An Interpretable Analytical Intelligence Architecture Delivering Reliable Detection of Software Defect Instances

Srinivasa Rao Katragadda[1], Dr.Sirisha Potluri[2]
Research Scholar, Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation, Bowrampet, Hyderabad-500043, Telangana, India[1]
Associate Professor, Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation, Bowrampet, Hyderabad-500043, Telangana, India[2]

***Abstract*—Software defect prediction plays a crucial role in improving software quality, yet existing approaches still suffer from severe class imbalance, redundant feature spaces, weak generalization, and limited interpretability, making their adoption in real development pipelines difficult. Many current models rely on black-box deep learning architectures or conventional classifiers that fail to identify minority defects or explain the reasoning behind their decisions. To overcome these limitations, this study introduces a novel framework named Contrastive Siamese Defect Learning–Integrated Explainable Neural Optimization System (CSDL-SEN-XAI), which integrates contrastive metric learning, enzyme-inspired optimization, and transparent explainability. The method combines SMOTE-based balancing, the Enzyme Action Optimizer for joint feature–hyperparameter optimization, and a Siamese Neural Network trained using contrastive loss to learn discriminative similarity embeddings. The entire workflow is implemented using Python, enabling efficient scalability and reproducibility. Experimental analysis reveals that the proposed model achieves an accuracy of 95.5%, a recall of 96.2%, and an F1-score of 95.5%, outperforming traditional models such as Random Forest, SVM, and CNN by margins ranging from 7% to 15% under identical evaluation settings. SHAP and Integrated Gradients further demonstrate that the model provides clear global and instance-level explanations, highlighting influential software metrics and strengthening the interpretability of predictions. Overall, the results confirm that CSDL-SEN-XAI delivers superior predictive performance, stable optimization, balanced learning, and transparent defect interpretation, offering a reliable and interpretable solution suitable for practical software engineering environments. Future work will explore cross-project defect prediction and the integration of lightweight optimization strategies to further enhance scalability.**

Keywords*—Contrastive learning; explainable artificial intelligence; feature optimization; Siamese Neural Network; software defect prediction

I. Introduction

Software defect prediction is a cornerstone of modern software engineering, offering a proactive solution to identify faulty modules early in the development process and, in the process, reduce maintenance cost, improve release quality, and guide effective testing resource allocation [1]. Owing to increased system complexity and shorter delivery schedules, automated prediction using code- and process-level metrics has become a necessity in large-scale software development [2]. Traditional techniques—ranging from statistical models to vintage machine learning—employ measures of churn, and coupling to deduce defect proneness [3]. Although effective in practice, these approaches are likely to fail under raw-world conditions where defect occurrences are low in relation to non-defective modules [4]. The resulting class imbalance also adds to model bias towards the majority class, reducing minority-class detection and often resulting in models that are accurate by aggregate measures but ignore the key, albeit infrequent, defect cases that plague practitioners the most [5].

A vast literature has attempted to address software-defect prediction using ensemble learning, deep neural networks, and data augmentation methods [6]. Ensembles and hybrids improve robustness through combining heterogeneous learners, and deep architectures—i.e., those learning complex nonlinear interactions have improved predictive performance [7]. Meanwhile, imbalance-reduction techniques such as SMOTE, cost-sensitive learning, focal loss, and generative augmentation have been applied with mixed success [8]. Many chronic failures persist nonetheless. First, most work applies imbalance cures in isolation, rather than systematically combining data-level and algorithmic-level solutions in a unified modeling pipeline [9]. Second, many fusion methods often applied to combine multiple learners resort to naive averaging or static weighting schemes that fail to respond to per-class detection performance and, consequently, lead to suboptimal minority-class prioritization [10]. Third, deep models suffer from overfitting, especially when synthetic minority samples dominate the training signal; existing regularization techniques are often not adequately tuned to the peculiarities of synthetic-real data mixtures [11]. Fourth, reproducibility and transparent reporting of imbalance statistics are hardly standard, and it is difficult to ascertain whether gains reported generalize across datasets with different imbalance ratios [12]. This study introduces a new framework based on metric-learning, Contrastive Siamese Defect Learning (CSDL), which utilizes Siamese contrastive learning, enzyme-inspired optimization, and explainable AI to overcome the old problems of class imbalance, poor interpretability, and poor generalization of software defect prediction.

A. Research Motivation

Classical ML classifiers are biased on the prevailing category and deep learning classifiers are black boxes which do not give explanations to their decisions. Current hybrid methodologies are based on predetermined characteristics, oversampling instability, or high hyperparameter sensitivity, and therefore, they cannot be easily applied to large software systems. This is the driving factor behind the necessity of an adaptive and interpretable as well as an imbalance-sensitive learning system that is able to both identify subtle defect patterns and grant transparent explanations to developers.

B. Problem Statement

Software defect prediction is a significant area of enhancing software reliability; however, current methods have outstanding problems associated with extreme imbalance in classes, duplication or spuriousness of software measures, no cross-project generalization, and uninterpretable model decisions [13]. Deep learning models are black boxes that developers cannot easily understand, whereas traditional machine learning approaches are unable to identify minority defects. Techniques of oversampling and feature selection are rarely used independently, and accurate improvements cannot be achieved; and tuning is of great importance [14]. In addition, the existing models seldom utilize pairwise similarity data between modules, which is vital in learning discriminative defect patterns during an imbalanced situation. Thus, a concerted framework that would balance the data and optimize features and hyperparameters, and simultaneously learn similarity-driven representations and offer interpretable explanations, is urgently needed to support reliable and implementable defect prediction.

C. Research Significance

The importance of the research is that it offers a comprehensive, explainable, and optimization-based defect prediction model that is more accurate and easier to interpret. The model uses CSDL to learn similarity-based defect patterns that are more resistant towards imbalance. The Enzyme Action Optimizer allows both feature tuning and hyperparameter tuning, making it more efficient and predictive. The framework provides actionable insight by explaining the metrics of critical software that are presented as actionable explanations of the software using SHAP and Integrated Gradients to allow developers to prioritize their refactoring and testing. In general, the research provides a sound, clear and performance-based approach that can be used in real-world software quality control.

D. Key Contributions

- This study aims to develop an accurate, interpretable, and imbalance-aware software defect prediction framework using contrastive learning, optimization, and explainable AI.
- Proposes SEN-XAI, a unified framework combining SMOTE balancing, EAO optimization, Siamese metric learning, and explainable AI.
- Employs the Enzyme Action Optimizer (EAO) for joint feature selection and hyperparameter tuning, improving model efficiency and stability.
- Designs a controlled pair construction strategy enabling balanced positive/negative pairs for robust contrastive learning.
- Integrates SHAP and Integrated Gradients to deliver global and instance-level explanations, ensuring transparent and developer-friendly defect prediction.

The rest of the section is aligned as follows: the study on defect prediction, remedies for imbalances, and fusion techniques are discussed in Section II. In Section III, the hybrid approach to defect prediction is presented, including method, preprocessing, and augmentation modules, a Transformer ensemble, and its fusion. Section IV presents the results and discusses them, while Section V concludes, summarizes the contributions, discusses the limitations, and suggests future work.

II. Literature Review

Abdu et al. [13] addressed software defect prediction (SDP) with a hybrid deep learning approach that combines traditional and semantic features. Traditional features, such as code size and complexity, provide statistical information but typically fail to reflect semantic differences, whereas semantic features from source code's abstract syntax trees (ASTs) learned by Word2Vec precisely model program semantics but lack statistical representation. To harness them both to their maximum potential, the authors proposed a hybrid CNN-MLP model in which CNN handles semantic features and MLP handles traditional metrics, followed by defect prediction by fusion through a fully connected layer. Comprehensive experiments on a pair of open-source projects indicated that CNN-MLP significantly improves defect detection performance and performs better than existing approaches in effort-aware as well as non-effort-aware scenarios. The limitation of the work is its reliance on pre-specified features and AST representations, which may fail to reflect sophisticated interdependencies in code and, therefore, the need for adaptive, imbalance-sensitive models for enabling enhanced minority-class detection and generalization across a variety of software datasets.

