Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.083166

ARTICLE

HAR-MLP: A Hybrid Attention–Residual MLP Architecture with Handcrafted Features for Software Bug Prediction

Isil Karabey Aksakalli*

Department of Computer Engineering, Erzurum Technical University, Erzurum, Türkiye

*Corresponding Author: Isil Karabey Aksakalli. Email: isil.karabey@erzurum.edu.tr

Received: 30 March 2026; Accepted: 16 June 2026; Published: 13 August 2026

ABSTRACT: Open-source platforms and issue tracking systems such as GitHub and Jira generate large volumes of issue reports and code changes, making effective bug identification a challenging task. This study investigates software bug prediction by integrating various feature extraction methods, including Word2Vec, TF-IDF, FastText, GloVe, and Doc2Vec, with several lightweight ML algorithms. Hybrid feature sets are further enhanced using Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) and empirical results indicate that Word2Vec and Multi-Layer Perceptron (MLP) provide comparatively stronger performance. The study proposes a Hybrid Attention-Residual Multilayer Perceptron (HAR-MLP) model to automatically classify software issue reports as bugs and feature requests. The model is equipped with a feature set consisting of a combination of TF-IDF reweighted embedding averages derived from Word2Vec embedding vectors, mean-maximum-standard deviation pooling statistics, and DCT spectral coefficients. This feature fusion is then combined with context-sensitive representations obtained from a multi-headed self-attention mechanism and fed as input to the Residual MLP model as a classifier. While traditional MLP has direct forward connections between layers, Residual MLP reduces gradient decay and prevents information loss in deep layers by combining intermediate layer outputs with the input using the skip connection method. This structure increases the model's learning capacity and provides higher accuracy, especially in hybrid structures where attention outputs and handcrafted features are processed together. Experimental results demonstrate that the HAR-MLP model achieves higher performance than traditional feature extraction and classification methods while remaining computationally competitive compared to pre-trained transformer-based approaches. Moreover, cross-platform evaluation on three independent Jira issue datasets further demonstrates the generalizability of the proposed architecture, achieving macro-F1 scores ranging from 86.61% to 97.48%. The source code of the proposed HAR-MLP model can be accessed via the following link: https://github.com/isilkarabeyaksakalli/HAR-MLP.

KEYWORDS: Software bug prediction; feature extraction; transformers; residual MLP; hybrid attention mechanism

1 Introduction

In software engineering, many open-source platforms have been developed where developers can collaborate, announce their work and participate in creative discussions in order to identify and fix software bugs in a timely manner, to use the features in applications in different projects and to get answers to the questions asked in case of difficulties. Github is one of the most common platforms used by software developers as an open source code repository. Although open source code platforms allow developers to collaboratively figure out software-related problems, the topics of commit messages are often indistinguishable due to information overload. Categorizing the commit messages associated with a bug, question or feature from a large number of posts is crucial for timely identification of software bugs. As a solution to the software bug prediction

problem, many studies in the literature focus on predicting issues on Github repository and apply natural language processing, ML and deep learning techniques by utilizing frequency-based vectorizations and word embedding methods to extract hidden semantic information in short texts obtained from open source projects [1–4]. However, word embeddings and frequency-based vectorization methods often face limitations in performance when used in isolation, as they struggle to address linguistic complexities such as polysemy (the multiple meanings of a word) and syntactic variations like word order inversions [5].

Rather than relying on computationally heavy architectures such as BERT or traditional ML-based models, HAR-MLP is built around a different premise such as semantic, statistical, and spectral information are drawn from the same token embeddings and merged into a single feature vector before classification. By using lightweight Residual MLP skip connections, which have a lower computational load compared to transformer models, training is accelerated and gradient distortions in deeper layers are prevented by protecting the model from overlearning. For feature extraction, Word2Vec, which achieved the highest performance from raw embedding vectors in preliminary experiments, is applied to handcrafted features as the embedding backbone. First, the output vectors are reweighted according to IDF scores to increase the weight of rare terms in the documents. Secondly, the mean, maximum, and standard deviation pooling statistics of the vectors obtained from the Word2Vec backbone are calculated, and in the final step, the token sequence is passed through DCT coefficients, combining the three feature sets to achieve high performance with low computational cost.

To the best of our knowledge, existing studies have not directly addressed this combination of handcrafted features and self-attention for large-scale, text-based open-source bug prediction tasks. Existing research in related domains suggests that integrating attention mechanisms with rich handcrafted features can enhance predictive performance in software defect prediction [6], yet such approaches have not been applied to open-source bug prediction from raw issue text.

The main contributions of this study are listed in the following:

- A Hybrid Attention-Residual MLP (HAR-MLP) architecture is proposed, which combines handcrafted statistical and spectral features with a self-attention branch to jointly capture global distributional patterns and localized contextual dependencies.
- The proposed model offers a handcrafted feature set by combining TF-IDF weighted average to give more weight to key words across Word2Vec word vectors, multiple pooling to extract statistical profiles of the text, and DCT coefficients to retain positional information of the words.
- The word order and semantic, syntactic relationships and long-range dependencies that cannot be fully preserved by pooling operations are explicitly modeled through a multi-head self-attention mechanism.
- Handcrafted features and attention output are combined and fed into Residual MLP, allowing for more balanced learning of the model through label smoothing, and generalization is expanded through MixUp optimization.
- The proposed HAR-MLP model offers similar accuracy rates to pre-trained models with millions of parameters and high computational complexity when compared to computationally intensive transformer architectures and the model enables the development of much faster and more scalable systems.

The remainder of the paper is organized as follows. In Section 2, related work regarding the software bug prediction in the literature is presented and compared the methods proposed in this field. In Section 3, the proposed methodology along with the open-source datasets used are detailed. This section also covers feature aggregation, the HAR-MLP mechanism, and the architecture overview, which is evaluated based on the performance of these techniques. Section 4 presents a comprehensive analysis of the methods and discusses the comparative results. Finally, the paper concludes with a discussion and conclusion in the Sections 5 and 6.

2 Related Work

2.1 Feature Extraction and Classification in Natural Language Processing

Software bug prediction is a significant research area in software engineering. Open-source platforms such as GitHub and Jira, which are extensively utilized by developers, face the challenge of information overload. On these platforms, developers upload their work, and relevant bugs are addressed by appropriate contributors. However, it is often unclear whether the information presented represents a bug, a question, or a feature. To address this issue, various approaches have been proposed for the automatic classification of information on GitHub. These approaches include feature extraction and classification methods such as statistical methods (TF, TF-IDF), word embedding models (Word2Vec, FastText, Glove), transformation methods (DCT, DWT), ML methods (Naive Bayes (NB), Random Forest (RF), Multilayer Perceptron (MLP), Decision Tree (DT)), evolutionary algorithms (genetic, Rao optimization algorithm [7], particle swarm optimization [8]), and transformer architectures (BERT, RoBERTA, LSTM, GPT, Graph Neural Network-GNN).

Statistical methods and vector-based approaches have gained popularity in recent years in natural language processing and ML based approaches such as sentiment analysis, question and answer tasks, text classification and large language models. Frequency-based statistical methods primarily analyze word frequencies while often disregarding the contextual meanings of words within sentences. Term Frequency (TF), Term Frequency-Inverse Document Frequency (TF-IDF), and Bag of Words (BoW) are examples of straightforward and computationally efficient frequency-based statistical methods commonly used for document embedding. The simplistic nature of these methods and the fact that they do not consider the semantic and syntactic information has encouraged the development of more sophisticated, vector-based approaches to obtain word and document embeddings. Among vector-based approaches, word embedding models operate by transforming words into vector representations, enabling contextually similar words to be positioned closer together within the vector space. Word embeddings are a well-established textual representation method consisting of expressing words as vectors in a multidimensional plane, especially in natural language processing and ML methods such as sentiment analysis, text classification, question and answer systems, and large language models [5]. In the word embeddings method, of which there are various types including Word2Vec [9], fastText [10] and GloVe [11], words that are contextually similar in a vector space have closer embeddings to each other by calculating a previously defined distance metric. Another vector-based approach, Doc2Vec proposed by Le and Mikolov [12], extends Word2Vec by aggregating word embeddings to generate a unique representation for entire documents. This model considers embeddings at the document level, finds similarity from word sequences by converting paragraphs in the document into vectors instead of handling the words [13]. Although word embedding methods are widely used in natural processing and text classification, they do not take into account that a word can have different meanings in different contexts due to the polysemy problem. Various vectorization approaches, such as Word2Vec, fastText, and GloVe, assign a single static vector representation to each word, limiting their ability to capture context-dependent variations. Furthermore, since word order and syntactic structures are disregarded in these approaches, grammatical relationships within sentences cannot be modeled. While Doc2Vec extends the Word2Vec model at the document level, it similarly misses fine-grained contextual nuances. For these reasons, their performance is significantly lower than transformer-based embedding approaches, especially with large-scale datasets.

Along with these feature extraction methods, Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) methods, which are mostly used in image processing, but have also been used in natural language processing in recent years to analyze word and sentence embeddings and represent semantic

information in an embedding vector [3,14,15]. Using these approaches, datasets represented by feature vectors can be classified using various ML algorithms.

In recent years, significant advancements have been achieved in natural language processing, particularly in model architecture, pre-training techniques, and ML approaches. Transformer architectures have emerged as pre-trained, high-capacity models, driving further progress in the field [16]. Transformer architectures represent an open-source library and community initiative aimed at facilitating user access to large-scale pre-trained models. This platform enables users to build upon, experiment with, and deploy these models in various applications, achieving state-of-the-art performance in subsequent tasks. Transformer-based pre-trained models such as BERT [17], RoBERTA [18], Sentence-BERT [19], and GPT [20] are able to capture contextual information from word or sentence-based embedded vectors using a bidirectional attention mechanism, thus exhibiting high performance. However, due to the number of parameters and computational complexity, the feature extraction processes are quite extensive, and integrating these models into systems with limited computational resources is a significant challenge. To address this problem, instead of transformer models, a feature extraction and ML-based MLP classifier built on lightweight Word2Vec embedded vectors is used in the feature extraction and classification phases. On the other hand, the multi-head attention mechanism is also compactly integrated into the proposed method to handle contextual relationships among the words. Moreover, handcrafted features (TF-IDF weighted embeddings, multi-pooling statistics, and DCT coefficients) provide complementary semantic, statistical, and spectral cues that improve data efficiency and interpretability, while the residual MLP stabilizes optimization with far fewer parameters than transformer-based alternatives.

