Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.060228

ARTICLE

SESDP: A Sentiment Analysis-Driven Approach for Enhancing Software Product Security by Identifying Defects through Social Media Reviews

Farah Mohammad[1,2,*], **Saad Al-Ahmadi**[3] **and Jalal Al-Muhtadi**[1,3]

[1]Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, 11543, Saudi Arabia
[2]Department of Computer Science, and Technology, Arab East Colleges, Riyadh, 11583, Saudi Arabia
[3]College of Computer & Information Sciences, King Saud University, Riyadh, 11543, Saudi Arabia

*Corresponding Author: Farah Mohammad. Email: fnazar@ieee.org

Received: 27 October 2024; Accepted: 05 February 2025; Published: 26 March 2025

ABSTRACT: Software defect prediction is a critical component in maintaining software quality, enabling early identification and resolution of issues that could lead to system failures and significant financial losses. With the increasing reliance on user-generated content, social media reviews have emerged as a valuable source of real-time feedback, offering insights into potential software defects that traditional testing methods may overlook. However, existing models face challenges like handling imbalanced data, high computational complexity, and insufficient integration of contextual information from these reviews. To overcome these limitations, this paper introduces the SESDP (Sentiment Analysis-Based Early Software Defect Prediction) model. SESDP employs a Transformer-Based Multi-Task Learning approach using Robustly Optimized Bidirectional Encoder Representations from Transformers Approach (RoBERTa) to simultaneously perform sentiment analysis and defect prediction. By integrating text embedding extraction, sentiment score computation, and feature fusion, the model effectively captures both the contextual nuances and sentiment expressed in user reviews. Experimental results show that SESDP achieves superior performance with an accuracy of 96.37%, precision of 94.7%, and recall of 95.4%, particularly excelling in handling imbalanced datasets compared to baseline models. This approach offers a scalable and efficient solution for early software defect detection, enhancing proactive software quality assurance.

KEYWORDS: Software defect; data balancing; feature extraction; RoBERTa; transformer

1 Introduction

In the rapidly evolving landscape of software development, ensuring the quality and reliability of software products is paramount. As software systems become increasingly complex, the identification and resolution of defects at an early stage are critical to maintaining user satisfaction [1]. A bug is traditionally known as a software defect, an imperfection in something that is normally part of or integrated into a software product [2]. A bug is a problem that causes the system or one of its components to produce an unintended or incorrect result. Software defects can be caused by a variety of sources, such as incorrect code, misunderstandings of requirements, or even defects within the design and architecture of the system [3]. These defects can appear as crashes, inaccuracies in data processing, security vulnerabilities, or even usability issues; all of these deeply affect the functionality and/or user experience of software [4].

We can broadly classify software defects into different types. When the software does not perform any function as expected per the requirements laid down, it is termed a functional defect. Anything from

functionality not working correctly to the software refusing to execute any task that it is supposed to execute falls in this category [5]. Performance defects: The performance defects are Achilles' heel, where the software performs the required functionality, but that takes more time than acceptable or uses more resources than acceptable, resulting in problems like slow response or high memory consumption. Usability defects: Those have to do with the user interface and user experience. Where perhaps the software works just fine but is hard for users to work with or those users perceive it as unintuitive. Compatibility defects deal with when different hardware, operating systems, or browsers result in the software performing in an unwanted manner [6]. A security defect is one where, if manipulated, it would lead to a security breach presenting one with unauthorized access or data loss. Logical defects occur when there is an error in the logic of the software such that it produces the wrong output though the input put into the system is correct and seemingly all parts work correctly.

Software defects must be identified as early as possible for many reasons. Early detection and fixing of the defects mean lower cost and effort according to the software development life cycle [7]. According to a study, the cost of rectification of a defect during the final stages of testing or production increases exponentially compared to its detection at the initial stages of development. As shown, for instance, a defect found during the requirement phase may require only clarification or a minor change, but the same defect found after the software has entered into production will most likely require some complicated patch or even a full redesign, impacting all dependent systems and causing some downtimes [8]. In general, early detection of defects also increases software quality and reliability, improving users' satisfaction. Less defective software reaches the end users may perform better and improves the expectations of users. It also reduces the risk of adverse publicity. Early detection of defects becomes not just conducive but extremely essential in safety-critical industries such as healthcare, finance, and aerospace for avoiding catastrophic failures.

Conventional methods include static code analysis, dynamic testing, and historical data modeling, which have been conventionally applied in the identification of potential problems in software development [9]. For instance, static code analysis tools like SonarQube and FindBugs check the source code for possible defects without executing it; hence, they help to detect issues early during development. These generally generate a lot of false positives and cannot comprehend the runtime context of the software; hence, they usually fail in defect detection. On the other hand, dynamic testing involves the execution of software and the observation of general behavior. Issues apparent in runtime may be noticed, but several tested scenarios and inputs are limited, leaving untested paths where defects might go undetected [10]. Data modeling traditionally relies on back-data about defects to drive predictions of future issues, which may not work as well for codebases that are new or have changed over time dramatically. The patterns in many ways do not hold. Aggregately, these traditional methods can be labor-intensive in the number of effort and man-hours required and may not fully capture the complex, evolutionarily changing nature of modern software systems, thus limiting their capability to predict and prevent all possible defects.

However, with the advancement of social media, people review their experiences good and bad on online review sites and discussions, and this can be considered a gold mine of unstructured [11]. This paper proposes a new approach to software defect prediction based on the sentiment analysis of social media reviews. This research proposes a model focusing on user sentiments expressed in reviews to predict early signs of software defects, which can enable developers to address problems before they grow. The approach integrates advanced natural language processing techniques through a Transformer-Based Multi-Task Learning model using Robustly Optimized Bidirectional Encoder Representations from Transformers Approach (RoBERTa), which will conduct sentiment analysis and defect prediction all at once. The model picks up subtle opinions by users and associates negative feelings with probable software issues.

Of all these, the importance of this work is the fact that it would be able to exploit real-time and user-generated content on social media platforms to improve software quality. Such early monitoring and analysis of reviews on social media allow the developers to understand emerging defects that traditional testing has not yet perceived. The paper is going to describe in detail the methodology with features extraction and the prediction phase and prove the efficiency of the model by experiments on real-world data. The results reflect that sentiment analysis can act as one of the best alternatives for early defect prediction that will help to build a robust and user-friendly software product.

1.1 Research Contribution

The key research contributions of SESDP (Sentiment Analysis-Based Early Software Defect Prediction) are:

- The SESDP model introduces an innovative feature extraction process that utilizes RoBERTa for generating contextual embeddings, which are further enhanced by incorporating sentiment scores. This combined approach creates a richer and more informative feature vector that captures both the nuanced meanings of user reviews and their associated sentiment, leading to more accurate defect predictions.
- SESDP leverages Transformer-Based Multi-Task Learning to simultaneously perform sentiment analysis and defect prediction. By sharing a RoBERTa backbone and employing task-specific heads, the model effectively uses sentiment information to improve the accuracy of defect prediction, demonstrating the interdependence between user sentiment and software quality.
- The experimental results demonstrate the superiority of the SESDP model over traditional methods. Through rigorous testing on real-world datasets, SESDP consistently outperforms baseline models in both sentiment analysis and defect prediction tasks, validating the effectiveness of the proposed approach and its practical applicability in early defect detection using social media reviews.

The remainder of this paper is structured as follows: Section 2 provides a review of existing techniques for defect prediction. Section 3 details the methodology of the proposed approach. Section 4 presents the experimental results and evaluations, while Section 5 concludes with a discussion and recommendations for future research directions.