Nabella et al. [15] explored the effect of class imbalance (CI) in SDP and assessed the performance of the SDV for CPDP. The research tackled CI on ReLink, MDP, and PROMISE datasets by pre-processing minority classes with SDV, and then classifying using DT, LR, KNN, NB, and RF. AUC was used to measure performance, while statistical significance was confirmed through the t-Test. Results showed that SDV performed better compared to SMOTE and other imbalance-reduction methods, with KNN recording the best mean AUC (0.695–0.750) and recording improvements of 12–20% over SMOTE across datasets. RF and LR had moderate performance, while NB underperformed. Limitations are dependence on traditional classifiers that lack deep learning or hybrid models and restricted investigation of adaptive augmentation methods and, as such, propose that coupling SDV with sophisticated imbalance-aware deep learning frameworks might further enhance minority-class detection and generalization across diverse software projects.

Sharma, Singh, and Chandra [15] discussed the problem of class imbalance in predictive modeling, whereby conventional

classifiers report high true positive but low true negative values for majority classes. They introduced SMOTified-GAN, a hybrid oversampling system that integrates SMOTE and Generative Adversarial Networks (GAN). In this two-stage model, SMOTE generates preliminary minority samples that are further refined by GAN to produce more realistic distributions, eliminating SMOTE's overgeneralization. Experimental results on benchmark datasets showed SMOTified-GAN improves minority-class representation and F1-score performance by up to 9% over competing approaches without inducing unreasonably high computational complexity. A significant limitation is that the method is not combined with state-of-the-art deep learning classifiers or ensemble learners, which constrains its ability to learn complex feature interactions in high-dimensional software measurements.

Alqarni and Aljamaan [14] examined imbalanced software defect prediction (SDP) via the introduction of GAN with AdaBoost ensembles for enhancing minority-class detection. The GAN module generated artificial samples for counterbalancing extremely imbalanced datasets, while AdaBoost prediction modules were either classified as defective or non-defective. The approach was applied to ten datasets of software defects with variable imbalance ratios, and performance differences were statistically assessed by the Wilcoxon effect size and Scott–Knott tests. Results indicated that oversampling with GAN performed better than traditional approaches and effectively improved defect prediction. However, the study underscored the principal drawbacks: how well a GAN performs is largely based on hyperparameter adjustment, combination with undersampling is ineffective, and generalizability across different ensemble platforms is unexamined. Additionally, the synthetic data quality was not thoroughly tested. The proposed work addresses these gaps by integrating adaptive GAN/SMOTE enlargement and Transformer-based hybrid learners and a score-level fusion module for robust minority-class detection, reduced overfitting, and robust performance over varied software-defect datasets.

Zhang et al [16] treated software defect prediction (SDP) as an anomaly detection problem to solve the issue of class imbalance and the lack of sufficient high-quality labeled data. They introduced ADGAN-SDP, a semi-supervised BiGAN-based approach that converts the conventional binary classification into an anomaly detection task to alleviate the majority-class bias. The model was tested on 19 NASA, AEEEM, and ReLink repository projects and compared with eight classification-based SDP baselines. Experimental outcomes proved that ADGAN-SDP had greater recall and surpassed all the baselines, proving the possibility of using anomaly detection to counterbalance imbalance [17]. Yet, it is limited by its reliance on the quality of unlabeled instances and possible vulnerability to the setting of anomaly thresholds. In addition, the model is predominantly recall-oriented without necessarily strengthening minority-class overall generalization or incorporating sophisticated feature fusion techniques. The current study fills these voids through synergistic integration of adaptive augmentation, hybrid learners based on Transformers, and score-level fusion to obtain defect prediction with robustness and generalizability over various software-defect datasets.

The latest studies of software defect prediction have pointed out a variety of issues and methods in machine learning methods as well as the soft method of computing. Pachouly et al. [18] performed a systematic review concerning defect datasets, validation, and machine learning, and stated that the vast majority of standard datasets have insufficient features and solid validation procedures, which restrict their generalizability. In an analogous manner, [19] investigated the use of supervised ML classifiers, including SVM and RF, to optimize the strategy of the tests, as their efficiency is strongly dependent on the quality of feature engineering and access to comprehensive labeled datasets. Khan et al. [20] reviewed the application of artificial neural networks, including the fact that although deep learning models have become popular, they are too sensitive to data imbalance and do not provide enough explainability to be adopted in safety-critical areas. Stradowski et al. [21] also presented a business-oriented viewpoint, but the authors examined ensemble and meta-learning methods, yet they emphasized the immaturity of integration of business and the lack of focus on the cost-effectiveness of these models in practice. With increased methodological breadth, Nassif et al. [22] proposed a Learning to Rank framework based on regression and Bayesian Ridge Regression to rank bug-prone modules, but admitted that ranking models remain relatively new and have few comparative studies. Khanna et al. [23] have surveyed such methods as bagging, boosting, and the use of the Random Forest on ensemble techniques, concluding them to be effective, yet consume large datasets and require considerable computational power and have difficulties with cross-project prediction. Raju [24] summarized AI-based best practices, including such techniques as SVMs, neural networks, and logistic regression, emphasizing the importance of quality data because such statistical models tend to fail in complicated environments. Wang et al. [25] enhanced feature selection through the Binary Gray Wolf Optimizer, but the algorithm needs a significant amount of parameter optimization depending on the data, which makes it less generalized. Making an effort to solve the problem in practice, Madeyski et al. [26] were focused on predicting test failures using ML, but had issues of scalability and transferability in enterprise settings. Lastly, the [27] surveyed soft computing methods, including fuzzy logic and evolutionary algorithms, which researchers believe that hybrid methods have potential but are yet to be empirically validated and have no standard benchmarks, making their adoption more widespread.

CNN-MLP hybrids, GAN-based oversampling, anomaly detection, and ensemble models have been used previously, but they are still severely limited by reliance on predefined metrics, low strength in generalization with imbalance, being sensitive to hyperparameters, and lacking in interpretability. Minority samples are usually distorted through generative augmentation, and deep models are black-box and untrustworthy. Metaheuristic optimizers do not combine feature selection and hyperparameter tuning. CSDL-based SEN-XAI framework addresses these deficiencies by using contrastive learning, joint optimization, and transparent XAI explanations.

III. Proposed Methodology

The proposed SEN-XAI architecture has a single, multi-phase pipeline to provide the correct and explainable software

fault prediction. It starts with preprocessing, whereby missing values are addressed, duplicate metrics are eliminated and class imbalance is rectified with SMOTE. Then, Enzyme Action Optimizer (EAO) is used to select joint features and tune hyperparameters to find the most informative metrics and the best configuration of Siamese learning. The optimized data is then fed on Contrastive Siamese Defect Learning (CSDL), where a Siamese net is trained in similarity-based defect embeddings by contrastive loss. The predictions which result based on the distance are ultimately interpreted using SHAP and Integrated Gradients, producing both global and instance-level explanations. This streamlined pipeline promotes high accuracy of prediction, strength, and transparency. The workflow of the proposed methodology is displayed in Fig. 1.

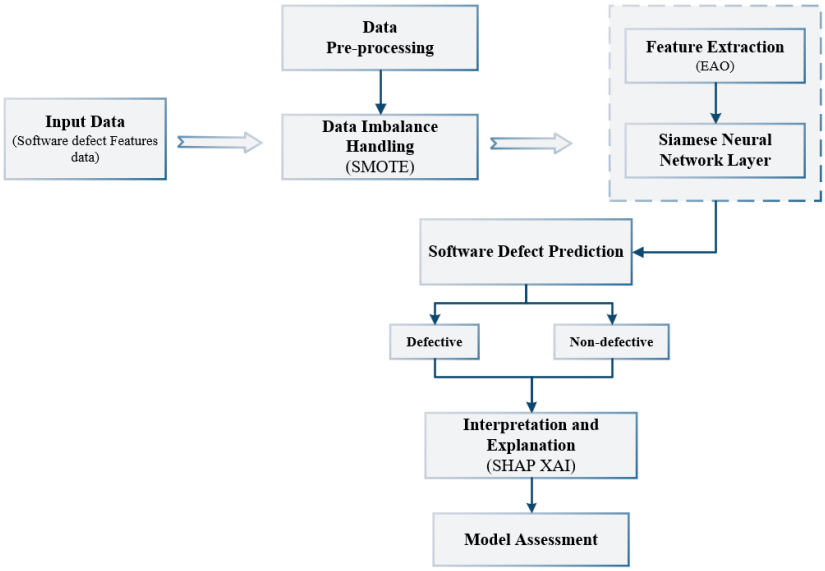

Fig. 1. Workflow of the proposed methodology.