2.2 Software Bug Identification and Prediction

Software bug detection and prediction remains a complex and time-intensive challenge for software engineers. Numerous methodologies have been proposed in the literature to address this issue. For example, Kallis et al. [21] developed a TicketTagger tool that automatically recognizes the types of reports submitted with a pre-trained fastText model to track issue handling activities in the Github repository and analyzes the issue title and descriptions using ML techniques to assign tags to each issue. The prediction performance of the tool was evaluated on about 30,000 Github issues and the experimental results show that TicketTagger gives at least 20% better F1 scores for each issue type than the J48 tool. In another study on Github issue classification, Bharadwaj and Kadam [2] proposes a neural architecture to classify open source Github issues into three types such as bug, enhancement and question using BERT-Style models and contextual embeddings. As a result of the experiments, the proposed architecture achieves better results than the competition organizer's prediction with an F1 score of 86.53%.

Tambe and Ragha [3] developed a dynamic feature selection method for predicting high and low priority bug categories and proposed an ensemble deep learning classifier with feature selection that provides high efficiency of multi-domain bug prediction in different use cases. This model is a combination of Deep Random Forest, kNN, Logistic Regression, Multilayer Perceptron, Support Vector Machine and 1D Convolutional Neural Network classifiers and uses DFT, DCT and Convolutional Features for feature extraction. In the feature selection layer, variance-based selection is applied to the Alpha, Beta, Gamma, and Delta Wolves. According to the experimental results, the proposed model achieves an accuracy of 99.9%. In another study on feature selection using deep learning and adaptive golden eagle optimizer [22] Long-Short-Term-Term Memory (LSTM) based recurrent neural network is proposed for bug prediction and the proposed method achieved the highest accuracy rates of 92.8% and 93.41% on NASA and PROMISE datasets. Panda and Nagwani [23] collected data from software bug repositories such as Mozilla, Eclipse, Apache and Netbeans and labeled them with severity and priority labels. To address the issue of class imbalance, the

authors employed the SMOTE method to generate synthetic data by utilizing feature space similarities within the underrepresented class. A Heuristic Fuzzy Similarity Measurement (IFSM) based severity estimation and priority estimation technique is proposed to estimate the severity and priority of bugs. Using NLP technique, severity-term and priority-term dictionaries are created by extracting the most frequently used terms. As a result of the experiments, it is stated that the accuracy rates of 92% and above are achieved in all repos, outperforming other state-of-the-art models. Kumar and Chaturvedi [24] proposes a reward-based weighted majority voting (WMV) ensemble technique to overcome the problem that all base classifiers alone in the simple majority voting (SMV) ensemble method may produce equal weights and to improve the performance of SMV. A basic classifier (BC) obtains a higher weight when it predicts more classes correctly. The proposed WMV ensemble method is compared with several ML techniques, including the SMV method. Experimental results demonstrate that the WMV method generally outperforms state-of-the-art majority-based methods in terms of accuracy, F-measure, and Matthews correlation coefficient. In another study applying ensemble learning [25], hyperparameter optimization and ensemble learning are applied to Vote, Bagging, Random Forest, AdaBoostM1 and Logistic Regression ML methods and it is observed that the performance of the methods increases according to the default parameters. Siachos et al. [5] proposed a Graph Attention Network based model by representing each text containing bug and feature features obtained from open source Github [26] and Jira repositories as a graph. This proposed model is compared with different graphs using accuracy, precision and recall metrics and it is seen that the proposed model reaches the highest accuracy values of 80.22% in Github and 95.64% in Jira datasets.

In this study, the proposed HAR-MLP model, tested on the open-source GitHub Bugs Prediction Dataset [26], achieved a *5.18%* higher test accuracy compared to the work by Siachos et al. [5], which is the study in the literature utilizing this dataset through graph neural networks. The proposed model also has at least 2.4% higher F1 score when compared to the results obtained from various feature extraction algorithms and ML methods.

3 The Proposed Approach

In this study, we propose a Hybrid Attention–Residual MLP (HAR-MLP) architecture that combines a self-attention mechanism with handcrafted statistical features to classify previously labeled texts as *bug* or *feature* reports from the title and description fields of open-source GitHub projects. The architecture is designed to leverage both contextual sequence representations and diverse non-contextual statistical descriptors for improved classification accuracy. The overall structure of the architecture and its phases are described in the following sub-sections.

3.1 Architecture Overview

In the architectural design, two parallel branches are applied as shown in the Fig. 1 for text-based feature extraction:

- *Sequence-Based Attention Branch*: Each pre-processed token sequence is mapped to a vector space via a Word2Vec vectorization model trained with Continuous Bag of Words (CBOW). The embedded vectors obtained after this application are passed through a multi-head self-attention mechanism to model semantic dependencies and relationships.
- *Handmade Feature Branch*: Simultaneously with the order-based attention branch, statistical pooling descriptors and DCT spectral coefficients are calculated from the same Word2Vec embedded vectors to find TF-IDF weighted embedded vector means, mean, standard deviation, and maximum values. This combination of features enables the evaluation of semantic information, which is not sufficient for the attention branch alone.

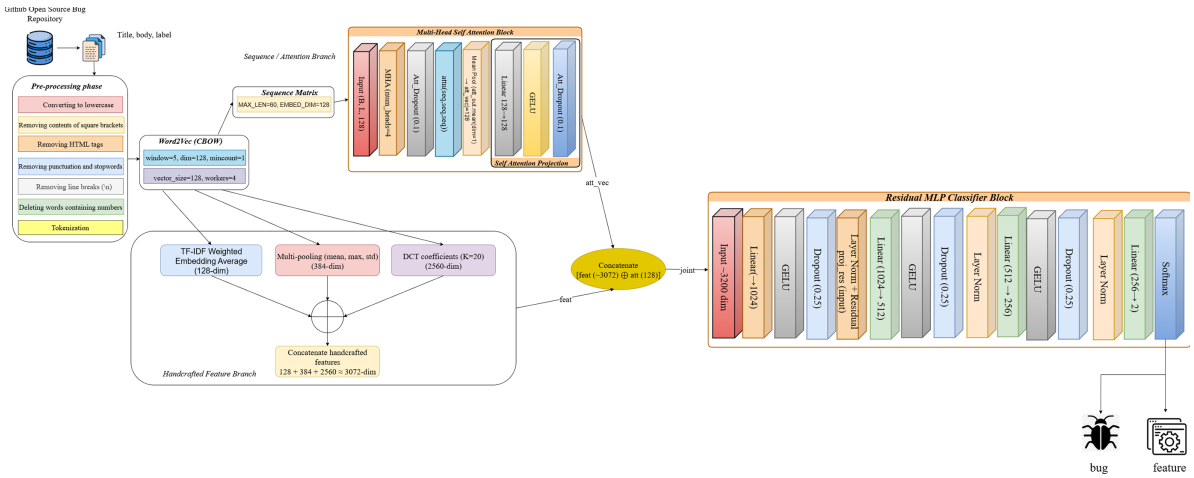


Figure 1: Architecture overview of the proposed HAR-MLP.

The outputs of both branches are combined along the feature dimension to create a unified representation and are given as input to the Residual Multilayer Perceptron (MLP) classifier. Thus, this two-branch design is expected to generate more distinct class boundaries by leveraging the statistical efficiency of lightweight ML and the sequence modeling capacity of the attention mechanism.

3.2 Pre-Processing Phase

This section describes the pre-processing steps applied to the datasets used in this study. The purpose of the pre-processing steps is to improve the classification efficiency of the structured text, remove unnecessary expressions, and provide the text as a cleaner input to the model. These steps are described in the following:

(A) Text Cleaning: A special function is used to clean the text, and this cleaning process is applied to each entry in the dataset.
- *Lowercase Conversion*: All text is converted to lowercase to standardize case sensitivity.
- *Removing Content from Square Brackets*: Information within square brackets is removed, eliminating redundant information.
- *Eliminating URLs*: Internet links are removed to exclude them from vectorization.
- *Removing HTML Tags*: HTML elements that do not contribute to text analysis are filtered.
- *Removing Words Containing Numbers*: Words containing numbers are removed to focus on meaningful text.

(B) Punctuation, Escape Character, and Stopword Removal: When the text is tokenized, punctuation marks, escape characters, and pause words are removed first. Common pause words are removed using a pre-defined list of words for English (e.g., the, is, a, in) from the NLTK library. Additionally, unnecessary escape characters like \\r are also removed as these characters do not convey any meaningful information within the text.

3.3 The Proposed Model

Fig. 1 illustrates the overall structure of HAR-MLP, which concatenates multi-head self-attention and a set of statistical and frequency-based handcrafted features for bug report classification. The attention component operates over token sequences from commit messages or code comments, picking up semantic dependencies between individual tokens, while the handcrafted features encode statistical and temporal-frequency properties of the same text. This combination improves the model's ability to handle both small and

medium-sized datasets where contextual and syntactic cues are each relevant. The model proceeds through the following steps in order:

1. **Token Embedding with Word2Vec:** In the first step, each text consisting of commit messages and code annotation is tokenized and stop word elimination is applied to remove the common English words. Then the remaining tokens are embedded using the Continuous Bag-of-Words (CBOW) variant of Word2Vec in Eq. (1):

$$E(w) : \mathcal{V} \to \mathbb{R}^d \tag{1}$$

producing a fixed-size sequence matrix S_i for each sample i in Eq. (2):

$$S_i = \text{PadTrunc}\left([E(w_{i1}), \ldots, E(w_{i\ell_i})], L\right) \in \mathbb{R}^{L\times d} \tag{2}$$

where L represents the maximum sequence length. Using this formula, semantic meaning is preserved which also enforces a uniform input size expected by downstream components.