2 Literature Review

Software defect prediction is an essential aspect of software quality assurance, aiming to identify and address defects before they escalate into significant issues. Various approaches have been proposed to enhance defect prediction accuracy and efficiency, each with its strengths and limitations. The SESDP model builds on these existing works, addressing key challenges and proposing innovative solutions. Wu et al. [12] proposed a novel software defect prediction approach, based on mutual information and correlation coefficient weighted class association rule mining (MCWCAR), which have made significant strides in improving defect prediction accuracy. They proposed a method that addresses the limitations of traditional association rule mining algorithms, particularly in handling the unbalanced distribution of defect data. The MCWCAR model employs a cost-sensitive strategy, integrating feature selection and itemset screening to enhance the prediction of defect-prone software. Despite the effectiveness demonstrated through experiments on 27 open-source datasets, the reliance on weighted frequent itemset mining still leaves room for improvement in handling complex, high-dimensional data.

Ali et al. [13] presented a software defect prediction model using an intelligent ensemble of multiple classifiers for improved prediction. The authors claim that the two-stage prediction with Random Forest, Support Vector Machine, Naïve Bayes, and Artificial Neural Network enhances the accuracy of the model

over conventional approaches. Indeed, their evaluation performed on several datasets from the NASA MDP has shown a high improvement in prediction performance. However, the challenge in integrating diverse classifiers and overfitting issues remain the challenges that may affect the generalization of the model in various contexts. And finally, in the class imbalance of the defect datasets, Chen et al. [14] proposed the dual ensemble software defect prediction concept. They discussed how a diverse ensemble, combined with neural networks, can come up with improved defect prediction models. The Dual Ensemble Software Defect Prediction (DE-SDP) model provides an alternative approach to the class imbalance problem and improves predictive accuracy based on a two-level ensemble method. The dual ensemble, on the other hand, has increased further computational complexity, and the performance of this approach may increase or decrease regarding further dataset features.

To address the challenge arising with high-dimensional features from the software defect datasets, Tang et al. [15] have proposed an approach known as learnable three-line hybrid feature fusion (LTHFFA). The authors justify the fact that LTHFFA combines multiple dimensionality-reduction techniques, which significantly enhances optimal feature selection in reducing redundancy. Experimental results across seventeen datasets confirm the superior performance of the LTHFFA method compared to other methods for dimensionality reduction. However, this may lead to high dependence on complex feature fusion processes, hence limiting the applicability in real scenarios where computational efficiency is of the essence. Some have argued for the utilization of Deep Q-learning network (DQN) in the realm of software defect prediction because of its potential to minimize false positives and hence improve the reliability of the prediction. Ismail et al. [16] stressed the importance of the reduction of false positives to prevent waste of resources on non-existent defects. The DQN model presents impressive gains in prediction accuracy through dynamic reward policies compared to baseline classifiers. While effective, this approach may require extensive computing and careful tuning of the reward policy to achieve high performance on different data sets. The literature review of existing studies has been shown in Table 1.

Table 1: Literature review of existing defect prediction model

Ref.	Methodology	Dataset	Two major limitations
[17]	Bidirectional Long Short-Term Memory (Bi-LSTM) network combined with oversampling techniques (SMOTE)	PROMISE repository datasets	• Imbalanced data problem despite oversampling efforts. • Potential overfitting due to reliance on synthetic data generated by SMOTE.
[18]	Five-stage framework with ensemble learning	NASA datasets	• High computational complexity in feature selection and ensemble learning stages.

(Continued)

Table 1 (continued)

Ref.	Methodology	Dataset	Two major limitations
[19]	Negative Correlation Learning (NCL)-based cost-sensitive ensemble learning approach (NCL_CSEL)	NASA dataset and AEEEM dataset	• Overhead in managing class imbalance and overlap simultaneously may complicate model interpretation and scalability.
[20]	Effort-Aware Zeta (EA-Z) ranking score calculation strategy	72 datasets	• Limited applicability outside of specific ranking-based defect prediction contexts.
[21]	Deep Q-learning Network (DQN) for feature extraction	NASA and PROMISE repository	• High dimensionality of data even after feature extraction can lead to computational inefficiency.
[22]	Improved Multi-Disciplinary Autoencoder Classification (IMDAC) model combining Information Maximizing Generative Adversarial Network (InfoGAN), Mean Squared Error Autoencoder (MSEA), Denoising Autoencoder (DAE), and Convolutional Neural Network (CNN)	15 software projects	• Dependence on multiple deep learning techniques increases model complexity and training time.

Therefore, a critical review of the available software defect prediction methods reveals some serious limitations linked to overcoming imbalanced data by the Bi-LSTM network, high computational complexity Ensemble learning frameworks, the applicability of methods such as cost-sensitive learning based on NCL, and ranking strategies with EA-Z, given that careful tuning and context dependencies are required. Most of the deep learning methods usually suffer from the main disadvantages of high-dimensional data and model complexity that reduce the practicality and efficiency of the approach. Search-based SDEP model, on the other hand, overcame these limitations, while a Transformer-based Multitask Learning with RoBERTa was able to conduct the tasks of sentiment analysis and defect prediction simultaneously. This approach helps in not only improving accuracy over imbalanced datasets but also generalizability and efficiency that make for a strong, scalable early software defect detection solution, outperforming traditional methods.

3 Methodology

This section elaborates in detail on the basic methodology of the SESDP model represented in Fig. 1. The whole process starts with data collection, followed by an intensive preprocessing step that involves text cleaning, tokenization, and label encoding of the text data. Later comes feature extraction, where RoBERTa is used for text embedding and computation and fusing of sentiment scores for a better feature set. Then the final sentiment analysis and defect forecasting are done using Transformer-Based Multi-Task Learning. Each of the steps has been described in detail in the following subsections.

