IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 July 2025, accepted 18 August 2025, date of publication 27 August 2025, date of current version 1 October 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3603451

RESEARCH ARTICLE

XAI Driven Software Defect Prediction Using Adaptive Feature Engineering Coupled With Autoencoder and Multi-Layer Perceptron: An Empirical Study

PARVATHANENI NAGA SRINIVASU[1], M. SAILAJA[2], SUJATHA CANAVOY NARAHARI[3], (Senior Member, IEEE), PAOLO BARSOCCHI[4], (Member, IEEE), AND AKASH KUMAR BHOI[5]

[1]Amrita School of Computing, Amrita Vishwa Vidyapeetham, Amaravati, Andhra Pradesh 522503, India
[2]Department of Computer Science and Engineering, Prasad V. Potluri Siddhartha Institute of Technology, Vijayawada, Andhra Pradesh 520007, India
[3]Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, Hyderabad, Telangana 501301, India
[4]Institute of Information Science and Technologies, National Research Council, 56124 Pisa, Italy
[5]eSupport for Research, Burla, Sambalpur 768018, India

Corresponding authors: Parvathaneni Naga Srinivasu (parvathanenins@gmail.com) and Paolo Barsocchi (paolo.barsocchi@isti.cnr.it)

ABSTRACT Software defect prediction is essential for ensuring the reliability and robustness of software prototypes during development. The current study has proposed a novel strategy that integrates explainable artificial intelligence (XAI) techniques with adaptive feature engineering and autoencoder neural networks to improve defect prediction accuracy and interpretability. Adaptive feature engineering processes input data to identify critical features, while autoencoders handle non-linear datasets, reduce noise, and generate meaningful latent representations by learning underlying data patterns. A Multi-Layer Perceptron (MLP) is employed to classify code snippets and localize defects, leveraging its ability to manage complex data patterns and diverse input features. To enhance transparency and trust in model predictions, the XAI component provides insights into feature significance and classification outcomes. Empirical evaluations were conducted on the PROMISE dataset, with performance assessed using metrics such as sensitivity, specificity, F1-score, and the Matthews correlation coefficient. The proposed approach has demonstrated superior accuracy compared to other defect prediction methods, highlighting its effectiveness in identifying defective code snippets and enhancing software quality.

INDEX TERMS Software defect prediction, explainable AI, multi-layer perceptron, feature engineering, autoencoders.

I. INTRODUCTION

Software defects are often characterized as discrepancies among the desired programming outcomes and software requirements. In the software project development lifecycle, the later the phases in which the defects are identified, the higher the expense of rectifying them. So, it is desired to identify the errors in the earlier phases to build a robust software model [1]. Considerable research is underway in the field of automating software defect prediction (SDP). A tremendous transformation has taken place in the software development field, where contemporary software models are more complex, often leading to complicated architectures and larger codebases. This results in an elevated probability of software defects, which impact the overall performance. Hence, rigorous evaluation models are needed to localize the defects, and using human intervention in defect identification is a tedious task [2]. According to the report released in September 2018 by the Consortium for IT Software Quality (CISQ), over 50% of total software expenditure is

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

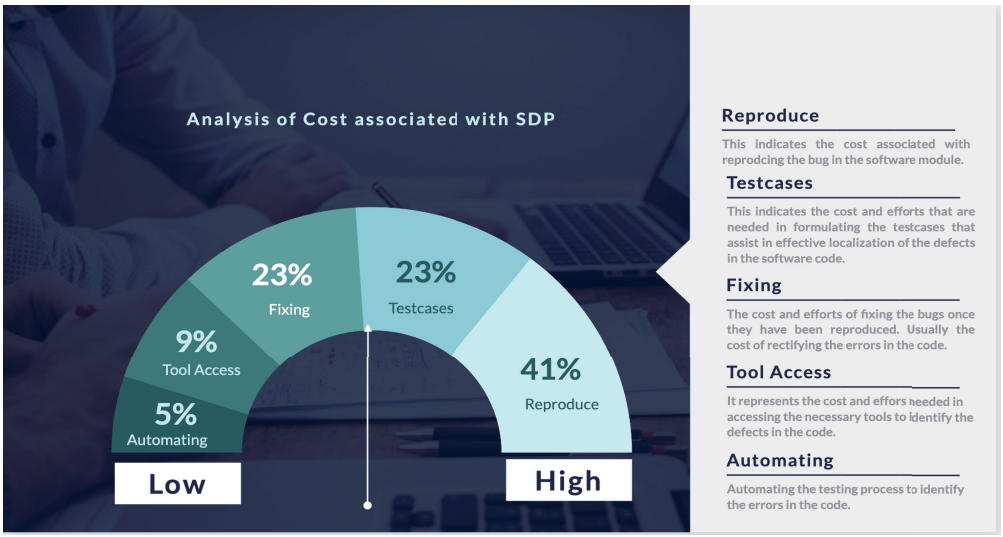


FIGURE 1. The impact of SDP on a software company.

allocated to bug identification and rectification, alongside the losses incurred due to software failures in the production environment [3], [4]. The impact of SDP on a software company is shown in Figure 1. The data is obtained from the website raygun.com, an error tracking and reporting software [5].

There are various machine intelligence models being used in the process of SDP [6], [7]. These advanced models are efficient in simultaneously reducing time and costs by effectively allocating necessary resources to the sections with a high likelihood of defects [8]. Traditional defect prediction approaches largely depend on handcrafted features and classical classification techniques, which often lack the decision interpretability and scalability for handling complex software models [9]. The conventional approach that is used in SDP is vulnerable to overfitting and underfitting, which would lead to poor generalization results and biased outcomes [10], [11]. Hence, the current study the use of the explainable artificial intelligence model (XAI) [12] in analyzing the feature dependencies and their contributions to the decision process.

Most of the SDP models are dependent on the features that are part of the training data, and the features are used in the prediction of the defect in the software code snippet. It is necessary to identify the essential features that assist in the precise classification of the code snippets. There are various research questions (RQ) that are associated with the research gaps that are being analyzed from the previous studies. Some of the research questions that are associated with the current field of study are listed below.

- **RQ1:** Does feature engineering have an impact on the classification accuracy of the model, especially when dealing with software defect detection datasets with distinct features?
- **RQ2:** Does Adaptive feature engineering yield better efficiency than the conventional feature engineering approaches?
- **RQ3:** How does the use of autoencoders affect the feature selection mechanism in SDP?
- **RQ4:** What insights can XAI provide into the decision-making process of the defect prediction model?

The aforementioned research questions are worth investigating to build a robust SDP model where feature engineering would have a considerable impact on the classification outcome of the model. Adaptive feature engineering recognizes the meaningful and domain-specific features from the code bases, which improve the relevance and quality of inputs for the classification models. Autoencoders are unsupervised feature extraction models, where the model does not need training. Moreover, they can identify the more significant features and non-linear dependencies among them [13]. The combination of the autoencoders with MLP would yield a better classification outcome over the other state-of-the-art (SOA) model. The listed below are the contributions of the current study.

- Adaptive feature engineering is performed alongside the autoencoders to identify the significant features in the data, thereby assisting the decision process with domain knowledge.
- The use of shapely values to determine the dependencies and contribution of the feature in the decision process is important to make the decision process interpretable.
- MLP technique is used to classify the code snippets with possible defects in the code.
- The current model is being assessed with respect to various standard metrics, and the results obtained are evaluated against various SOA models.

The major components of the proposed SDP model are depicted in Figure 2. The rest of the discussion on the study

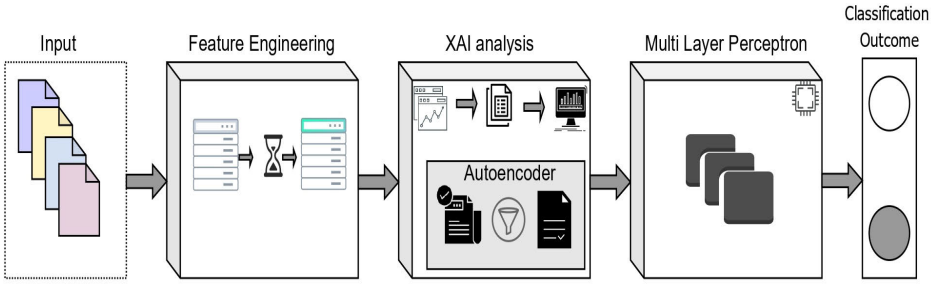


FIGURE 2. The components of the proposed SDP model.

is arranged as follows. Section II presents the systematic literature survey on various defect detection approaches. Section III presents the background of the study, which discusses the dataset, feature engineering, XAI analysis on the identified features, and the implementation settings. Section IV presents the MLP technique for classification and localizing the defective code base. Section V outlines the results and discussion, and finally, Section VI is the conclusion, along with the future scope of the proposed approach.

II. SYSTEMATIC LITERATURE REVIEW

Software defect prediction is a critical aspect of the software development lifecycle, where various ML and DL models are extensively used to localize software defects in the code. The use of ML techniques like support vector machine (SVM), random forest (RF), k-nearest neighbor (KNN), naïve Bayes (NB), and Atomic Rule Mining (ARM). Divergent machine intelligence approaches are being extensively used in SDP across studies [14], [15], [16], [17], [18]. Various studies are discussed in the current section to outline the advancement in the precise localization of the abnormalities in the code snippets.

Mohammad et al. [19] have investigated the impact and stability of four different SVM kernel functions in software defect prediction across 38 datasets, analyzing factors such as data granularity, imbalance ratios, and feature subsets through 1520 experiments. The findings recommend using the radial basis functions kernel, which consistently outperforms linear and other nonlinear kernels, particularly on imbalanced datasets and when using the top 40% of features. Goyel et al. [20] has proposed a novel filtering technique to use along with the SVM model for SDP. The model is efficient in dealing with the imbalanced datasets and results in, generating highly accurate classification results with around 16.73% improvement in accuracy. Decision Tree regression analysis [21] is being used in the SDP by making use of the requirement and agile-based metrics. The experimentation has shown that the requirement-based metric outperforms the agile-based metric in fault prediction. Thomas and Kaliraj [22] have proposed an optimized RF along with SMOTE to handle the class imbalances in the input dataset. Optimizing hyperparameters to improve accuracy, achieving 82.96% accuracy and an F1 score of 89.53% on the NASA JM1 dataset. Kun et al. [23] has proposed an SDP model, that has done feature selection using the whale optimization algorithm (WOA) and simulated annealing (SA) to construct an enhanced metaheuristic search based feature selection algorithm named EMWS. The model has been evaluated across 20 different projects and the study has proven that the feature selection would have a significant impact on classifying the defective code blocks.