2. **Handcrafted Feature Extraction:** For the handcrafted feature extraction, three feature sets are computed using S_i:
   - *TF-IDF weighted average embedding:* As a first feature, Word2Vec word vectors are reweighted according to their IDF scores using the TF-IDF weighted average embedding method. This gives higher influence to rare and informative words while reducing the impact of uninformative words on classification. The IDF scores serve solely as scalar weights shown in Eq. (3):

$$x_i^{(1)} = \frac{\sum_{w\in T_i} \text{idf}(w)\, E(w)}{\sum_{w\in T_i} \text{idf}(w)} \in \mathbb{R}^d \tag{3}$$

   where $E(w) \in \mathbb{R}^d$ is the Word2Vec embedding of token w, and $\text{idf}(w)$ is its inverse document frequency score. This is fundamentally different from standalone TF-IDF vectorization, which produces sparse frequency-based representations without semantic grounding.
   - *Multi-pooling statistics:* These statistics aggregate the Word2Vec embedding matrix $S_i \in \mathbb{R}^{L\times d}$ along the token axis to capture complementary global distributional properties of the sequence. Specifically, the mean pooling summarizes the average semantic direction of the token embeddings, max pooling retains the most activated feature in each embedding dimension across all tokens, and standard deviation pooling encodes the spread or diversity of token representations within the sequence shown in Eq. (4):

$$x_i^{(2)} = \left[\text{mean}(S_i),\ \max(S_i),\ \text{std}(S_i)\right] \in \mathbb{R}^{3d} \tag{4}$$

   where each operation is applied column-wise (axis = 0) over the L token embeddings, yielding a d-dimensional vector per statistic and a concatenated representation of size $3d$. Out-of-vocabulary tokens are represented as zero vectors to maintain a fixed-length output. This multi-pooling strategy ensures that both dominant and distributional patterns in the Word2Vec embedding space are captured, providing complementary cues to the TF-IDF weighted average and the DCT coefficients.
   - *DCT coefficients:* The coefficients transform the Word2Vec embedding matrix $S_i \in \mathbb{R}^{L\times d}$ into the frequency domain to capture global sequential patterns across token positions. The sequence is first padded or truncated to a fixed length of $L = 60$ tokens, with out-of-vocabulary tokens represented as zero vectors. The Type-II DCT with orthonormal normalization is then applied column-wise

(axis = 0), treating each of the d embedding dimensions as an independent discrete signal over token positions shown in Eq. (5):

$$C_i = \mathrm{DCT}(S_i, \mathrm{axis} = 0, \mathrm{norm} = \text{`ortho'}) \in \mathbb{R}^{L \times d} \tag{5}$$

Only the first $K = 20$ low-frequency coefficients are retained per embedding dimension, discarding higher-frequency components that correspond to rapid local fluctuations across token positions shown in Eq. (6):

$$C_i^{(K)} = C_i[0 : K, :], \quad x_i^{(3)} = \mathrm{vec}(C_i^{(K)}) \in \mathbb{R}^{Kd} \tag{6}$$

With this equation, a compact spectral representation of the document's embedding trajectory is obtained, and the low-frequency coefficients reflect broad, slowly shifting semantic patterns along the token sequence just like DCT-based dimensionality reduction in signal and image processing [14,15].

In this study, the selection of DCT as a third feature instead of DWT to ensure spectral discriminability is determined by the preliminary experiments. According to the preliminary experimental results, while the Word2Vec-DCT method has an accuracy of 82.87% with the MLP classifier, Word2Vec-DWT gives an accuracy of 81.16% with the same classifier. To maintain computational efficiency while preserving spectral discriminability, DCT alone was selected for integration into the HAR-MLP handcrafted feature branch.

The selected three complementary feature sets are concatenated into a single handcrafted vector shown in Eq. (7):

$$z_i = [x_i^{(1)}; x_i^{(2)}; x_i^{(3)}] \in \mathbb{R}^{d+3d+Kd} \tag{7}$$

where $d = 128$ is the Word2Vec embedding dimension, yielding a handcrafted feature vector of size 128 + 384 + 2560 = 3072.

3. **Self-Attention Branch:** The padded sequence $S_i \in \mathbb{R}^{L \times d}$ (with $L = 60$, $d = 128$) is fed into a multi-head self-attention (MHA) layer with $H = 4$ heads, using the sequence as queries, keys, and values simultaneously shown in Eq. (8):

$$\mathrm{MHA}(S_i) = \left[\mathrm{head}_1; \ldots; \mathrm{head}_H\right] W^O \in \mathbb{R}^{L \times d} \tag{8}$$

where each head computes in Eq. (9):

$$\mathrm{head}_h = \mathrm{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right) V_h, \quad d_k = d/H \tag{9}$$

with $Q_h = S_i W_h^Q$, $K_h = S_i W_h^K$, $V_h = S_i W_h^V \in \mathbb{R}^{L \times d_k}$. This mechanism captures token-to-token contextual dependencies regardless of position. The output is aggregated via mean pooling over the token dimension, followed by a linear projection with GELU activation and attention dropout ($p = 0.1$) shown in Eq. (10):

$$a_i = \mathrm{GELU}\left(W_p \cdot \frac{1}{L}\sum_{t=1}^{L} \mathrm{MHA}(S_i)_{t,:} + b_p\right) \in \mathbb{R}^d \tag{10}$$

Out-of-vocabulary tokens in S_i are represented as zero vectors, consistent with the handcrafted feature branch.

4. **Feature Fusion:** The handcrafted vector $z_i \in \mathbb{R}^{3072}$ and the attention representation $a_i \in \mathbb{R}^d$ are concatenated along the feature dimension shown in Eq. (11):

$$u_i = [z_i;\, a_i] \in \mathbb{R}^{3072+128} = \mathbb{R}^{3200} \tag{11}$$

This fused vector serves as the input to the Residual MLP classifier, enabling the model to jointly exploit global statistical descriptors from the handcrafted branch and context-aware token interactions from the self-attention branch.

5. **Residual MLP Classifier:** The fused vector $u_i \in \mathbb{R}^{3200}$ is passed through a Residual MLP with hidden layers of dimensions $(1024, 512, 256)$. A residual projection is applied from the input to the output of the first block only, aligning dimensions via a learned linear projection $R \in \mathbb{R}^{3200\times1024}$ shown in Eq. (12):

$$h_1 = \mathrm{LN}\big(\mathrm{Dropout}_{0.25}(\mathrm{GELU}(W_1 u_i + b_1)) + R\,u_i\big) \in \mathbb{R}^{1024} \tag{12}$$

Subsequent hidden layers apply linear transformation, GELU activation, dropout ($p = 0.25$), and layer normalization sequentially without residual connections shown in Eq. (13) and Eq. (14):

$$h_2 = \mathrm{LN}\big(\mathrm{Dropout}_{0.25}(\mathrm{GELU}(W_2 h_1 + b_2))\big) \in \mathbb{R}^{512} \tag{13}$$

$$h_3 = \mathrm{LN}\big(\mathrm{Dropout}_{0.25}(\mathrm{GELU}(W_3 h_2 + b_3))\big) \in \mathbb{R}^{256} \tag{14}$$

The final classification logits are produced by a linear head calculated in Eq. (15):

$$p_i = W_c h_3 + b_c \in \mathbb{R}^C, \quad \hat{y}_i = \mathrm{softmax}(p_i) \tag{15}$$

All linear layers are initialized with Xavier uniform initialization [27]; biases are initialized to zero.

6. **Loss Function and Regularization:** The model is trained with cross-entropy loss and label smoothing ($\varepsilon = 0.1$) to improve calibration and reduce overconfidence in Eq. (16):

$$\mathcal{L} = -\sum_{c=1}^{C} q_c \log \hat{y}_{ic}, \quad q = (1-\varepsilon)\,\mathrm{onehot}(y) + \frac{\varepsilon}{C} \tag{16}$$

MixUp regularization ($\alpha = 0.2$) is applied exclusively to the handcrafted feature vector z_i, leaving the sequence input to the self-attention branch unchanged as shown in Eq. (17):

$$z' = \lambda z_i + (1-\lambda) z_j, \quad \lambda \sim \mathrm{Beta}(\alpha, \alpha), \quad \mathcal{L}_{\mathrm{mix}} = \lambda\,\mathcal{L}(z_i) + (1-\lambda)\,\mathcal{L}(z_j) \tag{17}$$

Applying a dropout to a specifically designed branch enhances the ability to generalize along that path without affecting attention weights. For optimization, AdamW is used with a learning rate of $\mathrm{lr} = 2 \times 10^{-4}$ and weight decay of 10^{-4}, paired with a OneCycleLR scheduler that follows a cosine annealing profile, peaks at 5×10^{-4}, and warms up over the first 10% of training steps. To guard against unstable updates, gradient norms are clipped to $\|\nabla\|_2 \le 1.0$. Training is halted early if validation macro-F1 shows no improvement over five consecutive epochs, at which point the best recorded model state is restored.

The combined design allows token-level contextual signals from the self-attention branch and statistical-frequency descriptors from the handcrafted features to be jointly utilized during classification. Gradient flow through the residual MLP head remains stable across training, and applying MixUp to the handcrafted branch reduces the risk of overfitting, both of which contribute to stronger bug prediction outcomes.