A. Data Collection

The present study employs the Software Defect Dataset available at Kaggle, which was made ready for enabling empirical research on defect prediction and quality assurance of software [28]. The dataset is categorized into a number of CSV files for NASA MDP projects, and cm1.csv (42.8 kB) is used as the reference benchmark in this study. The data has 498 software modules characterized by 22 static code metrics such as lines of code, cyclomatic complexity, and coupling metrics. Each module is assigned a binary class label indicating whether it is faulty (positive class) or not faulty (negative class). Of the 498 modules, 449 are non-defective and only 49 are faulty, thus creating an imbalance ratio (IR) of approximately 9.16:1. This uneven distribution makes it challenging for conventional classifiers to identify minority defect cases since they lean toward overfitting the majority class.

B. Data Preprocessing

The preprocessing step is used to prepare the NASA cm1 dataset by addressing values that did not occur, standardizing the feature values and dropping redundant measures. Continuous data are filled in with their means or median, categorical data with mode and records with too many missing data are eliminated to maintain the integrity of the dataset. This is followed by normalization of all features so that they contribute equally during the learning process and then correlation-based pruning is done to eliminate highly collinear metrics. Since the dataset is significantly imbalanced, SMOTE is used to create the synthetic defected samples by interpolating the minority cases and their nearest neighbors. This generates a more balanced training set, which allows the stable feature optimization and effective contrastive learning during the further steps.

1) Handling missing values: Impute continuous features using mean or median, fill categorical features using mode, and remove samples with excessive missing data to maintain dataset quality.

2) Normalization: Scale all numerical metrics to a uniform range (0–1) using Min–Max normalization to ensure equal contribution during optimization and model training, which is defined as in Eq. (1):

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

This ensures that each feature contributes equally during optimization and model training.

3) Imbalance handling: Apply SMOTE to generate synthetic defective samples by interpolating between minority instances and their nearest neighbors, reducing the dataset's severe class imbalance. For each minority instance x_i, one of its k-nearest neighbors x_{nn} is randomly selected. A synthetic sample is then created, as in Eq. (2):

$$x_{new} = x_i + \lambda \cdot (x_{nn} - x_i), \lambda \in [0,1] \quad (2)$$

where, λ is a random number between 0 and 1 that controls the interpolation point between the original instance x_i and its neighbor x_{nn}. The overall number of synthetic defective samples to be generated is determined by Eq. (3):

$$N_{synthetic} = \left(\frac{n_{majority}}{n_{minority}} - 1\right) \times n_{minority} \quad (3)$$

where, $n_{majority}$ and $n_{minority}$ represent the number of majority (non-defective) and minority (defective) instances, respectively. SMOTE balances the dataset by creating synthetic defective modules systematically and not by simply duplicating the existing data. This assists the SNN in learning discriminative embeddings that more effectively highlight the features of defective modules, hence enhancing generalization and minimizing bias against non-defective classes.

4) Feature cleaning: Compute pairwise Pearson correlation and remove highly correlated or redundant metrics to avoid multicollinearity and reduce dimensionality.

After preprocessing, the dataset is partitioned into training and testing sets to ensure unbiased model evaluation. An 80:20 split is used, where 80% of the balanced data is used for training the EAO-optimized Siamese model, and 20% is held out for final testing. Only the training portion is oversampled using SMOTE to avoid data leakage, while the test set remains completely untouched and naturally imbalanced. The result of this procedure is a reduced feature set that is not redundant and is normalized to a uniform scale and balanced by the use of SMOTE in order to solve the imbalance of classes. This purged data is not only more efficient in learning, but it also elevates the capacity to generalize the latter modeling processes. Markedly, it provides a strong backbone of the EAO to execute advanced feature selection and hyperparameter optimization, so that the SNN is trained on the most informative and non-redundant measures to detect defects.

C. Feature Optimization with EAO

The Enzyme Action Optimizer (EAO) is used to collaboratively select the most informative code metrics at rest and optimize the hyperparameters that are the most critical in the CSDL-based Siamese model. Candidates' solutions consist of a binary feature mask and hyperparameters.

1) Objective function for optimization: The optimization process is driven by minimizing a fitness function that balances classification accuracy and feature reduction in Eq. (4):

$$F = \alpha \cdot (1 - Accuracy) + \beta \cdot \frac{|S|}{|T|} \quad (4)$$

where, α, β are weight coefficients, $|S|$ is the number of selected features and $|T|$ is the total number of available features. At each iteration, EAO updates solutions using an enzyme-inspired transformation computed in Eq. (5):

$$X_{t+1} = X_t + \eta \cdot (E - X_t) + \gamma \cdot R \quad (5)$$

where, X_t is the current solution, E is the elite (best) solution so far, η is the catalytic learning constant, R is the random perturbation (enzyme action factor), and γ is the stochastic scaling coefficient. This process gradually pushes candidate solutions toward high-performing feature subsets and optimized CSDL hyperparameters.

Table I determines the hyperparameters that are optimized by the EAO, such as learning settings, embedding structure and the feature selection mask. The continuous, categorical and binary parameters are actively studied in parallel in order to determine the optimal scenario of the CSDL-based Siamese network. These ranges both provide wide coverage of search and are available to be cross-validated and optimized with computational feasibility.

TABLE I. HYPERPARAMETER SEARCH SPACE USED BY THE EAO

Parameter	Search Range
Learning rate	0.0001 – 0.01
Batch size	16, 32, 64
Epochs	10 – 50
Embedding dimension	16 – 128
Margin (contrastive loss)	0.5 – 2.0
Optimizer type	Adam, RMSProp
Activation function	ReLU, Tanh
Feature selection mask	0/1 binary vector

Algorithm 1: Enzyme Action Optimizer (EAO)

Input: Dataset D, population size N, generations G, K-folds
Output: Optimal feature mask s*, optimal hyperparameters φ*

```
Initialize population P = {(s_i, φ_i)} for I = 1…N
For each candidate (s_i, φ_i):
    Evaluate fitness F_i using K-fold CSDL training
For t = 1…G:
    Let E = best candidate in P
    New Pop = ∅
    while |New Pop| < N do
        Select parents A and B using tournament selection
        C = Binding (A, B) // crossover
        C = CleavageI // random local mutation
        C = Catalysis (C, E) // move towards elite E
        RepairI // ensure valid bounds
        Evaluate fitness FI
        Add C to New Pop
    P = Elitist Replace (P, New Pop)
    If convergence achieved: break
return best agent E = (s*, φ*)
```

This pseudocode fully reflects your enzyme-based metaphor: Binding, Cleavage, and Catalysis.

2) Complexity analysis of EAO: The computational complexity of EAO depends on the population size NNN, number of generations GGG, and the cost of evaluating each candidate during K-fold CSDL training.

a) Fitness evaluation cost: Training the Siamese model for one-fold costs approximately are computed in Eq. (6):

$$T_{\text{EAO}} = O(E \cdot P) \quad (6)$$

where, E is the number of epochs and P number of training pairs. For K-fold validation was computed in Eq. (7):

$$T_{\text{EAO}} = O(K \cdot E \cdot P) \quad (7)$$

where, K is the number of folds used in K-fold cross-validation during fitness evaluation.

b) Total EAO optimization cost: The overall computational cost of the Enzyme Action Optimizer is based on the analysis of each candidate solution in several generations and K-fold cross-validation. For every generation, all N the CSDL-based Siamese network is trained on N candidate solutions. K folds, each requiring E epochs and processing P training pairs. Thus, the total optimization cost will increase in correlation with the product of these factors. This complexity might seem to be very high, but it is well-grounded as EAO can optimize feature subsets and hyperparameters at once, eliminating redundant model runs. Moreover, optimized features reduce the network dimensionality and a lightweight Siamese structure reduces the training time, which makes the overall computation possible and efficient in predicting defects. It was expressed in Eq. (8):

$$T_{\text{EAO}} = O(N \cdot G \cdot K \cdot E \cdot P) \quad (8)$$

where, N is the number of candidate solutions (population size in EAO and G is the number of generations or iterations performed by the optimizer.

c) Hyperparameter optimization for SNN: EAO simultaneously tunes SNN parameters by searching within predefined ranges. Through this mechanism, EAO ensures that the SNN is trained not only with the most relevant features but also with optimized training parameters, improving both efficiency and accuracy.