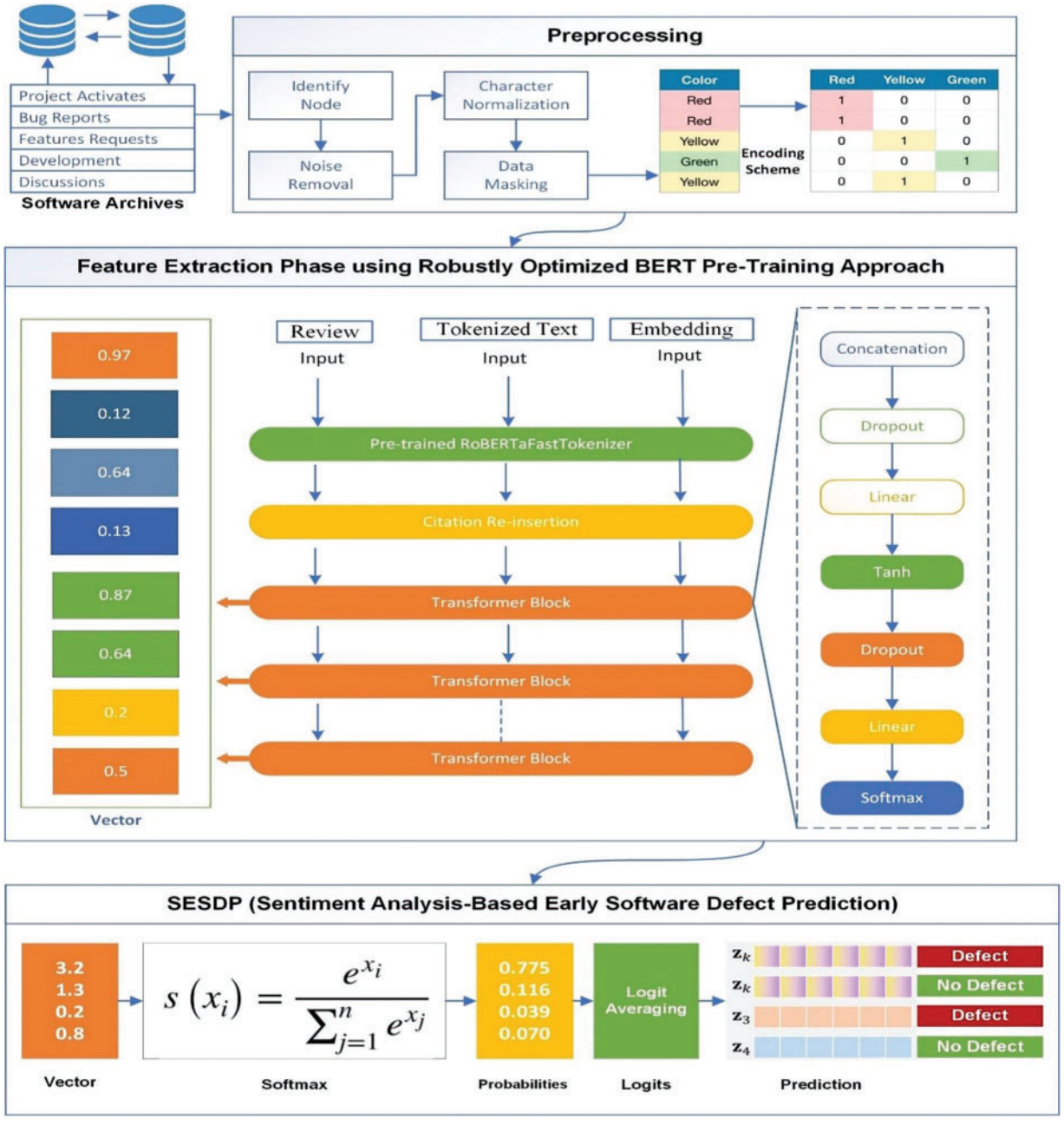


Figure 1: Proposed SESDP model

3.1 Data Collection

Three different datasets, represented in Fig. 2, are used for the evaluation of SESDP. The very first data set abbreviated as Software Bug Dataset I (SB-DSI) obtained from Socialist-DataSet-Issues. This dataset is the result of issues and pull request data collected from more than 30,000 repositories that cover over 1.5 million issues and pull requests. This dataset represents a wide range of activities happening in software projects, like bug reports, feature requests, and development discussions. Each issue has key fields such as title, description, labels, and state (open/closed), which provide rich context, averaging 50 comments per issue in how developers coordinate on the resolution of software defects. This dataset is particularly valuable for an analysis of the dynamics of issue-tracking and resolution processes in open-source software projects. The Bugzilla dataset of Mozilla contains over 1 million detailed bug reports, along with comments and attached metadata. A number of bug management attributes have been tracked in the dataset or bug repository, which includes bug ID, summary, product, component, severity, and resolution, among many others. On average, every bug report has 3 comments attached. Critical bugs take just about 30 days to resolve, while low-priority bugs take up to 90 days. This dataset is important in understanding the lifecycle of software defects and the effectiveness of resolution strategies in large software projects.

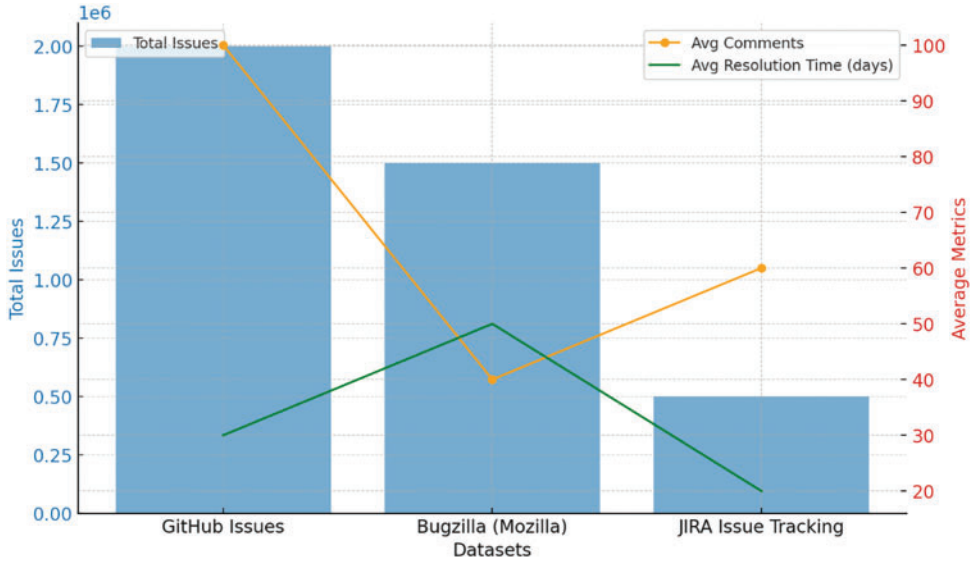


Figure 2: Dataset statistic

The Software Bug Dataset III (SB-DSIII) JIRA Issue Tracking Data involves over 8000 issue reports of various natures from different software development projects, ranging from bug forms and feature requests to improvement suggestions. Each issue is documented with an issue ID, project key, issue type, status, priority, description, and resolution. The dataset contains more than 10,000 issues, with an average resolution time of 15 days for critical bugs and as high as 60 days for non-critical ones. It is a dataset that will provide Program valuable insights into defect management processes within software development teams and help study and improve software quality assurance practices.

This research applied class weighting to the loss function during training to handle the class imbalance present in the datasets. This technique assigns higher weights to the minority classes, ensuring that the model gives appropriate attention to underrepresented instances. By incorporating class weighting, the SESDP

model mitigates the risk of bias toward the majority class, resulting in more accurate predictions, particularly in defect forecasting where defects are less frequent.

3.2 Preprocessing

Algorithm 1 illustrates some of the preprocessing steps involved in preparing textual data to make them ready for analysis and model training. Major steps involve cleaning the text, tokenization, and encoding labels. It begins with the removal of noise from the raw text data during the cleaning of the text. This includes eliminating special characters (e.g., @, #, &), URLs, and other non-alphanumeric content that does not contribute to the meaning of the text [23]. Additionally, stop words—common words like "and," "the," and "is" are removed because they generally do not carry significant information for the model. The cleaned text can be represented mathematically as:

$$CleanedText = RemoveNoise\,(T) - (SpecialChars + URLs + StopWords)$$

where T represents the original text data, and $RemoveNoise$ is the function applied to eliminate irrelevant content.