4 Experiments and Results

This section includes the dataset descriptions used in the study, preliminary experiments that contributed to the development of the proposed model (HAR-MLP), cross-dataset generalization on Jira datasets, ablation study, and a comparison of the HAR-MLP model with pre-trained transformer models in terms of computational time, accuracy, and F1 score.

4.1 Dataset

In this study, two open-source repositories of software-related textual data were utilized for bug/feature classification.

4.1.1 GitHub Bugs Prediction Dataset

The first dataset [26] comprises 450,000 GitHub issues collected from multiple repositories on the GitHub platform. Each instance includes the following attributes:

- *Title:* A brief summary of the issue.
- *Body:* A detailed description of the issue, including relevant explanations, references, or logs.
- *Label:* A categorical variable with three possible values: bug, feature, and question.

The following pre-processing steps were applied:

- *Text Pre-processing*: Standard NLP techniques including punctuation removal, stopword elimination, and unnecessary escape character filtering were applied to improve text quality, as described in Section 3.2.
- *Title and Body Combination*: The title and body fields are concatenated into a single textual representation to preserve contextual information.
- *Binary Classification*: Instances labeled as *question* are excluded, transforming the task into binary classification (bug vs. feature), resulting in 407,799 instances.

4.1.2 Jira Datasets

The second group of datasets comprises three Jira issue collections named jira_1, jira_2, and jira_3 retrieved from the Zenodo platform [28]. Each collection has approximately 814,000 software issues, with every entry labeled as either a bug or a feature request, which makes them immediately usable for binary classification without any additional filtering step. The text content of each issue is drawn from two fields, *title* and *body*, which are concatenated and passed through the same pre-processing pipeline applied to the GitHub Bugs Prediction dataset. All three collections are used to assess how well the proposed model generalizes across different issue-tracking platforms.

Although the three collections share an identical set of feature issues (verified by issue key overlap analysis: 349,922/349,922, 100% overlap), they differ in the source ecosystems from which their bug issues are drawn. Specifically, Jira_1 contains bug reports from Apache and Atlassian enterprise projects (e.g., Spark, Hive, Solr, Confluence); Jira_2 contains bugs from Qt framework, Apache big data, and JBoss projects (e.g., Qt, Flink, HBase, Cassandra); and Jira_3 contains bugs predominantly from Mojang/Minecraft game development projects (MC, MCPE, MCL, BDS), which account for approximately 53% of its bug issues. As shown in Table 1, this structural difference is also reflected in the average body length: bug reports in Jira_1 and Jira_2 average 81.4 and 80.6 words respectively, while Minecraft bug reports in Jira_3 are notably shorter at 57.6 words. The three collections are kept as separate benchmarks rather than merged in order to enable

domain-specific evaluation and to assess how well the proposed model generalizes across distinct software ecosystems. Table 2 summarizes the statistics of all datasets used in the experiments.

Table 1: Structural characteristics of the three Jira datasets.

Dataset	Bug Source Ecosystem	Avg Title (words)	Avg Body—Bug	Avg Body—Feature
Jira_1	Apache/Atlassian enterprise	8.3	81.4	62.5
Jira_2	Qt/Apache big data/JBoss	8.3	80.6	62.5
Jira_3	Mojang/Minecraft gaming	7.6	57.6	62.5

Feature issues are identical across all three datasets (100% key-level overlap).

Table 2: Statistics of the datasets used in the experiments.

Dataset	#documents	#training	#testing	#bugs	#features
GitHub	407,799	326,239	81,560	207,318	200,481
Jira_1	814,323	651,458	162,865	464,280	350,043
Jira_2	814,322	651,457	162,865	464,279	350,043
Jira_3	814,322	651,457	162,865	464,279	350,043

A common evaluation protocol was applied to all experiments. Each dataset is partitioned into an 80% training set and a 20% test set using stratified sampling with a fixed random seed (*random_state* = 42), so that the original class proportions are retained in both splits. The same partitioning scheme is used across every method and dataset reported in this study.

4.2 Experimental Results

In this section, various classification results using various feature extraction methods and ML algorithms are presented to determine the best feature extractor and ML classifier model. As feature extraction methods, Word2vec, TF-IDF, FastText, Glove, Doc2Vec, DCT, DWT and combinations of these methods are evaluated on the Github Bugs Prediction dataset [26]. Classification performances of these feature extraction methods and ML algorithms such as KNN, Naive Bayes, Random Forest, and MLP are presented on various metrics. For the experiments, The Type-II DCT which is the most widely used variant in the literature is applied and orthonormal normalization (norm = 'ortho') is used to ensure energy preservation across embedding dimensions and to maintain a consistent scale between original and transformed feature vectors. All DCT coefficients are retained, preserving the original dimensionality while concentrating dominant semantic variation into leading low-frequency coefficients. For the DWT method, the Daubechies 1 (db1) wavelet is employed with single-level decomposition. Only the approximation coefficients (coeffs [0]) are retained, as they capture the coarse-grained semantic structure of the embedding vector, while the high-frequency detail coefficients (coeffs [1]) are discarded as they may encode noise. The original embeddings, DCT-transformed, and DWT-transformed features are then concatenated to form the final feature matrix used as input to the classifiers to test different combinations of feature extraction methods.

Accuracy values for each feature extraction method, their combinations using ML algorithms are given in Table 3. Precision and Recall values are presented in Table 4.

Table 3: Accuracy (%) values obtained by different Feature Extraction Methods (FEM) and ML algorithms on Github Bugs Prediction dataset.

FEM	KNN	DT	LR	NB	RF	MLP
TF-IDF	66.91	**73.51**	81.08	76.52	81.16	77.12
FastText	78.79	69.55	81.56	77.19	80.22	82.64
GloVe	71.06	62.30	74.50	70.58	73.69	77.15
Doc2Vec	65.59	58.67	77.50	56.78	71.53	78.30
Word2Vec	**80.16**	70.55	**82.38**	**78.85**	81.35	**83.18**
Word2Vec-DCT	80.16	70.49	82.38	78.61	81.16	82.87
Word2Vec-DWT	78.12	68.43	79.32	74.80	79.03	81.16
TF-IDF-DCT	66.91	61.93	81.08	78.70	76.22	76.79
TF-IDF-DWT	67.11	69.80	76.85	71.58	78.33	77.17
FastText-DCT	78.79	69.35	81.55	77.34	80.19	82.55
FastText-DWT	77.81	67.74	79.00	75.68	78.84	80.48
GloVe-DCT	71.06	62.60	74.50	70.35	73.53	77.48
GloVe-DWT	68.14	60.31	70.42	68.46	71.13	73.68
Doc2Vec-DCT	65.59	58.24	77.50	56.78	71.47	78.33
Doc2Vec-DWT	63.80	56.90	68.11	56.54	67.77	71.11
Word2Vec-DCT-DWT	80.03	71.22	82.33	78.59	**81.65**	83.07
TF-IDF-DCT-DWT	67.20	70.99	81.09	76.72	79.57	77.05
FastText-DCT-DWT	78.71	69.76	81.54	77.37	80.37	82.60
GloVe-DCT-DWT	70.81	62.47	74.53	70.42	74.04	77.46
Doc2Vec-DCT-DWT	65.55	59.09	77.52	56.67	72.48	78.31

Note: **Bold** values indicate the best accuracy for each ML algorithm (column-wise).

Table 4: Precision (P) and Recall (R) values (%) obtained by various FEMs and ML algorithms on Github Bugs Prediction dataset.

FEM	KNN		DT		LR		NB		RF		MLP	
	P	R	P	R	P	R	P	R	P	R	P	R
TF-IDF	67	67	74	74	81	81	77	77	81	81	77	77
FastText	79	79	70	70	82	82	77	77	80	80	**83**	**83**
GloVe	72	71	62	62	75	75	71	71	74	74	78	77
Doc2Vec	66	66	59	59	78	78	58	57	72	72	78	78
Word2Vec	80	80	71	71	82	82	79	79	81	81	**83**	**83**
Word2Vec-DCT	80	80	70	70	82	82	79	79	81	81	**83**	**83**
Word2Vec-DWT	78	78	68	68	79	79	75	75	79	79	81	81
TF-IDF-DCT	67	67	62	62	81	81	79	79	76	76	77	77
TF-IDF-DWT	67	67	70	70	77	77	72	72	78	78	77	77
FastText-DCT	79	79	69	69	81	81	77	77	80	80	**83**	**83**
FastText-DWT	78	78	68	68	79	79	76	76	79	79	81	80
GloVe-DCT	72	71	63	63	75	75	71	71	74	74	78	78
GloVe-DWT	69	69	60	60	70	70	69	69	71	71	74	74

(Continued)

Table 4 (continued)

FEM	KNN		DT		LR		NB		RF		MLP	
	P	R	P	R	P	R	P	R	P	R	P	R
Doc2Vec-DCT	66	66	58	58	78	78	58	58	71	71	78	78
Doc2Vec-DWT	64	64	57	57	68	68	58	58	68	68	71	71
Word2Vec-DCT-DWT	80	80	71	71	82	82	79	79	82	82	**83**	**83**
TF-IDF-DCT-DWT	67	67	71	71	81	81	77	77	80	80	77	77
FastText-DCT-DWT	79	79	70	70	82	82	77	77	80	80	**83**	**83**
GloVe-DCT-DWT	72	72	62	62	75	75	71	71	74	74	77	77
Doc2Vec-DCT-DWT	66	66	59	59	78	78	58	58	72	72	78	78

Note: **Bold** values indicate the best precision and recall scores for each ML algorithm (column-wise).

When the results given in Table 3 are analyzed, it is seen that the Word2Vec vectorization method mostly gives better results than the other feature extraction methods. The highest performance is seen in the MLP model with Word2Vec feature extraction, which achieves an accuracy of 83.18%.