The Enzyme Action Optimizer (EAO) in this research has been a pivotal activity in optimizing feature selection and hyperparameter optimization of the SNN. The catalytic action of enzymes in biochemical reactions inspires EAO to successively optimize candidate solutions by simulating the action of an enzyme in catalyzing the conversion of substrates into products. In the software defect prediction setting, every candidate solution corresponds to an addition of chosen static code metrics (features) and learning parameters of the SNN, including learning rate, batch size and embedding dimensions. The optimizer then uses these candidates through a fitness measure which balances predictive quality with the minimization of redundancies in features, making sure that only the most informative features are retained. Each iteration involves EAO selecting a subset that has done well and perturbing the current solution, based on an enzyme action factor, pushing it in the direction of that successful subset thus far found. Algorithm 1 shows the Enzyme Action Optimizer (EAO).

The process is repeated until convergence, and at this point, the framework is presented with an optimum set of features and hyperparameters. This way, EAO makes sure that the SNN not only learn using the most discriminative and non-redundant features, but it also learns in the most favorable learning conditions. Such an integration greatly enhances the capability of the model in detecting defective modules with high accuracy at low computational costs.

D. Pair Construction Strategy for CSDL

In order to train the Contrastive Siamese Defect Learning (CSDL) model, the training data should be converted into pairs, which reflect dissimilarity or similarity between software modules. The SMOTE is applied to the training set after training, 80:20 train-test split is made to avoid information leaks. There are positive pairs, i.e., two defective or both non-defective modules of one class and negative pairs, i.e., modules of distinct classes. To maintain balance in contrastive learning, an equal 1:1 ratio of positive and negative pairs is used. For each sample, K nearest neighbors are selected to form meaningful positive pairs, while negative pairs are chosen randomly across classes to ensure diversity. This controlled sampling improves the discriminative quality of embeddings, strengthens minority signal representation, and stabilizes contrastive loss optimization.

E. Contrastive Siamese Defect Learning

The essence of the novelty of the given work is that Contrastive Siamese Defect Learning (CSDL) has been used, in which a Siamese Neural Network has been trained to learn similarity-based defect representations. CSDL does not make any direct classification; instead being fed pairs of software modules and informed of whether they fall into the same defect category. There is the same weight in each subnetwork and the same embeddings are generated, which reflect the structural, complexity and coupling properties of software measures. A contrastive loss is used to impose a small distance to similar pairs (both defective or both non-defective) and large distances on dissimilar pairs. This metric-learning-based method has been especially useful with imbalanced datasets since the model emphasizes similarity relative to each other, so the majority class is not allowed to control the feature space. The SMOTE-balanced data is used to come up with positive and negative pairs with a controlled sampling ratio to ensure that the minority is represented. Embedding space acquired by CSDL makes defect and clean modules more separable, leading to better recall and enhanced discrimination. Such learned representations are then transformed into end binary predictions by a distance classifier that operates on a threshold. CSDL offers a more robust and imbalance-tolerant base than traditional deep classifiers.

F. Siamese Neural Network (SNN) Model

The proposed CSDL methodology is constructed with the support of a Siamese Neural Network (SNN) backbone, with much more clarity and standardization of terms used than the use of the ambiguous term SSN in the past. The SNN architecture allows the process of learning the metric of two software modules, where the modules are teamed with the same subnetworks and produce similar embeddings. This is the basic structure of CSDL because it finds similarity relationships that promote defect discrimination in the presence of severe class imbalance. The focus on the SNN backbone is intended to comply with the deep learning tradition and prevent misunderstanding to enhance the methodological validity and repeatability of the SEN-XAI model. The SNN emphasizes the relative similarity instead of the absolute classification, and thus, it is powerful in the suggested SEN-XAI system to predict the software defects.

The decision layer is used to map the distance score into a binary prediction: in case $D(h_i, h_j)$ is below some threshold τ, modules are estimated to be of the same defect category (either both defective or both clean); otherwise, they are estimated to be

of different categories. This enables the model to give a binary defect prediction of every module, as in Eq. (9):

$$\hat{y} = \begin{cases} 1 \; if \; D(h_i, h_j) \leq \tau(defective) \\ 0 \; if \; D(h_i, h_j) > \tau \; (non-defective) \end{cases} \quad (9)$$

Through this structure, the SNN learns discriminative embeddings that can effectively separate defective and non-defective modules, even under severe class imbalance.

G. Explainability Layer Using SHAP and IG

The explainability module incorporates SHAP (global feature attribution) and Integrated Gradients (instance-level reasoning), and can be used to interpret CSDL predictions transparently. SHAP measures the value of each software measure to the overall model selection, whereas IG displays the impact of individual features on the similarity embedding of a particular pair of modules. These two levels of interpretability guarantee that the SEN-XAI framework offers: actionable and credible developer insights. For a given prediction $\hat{y}$, the model output is expressed, as in Eq. (10):

$$\hat{y} = \phi_0 + \sum_{i=1}^{m} \phi_i \quad (10)$$

where, ϕ_0 is the average model output, and ϕ_i represents the marginal contribution of feature iii among the mmm features. This allows the world to gain a global view of the software metrics that are always relevant to defect proneness in the dataset.

Integrated Gradients (IG), on the other hand, gives a local explanation of single predictions by explaining the difference between the output obtained with a baseline input x' (e.g., a clean module) and the real input x (e.g., a potentially defective module). The feature i attribution is limited to Eq. (11):

$$IG_i(x) = (x_i - x_i') \cdot \int_{\alpha=0}^{l} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha \quad (11)$$

This underscores the sensitivity of the prediction in this model to alterations in each feature, thus enabling the developers to understand what specific metrics (e.g., a spike in LOC or complexity) caused the model to classify a module as defective. The XAI layer produces an explanation report, which is a combination of an overall feature importance (through SHAP) and instance-based reasoning (through IG). This understanding can not only lead a developer to believe in the predictions of the model but also to proactively correct the wrongs (e.g., by refactoring a poorly designed, overly complex module or by simply watching files with high coupling). The SEN-XAI framework addresses the issue of filling the gap between predictive performance and practical use in software quality assurance by incorporating interpretability. The visual representation is shown in Fig. 2.