Algorithm 1: Preprocessing phase: text cleaning, tokenization, and label encoding

```
Input: Raw Text Data T, Sentiment Labels L_s, Defect Labels L_d
Output: Processed Tokens, Encoded Labels E_s, E_d
for each text sample t_i in T do
    t_i ← RemoveNoise (t_i) {Remove special characters, URLs, stop words}
    tokens_i ←Tokenize (t_i) {Tokenize the cleaned text}
    if L_s[i] is not encoded then
        E_s[i] ← LabelEncode (E_s[i]) {Encode sentiment label}
    end if
    If L_d[i] is not encoded then
        E_d[i] ←LabelEncode (L_d[i]){Encode defect label}
    end if
end for
return Processed Tokens, Encoded Labels E_s, E_d
```

The next step, after cleaning, will involve tokenization. Tokenization-in respect to the model's requirements means breaking down the cleaned text into words, subwords, or even characters [24]. This is an important step in preprocessing the text into a type of input that the transformer will understand, where every token will be processed separately. The tokenization process can be mathematically expressed as:

$$Tokens = Tokenize(CleanedText) \tag{1}$$

where $Tokenize$ is the function that segments the text into a sequence of tokens. Finally, label encoding is applied to convert categorical labels into numerical form, making them compatible with machine learning algorithms. In this context, sentiment labels (e.g., Positive, Negative, Neutral) and defect labels (e.g., Defect, No Defect) are encoded into numerical values. For example, a binary defect label could be encoded as 111 for "Defect" and 000 for "No Defect." The label encoding can be represented as:

$$EncodedLabels = LabelEncode(Sentiment\ Labels, Defect\ Labels) \tag{2}$$

where $LabelEncode$ maps the categorical labels to their corresponding numerical values. This phase transforms raw text data into a structured format suitable for model training. By systematically cleaning the text, tokenizing it into meaningful units, and encoding categorical labels into numerical values, this process ensures that the data is in an optimal state for further analysis and prediction tasks.

For long text inputs, truncation is performed to limit the sequence length to the maximum token limit supported by the transformer model while retaining critical information, typically focusing on the beginning or most relevant parts of the text. Alternatively, segmentation is applied, dividing lengthy texts into smaller, manageable chunks to preserve contextual information, with overlapping windows where necessary to maintain coherence between segments. To address class imbalances, oversampling techniques like SMOTE (Synthetic Minority Oversampling Technique) or undersampling of the majority class are employed, depending on the dataset's characteristics. Additionally, weight loss functions are utilized during model training to mitigate the impact of skewed class distributions, ensuring that minority class predictions are not overshadowed by the majority class. These strategies collectively enhance the quality and fairness of the training dataset, contributing to more accurate and reliable model predictions.

3.3 Feature Extraction

Feature extraction is one of the crucial phases in which raw text data is transformed into meaningful representations that the SESDP model would use for prediction. The process utilizes RoBERTa—a robustly optimized BERT pre-training approach for capturing both the contextual and sentiment-based features from text. RoBERTa is an optimized approach of BERT that seeks pre-training and fine-tuning approaches more effectively [25]. It removes Next Sentence Prediction from BERT, while it depends on dynamic masking in training and much larger mini-batch sizes and learning rates. This makes RoBERTa more effective at capturing the nuances of language in various contexts. The first step in the feature extraction process is to generate text embeddings using RoBERTa. These embeddings are dense vectors that represent the contextual meaning of the text, capturing subtle nuances and relationships between words [26]. Given an input text sequence $T = [t_1, t_2, \ldots, t_n]$, where t_i represents the tokens of the text, RoBERTa generates a sequence of embeddings $E_T = [e_1, e_2, \ldots, e_n]$ for each token. The embedding for the entire text sequence is often obtained by using the embedding of the special classification token $e_{[CLS]}$, which can be represented as:

$$e_{[CLS]} = RoBERTa(T) \tag{3}$$

Here, $e_{[CLS]}$ is a fixed-size vector that encapsulates the contextual meaning of the entire text sequence T. Once the text embeddings are obtained, the next step is to compute sentiment scores. A fine-tuned RoBERTa model, specifically trained for sentiment analysis, is used to extract sentiment information from the text. The sentiment score S for the text sequence T is calculated as:

$$S = \sigma\left(W_s \cdot e_{[CLS]} + b_s\right) \tag{4}$$

where W_s and b_s are the weights and bias parameters learned during fine-tuning for sentiment classification. σ is the softmax function, which converts the logits into a probability distribution over sentiment classes (e.g., Positive, Negative, Neutral).

The final step in the feature extraction process is to combine the text embeddings and sentiment scores into a single, comprehensive feature vector for each review. This fusion step ensures that both the contextual meaning and the sentiment information are considered by the model during prediction. The combined feature vector F is obtained by concatenating the text embedding $e_{[CLS]}$ and the sentiment score S:

$$F = Concat(e_{[CLS]}, S) \tag{5}$$

where F is the final feature vector representing the review. $Concat$ denotes the concatenation operation. This fused feature vector F is then used as input to the final prediction model, where it serves as a rich representation that captures both the content and sentiment of the review, enabling more accurate defect prediction. Table 2 shows the output of the proposed feature extraction process of SESDP.

Table 2: RoBERTA based feature extraction

Review	Tokenized text	Classification ([*CLS*]) embedding	Sentiment scores	Final feature vector
"The app crashes every time I open it."	["The", "app", "crashes", "every", "time", "I", "open", "it", "."]	[0.12, 0.34, −0.56, ..., 0.87]	[0.05,0.92,0.03] [Negative]	[0.12, 0.34, −0.56, ..., 0.87, 0.05, 0.92, 0.03]
"Great functionality, but it freezes occasionally."	["Great", "functionality", ",", "but", "it", "freezes", "occasionally", "."]	[−0.45, 0.22, 0.76, ..., −0.34]	[0.70,0.20,0.10] [Positive]	[−0.45, 0.22, 0.76, ..., −0.34, 0.70, 0.20, 0.10]
"Poor performance after the latest update."	["Poor", "performance", "after", "the", "latest", "update", "."]	[0.09, −0.31, 0.68, ..., 0.12]	[0.10,0.80,0.10] [Negative]	[0.09, −0.31, 0.68, ..., 0.12, 0.10, 0.80, 0.10]

3.4 Sentiment Analysis and Defect Prediction

The final phase of the SESDP model has been presented in Algorithm 2, which allows the simultaneous optimization of multiple analysis-related tasks such as sentiment analysis and defect prediction. By sharing a common transformer backbone (RoBERTa), the model can leverage the contextual knowledge gained from one task to enhance the performance of the other [27]. This is particularly useful when the tasks are related, as is the case with sentiment and defect prediction, where sentiment information can provide valuable context for identifying software defects. The shared backbone processes the input text sequence T and generates contextual embeddings. The [*CLS*] token embedding $e_{[CLS]}$ is the primary output, summarizing the entire text sequence:

$$e_{[CLS]} = RoBERTa(T) \tag{6}$$

where, $T = [t_1, t_2, \ldots, t_n]$ is the tokenized text sequence and $e_{[CLS]}$ is the output embedding corresponding to the [*CLS*]. The RoBERTa model itself is composed of multiple layers of transformers, each with its own self-attention and feed-forward networks. The embedding $e_{[CLS]}$ is obtained after passing the input sequence through all these layers:

$$e_{[CLS]} = TransformerLayers(T) \tag{7}$$

where $TransformerLayers$ represents the sequence of transformer layers in RoBERTa. In the task-specific head, the sentiment analysis head applies a fully connected layer to the [*CLS*] embedding to produce logits

for sentiment classification. The logits z_s are then converted into probabilities S using the softmax function:

$$z_s = W_s \cdot e_{[CLS]} + b_s \tag{8}$$

$$S = \sigma(z_s) = \frac{\exp(z_s^i)}{\sum_{j=1}^{k}(z_s^i)} \tag{9}$$

where W_s is the weight matrix for the sentiment analysis head, b_s is the bias term, z_s is the logits vector for sentiment classes. $S = [s_1, s_2, \ldots, s_k]$ represents the probabilities for each sentiment class, where k is the number of sentiment classes and σ is the softmax function.