Fig. 2 shows the average accuracy of various feature extraction methods over ML methods. According to the results obtained, Word2Vec based methods provide the highest accuracy rates, especially with DCT and DWT combinations, while Word2Vec and Word2Vec-DCT-DWT methods show the best average performance. FastText-based methods perform competitively, especially with DCT-DWT combinations. On the other hand, TF-IDF, GloVe and Doc2Vec based methods generally provide lower accuracy rates, and DCT and DWT transformations did not provide a significant improvement in GloVe and Doc2Vec methods. The MLP algorithm generally produces the best results, while the Decision Tree and Naive Bayes algorithms generally underperform.

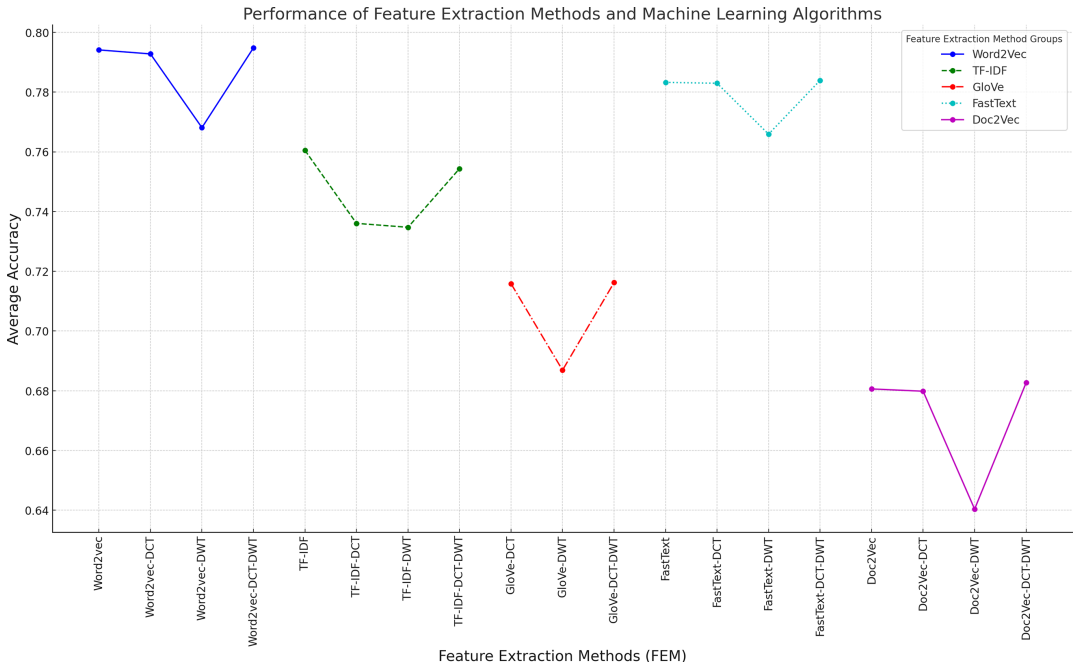


Figure 2: Average accuracy rates of the feature extraction methods grouped by Word2Vec, TF-IDF, GloVe, FastText,Doc2Vec.

When the precision and recall values obtained with the feature extraction methods and ML algorithms given in Table 4 are analyzed, it is seen that Word2Vec and FastText based feature extraction methods have generally higher precision and recall values in all algorithms. It is seen that the Word2Vec-DCT-DWT method gives the same results as Word2Vec in all algorithms except the RF algorithm. TF-IDF and GloVe based methods generally show average or low performance. Furthermore, consistent with the accuracy rates, the MLP model demonstrates superior performance compared to other algorithms across all feature extraction methods.

The boxplot shown in Fig. 3 compares the average F1 score distributions of different ML algorithms over all feature extraction methods. In general, the MLP (Multi-Layer Perceptron) algorithm has the highest average F1 Score values and also the narrowest distribution. The results imply that MLP performs consistently well over different feature extraction methods. While KNN, LR and RF algorithms perform relatively well in terms of average F1 Score, the widths of their distributions indicate instability in some cases. In contrast, the DT and NB algorithms generally have lower average F1 Score values and show wider distributions. This suggests that these two algorithms generally underperform over all feature extraction methods. As a result, the MLP algorithm provides the highest and most consistent performance with the proposed feature extraction methods.

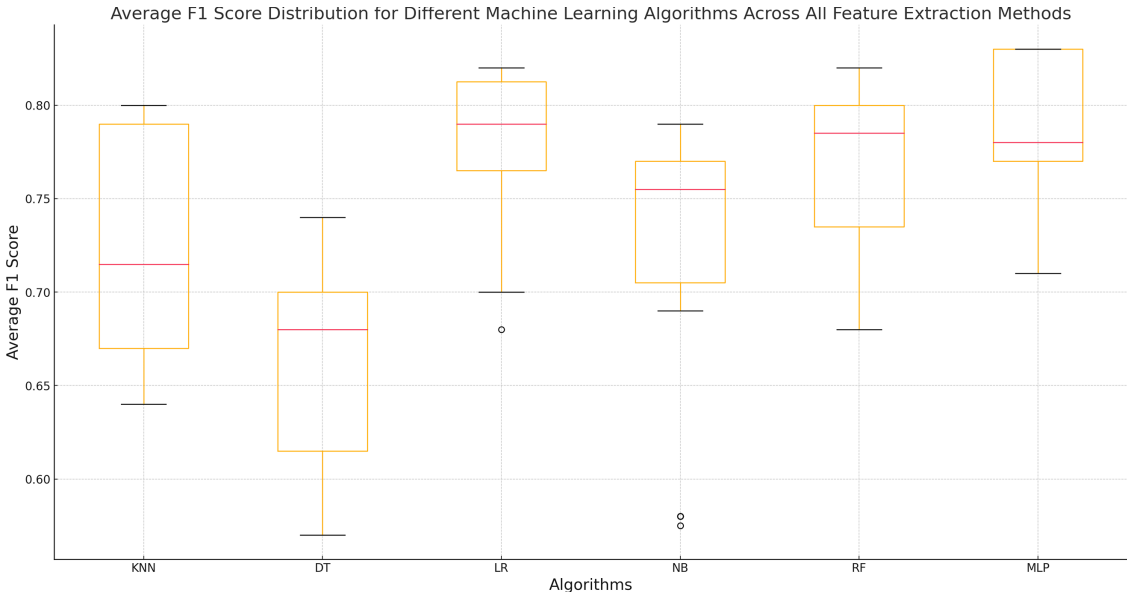


Figure 3: Average F1 scores of the ML algorithms.

Experimental results demonstrate that Word2Vec-based representation among embedding methods provides the best performance across all ML algorithms. However, Word2Vec embedding with MLP among ML methods is found to have the highest performance. Therefore, the proposed model is being developed based on Word2Vec and MLP.

To further enhance the representational capacity of the input, additional handcrafted features described in the Section 3.3 were incorporated. Besides the handcrafted features, the *Self-Attention branch* was introduced to explicitly capture token-to-token contextual dependencies that are not fully preserved by pooling operations. Token-level relevance is determined dynamically by the multi-head self-attention mechanism, with higher weights assigned to positions that carry bug-indicative signal.

The outputs of the handcrafted feature branch and the attention branch are concatenated through the *feature fusion* step described in Section 3.3, producing a joint representation that draws on both global statistical patterns and localized contextual cues. This fused vector is then fed into a *Residual MLP classifier*, whose residual connections help maintain stable gradient flow and reduce the risk of vanishing gradients; dropout and label smoothing are applied on top to limit overfitting. *MixUp regularization* is additionally restricted to the handcrafted feature branch as a targeted measure for improving generalization.

4.2.1 Comparison with Pre-Trained Transformer-Based Models

In this section, four pre-trained transformer models such as DistilBERT-base-uncased, BERT-base-uncased, RoBERTa-base, and XLNet-base-cased were evaluated on the same dataset under identical pre-processing, data splits, and evaluation metrics to establish strong neural baselines using binary class (bug/feature) and 3-class (bug/feature/question) representation. Each model was implemented using the identical hyperparameters listed in Table 5.

Table 5: Hyperparameters of the proposed HAR-MLP model.

Component	Hyperparameter	Value
Word2Vec	Embedding dim (d)	128
	Window size	5
	Training epochs	30
	Training algorithm	CBOW
Handcrafted Features	Max sequence length (L)	60
	DCT coefficients (K)	20
Self-Attention	Number of heads (H)	4
	Head dimension (d_k)	32
	Attention dropout	0.1
Residual MLP	Hidden layer dims	(1024, 512, 256)
	MLP dropout	0.25
	Activation function	GELU
	Residual connection	First block only
	Weight initialization	Xavier uniform
Training	Batch size	128
	Optimizer	AdamW
	Base (Peak learning rate)	$2 \times 10^{-4}/5 \times 10^{-4}$
	LR scheduler	OneCycleLR (cosine)
	Weight decay	10^{-4}
	Max epochs	100
	Early stopping patience	5 (macro-F1)
	Gradient clipping	$\|\nabla\|_2 \leq 1.0$
	Random seed	42
Regularization	Label smoothing (ε)	0.1
	MixUp α	0.2

Table 6 reports classification performance and training times for all evaluated models using binary classification. The four fine-tuned transformers reach accuracy figures between 87.18% and 87.47%, which translates to a macro-F1 advantage of roughly 2.0–2.2 percentage points over HAR-MLP. This difference is largely a consequence of large-scale pre-training. The contextual representations these models bring to the task are considerably richer than what task-specific Word2Vec embeddings can produce. However, HAR-MLP trains in 1087.7 s, which is 9.6×, 18.9×, 22.6×, and 32.8× faster than DistilBERT, BERT-base-uncased, RoBERTa-base, and XLNet-base-cased, respectively.

Table 6: Performance and training time comparison of HAR-MLP and pre-trained transformer-based models on the GitHub Bugs Prediction dataset using binary classification.