Arasteh et al. [24] have a hybrid SDP method combining an autoencoder for feature selection and dimensionality reduction with the K-means algorithm for clustering, which has achieved an accuracy of 96% in evaluating the NASA PROMIS dataset [25]. Hybrid Instance Selection Using the Nearest-Neighbor (HISNN) method, which is a combination of KNN and NB approach for achieving high performance, and improved detection probability [26]. SDP often involves larger-size datasets with many software snippets, making the computational burden of k-NN significant during prediction, as it requires calculating distances for all training instances. Similar to k-NN, the NB model struggles with class imbalance, as it tends to predict the majority class, leading to poor defect detection and low recall. Another study using RF and ARM was used simultaneously for SDP [27], which achieved reasonable accuracy with minimal computational time. The RF and ARM have demonstrated a 90.09% accuracy and a 54.14% reduction in prediction time compared to existing models. A study using Hierarchical Neural Network (DP-HNN) [28] that leverages the hierarchical structure of abstract syntax trees (ASTs) to improvised SDP has been evaluated over 11 open-source projects and the achieving an average improvement of Matthews correlation coefficient (MCC) and Area Under the Curve (AUC) scores.

There are various DL models being extensively used in SDP for enhanced performance and to use larger-size datasets. There are various generalized regression neural networks (GRNN) [29], which have been evaluated over 28 public datasets, and the results indicate that the model achieved higher accuracy. Convolutional Neural Network (CNN) is being used in an enhanced framework for SDP where pre-processes input data to extract various statistical and higher-order features, followed by feature selection using improved Principal Component Analysis

(PCA). The abnormal code snippets are being classified using the optimized CNN using the Seagull Adopted Ant Lion Optimization approach, and it is observed that the model performed well [30]. A study using a bidirectional long short-term memory (Bi-LSTM) network with the use of oversampling approach such as random oversampling and Synthetic Minority Oversampling Technique (SMOTE) has been used in the precise classification of abnormal code snippets [31]. Experimental outcome on PROMISE datasets demonstrate that the approach significantly improves SDP accuracy. Another method of combining the CNN model and gated recurrent units (GRU) with SMOTE Tomek, when experimenting with the PROMISE datasets, has been found to be efficient in the precise classification of abnormal code modules.

Daza [32] in a study on SDP using the multiclassifier with hyperparameters. Four hyperparameter-based stacking models are used to improve the classification performance, especially under imbalanced data conditions. The study has used the Kaggle dataset of over 10,000 defect records. The authors conducted a series of actions that involve preprocessing, model calibration, and evaluation through cross-validation and stacking techniques. The performing model—Stacking 3A with Gradient Boosting (GB) and oversampling has achieved the highest accuracy of 95.64% and ROC-AUC of 98.00%, demonstrating that ensemble learning significantly enhances early defect detection. Zhijing et al. [33] has introduced a hybrid approach that combine residual networks with an improved Fish Migration Optimization algorithm for better SDP by capturing both semantic and structural code properties. The model hybrid model has achieved about 93% of accuracy on evaluation.

Xu et al. [34] has proposed an explainable SDP model that simultaneously trains both the predictor and its interpreter to improve accuracy and interoperability. By incorporating feedback loops and penalizing unreliable interpretations in the loss function. By making use of the Knowledge Distillation process and incorporating interpretation results into the loss function, the model captures richer decision logic and outperforms baseline methods like SVM, RF, Deep Belief Network (DBN), and CNN in terms of F-measure and AUC across standard datasets. Awal et al. [35] evaluates the robustness the rule-based Explainable AI techniques, specifically PyExplainer and LIME, in generating consistent explanations for ML models across software analytics tasks like defect prediction and code review classification. Experimental outcome has outlined the inconsistencies concerning to the XAI framework, which highlights the need for more robust XAI methods in real-world software engineering applications. TRGNet [36] is a deep transfer learning model that transforms the software module metrics into images and make use of the pre-trained GoogLeNet with a meta-estimator to enhance both within-project and cross-project defect prediction. TRGNet approach has improved the prediction performance by over 13% in within-project defect prediction and 16% in cross-project defect prediction, while also reducing memory usage and maintaining efficient computational performance.

TABLE 1. Details of the software projects in the PROMISE dataset that are considered in the current study.

Project	Number of Instances	Defect Percentage	Number of Defective Instances
camel-v1.6	2137	27.52	588
ivy-v1.2	352	11.36	40
log4j-v1.0	244	15.16	37
ant-v1.7	745	22.28	166
Xerces-v1.3	588	13.10	77
jedit-v4.1	1029	6.12	63

Current studies in SDP often lack interpretability and struggle with effectively capturing complex relationships in the data. The adaptive feature processing would assist in retaining critical features that assist in better precision. The XAI model would assist in understanding the underlying dependencies among the features that would assist in obtaining a better comprehensibility model [12].

III. BACKGROUND

The current section outlines the dataset used in the current study, feature processing, XAI-driven feature dependency analysis, and the details of the implementation environment across multiple sub-sections. The architecture diagram that comprises all the components of the proposed model is depicted in Figure 3.

The architecture diagram of the proposed SDP model using the Autoencoder and MLP. It begins with the input layer, which accepts the software code artifacts formatted into structured data for further analysis. These inputs undergo Adaptive Feature Engineering for normalization and are standardized to ensure consistency and improve the effectiveness of learning processes. The processed data is then passed through an Autoencoder module, to perform the feature selection. This includes the encoder Phase that compresses input features into a lower-dimensional space and the Decoder Phase that reconstructs the original input to preserve information. Finally, the Loss Analysis step evaluates reconstruction quality and ensures that meaningful patterns are retained.

The architecture incorporates an XAI Analysis module, that used the sharply analysis to analyze the dependencies among the features. Finally, the Multi-Layer Perceptron (MLP) Classifier, which receives the learned representations and performs classification to determine whether a software module is defective or non-defective.

A. DATASET DESCRIPTION

In the current study, open-access publicly accessed imbalanced datasets from the PROMISE repository are being used [37]. The datasets comprise software metrics, such as lines of code (LOC), cyclomatic complexity, and coupling,

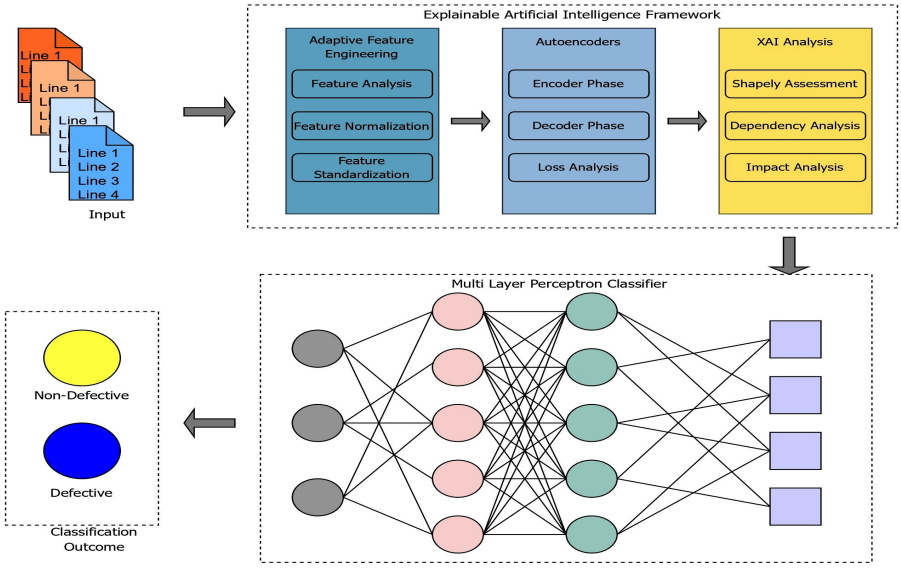


FIGURE 3. The architecture diagram of the proposed SDP model.

as well as defect labels indicating whether a software module is defective or non-defective. The data in the repository is imbalanced in nature, and it consists of diverse metrics that capture the complexity, design, and structural aspects of the software models. Some of the metrics include LOC, cyclomatic complexity, and coupling and defect labelling that assist in the training and evaluation of the learning models. The details of all the six projects that are being considered in the current study are discussed in Table 1.

Furthermore, the details of the various features that are part of the PROMISE dataset for fault localization are presented in Table 2. These features are processed using the adaptive feature engineering approach to recognize their significance in the decision process. The correlation across all the features listed in the dataset can be depicted from the heatmap image that is shown in Figure 4.

B. ADAPTIVE FEATURE ENGINEERING

The aforementioned features are being processed using the adaptive feature engineering strategy. Before the feature engineering is applied to the dataset, the feature normalization and scaling techniques are applied to the dataset to pre-process the training data. The operations that are presented on the input data are processed. Feature normalization is performed in the input data to confine the values in a given range. The primary purpose of data normalization is to ensure that all features contribute equally in the learning process. The corresponding formula for the data processing using the min-max normalization is presented in Equation 1.

$$v' = \frac{v - v_{min}}{v_{max} - v_{min}} \tag{1}$$

In the equation, the alphabet v denotes the actual value of the instance, the symbol v' designates the normalized value of the feature. The symbol v_{min} represents the minimum value among all the instances corresponding to the feature and, similarly, v_{max} designates the maximum value among all the instances. Then the feature standardization is performed across the instances using the Gaussian distribution. The associated formula for the feature standardization is identified by S_v is shown in Equation 2.

$$S_v = \frac{v' - \mu}{\sigma} \tag{2}$$

In the equation, the symbol μ represents the mean value of the feature value across all the instances. The symbol σ represents the standard deviation value associated with the feature.

1) FEATURE SELECTION

Feature analysis and selection are being performed using the Pearson correlation coefficient technique with Autoencoder approaches to precisely identify the features that contribute to the decision process. Initially, the Autoencoder technique is being used in the feature selection process [38]. The autoencoder-based feature selection is processed using the encoder and decoder phase for feature selection. Either of the phases would compress and uncompress the data as part of the feature selection, and finally, the loss measure is calculated to assess the robustness of the autoencoder model in the feature selection process. The encoder phase would compress the input data d into a smaller size representation z, which is known as the latent space. As it compresses the data, only the significant features that represent the input data are being retained, and lesser significant features are being discarded. The representation of the latent space in the encoder phase is shown in Equation 3.

$$z = f_e(d) = \gamma(\omega_e d + \beta_e) \tag{3}$$

TABLE 2. Details of the feature in the PROMISE software faulty detection repository.