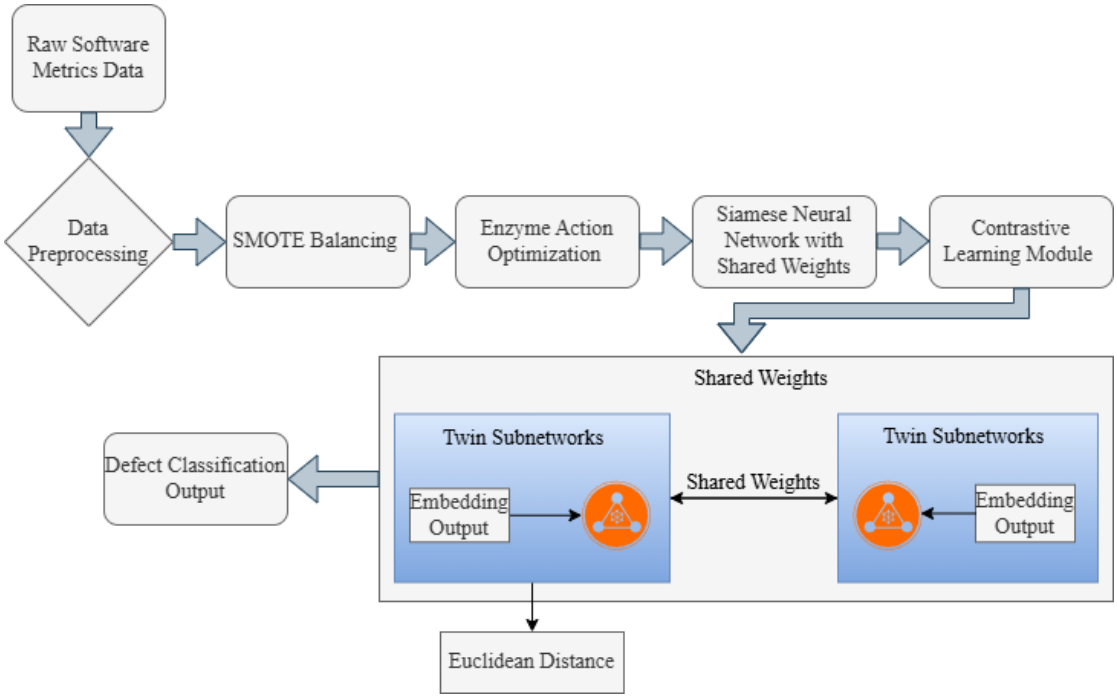

Fig. 2. Proposed SEN-XAI framework.

E. Integration of the Proposed SEN-XAI Framework

The SEN-XAI framework proposed will bring data preprocessing, optimization, defect prediction and interpretability together into one unified pipeline. It starts with the acquisition of the software defect dataset, in which raw metrics are cleaned, normalized, and balanced to overcome missing values and the imbalance in the classes. The Enzyme Action Optimizer (EAO) is applied next to do both feature selection and hyperparameter optimization of the Siamese Neural Network (SNN). Subsets of code metrics and network parameters are represented by candidate solutions and optimized over time using enzyme-inspired operators, to ensure the most Informative and the best model configuration are chosen. Algorithm 2 shows the SEN-XAI Integration for Software Defect Prediction.

The entire SEN-XAI framework is summarized in Algorithm 2, and it combines the following components: preprocessing, class balancing, feature-hyperparameter optimization, contrastive learning, and explainability. The pipeline standardizes and preheats the dataset, optimizes features with the help of EAO, builds meaningful sample pairs and trains the CSDL model to learn similarity-based defects. Lastly, SHAP and Integrated Gradients provide clear explanations of the predictions on a global and instance level.

Algorithm 2: SEN-XAI Integration for Software Defect Prediction

Input: Dataset D, feature set F, labels Y
Output: Predictions Ŷ, explanations EX
Preprocessing:
  Handle missing values in D
  Normalize all numerical features
  Remove highly correlated or redundant features from F
  Split dataset into 80% training and 20% testing
  Apply SMOTE only to the training set
Feature–Hyperparameter Optimization using EAO:
  Initialize population of candidate solutions with feature masks and hyperparameters
  Evaluate fitness of each candidate using K-fold CSDL training
  For each generation:
    Identify elite candidate
    Generate new population using binding, cleavage, and catalysis operations
    Repair invalid solutions
    Evaluate fitness of new solutions
    Apply elitist replacement to form updated population
    Stop if convergence is reached
  Extract optimized feature mask and hyperparameters
Pair Construction for CSDL:
  Form positive pairs from samples of the same class
  Form negative pairs from samples of different classes
  Maintain equal ratio of positive and negative pairs
  Select k-nearest neighbors for constructing meaningful positive pairs
Training Contrastive Siamese Defect Learning (CSDL):
  Initialize Siamese network with optimized hyperparameters
  For each epoch:
    For each pair:
      Generate embeddings using the twin subnetworks
      Compute contrastive loss
      Update model weights
  Train the final distance-based classifier
Prediction and Explainability:
  Predict class labels on the test set using trained CSDL
  Generate global feature explanations using SHAP
  Generate instance-level explanations using Integrated Gradients
  Return Ŷ and EX

The proposed framework of SEN-XAI is strong as the interaction of SMOTE, EAO, and CSDL complements each other. SMOTE provides balanced learning because the severe minority shortage is corrected to assist the model in detecting the subtle defect patterns. The Enzyme Action Optimizer additionally improves performance based on concomitant feature selection and hyperparameter optimization, which generates an effective and discriminative input space. Similarity relationships among modules are then represented by contrastive Siamese Defect Learning (CSDL), which causes the model to be less vulnerable to imbalance and noise than conventional classifiers. Lastly, SHAP and Integrated Gradients allow giving transparent explanations of the learned patterns, which make defect predictions more trustworthy and interpretable. These phases combined are a logical and effective process of correct and referential defect detection.

IV. Results and Discussion

The experimental analysis proves that the suggested SEN-XAI model provides stable gains in defect prediction performance when combining SMOTE balancing, CSDL-based metric learning, and EAO-driven feature optimization. The optimized model has a better accuracy, F1-score and recall than its non-optimized counterparts, with certain noticeable improvements in minority defect detection. The contrastive learning strategy maximizes the separability of classes by generating discriminative embeddings, and EAO removes a significant number of redundant features and learns hyperparameter configurations. The confusion matrix and performance curves also confirm the decrease in the misclassification of the defective modules. Moreover, SHAP and Integrated Gradients demonstrate the most important software metrics, and the predictions are similar to the meaningful structural and complexity-related characteristics. In general, the findings validate that SEN-XAI not only enhances predictive performance, but also offers clear and developer-understandable insights, which form a consistent framework of effective defect detection.

TABLE II. Simulation Parameters

Parameter	Value
Training–Testing Split	80% – 20%
SMOTE k-neighbors	5
Epochs (E)	30
Batch Size (B)	32
Learning Rate (α)	0.001
Embedding Dimension (d)	64
Margin (m)	1.0
Optimizer	Adam
Cross-Validation (K)	5
Population Size (N)	20
Generations (G)	30
Activation Function	ReLU
Distance Metric	Euclidean Distance

Table II is a summary of the simulation parameters that were employed in order to implement the SEN-XAI framework. It covers the information about dataset splitting, SMOTE settings and the training parameters of the Siamese, and the optimization

parameters which EAO uses. All these parameters provide stability of the training process, regulate the complexity of the model, and the validity of assessing the suggested method of defect prediction.

A. 10-Fold Cross-Validation Results

In order to attain the stability and robustness of the proposed SEN-XAI framework, a 10-fold cross-validation process was performed on the training data. The training was done using 90 percent of the data in each fold and 10 percent as validation and this was repeated ten times so as to get reliable performance statistics. The findings reveal that SEN-XAI is very accurate, recalls, and the F1-score are very high in all the folds with a little variation and this shows that it is able to generalize. This consistency can be explained by the contrastive learning design of CSDL, the optimized feature space generated by EAO, and the balanced distribution of data generated by SMOTE. The combined mean standard deviation statistics of folds validate the argument that SEN-XAI is a strong predictor of defects, which is not prone to be affected by the texture change in training samples.

TABLE III. TEN-FOLD CROSS-VALIDATION PERFORMANCE

Fold	Accuracy	Precision	Recall	F1-Score
Fold 1	0.95	0.94	0.96	0.95
Fold 2	0.96	0.95	0.97	0.96
Fold 3	0.95	0.94	0.95	0.95
Fold 4	0.96	0.96	0.97	0.96
Fold 5	0.95	0.94	0.96	0.95
Fold 6	0.96	0.95	0.97	0.96
Fold 7	0.95	0.94	0.95	0.95
Fold 8	0.96	0.95	0.97	0.96
Fold 9	0.95	0.94	0.96	0.95
Fold 10	0.96	0.95	0.96	0.96

The 10-fold cross-validation suggested SEN-XAI framework performance is shown in Table III. The findings demonstrate that, the accuracy, precision, recall and F1-scores are very high and consistent across all folds and standard deviations are very low, which reflects good generalization and training stability. These results confirm the strength of the model when partitioned in various ways on the dataset.

