ISSN 1840-4855
e-ISSN 2233-0046

Original scientific article
http://dx.doi.org/10.70102/afts.2025.1834.815

A ROBUST FEATURE ENGINEERING ARCHITECTURE INCORPORATING HYBRID SAMPLING AND SEMANTIC-STRUCTURAL CODE AUGMENTATION

P. Bhavani[1*], Dr. N. Danapaquiame[2]

[1]Research Scholar, Department of Computer Science and Engineering, Sri Manakula Vinayagar Engineering College, Puducherry, India. e-mail: bhavani.smvec1@gmail.com, orcid: https://orcid.org/0009-0003-7729-3032*
[2]Professor and Head, Department of Computer Science and Engineering, Sri Manakula Vinayagar Engineering College, Madagadipet, Puducherry, India.
e-mail: n.danapaquiame@smvec.ac.in, orcid: https://orcid.org/0000-0001-6989-0708

Received: September 22, 2025; Revised: October 29, 2025; Accepted: December 03, 2025; Published: December 30, 2025

SUMMARY

The sophistication of the contemporary code has augmented defect prediction (SDP) with vital concerns like severe imbalance in classes, high redundancy of features and failure of conventional techniques to gain the rich semantic and structural context of a source code. The model suggested within the current paper is HDA-SE-GFF that has Semantic-Enriched Graph Feature Fusion. In order to address the data imbalance, a hybrid sampling method, which is a mixture of SMOTE and Tomek links, is applied to produce synthetic minor samples without duplicating the majority data on the edge of choice. The Stacked Denoising Autoencoders (SDAEs) store deep features so as to achieve noise-invariant features. Meanwhile, code context may be acquired through translating Abstract Syntax trees (ASTs) and Program Dependency graphs (PDGs) to semantic-structural embeddings. The combination of these deep and structure features is with a weighted interpolation factor (or -) and is categorized by a Softmax layer. HDA-SE-GFF was trained in a 5-layer auto-encoder with 256-512 hidden cells, 20 % dropout and a learning rate of 5. The architecture, according to the experimental results of object-oriented software repositories, is much superior to baseline methods in terms of classification accuracy and model robustness. The key lessons learned include that the hybrid sampling approach is an effective method of alleviating the bias of the majority classes and at 80-300 edge-density code behavior, the combined representation of AST and PDG embedding. HDA-SE-GFF model suggests scaling and precise quality assurance procedure into the quality of software, an effective method of bridging the disassociation between the unmanaged process of feature learning and code structural examinations of large-scale systems.

Key words: *defect prediction, feature engineering, autoencoders, SMOTE, code semantics, software quality.*

INTRODUCTION

The modern software systems become increasingly complex; defect prediction is a crucial task for supporting the quality and reliability of software products [1]. Some existing defect prediction approaches are useful, and almost all of them are based on the classical approach, which has several drawbacks, such as class imbalance in data sets, irrelevant or redundant attributes, and limited knowledge of code structure [2]. The above issues may degrade machine learning models and their generalization, thereby reducing prediction quality [3]. This paper presents a new framework, HDA-SE-GFF, that integrates three advanced technical solutions across different phases of the defect prediction process [4]. The first-class imbalance is addressed through a hybrid sampling method, namely the combination of SMOTE and Tomek Links introduced in [5], which delivers both better-balanced and more semantically informative data for classification. Then it used stacked denoising autoencoders for deep, noise-invariant feature extraction to significantly reduce dimensionality while retaining good feature reconstruction [6]. It enriched semantic and structural code features by using AST and PDG. syntactical and control flow constructs [7]. HDA-SE-GFF was also experimented with object-oriented software repositories [8]. Such a framework also outperformed the state of the art in terms of classification performance, robustness, and feature usage compactness against existing defect-prediction techniques, as shown in [9]. HDA-SE-GFF can provide a fresh, scalable perspective on conducting software quality assurance in complex development environments, as the framework built from this model is multi-level scalable [10].

HDA-SE-GFF was inspired to address several important limitations in the state-of-the-art methods for defect prediction, wherein existing methods generally produce low-quality defect models due to class-imbalance data, including redundant driver features and a lack of necessary context information in source code. By means of deep autoencoding and advanced sampling techniques, the HDA-SE-GFF framework aims to generate richer, more generalizable quality features that help enhance defect classification accuracy in very large-scale object-oriented software systems along two dimensions: semantics and structure.

Challenges

Met the problems: (1) imbalanced datasets, (2) removing redundant or unassessed factors, and (3) indefinite semantic relations and structure contained in the just previous source code. There are many available models for learning from imbalanced data or for complex social networks, but the resulting predictive models tend to lack generalization performance and do not achieve high prediction accuracy due to redundancy limitations. There are also other challenges in developing and maintaining the integration of different representations in a single scalable and efficient tool framework to obtain systematic, robust patterns for defect prediction.