Feature Label	Full Abbreviation	Feature Description
amc	Average method complexity	The average size of methods within a particular class.
avg_cc	Average Cyclomatic complexity	The highest McCabe's cyclomatic complexity (CC) score assigned to a particular method.
ca	Afferent coupling	Measures the count of classes that rely on a specific class.
cam	Cohesion among methods of class	Ratio of the sum of all parameter types of every method within a class to the product of all methods in the class and the count of method parameter types in the entire class.
cbm	Coupling between methods	The total count of new or rewritten methods that all class-inherited methods are connected to.
cbo	Coupling between objects	The count of classes that are connected to a particular class.
ce	Efferent coupling	Quantifies the count of all classes on which a specific class relies.
dam	Data access metric	The ratio of private/protected attributes to total attributes in a class.
dit	Depth of inheritance tree	The maximum distance between a particular class and the root of an inheritance tree.
ic	Inheritance coupling	The count of parent classes that a class is connected to.
lcom	Lack of cohesion of methods	The count of method pairs in a class which do not have access to any class attributes.
lcom3	Lack of cohesion in methods	Structural and logical qualities of source code, including McCabe's cyclomatic, essential, and design complexity.
loc	Lines of code	The number of lines of code.
max_cc	Maximum Cyclomatic complexity	Quantifies the maximum number of linearly independent paths through a program's source code.
mfa	Measure of functional abstraction	Ratio of all methods inherited by a class to all methods that can be accessed by its member methods.
moa	Measure of aggregation	The count of attributes in a class with user-defined types.
noc	Number of children	The total number of subclasses of a class in an inheritance tree.
npm	Number of public methods	The number of public methods in a class.
rfc	Response for classes	Total number of different methods called by code in a specific class.
wmc	Weighted methods for class	Sum of the complexity values across all methods in a class, possibly weighted.

In equation, the symbol γ represents the activation function being used. In the current study, a Rectified Linear Unit (ReLU) is used. The symbol ω_e denotes the weight matrix at the encoder side, which assist in transforming the input into a new feature space. The symbol β_e represents the bias vector, that assists in adjusting the output value. The decoder reconstructs the input d from the latent space representation, which ideally performs the inverse operation, mapping back the bottleneck to its original dimension. The same was being represented using Equation 4.

$$d' = f_d(z) = \gamma(\omega_d z + \beta_d) \tag{4}$$

From the equation, the symbol d' represents the reconstructed version of the input d. The symbol ω_d designates the associated weight matrix at the decoder side. The symbol β_d designates the bias vector associated with the decoder component. The mathematical representation of the ReLU function is shown in Equation 5.

$$f(x) = max(0, x) \tag{5}$$

The ReLU activation is efficient in capturing complex patterns from the data, and it allows gradients to propagate without significant attenuation for positive inputs [39]. The diagram representing the encoder and decoder phase of the autoencoder component as depicted in Figure 5. In the process of selecting the precise features, the selection of the features is updated continuously until the minimal loss value is attained. The Binary Cross-Entropy (BCE) approach is being used in the current study for assessing the loss measure. The corresponding formula for the BCE approach is presented in Equation 6.

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[x^i log\hat{x}^i + (1-x^i)log(1-\hat{x}^i)\right] \tag{6}$$

In equation, the alphabet n designates the count of samples being considered. x^i is the actual label associated with the sample. The symbol x^i is the predicted label of the i^{th} sample of the data. The logarithmic value $log\hat{x}^i$ is used to penalize overconfident incorrect predictions of the positive class. $(1-x^i)$represents the ground truth of the negative class and $log(1-\hat{x}^i)$ penalize overconfident incorrect predictions for the negative class. The resultant outcome of the autoencoder is presented in a heatmap image in Figure 6. Now, the features that are being selected using the autoencoders are further processed using the Pearson correlation coefficient (PCC) [40] for approximating the correlation coefficient to determine the feature weight. The formula for PCC is shown in Equation 7. 7.

$$C_{i,T} = \frac{\sum_{j=1}^{N}\left(m_{i,j} - m_i'\right)\left(n_j - n'\right)}{\sqrt{\sum_{j=1}^{N}\left(m_{i,j} - m_i'\right)^2\left(n_j - n'\right)^2}} \tag{7}$$

From the equation, the symbol T designates the target variable, the symbol $m_{i,j}$ designates the j^{th} the instance of the feature m_i, Similarly, the n_j designated the j^{th} instance of the target value. The symbol m_i'represents the mean of values

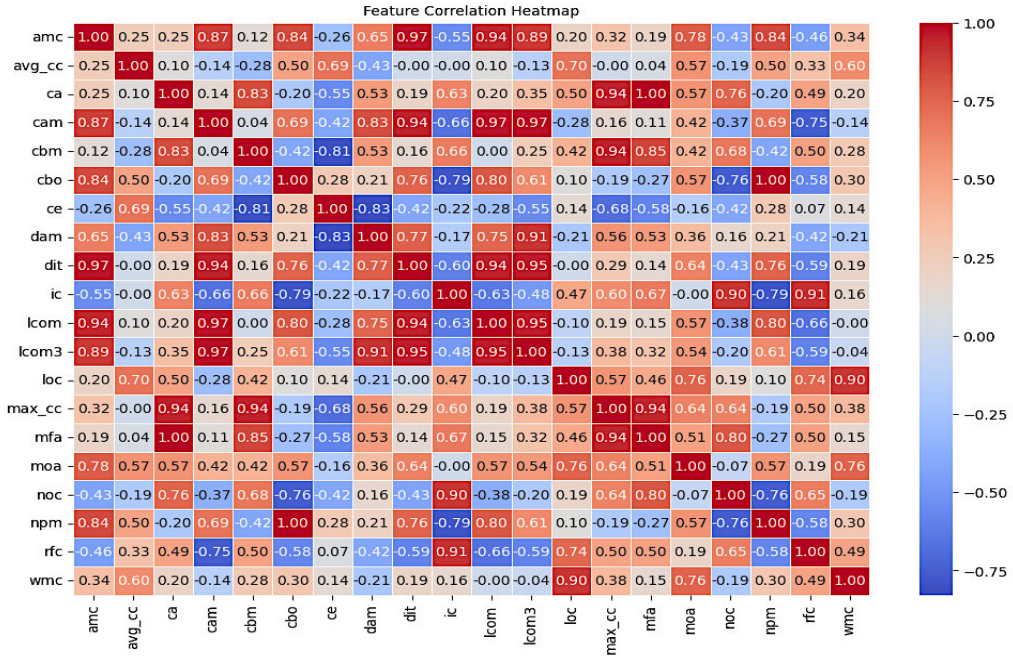


FIGURE 4. The feature correlation heatmap of the PROMISE dataset.

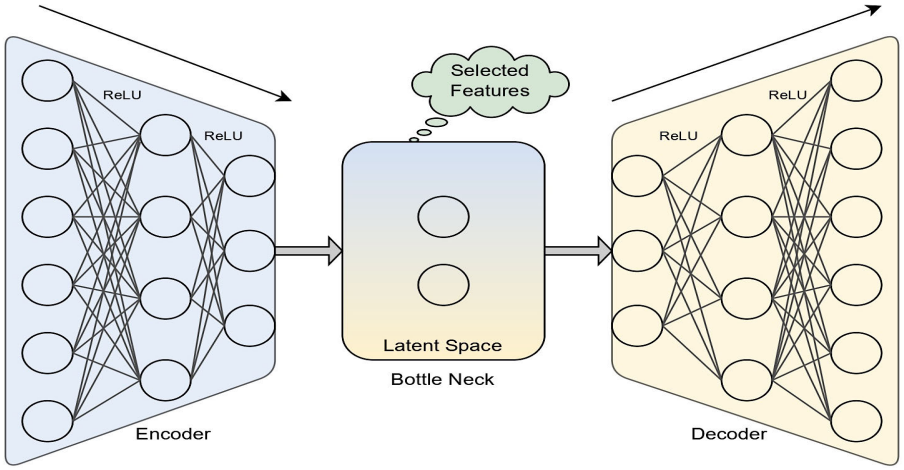


FIGURE 5. The architecture diagram of the autoencoder model for feature selection.

associated with the feature m_i, and n' represents the mean of the target value. Now, the initial rank $\mathbb{R}_{m_i}$ is assigned to the features based on the absolute correlation coefficient as shown in Equation 8.

$$\mathbb{R}_{m_i} = \frac{\left|\rho_{m_i,n}\right|}{\sum_{j=1}^{n}\left|\rho_{m_i,n}\right|} \tag{8}$$

2) FEATURE RANK UPDATION

The feature rank value is updated over the training process concerning the learning rate of the model. The corresponding formula for rank updating is shown in Equation (9).

$$\mathbb{R}'_{m_i} = \mathbb{R}_{m_i} - \eta\frac{\partial L}{\partial \mathbb{R}_{m_i}} \tag{9}$$

From the equation, the $\frac{\partial L}{\partial \mathbb{R}_{m_i}}$ designated the gradient of the loss function concerning to input $\mathbb{R}_{m_i}$. Furthermore, the feature dependency and significance are assessed using the XAI models for better interpretability of the current model.

C. XAI BASED ANALYSIS

The XAI-based analysis is exceptionally significant in understanding and trusting the decision-making approach in

FIGURE 6. Latent features extracted via autoencoder for model training.

the prediction model. The feature analysis in the current study is analyzed using the Shapley Additive exPlanations (SHAP) based analysis [41]. The SHAP-based analysis would assist in quantifying the contributions of every feature to the model's prediction. The use of SHAP in feature engineering to anticipate SDP bridges the gap between model performance and interpretability. The corresponding formula for the SHAP-based feature analysis is presented through Equation 10.

$$\phi_x = \sum_{p \subseteq F\{x\}} \frac{|p|!\,(k-|p|-1)!}{k!} [f(p \cup \{x\} - f\,(p))] \quad (10)$$

From the equation, the symbol p designates the subset of features excluding $x(p \subseteq F\{x\})$, and the symbol $|p|$ designates the count of features in the sub-set p. The symbol $f(p)$ represents the features in x that contribute to the model's prediction. The symbol $f(p \cup x)$ denotes the model's prediction over the features in p and x. The output of the model corresponding to the input is expressed as the sum of the Shapley values, and the same was shown in Equation 11.

$$f(x) = f(\phi) + \sum_{x=1}^{k} \phi_x \quad (11)$$

The feature significance in making the final assessment is assessed by assessing the mean absolute Shapley value for each feature across all instances, as shown in Equation 12.

$$s_f = \frac{1}{n} \sum_{i=1}^{n} \left| \phi_x^i \right| \quad (12)$$

In the equation, the notation ϕ_x^i denotes the Shapley value of feature x for the i^{th} Sample. The notation n designates the number of instances. The features that are exceptional in the decision process of defect detection are presented in Figure 7. The Shapley dependency graphs are good at providing a visual representation of feature dependencies and their contributions to model predictions. The positive and the negative dependencies among the features can be depicted using dependency graphs. Figure 8 presents the dependency graphs for some of the most significant features, offering insights into their individual and interactive effects on the model's performance.