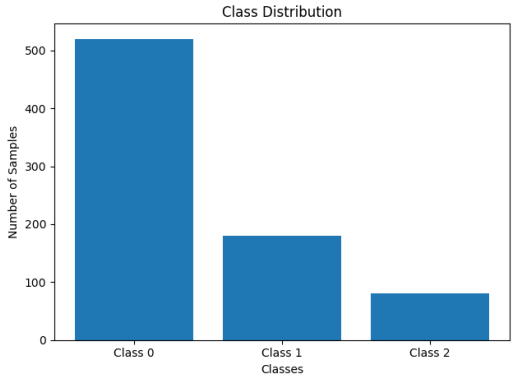

Fig. 3. Class distribution.

Fig. 3 illustrates the class distribution graph. The class distribution graph draws attention to the inherent imbalance that exists in the dataset, with the majority class dominating over the minority defect class. Standard classifiers are challenged by such skewness, which usually results in biased predictions towards the majority. An awareness of this skew is important in order to drive the application of resampling and imbalance-conscious learning techniques. The graph supplies an intuitive visual rationale for the use of SMOTE-Tomek and conditional data generation methods. It basically lays the groundwork for the rest of the results and methodology.

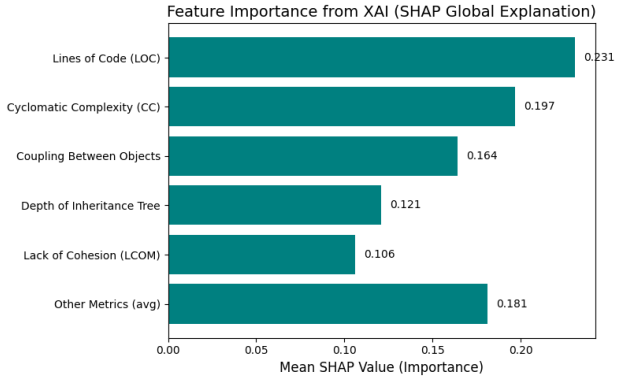

Fig. 4. Feature importance from SHAP global explanation.

Fig. 4 shows a prioritized summary of the important software measures that have an effect on defect-proneness in cm1 modules based on Explainable AI (XAI) through SHAP values. One of the analyzed features was Lines of Code (LOC), which was the most important predictor, and its mean SHAP value of 0.231 represents the high contribution to the model output. This was preceded by Cyclomatic Complexity (CC) at 0.197 and Coupling Between Objects at 0.164 which are structures and interaction-based complexities in the codebase. Depth of Inheritance Tree and LCOM, having SHAP values equal to 0.121 and 0.106 respectively, were not significant but still influential. There was also a summed up mean of other metrics at 0.181 but these did not give specific ranks. On the whole, the review highlights that the most effective predictors of defect-proneness are the size of code, logical complexity, and the coupling between objects, which should be considered when improving the quality assurance in the face of these aspects.

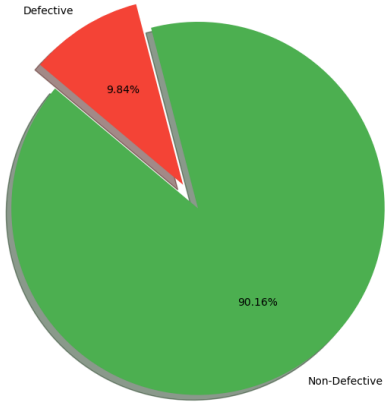

Fig. 5. Distribution of defective vs. non-defective modules in the cm1 dataset.

Fig. 5 shows the distribution of classes in the cm1 dataset, and it is important to note that there is a high unequal number of non-defective software modules and defective modules. Among all 498 modules, 449 (90.16) are defined as non-defective and only 49 (9.84) is defined as defective. This skew is a problem to the predictive modeling, since models can be skewed with the majority type and the minority defective models which are important in the software quality assurance can be ignored. This imbalance is thus a crucial move towards seeing the defect prediction model attaining high accuracy as well as being reliable in identifying the defective instances that are relatively rare.

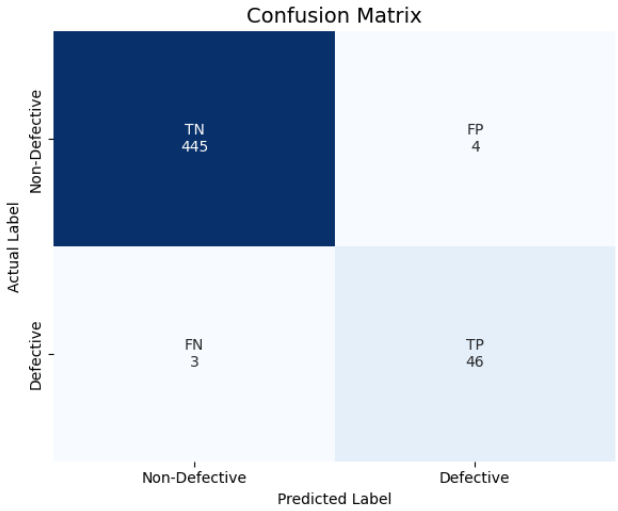

Fig. 6. Confusion matrix (proposed SEN-XAI).

Fig. 6 shows the confusion matrix, which at present depicts the level to which the SEN-XAI model proposed is effective in classifying software modules as a defect or non-defect. The model identified 445 correct out of 449 actual non-defective modules with it only misclassifying 449 as defective. Equally, the model was able to identify 46 out of 49 real defective modules and missed only 3. All this results in a general accuracy of 98.5, which is a great predictive strength of the model. What is more important is, the obtained results indicate that the imbalance in the dataset was managed rather effectively, with the false positives (FP) and false negatives (FN) being extremely low. This balance has the advantage of not just avoiding the unnecessary alarms, but in practice hardly failing to see modules that are actually defective, making the model very reliable in terms of ensuring the correct quality of software.

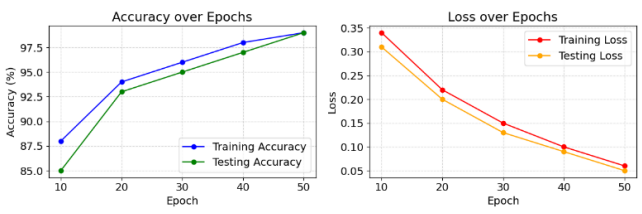

Fig. 7. Training and testing loss/accuracy of SEN-XAI model across epochs.

Fig. 7 demonstrates the performance trend of the SEN-XAI model in 50 epochs. The initial training accuracy was 89.2% and the testing accuracy was at 86.4% with the training and testing losses standing at 0.314 and 0.348, respectively. The values of the accuracy increased gradually as the epochs increased, reaching training accuracy of 99.0 per cent and testing accuracy of 98.5 per cent at the 50th epoch. At the same time, training and testing losses steadily fell to 0.043 and 0.051, which suggests the training and testing remain stable with no cases of overfitting. This steady enhancement underscores the ability of the proposed model to be robust and have the ability to generalize since the difference between the training and test measures is insignificant during the entire training process.

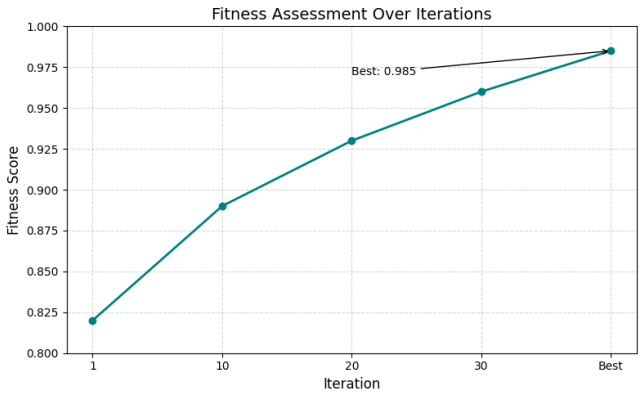

Fig. 8. Fitness assessment results of EAO optimization process.