Model	Acc (%)	Prec (%)	Rec (%)	F1 (%)	Training Time (s)
DistilBERT-base-uncased	87.25	87.25	87.24	87.24	10,446.7
BERT-base-uncased	87.47	87.48	87.48	87.47	20,527.4
RoBERTa-base	87.40	87.45	87.37	87.39	24,542.5
XLNet-base-cased	87.18	87.18	87.18	87.18	35,672.5
HAR-MLP (proposed)	85.41	85.42	85.39	85.40	1087.7

The confusion matrix of the proposed HAR-MLP model is presented in Fig. 4 on the Github Bugs Prediction [26] test dataset. The balanced distribution of 6205 false positives and 5695 false negatives indicates that the model does not exhibit a systematic bias toward either class.

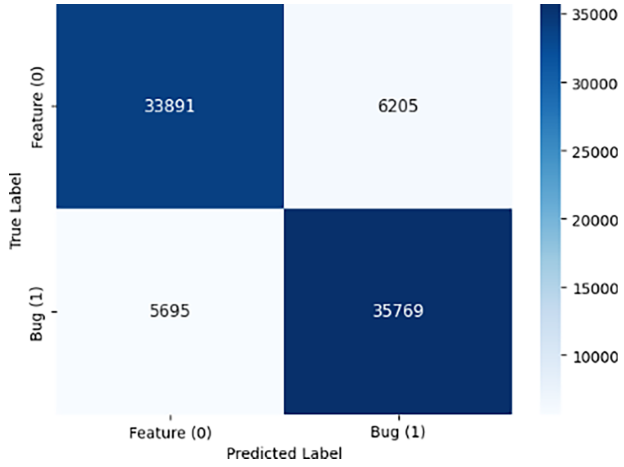


Figure 4: Confusion matrix of the proposed HAR-MLP model on the GitHub Bugs Prediction dataset.

Furthermore, HAR-MLP requires no pre-trained checkpoint and imposes no dependency on external large-scale corpora, making it suitable for deployment in resource-constrained environments. In terms of inference latency as seen in Table 7, HAR-MLP processes a single sample in 0.545 ms, compared to 3.573, 5.925, 6.050, and 11.290 ms for DistilBERT, BERT-base, RoBERTa-base, and XLNet-base, respectively, corresponding to speedups of 6.6×, 10.9×, 11.1×, and 20.7×.

The theoretical attention complexity represented in Table 7 further contextualizes these differences. The attention FLOP count of HAR-MLP employs a single self-attention layer over sequences of length $L = 60$ and embedding dimension $d = 128$, requiring $2L^2d = 921{,}600$ operations. In contrast, DistilBERT (6 layers,

$L = 128$, $d = 768$) and BERT-base/RoBERTa-base/XLNet-base (12 layers, $L = 128$, $d = 768$) require 163.8× and 327.7× more attention FLOPs, respectively.

Table 7: Theoretical complexity and inference efficiency comparison. Attention complexity is expressed as $\mathcal{O}(\text{layers} \times L^2 \times d)$, where L is the sequence length and d is the hidden dimension.

Model	Params	Infer. (ms)	Layers	L	d	$\mathcal{O}(\cdot)$
DistilBERT	66,955,010	3.573	6	128	768	$\mathcal{O}(6L^2d + L.d.512)$
BERT-base-uncased	109,483,778	5.925	12	128	768	$\mathcal{O}(12L^2d)$
RoBERTa-base	124,647,170	6.050	12	128	768	$\mathcal{O}(12L^2d)$
XLNet-base-cased	117,310,466	11.290	12	128	768	$\mathcal{O}(12L^2d)$
HAR-MLP (proposed)	7,298,434	0.545	1	60	128	$\mathcal{O}(L^2d)$

Overall, HAR-MLP achieves macro-F1 within 2.24% of the strongest transformer baseline while requiring no pre-trained checkpoint, training up to 32.8× faster, and performing inference up to 20.7× faster. These results are consistent with the resource-aware design motivation outlined in Section 2 and demonstrate that HAR-MLP is a practical alternative for large-scale bug prediction scenarios where computational efficiency is critical.

Three-class Evaluation using Github Bugs Prediction Dataset

To address the concern regarding the exclusion of the *question* class, all models additionally are evaluated under the original three-class setting using the same dataset, pre-processing pipeline, and evaluation protocol. To mitigate class imbalance (Bug: 133,654; Feature: 138,212; Question: 28,134), soft-balanced class weighting is applied to the HAR-MLP training objective, where weights are computed as the square root of balanced weights and normalized to unit mean. Transformer baselines are fine-tuned with *num_labels* = 3 under identical hyperparameters. Table 8 reports per-class F1 scores for all models. The Question class proves challenging across all architectures, even the strongest transformer (RoBERTa-base) achieves only 58.39% F1 on this class, while HAR-MLP achieves 42.46%. In contrast, for the Bug and Feature classes, HAR-MLP achieves competitive F1 scores of 79.14% and 80.02%, following transformers by only 5–6 percentage points. The low Question F1 observed across all models indicates that the difficulty originates from the inherent linguistic ambiguity of the Question label, question-type issues frequently overlap with bug reports and feature requests in vocabulary and phrasing, rather than from a specific limitation of the proposed method.

Table 8: Per-class F1 scores (%) in the 3-class setting (Bug/Feature/Question). HAR-MLP uses soft-balanced class weighting; transformer models are fine-tuned with *num_labels* = 3.

Model	Bug F1	Feature F1	Question F1	Macro F1
DistilBERT-base	85.05	84.95	55.75	75.25
BERT-base-uncased	85.24	85.32	56.34	75.63
RoBERTa-base	84.62	85.64	58.39	76.21
XLNet-base-cased	84.70	84.97	57.20	75.63
HAR-MLP (proposed)	79.14	80.02	42.46	67.21

The confusion matrix for three-class behavior is represented in Fig. 5. Bug and Feature classes are classified correctly at rates of 78.09% and 80.36%, respectively, with most errors occurring as mutual misclassifications between these two classes, 1996 Bug label is predicted as Feature and 1841 Feature label is

predicted as Bug. The Question class shows lower recall at 44.26% in total of the 2813 question-type issues, 732 (26.0%) are misclassified as Bug and 836 (29.7%) as Feature. This symmetric distribution of Question errors across both majority classes confirms that the difficulty arises from linguistic overlap rather than a systematic bias toward one class, and it is consistent with the low Question F1 observed across all models including large pre-trained transformers.

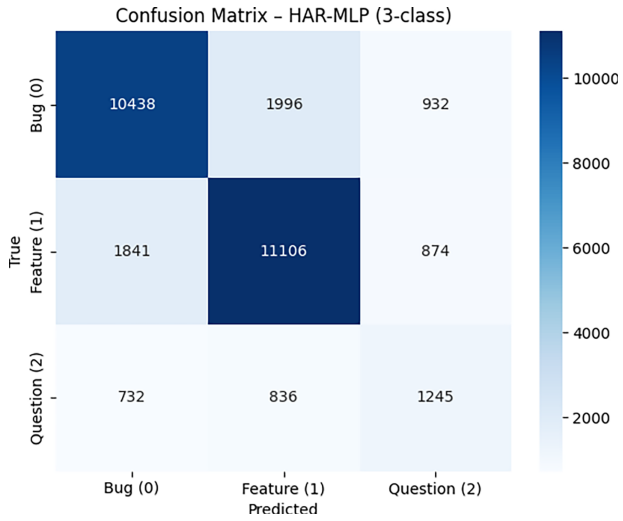


Figure 5: Confusion matrix of the proposed HAR-MLP model in the 3-class setting (Bug/Feature/Question) on the GitHub Bugs Prediction dataset.

4.2.2 Cross-Dataset Generalization on Jira Datasets

To evaluate the performance of the proposed model when applied to several datasets rather than a single one, and to generalize the model, three Jira issue datasets published by the Zenodo platform are used. The public data collected from the Jira platform, an issue tracking system, contains software issues and is completely independent of the Github dataset used for the main experiments. This enables cross-platform generalization, and the same model architecture, data splitting method, and hyperparameters are applied to the Jira dataset as well.

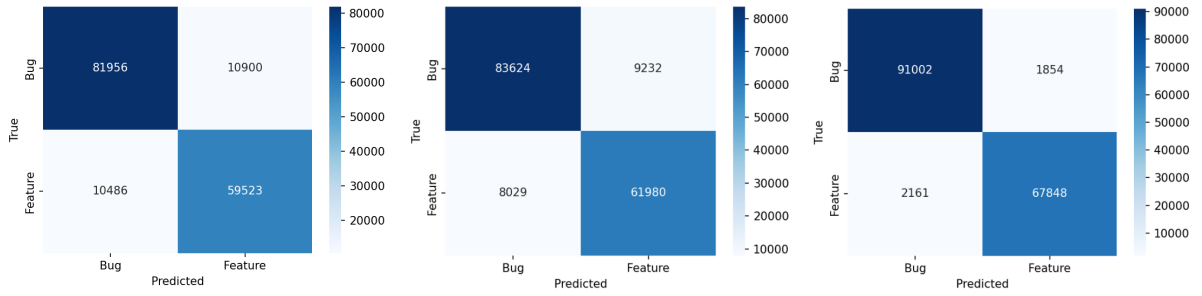
According to the results given in Table 9, the proposed model shows balanced performance between Bug and Feature classes in all three datasets. In Jira_1, Bug detection, with an F1 score of 88.46%, performs better than Feature detection, which has an F1 score of 84.77%. This result is consistent with the imbalance in the number of feature-based data points in the dataset. In the Jira_2 dataset, Bug detection has an F1 score of 90.64%, which is better than Feature detection with an F1 score of 87.78%. Finally, Jira_3 shows higher performance compared to the other datasets, with Bug detection reaching an F1 score of 97.84% and Feature detection reaching an F1 score of 97.13%. The notably higher performance on Jira_3 is attributable to the domain-specific characteristics of its bug issues. As reported in Table 1, Minecraft bug reports in Jira_3 are significantly shorter on average (57.6 words) compared to those in Jira_1 (81.4 words) and Jira_2 (80.6 words), and employ game-specific vocabulary. Since the feature issues are identical across all three datasets, this creates a more distinctive linguistic boundary between bug and feature classes in Jira_3, making the classification task inherently easier in the gaming domain than in enterprise software environments.