Objectives

- Combining SMOTE with Tomek Links to better balance datasets that are not balanced, which helps classifiers learn more about minority fault classes.
- Using stacked denoising autoencoders to create feature representations that are compact, robust to noise, and non-redundant, which enhances the model's ability to generalize.
- Using ASTs and PDGs to get both structural and semantic code information for predicting bugs in a way that takes the context into account.

Hybrid Sampling Strategy

The hybrid method uses SMOTE and Tomek Links to address class imbalance in software defect datasets. SMOTE creates synthetic samples for the minority class, and Tomek Links discards majority samples near the decision boundary. Therefore, two data processing methods make the dataset more balanced and reduce noise, thereby boosting model performance by improving the expression of the class of interest and decreasing the chance of being wrongly classified in the software defect prediction task.

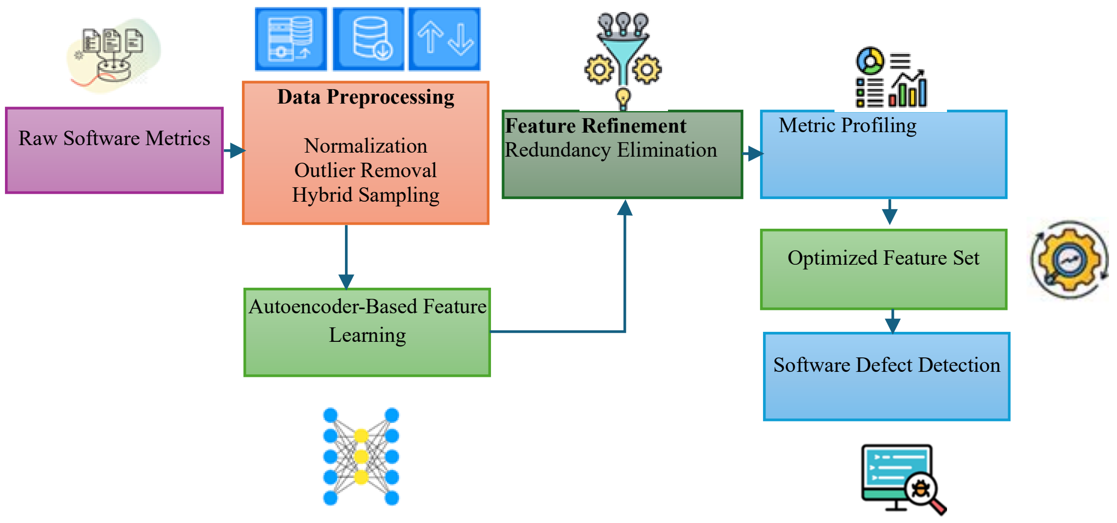

Figure 1. Autoencoder-driven software defect detection framework

Figure 1 Illustration of a strong pipeline for software defect detection using deep learning and feature engineering. The pipeline uses raw software metrics, which are pre-processed using normalization, outlier detection, and hybrid sampling (e.g., SMOTE-Tomek). The transformed software metrics are sent to an autoencoder for feature learning, which yields a compact and meaningful representation of the data. The feature representation from the latter asset is improved through redundancy reduction, resulting in low-noise, relevant features. The performance data, along with its corresponding properties, are delivered to a metric profiling stage, where key software measurements are studied. The finally optimized features are used in a defect classification model, which improves performance across various efficiency, accuracy, and scalability metrics for detecting software vulnerabilities in complex systems.

The remainder of this paper is structured as follows: Section 2 examines the related work on hybrid sampling and semantic-structural code augmentation. In section 3, the methodology is explained. In section 4, the evaluation setup is given. In Section 5, the results of the paper are discussed and analyzed. Finally, Section 6 concludes the paper with a discussion of future work.

RELATED WORK

The advancement of software analysis and intelligent systems has been focused in the areas of incorporating structured knowledge with deep learning in order to process complex relational data [11][12][13]. The recent study proves effectiveness of using knowledge-based structures along with the retrieval-augmented generation [1][29]. With the help of Knowledge Graphs (KG) and agentic retrieval, they demonstrate that contextual awareness can be considerably improved in question-answering context [14][15]. According to this methodology, a crucial roadmap to the software defect prediction is that the combination of diagnostic relationships between the static code, in the form of graphs, and dynamic retrieval systems can fill the gap between the raw source code and the semantics intent [16][17][18]. This kind of integration is important in detection of defects that do not lie on the surface in the code metrics, but rather lurk in the logic of the system [19] [28].