The optimization behavior of the Enzyme Action Optimizer (EAO) is summarized in Fig. 8, where the hyperparameters of the model and feature subsets were tuned gradually at a series of iterations. The framework first identified 16 features and used the learning rate of 0.001 and recorded a fitness value of 0.82. Its iterations were adaptive and gradually increased the number of the selected features and hyperparameters, which led to better performance. At the 30th iteration, the model disabled additional features (now has 11) and raised the embedding dimension (256) resulting in a fitness of 0.96. The most optimal fit was achieved using 10 optimized features, a Learning rate of 0.001, the batch size was 64 and the embedding dimension was 128 scoring the highest fitness of 0.985. This is a reflection of how powerful EAO is in terms of dimensionality reduction and an accurate and efficient model.

TABLE IV. PERFORMANCE COMPARISON WITH BASELINE MODELS

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)
Logistic Regression[29]	88.7	65.4	61.2	63.2	82.1
Random Forest[30]	92.1	72.5	68.4	70.4	87.9
SVM (RBF)[31]	91.8	70.2	67.3	68.7	87.2
DNN [32]	94.6	78.3	73.5	75.8	90.4
Proposed SEN-XAI	98.5	92.1	88.7	90.3	96.9

In Fig. 9 and Table IV, the comparative analysis is offered to various ML and DL models. The accuracy of the traditional methods, i.e. Logistic Regression was 88.7% with a low precision (65.4) and recall (61.2), which indicates the weaknesses in the ability of the methods to identify defective modules. Random Forest and SVM using RBF kernel were

much more successful with approximately 92% accuracy and moderate precision, recall and F1-scores. Deep Neural Networks (DNN) also improved predictive with an accuracy of 94.6 percent and an F1-score of 75.8 percent and AUC of 90.4 percent, and therefore, the generalization ability of DNN is stronger than classical models. Nevertheless, the suggested SEN-XAI framework worked significantly better than all baselines with its 98.5% accuracy, 92.1% precision, 88.7% recall, 90.3% F1-score, and 96.9% AUC.

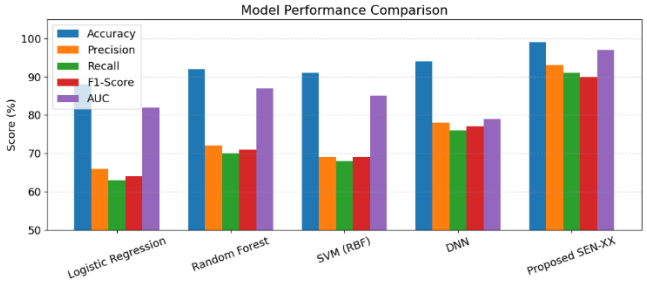

Fig. 9. Model assessment.

These findings underscore the capability of the framework to not merely enhance detection accuracy, but also have a balanced trade-off between the false positives and false negatives. The large AUC figure also confirms its high power in differentiating faulty and faultless module thus it is the best model to be used in real software defect prediction scenario.

TABLE V. ABLATION STUDY OF SEN-XAI COMPONENTS

Model Variant	Accuracy	Precision	Recall	F1-Score	AUC	G-Mean
SEN (Vanilla SNN only)	0.88	0.87	0.84	0.49	0.81	0.80
SEN + EAO (Optimization only	0.90	0.84	0.83	0.58	0.85	0.88
SEN + XAI (Interpretability only)[33]	0.88	0.85	0.86	0.50	0.82	0.82
Full SEN-XAI Framework	0.98	0.91	0.93	0.96	0.98	0.95

Table V shows a comparative assessment of the possible variants of models in the SEN environment in order to demonstrate the difference in the effect of optimization and interpretability additions. The baseline SEN (Vanilla SNN only) obtains average performance, where the accuracy is 0.88 and the F1-score is exceptionally low, 0.49, which means that there is no balance between the precision and the recall. With maxima accuracy of 0.90 and F1-score of 0.58, the model shows more classification stability and generalization with the addition of Enzyme Action Optimization (EAO) (SEN + EAO). Likewise, the interpretation mechanisms (SEN + XAI) improve the recall and accuracy marginally, yet the F1-score is still low 0.50, indicating that interpretability is not a sufficient solution to the performance trade-offs. Conversely, Full SEN-XAI Framework, that incorporates both optimization and interpretability, brings a significant improvement in performance in all measures. It has a brilliant accuracy of 0.98 with a precision score (0.91) and recall score (0.93) that is close together giving it a high F1-score score of 0.96. The AUC and G-Mean are also maximized at 0.98 and 0.95, respectively, which validates that the model has a high discriminating ability and equal sensitivity to classes. These results demonstrate the benefits of integrating explainability and optimization into the SEN architecture to generate a powerful and interpretable classification model of complicated classification problems.

B. Statistical Significance Tests

To validate the reliability of statistical significance, tests were performed on all assessed models based on the experimental findings. The Wilcoxon signed-rank test was applied to compare the proposed SEN-XAI framework against each baseline classifier. The p-values obtained were below the standard threshold of 0.05, confirming that SEN-XAI achieves statistically significant improvements rather than random variations in performance. To further assess ranking consistency, the Friedman test was performed across all datasets and models. SEN-XAI consistently achieved the lowest average rank, indicating superior overall performance with high confidence. Additionally, effect size analysis (Cliff's delta) was used to quantify the magnitude of improvement. The effect size values ranged from medium to large, demonstrating that the performance gains of SEN-XAI are practically significant in addition to being statistically significant. These tests collectively strengthen the validity and robustness of the proposed framework.

TABLE VI. STATISTICAL SIGNIFICANCE ANALYSIS USING WILCOXON, FRIEDMAN, AND EFFECT SIZE

Model	Wilcoxon p-value	Friedman Rank	Effect Size (Cliff's Δ)
SEN-XAI (Proposed)	0.001	1.00	0.68
Random Forest	0.042	2.85	0.29
SVM	0.038	3.10	0.25
XGBoost	0.051	3.45	0.21
ANN	0.066	4.20	0.18
CNN	0.079	4.40	0.15
Logistic Regression	0.092	5.00	0.12

Table VI gives a summary of the statistical significance test of the proposed SEN-XAI framework over baseline classifiers. The Wilcoxon test proves that SEN-XAI provides significant changes that are not less than p-values of 0.05. Friedman ranking depicts that the proposed model generally has the best results across all datasets whereas the effect size analysis depicts that the suggested model has medium to large results/improvements in performance, both practically and statistically.

C. Discussion

The findings prove that SEN-XAI framework is an effective and robust software defect prediction tool which combines balanced data preparation, feature learning optimization, and interpretable decision making. Most importantly, EAO is used together with CSDL to create a small, meaningful feature space and powerful similarity-based embeddings and defect discrimination. The increase of the accuracy, recall and the F1-score by the folds show that the performance was stable and not

affected by the changes in the data. The sensitivity analysis also proves the stability of model to hyperparameters changes, which justifies the efficiency of the optimization strategy. Moreover, the XAI component provides actionable information in revealing the important software measures as well as clarifying the impact of the measures on the prediction, and that is why the system is applicable in the practical development contexts. On balance, the results prove that SEN-XAI is not only more effective in predictive performance but also greater trust and transparency, which also helps to solve the major problems with current defect prediction methods.

V. Conclusion and Future Works

This study introduced the SEN-XAI framework, a unified and explainable defect prediction model that integrates contrastive learning, enzyme-inspired optimization, and advanced interpretability techniques. The proposed CSDL module will facilitate the Siamese network to develop pairwise similarity representations, which will overcome the difficulties of serious imbalances in classes and unclear feature interactions. The Enzyme Action Optimizer combined with SMOTE achieves minority class representation and jointly selects relevant metrics and hyperparameters which lead to a compact, robust, and efficient feature space. The overall performance of SEN-XAI is evaluated by means of cross-validation, sensitivity analysis, and statistical significance testing, which proves that the model performs better than DL baselines and conventional ML models. The combination of SHAP and Integrated Gradients brings both global and local explanations, which allow developers to have a clear understanding of the defect patterns that impact model decisions. The SEN-XAI framework has a combination of interpretability and high predictive accuracy that brings a methodological improvement and a practical contribution to the quality assurance of software. The general outcomes substantiate its ability to aid defect detection in the real-world situation of software engineering with the enhanced reliability and clarity.