Based on the results obtained, consistent improvements were observed across all datasets created from different training sources and different platforms, indicating that the proposed HAR-MLP model provides real representative improvements.

Table 9: Per-class classification results (%) of the proposed HAR-MLP on the Jira datasets.

Dataset	Class	Precision	Recall	F1-Score	Support
Jira_1	Bug (0)	88.66	88.26	88.46	92,856
	Feature (1)	84.52	85.02	84.77	70,009
	Macro avg	86.59	86.64	86.61	162,865
Jira_2	Bug (0)	91.24	90.06	90.64	92,856
	Feature (1)	87.04	88.53	87.78	70,009
	Macro avg	89.14	89.29	89.21	162,865
Jira_3	Bug (0)	97.68	98.00	97.84	92,856
	Feature (1)	97.34	96.91	97.13	70,009
	Macro avg	97.51	97.46	97.48	162,865

When the confusion matrices given in Fig. 6 are examined, it is seen that Jira_1 has 10,900 false negatives and 10,486 false positives, indicating that the model does not have a systematic bias towards any class. Similar results are observed in Jira_2, with 8029 false positives and 9232 false negatives. In Jira_3, the model obtained only 1854 false negatives and 2161 false positives from 162,865 test samples, achieving a separation performance of over 97% between error and feature reports. The experimental results indicate that HAR-MLP consistently outperforms FastGATConv which is proposed by Siachos et al. [5] on all datasets in both accuracy and F1 score metrics. On the GitHub dataset, HAR-MLP achieves 85.41% accuracy and 85.40% F1, surpassing FastGATConv by 5.19 and 5.07 percentage points, respectively. Besides, the improvements of 4.09%–7.34% on Jira_1, 3.26%–5.69% on Jira_2, and 1.89%–2.47% on Jira_3 are observed on the JIRA datasets. Notably, the F1 gap on Jira_1 (7.34 pp) is substantially larger than the accuracy gap (4.09 pp), which reflects the precision–recall imbalance reported for FastGATConv on that dataset (82.08% vs. 76.64%), indicating that the graph-based model struggles with minority-class recall. In contrast, HAR-MLP maintains well-balanced precision and recall across all datasets, suggesting that the combination of TF-IDF-weighted Word2Vec embeddings, multi-pooling, DCT coefficients, and self-attention provides more discriminative and robust textual representations than graph-of-words constructions.



(**a**) Jira_1 (**b**) Jira_2 (**c**) Jira_3

Figure 6: Confusion matrices of the proposed HAR-MLP on the three Jira datasets.

4.2.3 Ablation Study

To quantify the contribution of each component in the proposed model, an ablation study was performed using four configurations as listed below:

1. Baseline: Word2Vec embeddings with a traditional MLP classifier (no attention, no handcrafted features).
2. Baseline + Attention: Addition of a self-attention branch without handcrafted features.
3. Handcrafted + MLP: Handcrafted features (TF-IDF weighted embeddings, multi-pooling, DCT) fed directly into the residual MLP without any attention mechanism.
4. HAR-MLP: Combination of handcrafted features with the self-attention branch, fused via residual MLP, trained with label smoothing and MixUp regularization.

Table 10 presents the ablation results, where each successive modification leads to a clear gain in classification performance. All metrics are reported as mean values over five independent runs with different random seeds. In the simplest configuration (Model 1), Word2Vec embeddings paired with a standard MLP reaches 83.52% mean validation accuracy and 83.51% mean macro-F1. In the second configuration, adding a self-attention branch without any handcrafted features (Model 2) brings the mean macro-F1 up to 84.68%, a 1.16% gain over the baseline. The improvement suggests that attention over token-level representations introduces useful relational structure that a traditional MLP cannot recover on its own. On the other hand, Model 3 takes a different path bypassing the attention and routing the TF-IDF weighted embeddings, multi-pooling statistics, and DCT coefficients straight into the residual MLP. Despite its architectural simplicity, this setup achieves 85.07% mean macro-F1, 1.55% above Model 1, indicating that the handcrafted features alone carry strong discriminative signal. A further observation is that Model 3 converges after just 7–8 epochs on average, suggesting that structured feature representations provide more direct optimization cues than raw sequential input.

Table 10: Ablation study results (%) of the proposed model on the Github Bugs Prediction dataset (mean over 5 independent runs).

Model No	Val Acc	Precision	Recall	F1 ± Std	Avg. Epoch
1. Baseline (Word2Vec + MLP)	83.52	83.52	83.52	83.51 ± 0.08	25
2. Baseline + Attention	84.68	84.68	84.68	84.68 ± 0.06	15
3. Handcrafted + MLP (no attention)	85.07	85.08	85.07	85.07 ± 0.06	9
4. **The proposed HAR-MLP**	**85.32**	**85.35**	**85.33**	**85.31 ± 0.07**	**8**

Note: **Bold** values indicate the best results across all ablation configurations. Results averaged over 5 runs with different random seeds (42, 123, 456, 789, 2024). Std: standard deviation of F1-score across runs.

Finally, the proposed HAR-MLP (Model 4) combines both components and achieves the highest mean macro-F1 of 85.31%, representing a 1.80% gain over the baseline. Training completes in approximately 319 s over 7–9 epochs, requiring only modest additional computation compared to Model 3 to incorporate the attention mechanism. While the accuracy margin over Model 3 is modest (+0.25%), the attention branch provides complementary representational capacity at a negligible computational cost.

Furthermore, all experiments were repeated over five independent runs with different random seeds (42, 123, 456, 789, 2024) to verify that these improvements are not attributable to random variation. The low standard deviations across all models (ranging from ±0.06% to ±0.08%) confirm that the reported results are stable and reproducible. In addition, pairwise McNemar's tests with Edwards' continuity correction were conducted on the validation set (N = 81,560). All pairwise comparisons are statistically significant at the $p < 0.01$ level across all seeds. Notably, even the modest improvement of the proposed HAR-MLP (Model 4) over Model 3 (+0.25% mean F1) is consistently confirmed to be statistically significant ($p < 0.01$, $b = 1273$, $c = 1013$), ruling out the possibility that the gain is attributable to random variation. Overall, these results provide a meaningful statistical evidence that both the attention mechanism and the handcrafted feature set

contribute independently, and that their combination yields the most discriminative representation space within a competitive training time.

5 Discussion

In this study, the proposed HAR-MLP model combines handcrafted statistical features with a self-attention branch to classify bug and feature reports using title and body information from open-source projects. Compared to the baseline Word2Vec+MLP configuration on the same dataset, HAR-MLP achieves an absolute improvement of 2.06% in validation accuracy and macro-F1 score.

The Word2Vec embeddings used in this study are not randomly initialized; they are trained for 30 epochs using the CBOW variant on the task-specific corpus, yielding semantically meaningful vector representations prior to any spectral transformation. Given token embeddings arranged in a matrix of shape $(L \times d)$, each column along the embedding dimension constitutes a discrete signal indexed by token position. By decomposing this signal with DCT coefficients, the distribution trends at the subject level represent gradually changing semantic patterns, and higher frequency coefficients better reflect the fluctuations at the marker level. Therefore, considering the first K coefficient of the generated sequence frequency component reduces the sequence to a compact spectral representation of the embedding trajectory, enabling the application of spectral transformation to word and sentence embedding operations. This transformation process is also seen in NLP studies in the literature [14,15].

When the results obtained within the scope of the study are examined, as given in Table 3, DCT is not used alone as an embedded representation vector. When Word2Vec-DCT and plain Word2Vec are compared, DCT, which gives the best performance in the MLP classifier, achieves accuracy values of 83.18% and 82.87% respectively, adding frequency domain patterns that other features cannot capture, thus performing better than the plain Word2Vec model. The use of DCT alone was not preferred because it would eliminate high-frequency components that can carry locally distinctive signals. It is observed that efficient gains are obtained when it is used together with the Word2Vec backbone and other handcrafted features within the HAR-MLP architecture.

While existing software bug prediction approaches [3,5,21,22] in the literature often rely on pre-trained deep learning networks as classifiers and these networks typically require longer end-to-end training times, HAR-MLP offers a strong trade-off between accuracy and training efficiency in binary classification. As empirically demonstrated in Table 6, HAR-MLP achieves macro-F1 within 2.24% of the best fine-tuned model (BERT-base-uncased, F1 = 87.47%) while training 18.9× faster and requiring no large-scale pre-training. The results indicate that HAR-MLP is a promising alternative for scenarios where both predictive performance and computational resource efficiency are critical considerations. On the other hand, when the evaluation is extended to the three-class setting (Bug/Feature/Question) with soft-balanced class weighting, the performance gap between HAR-MLP and transformer models widens from approximately two percentage points to approximately nine percentage points in macro-F1 (67.21% vs. 76.21% for RoBERTa-base), indicating that pre-trained contextual representations provide greater benefit as task complexity increases. Nevertheless, the Question class proves difficult for all architectures: even the strongest baseline achieves only 58.39% F1, and misclassifications are distributed symmetrically across Bug and Feature classes, suggesting that the challenge originates from inherent label ambiguity rather than a model-specific weakness. For the Bug and Feature classes, HAR-MLP remains competitive with a gap of only 5–6 percentage points, while preserving its computational advantages. These findings confirm that the binary Bug/Feature formulation represents the more tractable and practically relevant task for lightweight deployment scenarios.