D. HYPERPARAMETERS

The current section outlines the hyperparameters and the experimental configuration that are used in evaluating the current model. The settings are set to be the same throughout the experimental process without changing the configuration. The finetuning of the parameters would yield better accuracies [42]. However, that analysis has not been done in the current study. The hyperparameters are described in Table 3.

E. IMPLEMENTATION SETUP

The sub-section discusses the specifications of the execution environment that is being used in the current study. The evaluation of the model is done through the Google Colab platform. The details of the standalone machine for implementation and various libraries that are used in the implementation process are presented in Table 4.

IV. MULTI-LAYER PERCEPTRON FOR SDP

The current section outlines the role of the multi-layer perceptron model in classifying abnormal code snippets. MLP classifier has been proven to be efficient in handling non-linear data in a much more effective way [12], and it works well with multi-feature datasets. MLPs are efficient in assessing the integrals, making it possible to handle the non-linearly separable problems. Built from units called perceptron or neurons, an MLP takes input features and assigns a specific weight to each one during processing. The input layer at the MLP would accept the input, where every single input variable a neuron is being assigned. The hidden layers do have the business logic, which performs all

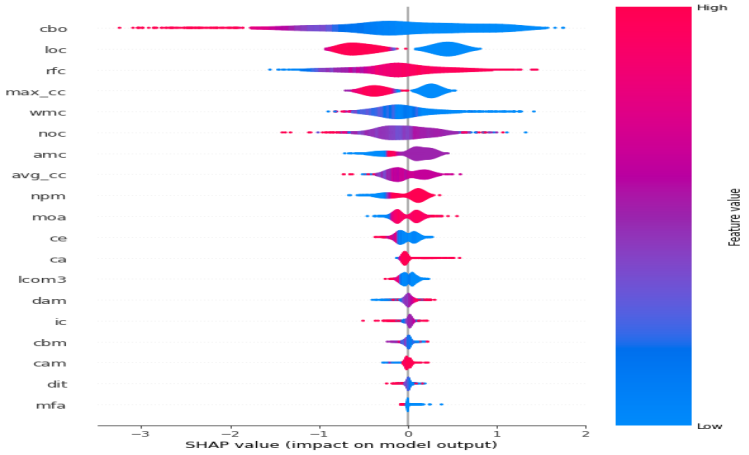


FIGURE 7. The graphs represent the feature significance in the model output.

TABLE 3. Experimental configuration parameters.

Parameter	Description
Number of Epochs	50 (Same for Autoencoder and MLP)
Activation Function - Autoencoder	ReLU (Encoder), Sigmoid (Decoder)
Activation Function - MLP	ReLU
Optimizer Used	Adam
Batch Size	32
Learning Rate	0.001 (Same for Autoencoder and MLP)
Dropout Factor	0.2
Loss Function - MLP	Binary Cross-Entropy
Loss Function - Autoencoder	Mean Squared Error

TABLE 4. System environment and experimental setup.

Environment	Details
Machine and Make	Windows 11 Asus Vivobook S 15
Processor	Core i9 13th Generation 2.6GHz
Storage	16 GB LPDDR5 RAM
Memory Allotment	30 GB
Online Platform	Google Colab
GPU	NVIDIA Tesla T4
Programming Language	Python
Libraries Used	Scikit-learn, PyTorch, Pandas, Matplotlib, SHAP

processes. The output layer returns the belongingness of out across various classes. Let us assume there are F features that are being scaled down to k features by using the autoencoder from the input I having the instances $i_1, i_2, \ldots, i_n$ for all the n inputs. A function is applied over the input samples that assess the value by adding up the corresponding values of all the input samples, the formula is shown in Equation 13.

$$f(I) = \sum_{x=1}^{n} \omega_x i_x \tag{13}$$

From the equation, the ω_x designates the weights that are associated to the data sample x. The weights are labelled as ω_{ih} for all connections from the input to the hidden layer and similarly, ω_{ho} for all connections from the hidden to the output layer. The network adjusts the weights and biases to learn the connection between input units and the predicted outputs based on feedback. The corresponding formula for MLP approach over the r^{th} neuron corresponding to the m^{th} node over the presumed inputs $s_1, s_2, \ldots, s_x$ at the corresponding layer is shown in Equation 14.

$$O_m = \sum_{r=1}^{m} \omega_{h0}^{2} f\left(\sum \omega_{rh}^{1} s_v(m) + t_h\right) \tag{14}$$

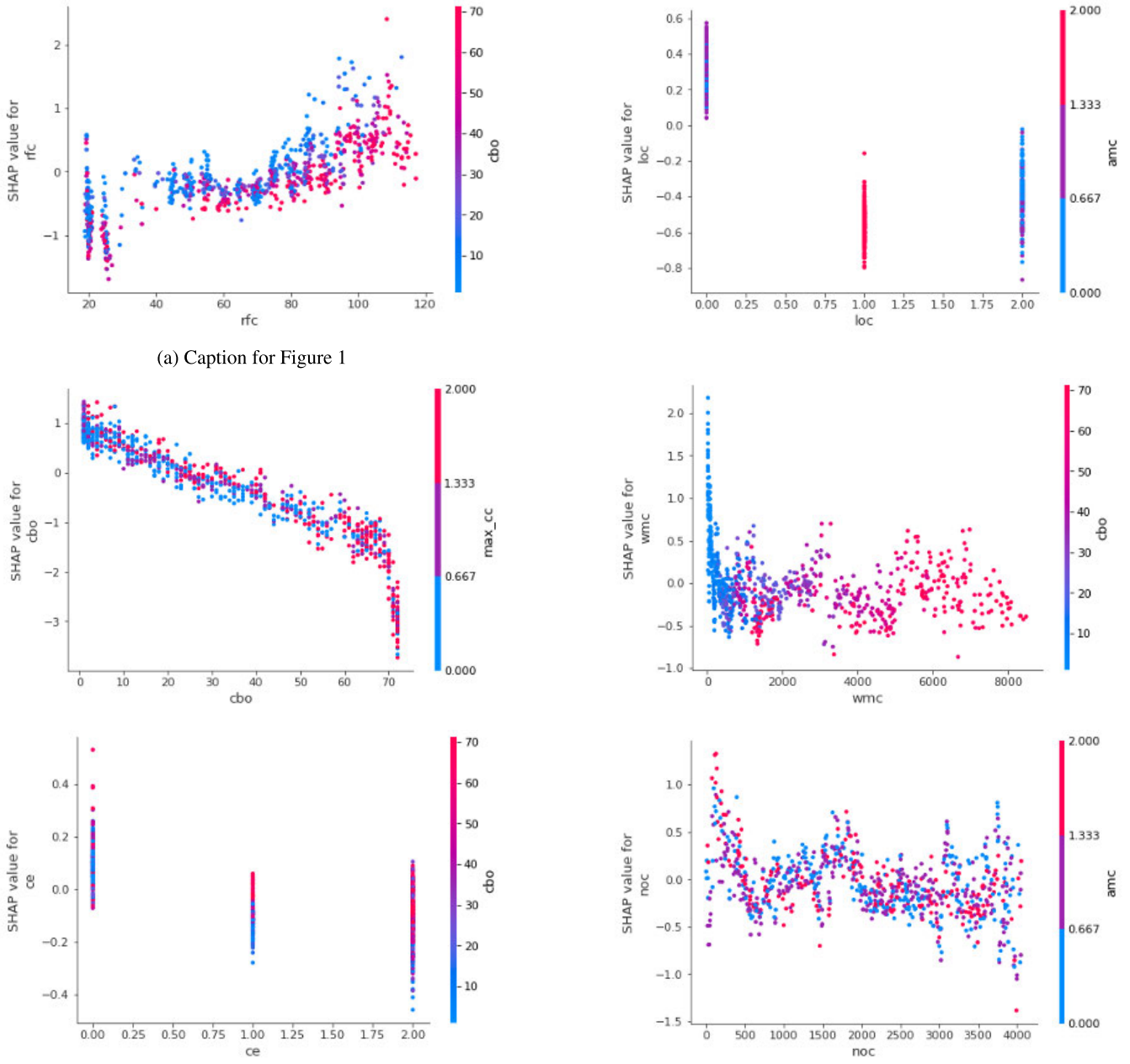
From the equation, the notation t_h designated the assessed threshold at the hidden layer, where it is assumed to be the bias term. The $f()$ is the activation function applied a non-linearity to the weighted sum of inputs. The ReLU function is being used, and the corresponding formula is shown in Equation 15.

$$f(x) = max(0, x) \tag{15}$$

The next sub-layer in the hidden layer will take the output of previous layer fed as input for additional processing. The values from the output layer are used to calculate the neuron's output, as shown in Equation 16.

$$O_m = \sum_{r=1}^{m} \omega_{h0} \times O_x\,(m) + \beta_{h0} \tag{16}$$

From the equation, the notation β_{h0} bias value that is associated with the hidden output layer. The architecture diagram of the MLP can be depicted through Figure 9, which is composed of input, hidden, and output layers.



(a) Caption for Figure 1

FIGURE 8. **The dependency graphs of various significant features in the decision process.**

A. MODEL EVALUATION

Furthermore, the model is evaluated concerning the hyper-parameters like the loss and accuracy measures at both the training and validation phases. The corresponding graphs would assist us in understanding the generalizability of the model. The corresponding loss and accuracy graphs over the 50 epochs are shown in Figure 10. It can be observed from the loss and the accuracy graphs that the model has been appropriately trained, where the model performs well during the training phase as well as during the validation phase.

V. EMPIRICAL STUDY

To validate the effectiveness of the proposed XAI-driven SDP using the multi-layer perception and autoencoders. Initially, after the features are analyzed using the Autoencoders, the most significant features are selected to carry out the classification process. The significance of the features are analyzed using statistical ranking and model-based relevance scores, and furthermore, the feature dependencies are being analyzed. This analysis was critical in ensuring that the autoencoder preserved insightful feature representations during the process.

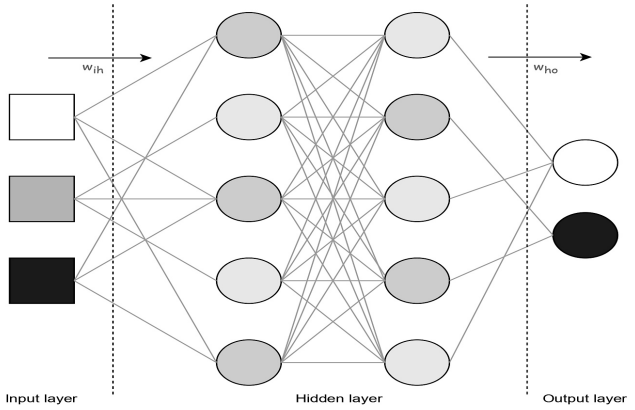


FIGURE 9. The architecture diagram of the multi-layer perceptron neural network.