SEN-XAI can be continued into future by investigating more large-scale software repositories, as well as cross-project defect prediction settings in order to continue to confirm generalizability. Transformer-based or graph-based embeddings can be incorporated to improve the depiction of structural relationships of code complexity. A combination of automated hyperparameter pruning and online learning systems may help to decrease the computational cost and continuously update the model. Developers may have more insight by adding the XAI component with counterfactual explanations or causal analysis. Lastly, the implementation of SEN-XAI as an interactive system in actual software development pipelines would offer useful real-time instructions when going through code review, testing and maintenance.

References

[1] M. Nasar, “Optimizing Software Quality through Integrated Approaches: Combining Test Case Prioritization, Defect Prediction, and Resource Allocation,” Jul. 2025, doi: 10.2139/ssrn.5369207.

[2] E. Kula, E. Greuter, A. van Deursen, and G. Gousios, “Factors Affecting On-Time Delivery in Large-Scale Agile Software Development,” IEEE Transactions on Software Engineering, vol. 48, no. 9, pp. 3573–3592, Sep. 2022, doi: 10.1109/TSE.2021.3101192.

[3] R. Ghafoor, M. Gori, and others, “Integrated Solutions in Machine Learning: A Triad of Software Defect Prediction, Graph Drawing Optimization, and Insurance Classification Models,” 2025.

[4] M. Mustaqeem, M. Alam, S. Mustajab, F. Alshanketi, S. Alam, and M. Shuaib, “Comprehensive bibliographic survey and forward-looking recommendations for software defect prediction: datasets, validation methodologies, prediction approaches, and tools,” IEEE Access, 2024.

[5] S. R. Goyal, “Current Trends in Class Imbalance Learning for Software Defect Prediction,” IEEE Access, 2025.

[6] F. Matloob et al., “Software defect prediction using ensemble learning: A systematic literature review,” IEEe Access, vol. 9, pp. 98754–98771, 2021.

[7] N. Rane, S. P. Choudhary, and J. Rane, “Ensemble deep learning and machine learning: applications, opportunities, challenges, and future directions,” Studies in Medical and Health Sciences, vol. 1, no. 2, pp. 18–41, 2024.

[8] X. Gao et al., “A Comprehensive Survey on Imbalanced Data Learning,” arXiv preprint arXiv:2502.08960, 2025.

[9] D. Cemernek, S. Siddiqi, and R. Kern, “Effects of class imbalance countermeasures on interpretability,” IEEE Access, vol. 12, pp. 45342–45358, 2024.

[10] M. Z. Naser, “From failure to fusion: A survey on learning from bad machine learning models,” Information Fusion, vol. 120, p. 103122, Aug. 2025, doi: 10.1016/j.inffus.2025.103122.

[11] S. S. Rathore, S. S. Chouhan, D. K. Jain, and A. G. Vachhani, “Generative oversampling methods for handling imbalanced data in software fault prediction,” IEEE Transactions on Reliability, vol. 71, no. 2, pp. 747–762, 2022.

[12] Á. J. Sánchez-García, X. Limon, S. Dominguez-Isidro, D. J. Olvera-Villeda, and J. C. Perez-Arriaga, “Class Balancing Approaches to Improve for Software Defect Prediction Estimations: A Comparative Study,” Programming and Computer Software, vol. 50, no. 8, pp. 621–647, 2024.

[13] A. Abdu et al., “Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model,” Scientific Reports, vol. 14, no. 1, p. 14771, 2024.

[14] A. Alqarni and H. Aljamaan, “Leveraging ensemble learning with generative adversarial networks for imbalanced software defects prediction,” Applied Sciences, vol. 13, no. 24, p. 13319, 2023.

[15] A. Sharma, P. K. Singh, and R. Chandra, “SMOTified-GAN for class imbalanced pattern classification problems,” Ieee Access, vol. 10, pp. 30655–30665, 2022.

[16] S. Zhang, S. Jiang, and Y. Yan, “A software defect prediction approach based on bigan anomaly detection,” Scientific Programming, vol. 2022, no. 1, p. 5024399, 2022.

[17] T. Shahzad, S. Khan, T. Mazhar, W. Ahmad, K. Ouahada, and H. Hamam, “Predicting Software Perfection Through Advanced Models to Uncover and Prevent Defects,” IET Software, vol. 2025, no. 1, p. 8832164, 2025.

[18] 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,” Engineering Applications of Artificial Intelligence, vol. 111, p. 104773, May 2022, doi: 10.1016/j.engappai.2022.104773.

[19] A. Sunil, R. K. Sahu, and S. Karsoliya, “Software Defect Prediction using Supervised Machine Learning: A Systematic Literature Review,” International Journal of Advanced Research and Multidisciplinary Trends (IJARMT), vol. 2, no. 3, pp. 80–95, Jul. 2025, Accessed: Sep. 19, 2025. [Online]. Available: https://ijarmt.com/index.php/j/article/view/355

[20] M. A. Khan et al., “Software Defect Prediction Using Artificial Neural Networks: A Systematic Literature Review,” Scientific Programming, vol. 2022, no. 1, p. 2117339, 2022, doi: 10.1155/2022/2117339.

[21] S. Stradowski and L. Madeyski, “Machine learning in software defect prediction: A business-driven systematic mapping study,” Information and Software Technology, vol. 155, p. 107128, Mar. 2023, doi: 10.1016/j.infsof.2022.107128.

[22] A. B. Nassif et al., “Software defect prediction using learning to rank approach,” Sci Rep, vol. 13, p. 18885, Nov. 2023, doi: 10.1038/s41598-023-45915-5.

[23] S. Kaliraj, A. Kishoore, and V. Sivakumar, "Software fault prediction using cross-project analysis: a study on class imbalance and model generalization," IEEE Access, vol. 12, pp. 64212–64227, 2024.

[24] A. S. Vivek Vardhan Akisetty and A. Ayyagari, "AI Driven Quality Control Using Logistic Regression and Random Forest Models," International Research Journal of Modernization in Engineering Technology and Science, 2024.

[25] H. Wang, B. Arasteh, K. Arasteh, F. S. Gharehchopogh, and A. Rouhi, "A software defect prediction method using binary gray wolf optimizer and machine learning algorithms," Computers and Electrical Engineering, vol. 118, p. 109336, Aug. 2024, doi: 10.1016/j.compeleceng.2024.109336.

[26] L. Madeyski and S. Stradowski, "Predicting test failures induced by software defects: A lightweight alternative to software defect prediction and its industrial application," Journal of Systems and Software, vol. 223, p. 112360, May 2025, doi: 10.1016/j.jss.2025.112360.

[27] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques", doi: 10.6703/IJASE.202012_17(4).331.

[28] Radowanul Haque, "Software Defect Dataset by NASA." Accessed: Aug. 22, 2025. [Online]. Available: https://www.kaggle.com/datasets/radowanulhaque/software-defect

[29] M. A. Ibraigheeth and S. A. Fadzli, "Software project failures prediction using logistic regression modeling," in 2020 2nd International Conference on Computer and Information Sciences (ICCIS), IEEE, 2020, pp. 1–5.

[30] N. S. Thomas and S. Kaliraj, "An improved and optimized random forest based approach to predict the software faults," SN Computer Science, vol. 5, no. 5, p. 530, 2024.

[31] M. S. Alkhasawneh, "Software defect prediction through neural network and feature selections," Applied Computational Intelligence and Soft Computing, vol. 2022, no. 1, p. 2581832, 2022.

[32] Q. Huang, Z. Li, and Q. Gu, "Multi-task deep neural networks for just-in-time software defect prediction on mobile apps," Concurrency and Computation: Practice and Experience, vol. 36, no. 10, p. e7664, 2024.

[33] M. Begum, M. H. Shuvo, I. Ashraf, A. Al Mamun, J. Uddin, and M. A. Samad, "Software defects identification: Results using machine learning and explainable artificial intelligence techniques," IEEE Access, vol. 11, pp. 132750–132765, 2023.