Algorithm 2: Final sentiment analysis and defect prediction using transformer-based multi-task learning

Input: Tokenized Text Data $T = \{t_1, t_2, \ldots, t_n\}$
Output: Sentiment Predictions S, Defect Predictions D
for each tokenized text sequence T_i in T **do**
    $e_{[CLS]}$ROBERTa (T_i) {Extract [*CLS*] embedding using shared trans-former backbone}
    $Z_s \leftarrow W_s.e_{[CLS]} + b_s$ {Compute sentiment logits}
    $S_i \leftarrow \sigma(z_s)$. {Compute sentiment probabilities using softmax}
    $Z_d \leftarrow W_d.e_{[CLS]} + b_d$ {Compute defect logits}
    $D_i \leftarrow \sigma(z_d)$ {Compute defect probabilities using softmax or sigmoid}
    if S_i and D_i are valid predictions **then**
        Store S_i; in S {Store sentiment prediction}
        Store D_i in D {Store defect prediction}
    end if
end for
return S, D {Return the final sentiment and defect predictions}

Similarly, the defect prediction head uses the [*CLS*] embedding to generate logits for defect prediction. These logits z_d are converted into probabilities D using a softmax or sigmoid function, depending on the nature of the classification task:

$$z_d = W_d \cdot e_{[CLS]} + b_d \tag{10}$$

$$S = \sigma(z_s) = \frac{\exp(z_s^i)}{\sum_{j=1}^{k}(z_s^j)} \tag{11}$$

The overall loss L in multi-task learning is a weighted sum of the individual losses for sentiment analysis $L_{sentiment}$ and defect prediction L_{defect}. The loss for each task is typically computed using cross-entropy:

$$L_{sentiment} = -\sum_{i=1}^{k} y_s^i \log(S^i) \tag{12}$$

$$L_{defect} = -\sum_{i=1}^{k} y_d^i \log(D^i) \tag{13}$$

where y_s^i and y_d^i are the ground truth labels for sentiment and defect prediction, respectively. S^i and D^i are the predicted probabilities for each class in sentiment and defect prediction. The combined multi-task loss is then:

$$L = \alpha \cdot L_{sentiment} + \beta \cdot L_{defect} \tag{14}$$

where α and β are hyperparameters that control the contribution of each task to the overall loss. During inference, given an input text sequence Test-Time Training (TTT), the model produces both sentiment classification and defect prediction simultaneously. The shared RoBERTa backbone ensures that the contextual information extracted from the text is leveraged in both tasks, enhancing the accuracy of the predictions.

3.5 Model Deployment

In the deployment phase of the SESDP model, the focus is on transitioning the trained model from the development environment to a production environment, where it can be utilized to make real-time predictions on new data. The process begins with exporting the trained model into a format suitable for deployment in the ONNX (Open Neural Network Exchange) format. The developed PyTorch, the ONNX format, offers a similar capability, allowing the model to be exported with "$torch.onnx.export(model, SESDP\_input, "model.onnx")$, where the ONNX file can be deployed in various environments.

Once the model is exported, the next step involves developing an API (Application Programming Interface) to serve the model, enabling real-time predictions on new social media reviews. A common approach is to use Flask, a lightweight web framework in Python, to build the API. The API acts as an intermediary, receiving input data, such as new text data, processing it, and returning predictions. For instance, the API might use a Flask application where the model is loaded using $tf.keras.models.load\_model('path/to/saved\_model')$. When a Power-On Self Test (POST) request with new text data is received, the text is preprocessed, fed into the model, and the predictions are returned in JSON format. The API's prediction pipeline typically involves several steps: receiving input, preprocessing the text (such as tokenization and embedding), running the preprocessed input through the model to generate predictions, and returning these predictions to the client. This setup allows for real-time analysis and prediction of sentiment and potential defects in software based on user reviews.

The final step in the deployment process is to integrate the API into a larger system that monitors software quality. This system could include components like a data collection module that scrapes social media platforms for new reviews, an API endpoint that processes these reviews, and a monitoring dashboard that displays real-time analytics on software quality. Additionally, the system might feature an alert system that automatically notifies developers when a high likelihood of defects is predicted, allowing for early intervention. Mathematically, this process can be described as follows: for a new social media review T_{new}, the system first preprocesses the review to generate a contextual embedding $e^{new}_{[CLS]}$ using RoBERTa. The sentiment score S_{new} and defect prediction D_{new} are then computed using the task-specific heads of the model:

$$S_{new} = \sigma(W_s \cdot e^{new}_{[CLS]} + b_s) \tag{15}$$

$$D_{new} = \sigma(W_d \cdot e^{new}_{[CLS]} + b_d) \tag{16}$$

where W_s and W_d are the weights, b_s and b_d are the biases, and σ is the sigmoid function. If the defect likelihood D_{new} exceeds a certain threshold τ, the system flags the review as a potential defect, signaling the need for developer attention.

$$Flage = \begin{cases} 1, & if D_{new} > \tau \\ 0, & otherwise \end{cases} \tag{17}$$

This deployment process ensures that the SESDP model is not only ready for real-time application but also seamlessly integrated into a broader software quality monitoring framework. By automating the

prediction and alerting processes, organizations can effectively monitor and mitigate software defects as they arise, enhancing the overall user experience and software reliability.

4 Experimental Results and Evaluation

This section provides an overview of the experiments conducted and their results. It covers the dataset employed, the performance metrics used, and the baseline approaches considered and offers a detailed presentation of the outcomes.

4.1 Results

The performance of the proposed models, SB-DSSI, SB-DSSII, and SB-DSSIII, was evaluated using three key metrics: accuracy, precision, and recall as shown in Fig. 3. The results show that SB-DSSII outperforms the other models across all metrics, achieving the highest accuracy at 97.34%, precision at 95.98%, and recall at 96.56%. These results indicate that SB-DSSII is the most reliable model, with superior classification abilities. SB-DSSIII also performs well, with an accuracy of 96.12%, precision of 95.01%, and recall of 95.67%, making it a strong alternative. Meanwhile, SB-DSSI, while slightly lower, still exhibits solid performance with an accuracy of 95.66%, precision of 93.12%, and recall of 94.97%. Overall, these results highlight that SB-DSSII addresses the challenges of defect prediction more effectively, particularly by improving precision and recall, which suggests it is better at reducing false positives and false negatives. SB-DSSIII, although not the top performer, provides competitive results and offers a balanced trade-off across all metrics. These findings validate the effectiveness of the proposed models in utilizing sentiment-based analysis for software defect prediction, with SB-DSSII emerging as the most optimal solution.