The model's performance was rigorously evaluated using widely used metrics for classification, which include Accuracy, Precision, Recall, F1-Score, and the Area Under the Receiver Operating Characteristic Curve (ROC-AUC). These metrics would outline the holistic view of the classifiers ability in distinguishing the defective and non-defective modules. The empirical results demonstrated that the integration of adaptive feature selection and non-linear feature compression significantly enhanced the MLP's generalization capability.

The empirical results demonstrated that the integration of adaptive feature selection and non-linear feature compression significantly enhanced the MLP's generalization capability. The performances are evaluated across various configurations as shown in the Table 5.

The steps involved in the empirical process are listed below.

- The first approach involve working with MLP model directly on raw input features, without any feature engineering or selection, serving as the baseline.
- In the second approach, relevant features engineering approaches like min-max normalization, and feature standardization of performed. Then the resultant dataset is given as input to the MLP model.
- The third approach has applied an autoencoder to compress the raw feature space and identifying the significant feature that contributed better data classification and MLP is used along with them.
- Finally, the MLP model is applied over the data that is processed using the adaptive feature engineering approach and selected features using the Autoencoder.

VI. RESULTS AND DISCUSSION

The current section presents the statistical analysis of the performance of the proposed approach along with the outcomes of the SOA techniques used for SDP. In the current study, the model is evaluated in terms of standard metrics such as sensitivity, precision, recall, precision, and F1 score [43]. The dataset was first divided into training set of 70%, validation set of 10%, and testing set of 20% sets to ensure the robustness of the model evaluation and generalization. Adaptive feature engineering was applied to the input data, which involved feature normalization and standardization to eliminate scale disparities and improve convergence during training. Multiple model configuration were considered, including baseline MLPs and enhanced architectures with adaptive feature engineering and autoencoders. The training process utilized the training set to optimize the model weights, while the validation set was employed for hyperparameter tuning and early stopping.

The performance of the model is evaluated individually for each of the projects. The current study considers six different projects, including camel, ivy, log4j, ant, Xerces, and jedit. The ablation study is also being performed to assess the efficiency of the model alone without the feature engineering task. The complete input data is considered to be across binary classes, which include the code snippets that are normal and have defects. The same was assessed across the projects, and the obtained confusion matrices are presented in Figure 11, and the associated values are presented in Table 6. It can be observed from the obtained results the proposed model has performed well, and the overall accuracy across all the projects is close to 98.3%, and precision is close to 99.8%, which is assumed to be good accuracy in SDP. Furthermore, the current model is also being evaluated concerning the receiver operating characteristic (ROC) curves [44], which outlines how well the proposed classification model is able to classify both classes. The ROC curve illustrates a classifier's efficacy in distinguishing both classes by displaying the true positive rate against the false positive rate at various threshold values. The ROC curves across all the projects are presented in a single graph in Figure 12.

A. ABLATION STUDY

The ablation study is conducted to analyze the robustness of the MLP model in the SDP, without the role of adaptive feature engineering using an Autoencoder. The observed results in the form of a confusion matrix are presented in Figure 13, and their associated values are presented in Table 7. The analysis of the performances for both the current model and the model that is presented in the ablation study without the feature processing component is being compared across all the projects, and the same has been presented as bar graphs in Figure 14. To evaluate the individual contribution of each component within the proposed SDP framework, the ablation study was further extended using four configurations as discussed in the empirical analysis as shown in Table 5. These configurations includes a basic MLP trained on raw features to the complete pipeline integrating both adaptive feature engineering and autoencoder-based feature selection. This analysis aims to highlight the incremental performance gains achieved at each component, and to justify the

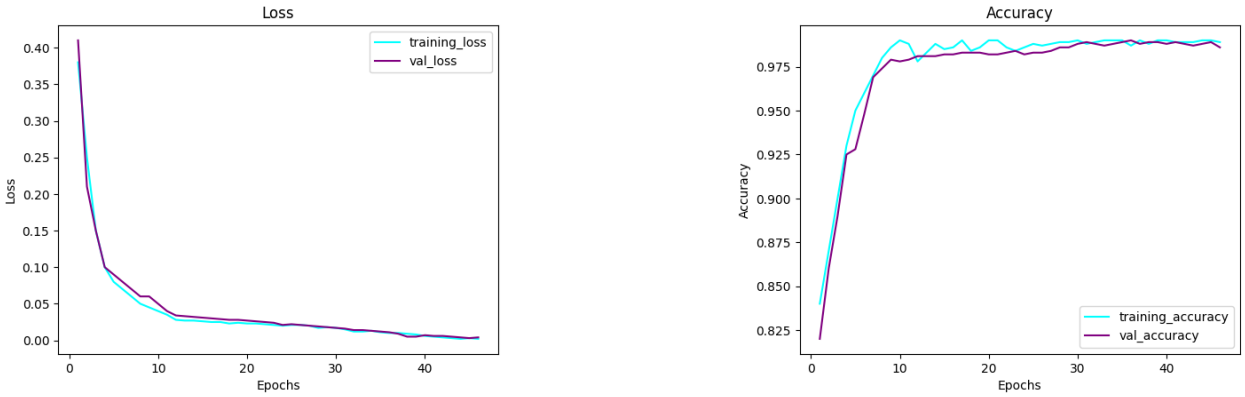


FIGURE 10. Training performance of the proposed model: (a) Loss curve and (b) Accuracy curve over epochs.

TABLE 5. Model approaches considered in the empirical analysis.

Approach	Configuration Used	Description
Configuration-1	Raw features + MLP	No feature engineering or the autoencoder being used; Data being directly classified using MLP.
Configuration-2	Selected features + MLP	Adaptive feature engineering being applied, but no autoencoder. Followed by an MLP-based classification.
Configuration-4	Autoencoder + MLP	Autoencoder are used for feature selection before final classification using MLP. But the feature engineering is not performed
Proposed Approach	Feature Engineering + Autoencoder + MLP	Autoencoder is performed after the feature engineering and final classification using MLP.

TABLE 6. Experimental results of the proposed model.

Project	Precision	Recall	Accuracy	F1-Score
camel	0.991	0.943	0.981	0.966
ivy	1.000	0.800	0.971	0.888
log4j	1.000	0.875	0.979	0.933
ant	1.000	0.942	0.986	0.970
xerces	1.000	0.937	0.991	0.968
jedit	1.000	0.866	0.990	0.928
Overall	**0.998**	**0.893**	**0.983**	**0.942**

TABLE 7. Evaluation outcomes of the proposed model without feature processing.

Project	Precision	Recall	Accuracy	F1-Score
camel	0.991	0.928	0.976	0.959
ivy	0.750	0.750	0.943	0.750
log4j	0.857	0.750	0.938	0.800
ant	0.909	0.909	0.959	0.909
xerces	0.800	0.800	0.949	0.800
jedit	0.692	0.692	0.961	0.692
Overall	**0.833**	**0.804**	**0.954**	**0.818**

TABLE 8. Performance comparison of different model configurations.

Approach	Precision	Recall	Accuracy	F1-Score
Configuration-1	0.755	0.702	0.872	0.728
Configuration-2	0.785	0.751	0.905	0.768
Configuration-3	0.812	0.778	0.931	0.794
Proposed Approach	0.998	0.893	0.983	0.942

design of the proposed approach. The obtained experimental results are presented in the Table 8. The outcome of the ablation study, demonstrates the incremental performance improvements achieved by progressively enhancing the model configuration. Configuration-1, which has used the raw features directly with an MLP, achieved the lowest scores across all metrics, highlighting the limitations of using random features. Introducing adaptive feature engineering has resulted in noticeable gains in precision with 0.781, recall with 0.744, and overall accuracy 0.902, confirming the significance of feature. Similarly, Configuration-3, which has used autoencoder without prior feature selection, showed further improvement. The best results were obtained with the proposed approach, which combined both adaptive feature engineering and autoencoder-based configuring achieving a better accuracy of 0.954. These findings validate the effectiveness of integrating both feature engineering strategies in enhancing model robustness and efficiency.

Furthermore, the proposed model is evaluated concerning to the wilcoxon signed rank (WSR) [45] for the proposed model. To assess the statistical significance of the performance between proposed model and the other baseline configurations as discussed in empirical study. The current