5.1 Threads to Validity

Although the proposed HAR-MLP model shows significant performance improvements in software bug prediction, it has the following limitations and validity threats.

- *Generalizability*: Although the proposed model has been evaluated on Github Bugs Prediction Dataset [26] and three external Jira datasets [28] demonstrating cross-platform generalizability, it has not been tested on datasets from domains beyond software issue tracking, such as mobile app reviews or forum-based bug reports. Future studies will aim to assess the model's performance across more diverse sources and domains.
- *Dynamic software bug detection*: Since the model is tested on an open source dataset, it is not fully tested for real-time applicability in continuous integration/continuous deployment (CI/CD) environments and therefore cannot dynamically detect bugs. In future work, we will focus on this shortcoming and integrate the developed model into the live environment and perform software bug detection tests on real-world scenarios.
- *Performance gap in multi-class settings in an imbalance dataset*: While HAR-MLP achieves macro-F1 within 2.24% of the best transformer baseline in the binary setting, the gap widens to approximately 9 percentage points in the three-class setting, where HAR-MLP achieves 67.21% compared to RoBERTa-base's 76.21%. This gap is largely attributable to the class imbalance due to the slightly low number of Question labels compared to the other labels, which constitutes only 9.4% of the dataset, combined with its inherent linguistic ambiguity with Bug and Feature reports. For this reason, HAR-MLP is most suitable for balanced or binary classification scenarios where computational efficiency is a priority, whereas transformer-based models may be preferred when class-imbalanced multi-label settings demand higher minority-class recall and computational resources are available.

6 Conclusion

In this study, a Hybrid Attention-Residual Multi-Layer Perceptron (HAR-MLP) model is proposed to improve open-source software bug prediction. The model combines semantic representations from Word2Vec embeddings with handcrafted statistical and spectral features namely TF-IDF weighted embeddings, multi-pooling statistics, and DCT coefficients. After the merging process, the resulting handcrafted features are integrated through a series matrix using a multi-headed self-attention mechanism to capture inter-token dependencies. Thus, the features now input to the MLP classifier are processed, improving the model's gradient stability and generalization ability.

When the ablation studies are examined, it is seen that the HAR-MLP model achieved an accuracy of 85.41% and an F1 score of 85.40% on the GitHub Error Prediction dataset, and it provides an increase of 1.82% in accuracy and 1.81% in F1 score compared to the Word2Vec+ raw MLP model, which was determined as the baseline method. However, to the best of our knowledge, only one standalone study has utilized this dataset. Compared with the graph attention-based method (FastGATConv) proposed by Siachos et al. [5], the proposed model improves the accuracy on this dataset by 5.18%. Moreover, cross-platform evaluation on three Jira datasets further confirms the generalizability of HAR-MLP beyond the GitHub domain, achieving macro-F1 scores of 86.61%, 89.21%, and 97.48% on Jira_1, Jira_2, and Jira_3, respectively, without any dataset-specific tuning. In terms of accuracy and F1 score, improvements of 4.09%–7.34%, 3.26%–5.69%, and 1.89%–2.47% were observed in the Jira_1, Jira_2, and Jira_3 datasets compared to the FastGATConv method. Furthermore, the ablation study confirms that while self-attention alone yields marginal gains, its combination with handcrafted features significantly enhances representational richness and predictive performance. Additionally, the proposed model achieves convergence much faster than transformer models with a shorter inference rate, fewer layers and parameters, despite having a higher

computational cost per period. Furthermore, three-class experiments with soft-balanced class weighting confirm that while transformer models hold a larger advantage in this more complex setting, the Question class remains challenging across all architectures (best transformer F1: 58.39%) due to the imbalanced class distribution, reinforcing the practical motivation for the balanced binary classification.

In conclusion, HAR-MLP combines handcrafted feature engineering with attention-based sequence modeling, achieving accuracy competitive with stronger baselines while requiring substantially lower computational resources. These properties make it a plausible option for deployment in real-world software maintenance pipelines.

Acknowledgement: The author acknowledges that Claude, an AI assistant developed by Anthropic, was used during the preparation of this manuscript to support academic writing refinement, particularly for grammar correction and phrasing improvements. All scientific content, analysis, and interpretations and reported findings are entirely the responsibility of the author.

Funding Statement: The author received no specific funding for this study.

Availability of Data and Materials: The source code of the proposed model is available in the following public repository: https://github.com/isilkarabeyaksakalli/HAR-MLP.

Ethics Approval: Not applicable.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Liao Z, Wang K, Zeng Q, Liu S, Zhang Y, He J. Classification of open source software bug report based on transfer learning. Expert Syst. 2024;41(5):e13184. doi:10.1111/exsy.13184.
2. Bharadwaj S, Kadam T. GitHub issue classification using BERT-style models. In: Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering; 2022 May 21; Pittsburgh, PA, USA. p. 40–3. doi:10.1145/3528588.3528663.
3. Tambe D, Ragha L. ECBFMBP: design of an ensemble deep learning classifier with bio-inspired feature selection for high-efficiency multidomain bug prediction. J Cogn Sci. 2023;24(3):313–36.
4. Yildirim Taser P. A novel multi-view ordinal classification approach for software bug prediction. Expert Syst. 2022;39(7):e13044. doi:10.1111/exsy.13044.
5. Siachos I, Kanakaris N, Karacapilidis N. Software bug prediction using graph neural networks and graph-based text representations. Expert Syst Appl. 2025;259(1):125290. doi:10.1016/j.eswa.2024.125290.
6. Qiu S, Bicong B, He J. Features extraction and fusion by attention mechanism for software defect prediction. PLoS One. 2025;20(4):e0320808. doi:10.1371/journal.pone.0320808.
7. Thirumoorthy K, Jerold John Britto J. A feature selection model for software defect prediction using binary Rao optimization algorithm. Appl Soft Comput. 2022;131(11):109737. doi:10.1016/j.asoc.2022.109737.
8. Anju AJ, Judith JE. Hybrid feature selection method for predicting software defect. J Eng Appl Sci. 2024;71(1):124. doi:10.1186/s44147-024-00453-3.
9. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. Adv Neural Inf Process Syst. 2013;26:3111–9. doi:10.32614/cran.package.word2vec.
10. Joulin A, Grave E, Bojanowski P, Douze M, Jégou H, Mikolov T. Fasttext. zip: compressing text classification models. arXiv:1612.03651. 2016.
11. Pennington J, Socher R, Glove MC. Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); 2014 Oct 25–29; Doha, Qatar. p. 1532–43. doi:10.3115/v1/d14-1162.

12. Le Q, Mikolov T. Distributed representations of sentences and documents. In: Proceedings of the International Conference on Machine Learning. PMLR; 2014 Jun 21–26; Beijing, China. p. 1188–96.
13. Lau JH, Baldwin T. An empirical evaluation of doc2vec with practical insights intoDocument embedding generation. In: Proceedings of the 1st Workshop on Representation Learning for NLP; 2016 Aug 12; Berlin, Germany. p. 78–86. doi:10.18653/v1/w16-1609.
14. Dahab MY, Kamel M, Alnofaie S. An empirical study of documents information retrieval using DWT. In: Intelligent natural language processing: trends and applications. Cham, Switzerland: Springer; 2018. p. 251–64. doi:10.1007/978-3-319-67056-0_13.
15. Salama R, Youssef A, Diab M. Semantic compression for word and sentence embeddings using discrete wavelet transform. In: Proceedings of the Findings of the Association for Computational Linguistics ACL 2024; 2024 Aug 11–16; Bangkok, Thailand. p. 15963–77. doi:10.18653/v1/2024.findings-acl.945.
16. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, et al. Transformers: state-of-the-art natural language processing. In: Proceedings of the 1st Workshop on Representation Learning for NLP; 2016 Aug 12; Online. p. 38–45.
17. Devlin J, Chang MW, Lee K, Bert TK. Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT; 2019 Jun 2–7; Minneapolis, MN, USA.
18. Liu Y. Roberta: a robustly optimized bert pretraining approach. arXiv:1907.11692. 2019.
19. Reimers N. Sentence-BERT: sentence embeddings using siamese BERT-networks. arXiv:1908.10084. 2019.
20. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I, et al. Language models are unsupervised multitask learners. OpenAI Blog. 2019;1(8):9.
21. Kallis R, Di Sorbo A, Canfora G, Panichella S. Predicting issue types on GitHub. Sci Comput Program. 2021;205(3):102598. doi:10.1016/j.scico.2020.102598.
22. Siva R, Hariharan B, Premkumar N. Automatic software bug prediction using adaptive golden eagle optimizer with deep learning. Multimed Tools Appl. 2024;83(1):1261–81. doi:10.1007/s11042-023-16666-2.
23. Panda RR, Nagwani NK. Software bug severity and priority prediction using SMOTE and intuitionistic fuzzy similarity measure. Appl Soft Comput. 2024;150(2):111048. doi:10.1016/j.asoc.2023.111048.
24. Kumar R, Chaturvedi A. Software bug prediction using reward-based weighted majority voting ensemble technique. IEEE Trans Rel. 2024;73(1):726–40. doi:10.1109/tr.2023.3295598.
25. Al-Fraihat D, Sharrab Y, Al-Ghuwairi AR, Alshishani H, Algarni A. Hyperparameter optimization for software bug prediction using ensemble learning. IEEE Access. 2024;12(2):51869–78. doi:10.1109/access.2024.3380024.
26. Kumar A. GitHub bugs prediction dataset. 2025 [cited 2025 Feb 3]. Available from: https://www.kaggle.com/datasets/anmolkumar/github-bugs-prediction.
27. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings; 2010 May 13–15; Chia Laguna, Italy. p. 249–56.
28. Montgomery L, Lüders C, Maalej W. The Public Jira Dataset. Zenodo. 2022 [cited 2026 Jan 1]. Available from: https://zenodo.org/records/5901804.