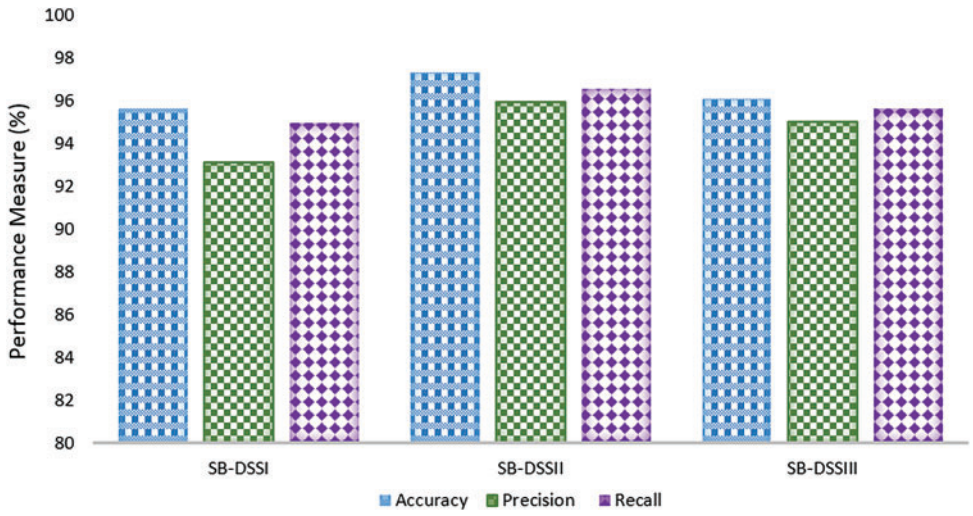


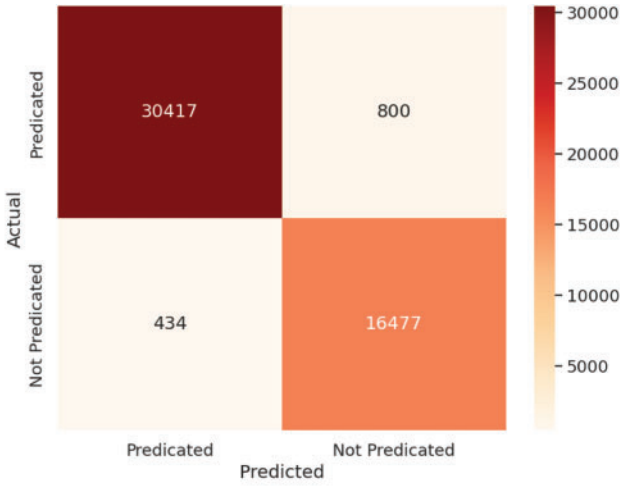
Figure 3: Experimental results on SB-DSSI, SB-DSSII and SB-DSSIII

In another experiment, confusion matrices were employed to evaluate the effectiveness of the proposed approach in distinguishing between predicated and not non-predicated instances, as illustrated in Fig. 4. The model demonstrated a notable average accuracy of 96.12% across all datasets, indicating a high true positive rate while maintaining a low false positive rate across various classification thresholds.

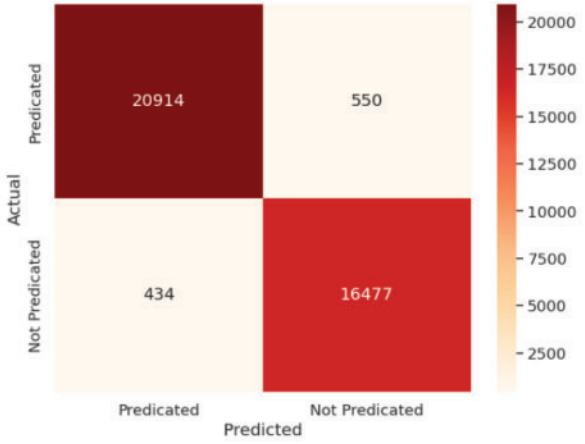
The Receiver Operating Characteristic (ROC) curves of Fig. 5 also validate the effectiveness of the proposed SESDP model. For SB-DSSI, the ROC curve shows an area under the curve (AUC) of 0.88, with

macro-average and micro-average ROC values close to 0.92, indicating strong model performance. In SB-DSSII and SB-DSSIII, the AUC values improve to 0.94 and 0.92, respectively, reflecting the model's consistent ability to discriminate between classes. The ROC curves confirm that the proposed model not only achieves high accuracy but also maintains a reliable balance between sensitivity and specificity, making it highly suitable for practical applications in software defect prediction.

(a) Confusion Matrix of SB-DSSI



(b) Confusion Matrix of SB-DSSII



(c) Confusion Matrix of SB-DSSIII

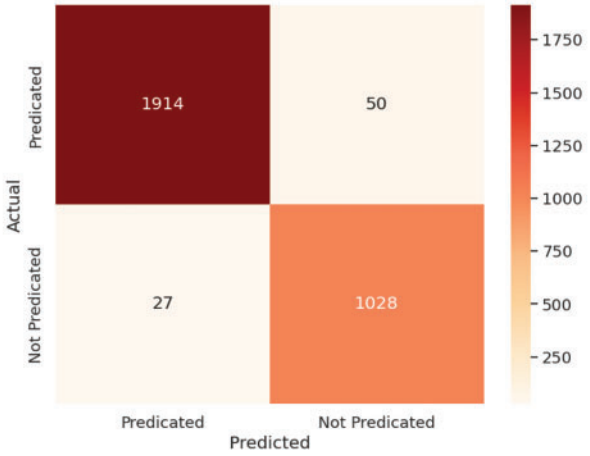
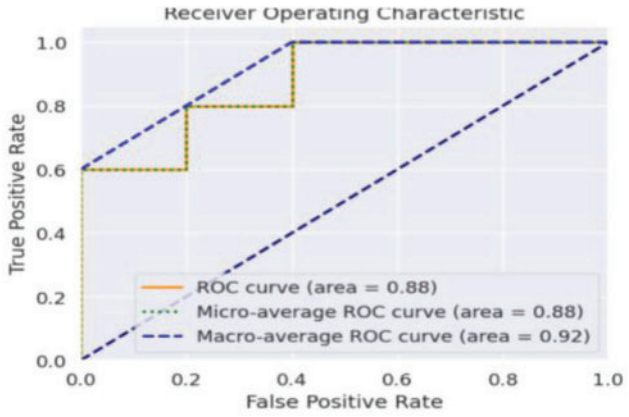
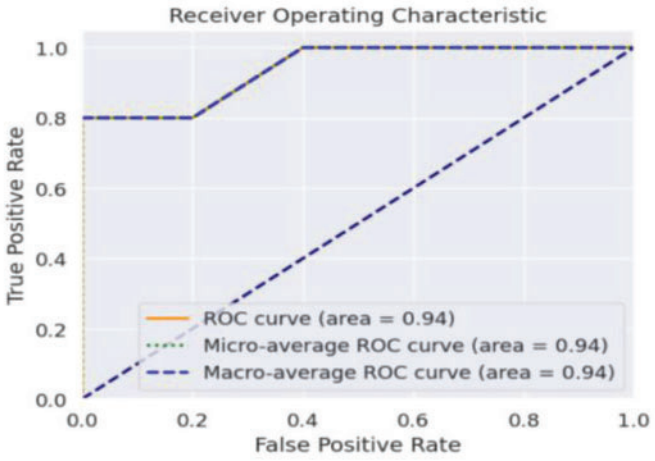


Figure 4: Confusion matrix of SB-DSSI, SB-DSSII and SB-DSSIII

(a) ROC Curve of Proposed Model on SB-DSSI



(b) ROC Curve of Proposed Model on SB-DSSII



(c) ROC Curve of Proposed Model on SB-DSSIII

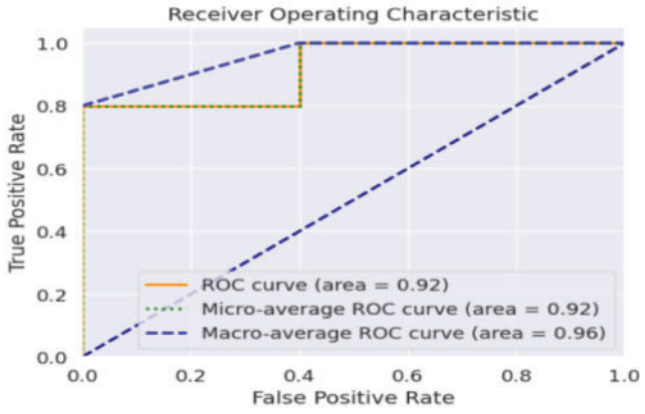


Figure 5: ROC curve of SB-DSSI, SB-DSSII and SB-DSSIII

The proposed Sentiment Analysis-Based Early Software Defect Prediction (SESDP) model is also compared with the baseline models as shown in Fig. 6. The SESDP model, which integrates a Transformer-Based Multi-Task Learning approach using RoBERTa for defect prediction and sentiment analysis, outperforms the baselines across all key metrics. The comparative performance results indicate that the proposed SESDP model achieves superior accuracy, precision, and recall. Specifically, the SESDP model attains an accuracy of 96.37%, a precision of 94.7%, and a recall of 95.4%, which represent substantial improvements over the baseline models. Zhang et al.'s model, for instance, achieves an accuracy of 89.23%, precision of 87.28%, and

recall of 89.21%. Similarly, Draz et al.'s model shows only marginal gains with an accuracy of 90.15% and lower recall at 87.2%. Even Balasubramaniam et al.'s optimized CNN model, which improves on traditional models with an accuracy of 93.49%, is outperformed by the SESDP model.