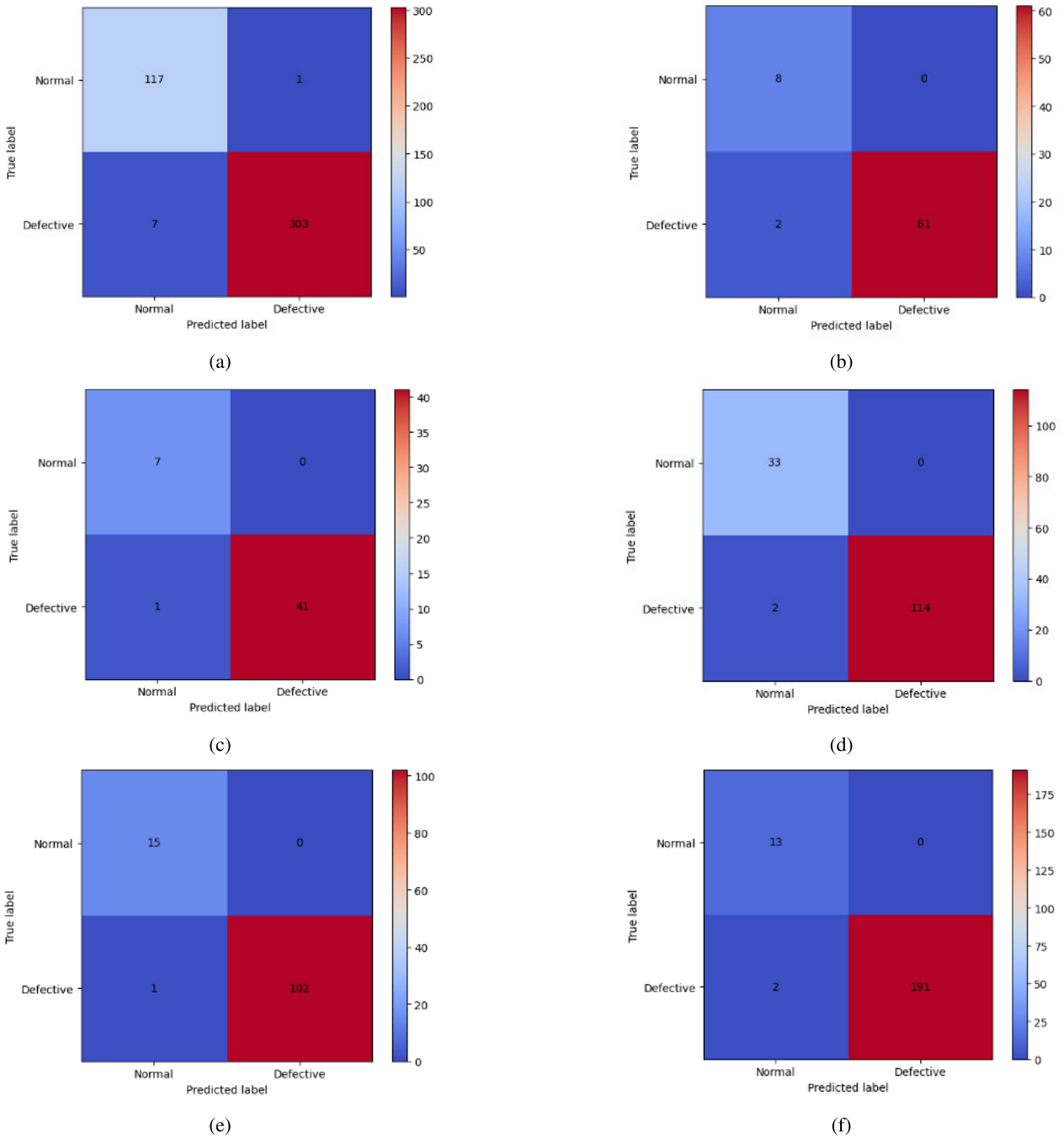


FIGURE 11. The loss and accuracy graphs of the proposed model. The sub-figures Figure corresponding to the camel project, corresponding to the ivy project, (c) corresponds to the log4j project, (d) corresponds to the ant project, (e) corresponds to the Xerces project, and (f) corresponding to jedit project.

study has considered assessing the model using WSR test, which is a non-parametric statistical test used for comparing two related samples. The corresponding formula for WSR is shown in Equation 17.

$$W_i = X_i - Y_i \tag{17}$$

From the equation, the X_i is corresponding to the proposed model and Y_i is corresponding to the baseline configuration. The test is run for multiple cross-folds repeatedly. In the current study the evaluation is done for 5 fold and the observed values are presented in Table 10.

B. ANALYSIS WITH SOA MODELS

The proposed Multi-Layer Perceptron with adaptive feature processing with autoencoder is being analyzed in conjunction with the other SOA models used in SDP. The obtained

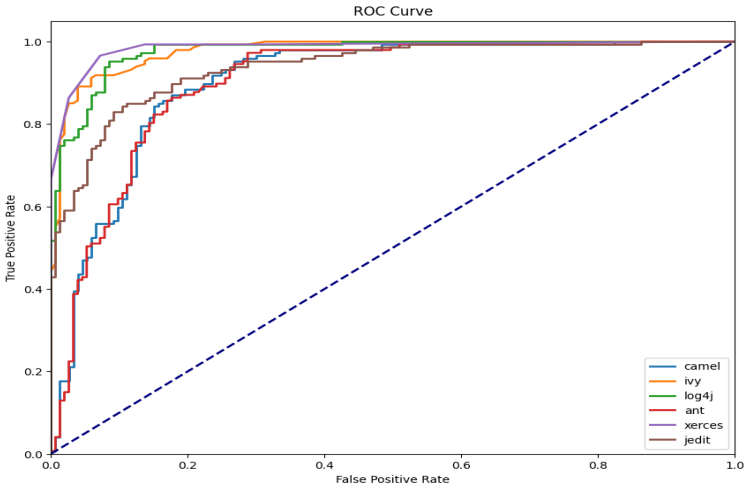


FIGURE 12. The architecture diagram of the multi-layer perceptron neural network.

TABLE 9. Evaluation out of wilcoxon signed-rank test across different configurations.

Model Compared	Folds	p-value
Configuration-1	5	0.005
Configuration-2	5	0.014
Configuration-3	5	0.019

TABLE 10. Comparative analysis with SOA models.

Approaches	Precision	Recall	Accuracy	F1-Score
Dynamic Classifier [48]	0.924	0.942	0.943	0.942
Deep Q-Learning [49]	0.790	N/A	0.830	0.780
LSTM [50]	0.920	0.920	0.910	0.920
Bi-LSTM [50]	0.925	0.921	0.929	0.921
RBFN [50]	0.815	0.811	0.818	0.811
DBN [51]	0.534	0.816	N/A	0.645
CNN [51]	0.534	0.759	N/A	0.627
ARNN [51]	0.538	0.618	N/A	0.575
GNN [51]	0.639	0.713	N/A	0.673
BERT [51]	0.769	0.626	N/A	0.689
SLDeep [52]	0.570	0.979	0.980	0.920
NF [53]	0.508	0.612	N/A	0.566
NN [54]	0.949	0.978	0.936	0.946
DT [48]	0.846	0.963	0.894	0.901
BAG [48]	0.888	0.963	0.921	0.924
Adaboost [48]	0.711	0.747	0.721	0.728
GaussianNB [48]	0.782	0.280	0.601	0.412
RF [48]	0.898	0.972	0.923	0.934
Proposed Model	**0.998**	**0.893**	**0.983**	**0.942**

performances are being compared with various existing studies like LSTM, Bi-LSTM, Radial basis network (RBFN), Deep Belief Networks (DBN), CNN, Adaptive Recurrent Neural Network (ARNN), Graph neural networks (GNN) and Bidirectional Encoder Representations from Transformers (BERT), neural network(NN), Decision tree (DT), Bagging (BAG), Random forest (RF) and neural forest (NF) in Table 10.

C. POTENTIAL LIMITATIONS

Despite the promising outcomes the study have some potential limitations. one of the key limitation of the study lies with the dataset used for evaluating the proposed model, that is the PROMISE dataset, which is widely adopted in academic research, is relatively outdated. Modern software programs are comparatively more complex and generate much advanced and context-dependent bugs. As a result, the performance of the proposed model may not translate directly to real-world. Additionally, the PROMISE dataset lacks dynamic behavioral attributes, and the feature engineering that is used in the current study did not include data balancing, which would reduce model robustness. The class balancing has not been performed in the current study, which would have a considerable impact on the performance of the model's outcome. In the comparative analysis with other SOTA models, different datasets have been used in some of the studies, which limits direct comparability.

The evaluation of the model is limited only to the PROMISE dataset, but the performance of the model is not evaluated over a large dataset, and the model's ability to handle diverse programming languages, which is another potential limitation of the current study. From the explainability perspective, the current study has employed a Shapley based approach to quantify the individual feature contributions. Analysis is primarily limited to analyzing feature dependencies and independence. It does not fully capture the underlying model decision logic or complex feature interactions. Furthermore, the decision process of the model remains partially interpretable, and techniques like LIME, which offer a local approximation of the model's behavior, provide more comprehensive and instance-specific interpretations.

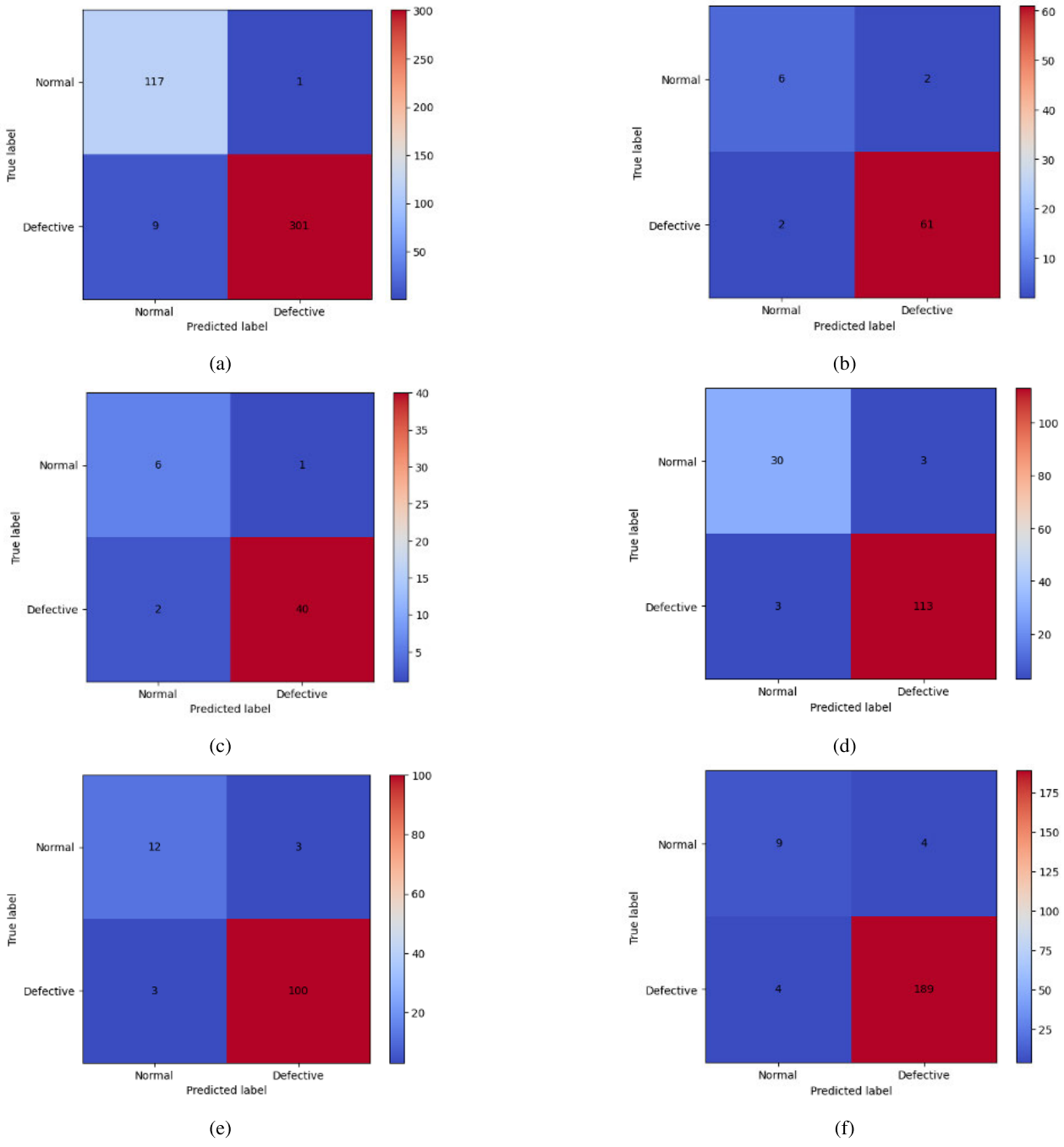


FIGURE 13. The loss and accuracy graphs of the proposed model without the feature processing task. The sub-figures Figure (a) corresponds to the camel project, (b) corresponds to the ivy project, (c) corresponds to log4j project, (d) corresponds to the ant project, (e) corresponding to the xerces project, and (f) corresponding to jedit project.

