Engineering and Technology. His research interests include software engineering, and machine learning.

Mengao Li was born in Shaoyang, China in 1983. He received the M.S. degree in computer system architecture from Jilin University in China in 2008. Since 2008, he has been working in CCSC Systems Engineering Research Institute. He is the author of many excellent articles. His research interests include system architecture, software development and artificial intelligence.