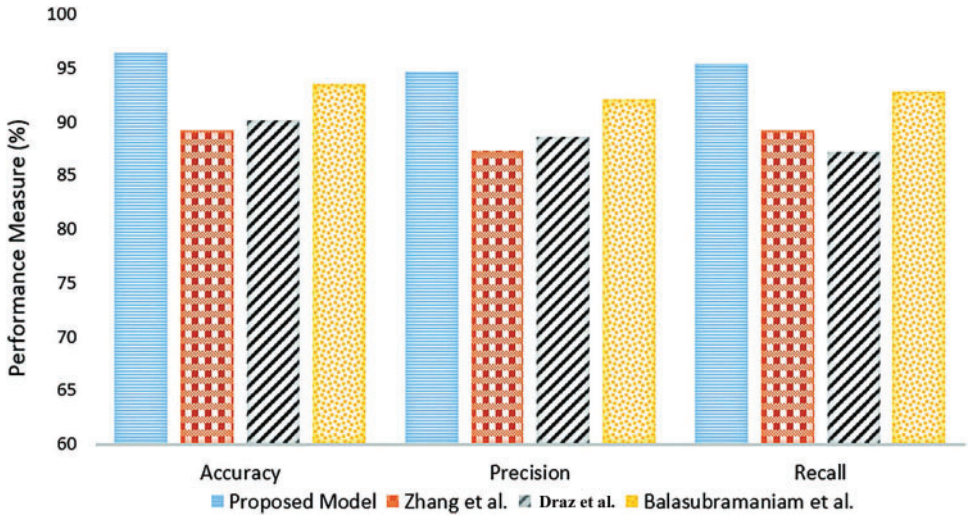


Figure 6: Comparison with baseline approaches in terms of Accuracy, Precision, and Recall [28,29,30]

The higher precision and recall of the proposed SESDP model indicate that it is more effective in identifying true positives and reducing false positives and false negatives. This suggests that the integration of sentiment analysis, which captures contextual information from user reviews, provides a significant advantage over traditional approaches that rely solely on statistical or machine learning techniques. In contrast to the baseline models, the SESDP model's ability to process and analyze sentiment scores alongside defect prediction ensures that the nuances of user-generated content are leveraged to enhance prediction accuracy. This results in more reliable defect predictions, even in cases where data is imbalanced or complex, challenges that often undermine the performance of baseline models.

The log loss comparison of the proposed SESDP model with the baseline models (Zheng et al. [28], Draz et al. [29], Balasubramaniam et al. [30]) across three datasets—SB-DSSI, SB-DSSII, and SB-DSSIII—demonstrates the superior performance of the SESDP model as shown in Table 3. In the SB-DSSI dataset, the SESDP model achieved the lowest log loss at 0.325, outperforming Zhang et al. (0.340), Draz et al. (0.530), and Balasubramaniam et al. (0.380). This result indicates that SESDP is more accurate and makes less uncertain predictions compared to the baselines. Similarly, in the SB-DSSII dataset, SESDP recorded a log loss of 0.335, once again surpassing Zhang et al. (0.350), Draz et al. (0.540), and Balasubramaniam et al. (0.405). The trend continues in the SB-DSSIII dataset, where the SESDP model achieved a log loss of 0.330, while Zhang et al., Draz et al., and Balasubramaniam et al. recorded 0.355, 0.515, and 0.398, respectively.

This comparative analysis highlights the effectiveness of SESDP in providing more accurate and reliable predictions with reduced uncertainty. The higher log loss values of Draz et al.'s model across all datasets suggest that it struggles with confident predictions, while Zhang et al.'s model, although better than He et al.'s, still lags behind SESDP. Balasubramaniam et al.'s model shows reasonable performance but is consistently outperformed by SESDP, particularly in the SB-DSSII dataset, where SESDP achieves a significantly lower log loss.

Table 3: Log loss comparison of proposed model with baselines

Dataset	Zheng et al. [28]	Draz et al. [29]	Balasubramaniam et al. [30]	SESDP (Proposed model)
SB-DSSI	0.340	0.530	0.380	0.325
SB-DSSII	0.350	0.540	0.405	0.335
SB-DSSIII	0.355	0.515	0.398	0.330

An important aspect of validating the SESDP model's effectiveness is understanding the contribution of its various components. To achieve this, an ablation study was conducted by progressively removing or modifying core elements of the model architecture. The goal of this analysis was to evaluate the significance of each component by examining its impact on the overall model performance. By disabling specific features and comparing the results with the fully integrated SESDP model, we can determine which components are essential for maximizing accuracy, precision, and recall in defect prediction tasks. Each of these configurations was evaluated using standard performance metrics: accuracy, precision, and recall. The results are presented in Table 4.

Table 4: Ablation study

Configurations	Accuracy (%)	Precision (%)	Recall (%)
Full SESDP model	96.37	94.7	95.4
With BERT	94.85	93.2	92.8
Without softmax	91.68	90.4	89.6
Without text embeddings	89.54	88.2	87.3

The ablation study reveals that the text embedding extraction using RoBERTa is the most important component of the SESDP model, as its removal results in the largest performance drop, reducing accuracy to 89.54%. This underscores the importance of capturing contextual information from user reviews for accurate software defect prediction. The softmax function also plays a significant role, with its exclusion leading to a notable decrease in accuracy to 91.68%, showing its importance in generating reliable classification probabilities. Additionally, the comparison between RoBERTa and BERT demonstrates that while BERT is effective, RoBERTa provides superior performance, likely due to its enhanced handling of language features and fine-tuning. Overall, the study highlights that each component contributes to the model's overall accuracy, and the full integration of all elements delivers the best results.

5 Conclusion and Future Work

The SESDP model presents significant advancement in the field of software defect prediction by leveraging sentiment analysis of social media reviews. Through the integration of Transformer-Based Multi-Task Learning with RoBERTa, SESDP effectively addresses the limitations of existing models, such as handling imbalanced datasets and capturing the contextual nuances of user feedback. The experimental results demonstrate that SESDP outperforms traditional methods in terms of accuracy, precision, and recall, validating its effectiveness in early defect detection. By utilizing real-time user-generated content, SESDP provides a proactive approach to maintaining software quality, ultimately reducing the cost and time associated with identifying and fixing defects. Future research directions for SESDP could involve exploring the integration of additional data sources, such as bug-tracking systems and developer comments,

to further enhance the model's predictive capabilities. Additionally, investigating the application of SESDP in different domains and software types could provide insights into its generalizability and adaptability. Further refinement of the model's architecture, particularly in optimizing the balance between sentiment analysis and defect prediction tasks, could also lead to even greater improvements in prediction accuracy and efficiency. Finally, implementing advanced techniques for real-time data processing and incorporating user feedback loops could make SESDP more responsive and dynamic in rapidly changing software environments.