D. THREATS TO THE VALIDITY

In the process of conducting the comparative study with SOA models in the SDP process, the current study approximates the overall performance calculated as the mean of the performances across all the projects. However, in the comparative analysis, the other model may not consider all the projects or different projects. However, to showcase the superiority of the current model, it is compared with the other projects irrespective of the projects that are being considered in the other studies. This is considered to be one of the potential limitations of the comparative analysis phase of the study. The PROMISE dataset has ten different projects, out of which only six are being considered in the current study, which is another potential limitation of the current study.

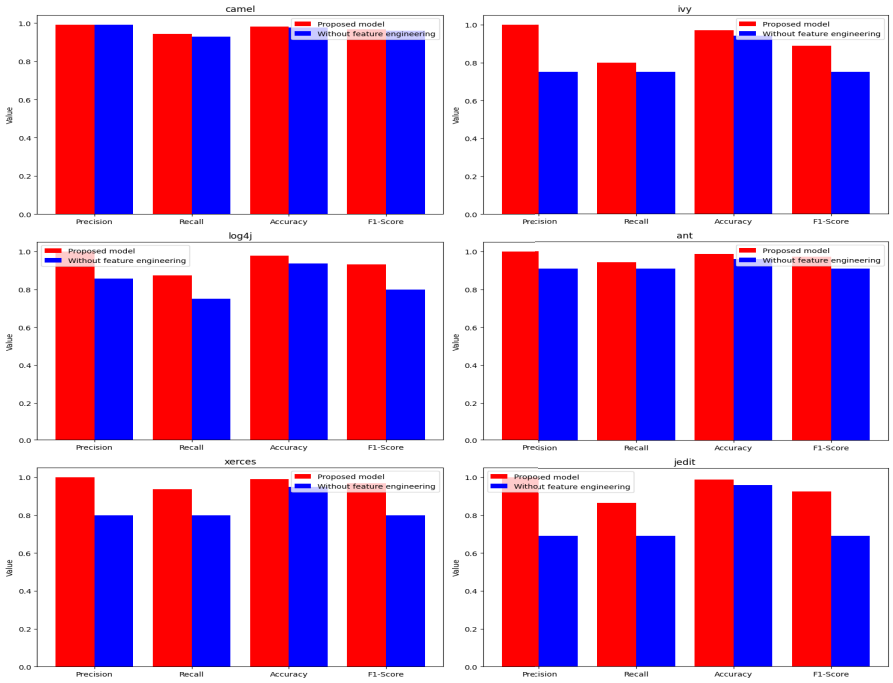


FIGURE 14. The graph represents the comparative analysis of the current model.

VII. CONCLUSION

The current study on software fault prediction uses the MLP with adaptive feature engineering with an autoencoder. The model is evaluated across six different projects in the PROMISE dataset, and the obtained results on evaluating across the standard metrics like precision, recall, accuracy, F1-score, and the ROC curves seem to be reasonably fair, and results across the majority of the projects outperforms the existing models used in SDP. The model is also evaluated without feature engineering as part of the ablation study, and it was observed that the current model yielded reasonable accuracy with feature engineering. The model interpretability is being analyzed using the shapely values, the significant features that have contributed to the decision process of the model, and an outline of the dependencies among various features in the dataset. The current model can be used as the future assistive model for defect prediction in the real-time scenario. The future research direction concerning the current study involves involving divergent projects in the evaluation process and using the cognitive machine intelligent models for more precise outcomes. The LIME based analysis can also be used in assessing the feature significance of the features in the decision-making process. Furthermore, the large language models can also be used to analyze the code snippets in real-time scenarios along with the current strategy for better, more precise outcomes.

REFERENCES

[1] H. Chen, L. Yang, and A. Wang, "Efficient cross-project software defect prediction based on federated meta-learning," *Electronics*, vol. 13, no. 6, p. 1105, Mar. 2024, doi: 10.3390/electronics13061105.

[2] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," *Eng. Appl. Artif. Intell.*, vol. 111, May 2022, Art. no. 104773, doi: 10.1016/j.engappai.2022.104773.

[3] H. Krasner. (2018). *The Cost of Poor-Quality Software in the U.S.: A 2018 Report*. [Online]. Available: https://www.it-cisq.org/wp-content/uploads/sites/6/2023/09/The-Cost-of-Poor-Quality-Software-in-the-U.S.-2018-Report.pdf

[4] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA, May 2016, pp. 297–308.

[5] L. Marwick. *Raygun.com*. Accessed: Dec. 5, 2024. [Online]. Available: https://raygun.com/blog/cost-of-software-errors/

[6] E. N. Akimova, A. Y. Bersenev, A. A. Deikov, K. S. Kobylkin, A. V. Konygin, I. P. Mezentsev, and V. E. Misilov, "A survey on software defect prediction using deep learning," *Mathematics*, vol. 9, no. 11, p. 1180, May 2021, doi: 10.3390/math9111180.

[7] M. Nevendra and P. Singh, "A survey of software defect prediction based on deep learning," *Arch. Comput. Methods Eng.*, vol. 29, no. 7, pp. 5723–5748, Nov. 2022, doi: 10.1007/s11831-022-09787-8.

[8] Y. Zhao, K. Damevski, and H. Chen, "A systematic survey of just-in-time software defect prediction," *ACM Comput. Surveys*, vol. 55, no. 10, pp. 1–35, Oct. 2023.

[9] B. G. Geçer and A. K. Tarhan, "Explainable AI framework for software defect prediction," *J. Softw., Evol. Process*, vol. 37, no. 4, pp. 1–32, Apr. 2025, doi: 10.1002/smr.70018.

[10] S. Goyal, "Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction," *Artif. Intell. Rev.*, vol. 55, no. 3, pp. 2023–2064, Mar. 2022, doi: 10.1007/s10462-021-10044-w.

[11] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Yasin Ghadi, and M. Amir Khan, "Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning," *PeerJ Comput. Sci.*, vol. 10, p. e1860, Feb. 2024, doi: 10.7717/peerj-cs.1860.

[12] P. N. Srinivasu, G. J. Lakshmi, A. Gudipalli, S. C. Narahari, J. Shafi, M. Woźniak, and M. F. Ijaz, "XAI-driven CatBoost multi-layer perceptron neural network for analyzing breast cancer," *Sci. Rep.*, vol. 14, no. 1, p. 28674, Nov. 2024, doi: 10.1038/s41598-024-79620-8.

[13] X. Li, W. Chen, Q. Zhang, and L. Wu, "Building auto-encoder intrusion detection system based on random forest feature selection," *Comput. Secur.*, vol. 95, Aug. 2020, Art. no. 101851, doi: 10.1016/j.cose.2020.101851.

[14] M. Assim, Q. Obeidat, and M. Hammad, "Software defects prediction using machine learning algorithms," in *Proc. Int. Conf. Data Analytics Bus. Industry: Way Towards Sustain. Economy (ICDABI)*, Sakheer, Bahrain, Oct. 2020, pp. 1–6, doi: 10.1109/ICDABI51230.2020.9325677.

[15] T. Yu, W. Wen, X. Han, and J. H. Hayes, "ConPredictor: Concurrency defect prediction in real-world applications," *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, pp. 558–575, Jun. 2019, doi: 10.1109/TSE.2018.2791521.

[16] X. Yu, J. Rao, L. Liu, G. Lin, W. Hu, J. W. Keung, J. Zhou, and J. Xiang, "Improving effort-aware defect prediction by directly learning to rank software modules," *Inf. Softw. Technol.*, vol. 165, Jan. 2024, Art. no. 107250, doi: 10.1016/j.infsof.2023.107250.

[17] S. Qiu, H. Huang, W. Jiang, F. Zhang, and W. Zhou, "Defect prediction via tree-based encoding with hybrid granularity for software sustainability," *IEEE Trans. Sustain. Comput.*, vol. 9, no. 3, pp. 249–260, May 2024, doi: 10.1109/TSUSC.2023.3248965.

[18] C. López-Martín, Y. Villuendas-Rey, M. Azzeh, A. Bou Nassif, and S. Banitaan, "Transformed K-nearest neighborhood output distance minimization for predicting the defect density of software projects," *J. Syst. Softw.*, vol. 167, Sep. 2020, Art. no. 110592, doi: 10.1016/j.jss.2020.110592.

[19] M. Azzeh, Y. Elsheikh, A. B. Nassif, and L. Angelis, "Examining the performance of kernel methods for software defect prediction based on support vector machine," *Sci. Comput. Program.*, vol. 226, Mar. 2023, Art. no. 102916, doi: 10.1016/j.scico.2022.102916.

[20] S. Goyal, "Effective software defect prediction using support vector machines (SVMs)," *Int. J. Syst. Assurance Eng. Manage.*, vol. 13, no. 2, pp. 681–696, Apr. 2022, doi: 10.1007/s13198-021-01326-1.

[21] G. Kaur, J. Pruthi, and P. Gandhi, "Decision tree regression analysis of proposed metric suite for software fault prediction," *Social Netw. Comput. Sci.*, vol. 5, no. 1, p. 69, Dec. 2023, doi: 10.1007/s42979-023-02386-9.

[22] N. S. Thomas and S. Kaliraj, "An improved and optimized random forest based approach to predict the software faults," *Social Netw. Comput. Sci.*, vol. 5, no. 5, p. 530, May 2024, doi: 10.1007/s42979-024-02764-x.

[23] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *J. Syst. Softw.*, vol. 180, Oct. 2021, Art. no. 111026, doi: 10.1016/j.jss.2021.111026.

[24] B. Arasteh, S. Golshan, S. Shami, and F. Kiani, "Sahand: A software fault-prediction method using autoencoder neural network and K-means algorithm," *J. Electron. Test.*, vol. 40, no. 2, pp. 229–243, Apr. 2024, doi: 10.1007/s10836-024-06116-8.

[25] *Promise Software Engineering Repository*. Accessed: Jun. 10, 2025. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[26] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 969–980, Sep. 2015, doi: 10.1007/s11390-015-1575-5.

[27] S. Thapa, A. Alsadoon, P. W. C. Prasad, T. Al-Dala'in, and T. A. Rashid, "Software defect prediction using atomic rule mining and random forest," in *Proc. 5th Int. Conf. Innov. Technol. Intell. Syst. Ind. Appl. (CITISIA)*, Sydney, NSW, Australia, Nov. 2020, pp. 1–8, doi: 10.1109/citisia50690.2020.9371797.

[28] H. Yu, X. Sun, Z. Zhou, and G. Fan, "A novel software defect prediction method based on hierarchical neural network," in *Proc. IEEE 45th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Madrid, Spain, Jul. 2021, pp. 366–375, doi: 10.1109/COMPSAC51774.2021.00059.

[29] F. Alghanim, M. Azzeh, A. El-Hassan, and H. Qattous, "Software defect density prediction using deep learning," *IEEE Access*, vol. 10, pp. 114629–114641, 2022, doi: 10.1109/ACCESS.2022.3217480.

[30] D. S. Balasubramaniam and D. S. G. Gollagi, "Software defect prediction via optimal trained convolutional neural network," *Adv. Eng. Softw.*, vol. 169, Jul. 2022, Art. no. 103138, doi: 10.1016/j.advengsoft.2022.103138.

[31] N. A. A. Khleel and K. Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," *Cluster Comput.*, vol. 27, no. 3, pp. 3615–3638, Jun. 2024, doi: 10.1007/s10586-023-04170-z.

[32] A. Daza, "Software defect prediction based on a multiclassifier with hyperparameters: Future work," *Results Eng.*, vol. 25, Mar. 2025, Art. no. 104123, doi: 10.1016/j.rineng.2025.104123.

[33] Z. Liu, T. Su, M. A. Zakharov, G. Wei, and S. Lee, "Software defect prediction based on residual/shuffle network optimized by upgraded fish migration optimization algorithm," *Sci. Rep.*, vol. 15, no. 1, pp. 1–15, Feb. 2025, doi: 10.1038/s41598-025-91784-5.

[34] G. Xu, Z. Zhu, X. Guo, and W. Wang, "A joint learning framework for bridging defect prediction and interpretation," 2025, *arXiv:2502.16429*.

[35] M. A. Awal and C. K. Roy, "EvaluateXAI: A framework to evaluate the reliability and consistency of rule-based XAI techniques for software analytics tasks," *J. Syst. Softw.*, vol. 217, Nov. 2024, Art. no. 112159, doi: 10.1016/j.jss.2024.112159.

[36] M. Nevendra and P. Singh, "TRGNet: A deep transfer learning approach for software defect prediction," *Expert Syst. Appl.*, vol. 282, Jul. 2025, Art. no. 127799, doi: 10.1016/j.eswa.2025.127799.

[37] R. Malhotra, A. Rajpal, and D. Rathore, "Software defect," *IEEE Dataport*, Feb. 2018, doi: 10.21227/H2K078. [Online]. Available: https://ieee-dataport.org/documents/software-defect

[38] X. Xu, H. Gu, Y. Wang, J. Wang, and P. Qin, "Autoencoder based feature selection method for classification of anticancer drug response," *Frontiers Genet.*, vol. 10, p. 233, Mar. 2019, doi: 10.3389/fgene.2019.00233.

[39] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: A survey," *Artif. Intell. Rev.*, vol. 57, no. 2, p. 28, Feb. 2024, doi: 10.1007/s10462-023-10662-6.

[40] I. M. Nasir, M. A. Khan, M. Yasmin, J. H. Shah, M. Gabryel, R. Scherer, and R. Damaševičius, "Pearson correlation-based feature selection for document classification using balanced training," *Sensors*, vol. 20, no. 23, p. 6793, Nov. 2020, doi: 10.3390/s20236793.

[41] P. N. Srinivasu, M. F. Ijaz, and M. Woźniak, "XAI-driven model for crop recommender system for use in precision agriculture," *Comput. Intell.*, vol. 40, no. 1, p. 12629, Feb. 2024, doi: 10.1111/coin.12629.

[42] S.-W. Chung, S.-S. Hong, and B.-K. Kim, "Hyperparameter tuning technique to improve the accuracy of bridge damage identification model," *Buildings*, vol. 14, no. 10, p. 3146, Oct. 2024, doi: 10.3390/buildings14103146.

[43] O. G. Ajayi and J. Ashi, "Effect of varying training epochs of a faster region-based convolutional neural network on the accuracy of an automatic weed classification scheme," *Smart Agricult. Technol.*, vol. 3, Feb. 2023, Art. no. 100128, doi: 10.1016/j.atech.2022.100128.

[44] S. Morasca and L. Lavazza, "On the assessment of software defect prediction models via ROC curves," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3977–4019, Sep. 2020, doi: 10.1007/s10664-020-09861-4.

[45] M. Fetaji and B. Fetaji, "Assessing data processing using Wilcoxon signed-ranks test for polyclinic management system—Case study," in *Proc. 7th Medit. Conf. Embedded Comput. (MECO)*, Budva, Montenegro, Jun. 2018, pp. 1–6, doi: 10.1109/MECO.2018.8405979.

[46] S. Kaliraj, V. G. P. Sahasranth, and V. Sivakumar, "A holistic approach to software fault prediction with dynamic classification," *Automated Softw. Eng.*, vol. 31, no. 2, p. 70, Nov. 2024, doi: 10.1007/s10515-024-00467-4.

[47] Q. Zhang, J. Zhang, T. Feng, J. Xue, X. Zhu, N. Zhu, and Z. Li, "Software defect prediction using deep Q-learning network-based feature extraction," *IET Softw.*, vol. 2024, no. 1, Jan. 2024, Art. no. 3946655, doi: 10.1049/2024/3946655.

[48] I. Batool and T. A. Khan, "Software fault prediction using deep learning techniques," *Softw. Quality J.*, vol. 31, no. 4, pp. 1241–1280, Dec. 2023, doi: 10.1007/s11219-023-09642-4.

[49] A. Abdu, Z. Zhai, R. Algabri, H. A. Abdo, K. Hamad, and M. A. Al-antari, "Deep learning-based software defect prediction via semantic key features of source code—Systematic survey," *Mathematics*, vol. 10, no. 17, p. 3120, Aug. 2022, doi: 10.3390/math10173120.

[50] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Expert Syst. Appl.*, vol. 147, Jun. 2020, Art. no. 113156, doi: 10.1016/j.eswa.2019.113156.

[51] Y. Qiu, Y. Liu, A. Liu, J. Zhu, and J. Xu, "Automatic feature exploration and an application in defect prediction," *IEEE Access*, vol. 7, pp. 112097–112112, 2019, doi: 10.1109/ACCESS.2019.2934530.

[52] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Comput.*, vol. 22, no. 1, pp. 77–88, 2019, doi: 10.1007/s10586-018-1730-1.

PARVATHANENI NAGA SRINIVASU received the B.Tech. degree from JNTU Kakinada, the M.Tech. degree from GITAM University, and the Ph.D. degree in biomedical imaging from GITAM University. He is an Associate Professor with the Amrita School of Computing, Amrita Vishwa Vidyapeetham, Amaravati, India. He holds a post-doctoral fellowship from the Federal University of Ceará, Brazil. He is also a Research Fellow with INTI International University, Malaysia. His expertise spans biomedical imaging, soft computing, explainable AI, and healthcare informatics, with several publications in top-tier journals. He has recognized as a Stanford Top 2% Scientist, in 2024. He serves as an editorial board member, an associate editor, and a reviewer for over 100 Scopus- and Web of Science-indexed journals, actively contributing to research and academia.

M. SAILAJA received the Ph.D. degree from ANU, Guntur. She is an accomplished Academician and a Researcher in computer science and engineering, with over 16 years of teaching experience. She is passionate about emerging technologies, with expertise in soft computing, machine learning, deep learning, and engineering optimization. She has authored 39 research publications, that are indexed in reputed databases, and holds two patents. She has applied for three major funded research projects and was honored with the Best Women Researcher award by Elsevier SSRN, in 2022. A dedicated educator, she has designed five academic courses and completed 15 professional certifications. Her dynamic contributions extend beyond teaching, encompassing academic leadership, curriculum innovation, and student mentorship.

SUJATHA CANAVOY NARAHARI (Senior Member, IEEE) received the B.Tech. degree in electronics and communication engineering, the M.Tech. degree in electronic instrumentation and communication systems, and the Ph.D. degree in image processing from S. V. University, Tirupati, India, in 2001, 2005, and 2018, respectively. She started her career in 2001, as a Lecturer with the Annamacharya Institute of Technology and Sciences, Kadapa. She is currently with the Department of ECE, Sreenidhi Institute of Science and Technology, Hyderabad, Telangana, as a Professor. She has presented 19 papers in international and national conferences and published several articles in international journals. She has guided several UG and PG projects, encouraged many of her students to publish articles in reputed journals. She has authored four books and published eight patents, out of which two are granted. Her current research interests include image and video processing, speech and signal processing, machine learning, deep learning, quantum computing, and antenna design. She is a an IETE-Fellow and a member of IAENG. She is a reviewer of various international journals and conferences.

PAOLO BARSOCCHI (Member, IEEE) received the M.Sc. and Ph.D. degrees in information engineering from the University of Pisa, Italy, in 2003 and 2007, respectively. He is currently a Researcher with the Information Science and Technologies Institute, National Research Council (CNR), Pisa, Italy. He has co-authored over 150 articles published in international journals and conference proceedings. His research interests primarily include the IoT, cyber-physical systems, indoor localization, and radio signal processing. He is also an active member of several program committees and editorial boards of international journals. He has served as the program chair and the co-chair for several conferences.

AKASH KUMAR BHOI received the B.Tech., M.Tech., and Ph.D. degrees. He is the Founder of eSupport for Research, an MSME UDYAM-registered enterprise, and manages an academic YouTube channel dedicated to research awareness. He was a Research Associate with the Wireless Networks (WN) Research Laboratory, ISTI-CNR, Pisa, Italy, for three years, and was conferred the honorary title of an Adjunct Fellow with the Institute for Sustainable Industries and Liveable Cities (ISILC), Victoria University, Melbourne, Australia. At Sikkim Manipal University, he coordinated the Ph.D. course on research and publication ethics (RPE) and served as an Assistant Professor (Selection Grade) with SMIT, for nearly ten years. He has authored more than 180 research publications, 30 Scopus-indexed books, and holds an H-index of 39 with over 5700 citations. He is recognized in the World's Top 2% Scientists list for single-year impact (2022 and 2023), compiled by Stanford University and published by Elsevier BV. He is actively engaged as an editor for books with Springer Nature, Elsevier, Routledge, and CRC Press. He serves as a guest editor for reputed journals, including Springer Nature, Wiley, and Inderscience.