Acknowledgement: This research was supported by a grant from the Center of Excellence in Information Assurance (CoEIA), King Saud University (KSU).

Funding Statement: This research work is funded by a grant from the Center of Excellence in Information Assurance (CoEIA), King Saud University (KSU).

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Farah Mohammad; data collection: Saad Al-Ahmadi; analysis and interpretation of results: Jalal Al-Muhtadi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All data generated or analyzed during this study are included in this published article.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Qiao L, Li X, Umer Q, Guo P. Deep learning based software defect prediction. Neurocomputing. 2020;385(2):100–10. doi:10.1016/j.neucom.2019.11.067.
2. Muhammad ZZ, Sapiah S, Asmak INH. Application of deep learning in software defect prediction: systematic literature review and meta-analysis. Inf Softw Technol. 2023;158:107175. doi:10.1016/j.infsof.2023.107175.
3. Feng S, Keung J, Yu X, Xiao Y, Bennin KE, Kabir MA, et al. COSTE: complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. Inf Softw Technol. 2021;129(1):106432. doi:10.1016/j.infsof.2020.106432.
4. Feng S, Keung J, Yu X, Xiao Y, Zhang M. Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Inf Softw Technol. 2021;139(06):106662. doi:10.1016/j.infsof.2021.106662.
5. Gong L, Zhang H, Zhang J, Wei M, Huang Z. A comprehensive investigation of the impact of class overlap on software defect prediction. IEEE Trans Softw Eng. 2023;49(4):2440–58. doi:10.1109/TSE.2022.3220740.
6. Jin C. Cross-project software defect prediction based on domain adaptation learning and optimization. Expert Syst Appl. 2021;171(1):114637. doi:10.1016/j.eswa.2021.114637.
7. Zhao Y, Damevski K, Chen H. A systematic survey of just-in-time software defect prediction. ACM Comput Surv. 2023;55(10):1–35. doi:10.1145/3567550.
8. Sharma T, Jatain A, Bhaskar S, Pabreja K. Ensemble machine learning paradigms in software defect prediction. Procedia Comput Sci. 2023;218(5):199–209. doi:10.1016/j.procs.2023.01.002.
9. Giray G, Bennin KE, Köksal Ö, Babur Ö, Tekinerdogan B. On the use of deep learning in software defect prediction. J Syst Softw. 2023;195(11):111537. doi:10.1016/j.jss.2022.111537.
10. Khalid A, Badshah G, Ayub N, Shiraz M, Ghouse M. Software defect prediction analysis using machine learning techniques. Sustainability. 2023;15(6):5517. doi:10.3390/su15065517.
11. Rizwan Rashid Rana M, Ur Rehman S, Nawaz A, Ali T, Imran A, Alzahrani A, et al. Aspect-based sentiment analysis for social multimedia: a hybrid computational framework. Comput Syst Sci Eng. 2023;46(2):2415–28. doi:10.32604/csse.2023.035149.

12. Wu W, Wang S, Liu B, Shao Y, Xie W. A novel software defect prediction approach via weighted classification based on association rule mining. Eng Appl Artif Intell. 2024;129(2):107622. doi:10.1016/j.engappai.2023.107622.
13. Ali M, Mazhar T, Arif Y, Al-Otaibi S, Ghadi YY, Shahzad T, et al. Software defect prediction using an intelligent ensemble-based model. IEEE Access. 2024;12(2):20376–95. doi:10.1109/ACCESS.2024.3358201.
14. Chen J, Xu J, Cai S, Wang X, Chen H, Li Z. Software defect prediction approach based on a diversity ensemble combined with neural network. IEEE Trans Reliab. 2024;73(3):1487–501. doi:10.1109/TR.2024.3356515.
15. Tang Y, Dai Q, Du Y, Chen L, Niu X. A software defect prediction method based on learnable three-line hybrid feature fusion. Expert Syst Appl. 2024;239(A):122409. doi:10.1016/j.eswa.2023.122409.
16. Ismail AM, Hamid SHA, Sani AA, Daud NNM. Toward reduction in false positives just-in-time software defect prediction using deep reinforcement learning. IEEE Access. 2024;12:47568–80. doi:10.1109/ACCESS.2024.3382991.
17. Khleel NAA, Nehéz K. Software defect prediction using a bidirectional LSTM network combined with oversampling techniques. Clust Comput. 2024;27(3):3615–38. doi:10.1007/s10586-023-04170-z.
18. Ali M, Mazhar T, Al-Rasheed A, Shahzad T, Yasin Ghadi Y, Khan MA. Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. PeerJ Comput Sci. 2024;10(17):e1860. doi:10.7717/peerj-cs.1860.
19. Li L, Su R, Zhao X. Neighbor cleaning learning based cost-sensitive ensemble learning approach for software defect prediction. Concurr Comput. 2024;36(12):e8017. doi:10.1002/cpe.8017.
20. Guo Y, Shepperd M, Li N. Improving classifier-based effort-aware software defect prediction by reducing ranking errors. In: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering; 2024; Salerno, Italy: ACM. p. 160–9. doi:10.1145/3661167.3661195.
21. Zhang Q, Zhang J, Feng T, Xue J, Zhu X, Zhu N, et al. Software defect prediction using deep Q-learning network-based feature extraction. IET Softw. 2024;2024(1):3946655. doi:10.1049/2024/3946655.
22. Zhu K, Zhang N, Jiang C, Zhu D. IMDAC: a robust intelligent software defect prediction model via multi-objective optimization and end-to-end hybrid deep learning networks. Softw Pract Exp. 2024;54(2):308–33. doi:10.1002/spe.3274.
23. Alsaedi T, Rizwan Rashid Rana M, Nawaz A, Raza A, Alahmadi A. Sentiment mining in e-commerce. Int J Electr Comput Eng Syst. 2024;15(8):641–50. doi:10.32985/ijeces.15.8.2.
24. Choo S, Kim W. A study on the evaluation of tokenizer performance in natural language processing. Appl Artif Intell. 2023;37(1):2175112. doi:10.1080/08839514.2023.2175112.
25. Rana MRR, Nawaz A, Ali T, Alattas AS, AbdElminaam DS. Sentiment analysis of product reviews using transformer enhanced 1D-CNN and BiLSTM. Cybern Inf Technol. 2024;24(3):112–31. doi:10.2478/cait-2024-0028.
26. Galal O, Abdel-Gawad AH, Farouk M. Rethinking of BERT sentence embedding for text classification. Neural Comput Appl. 2024;36(32):20245–58. doi:10.1007/s00521-024-10212-3.
27. Wolff B, Seidlmayer E, Förstner KU. Enriched BERT embeddings for scholarly publication classification. In: International Workshop on Natural Scientific Language Processing and Research Knowledge Graphs; 2014; Cham: Springer Nature Switzerland. p. 234–43.
28. Zheng L, He Z, He S. An integrated probabilistic graphic model and FMEA approach to identify product defects from social media data. Expert Syst Appl. 2021;178(6):115030. doi:10.1016/j.eswa.2021.115030.
29. Draz MM, Emam O, Azzam SM. Software cost estimation predication using a convolutional neural network and particle swarm optimization algorithm. Sci Rep. 2024;14(1):13129. doi:10.1038/s41598-024-63025-8.
30. Balasubramaniam DS, Gollagi DSG. Software defect prediction via optimal trained convolutional neural network. Adv Eng Softw. 2022;169(5):103138. doi:10.1016/j.advengsoft.2022.103138.