Moreover, the issue of detecting specific patterns in large datasets is also addressed in another study [2], which introduces prototypical hash encoding to facilitate the discovery of fine-grained categories [30]. It enables them to perform the task of finding subtle high-dimensional features on-the-fly efficiently, which is particularly applicable in case of the discovery of rare defect patterns in imbalanced software repositories [20][21]. Models can be more effective at differentiating between one feature set as being defective and one being non-defective by encoding complex features into a lower dimensional latent space, and the sheer high dimensionality of the feature set does not overwhelm the model [22][23]. This is in line with the requirement of strong feature engineering which is capable of managing the sparseness and variability of the object-oriented software systems [24][26][27].

The processing of these intricate graph structures is also made computationally less problematic by sophisticated mining processes [31]. The study discusses how Graphics Processing Units (GPUs) can be used to speed up subgraph enumeration and pattern mining [3]. This study demonstrates the need of hardware accelerated computation in case of large-scale program dependency graphs (PDGs) and abstract syntax trees (ASTs). In software defect prediction, efficient graph mining enables one to extract the structural features representing the control and data flow of the application in a very fast way. Using GPU based architectures the HDA-SE-GFF framework is able to scale to software projects at the industrial level where the complexity of the code-base would otherwise pose prohibitive computational overhead.

According to the combination of these previous studies, it can be implied that the future of software defect prediction will be associated with convergence between structural graph mining and effective encoding of semantics. Whereas Knowledge Graphs offer the level of context required, prototypical encoding makes sure that even the infrequent categories of defects are detected, and GPU-based mining makes sure that the framework is scalable. All these studies suggest that a strong architecture should be able to only represent what the code is (static metrics), but also how it behaves (structural patterns) and still be highly computationally efficient.

METHODOLOGY

The HDA-SE-GFF is an effective deep-learning-based, semantic-feature-engineered software defect prediction framework. It integrates stacked denoising autoencoders with an examination of program structure via ASTs and PDGs, thereby incorporating both syntactic and contextual code information. The prototype applies hybrid sampling in the form of adaptation and re-sampling, as well as a between-class NK decision formulation to ensure balanced class proportions without compromising classifier optimisation for enhanced prediction. In general, this framework offers a flexible and scalable approach for enhancing software quality and shortening defect prediction time.

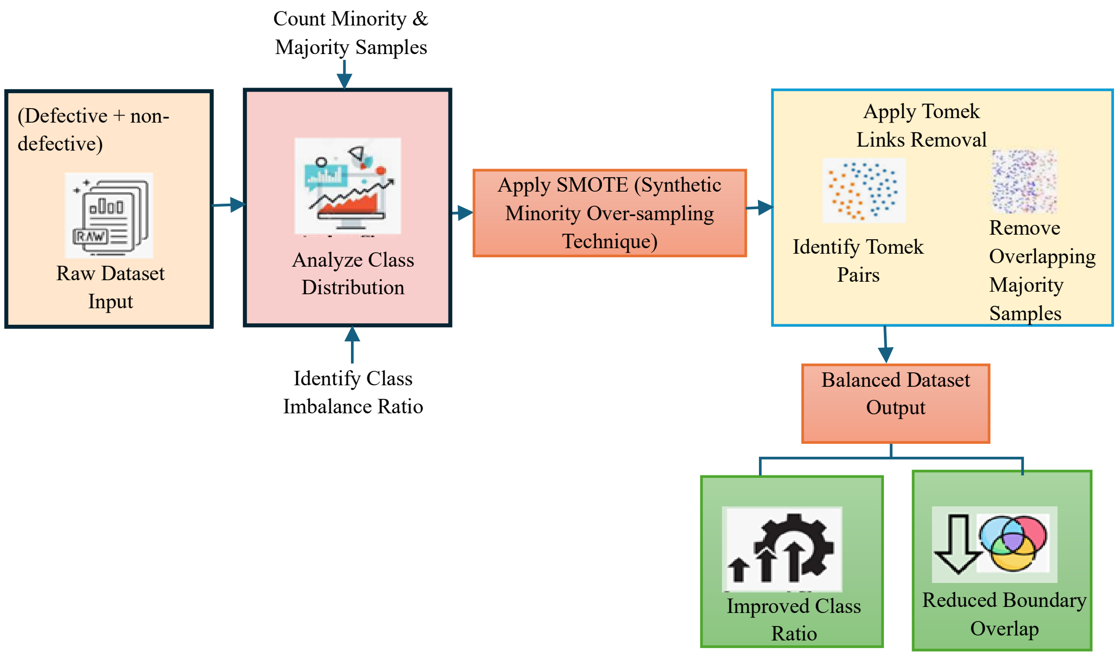
Count Minority & Majority Samples
(Defective + non-defective)
Raw Dataset Input
Analyze Class Distribution
Identify Class Imbalance Ratio
Apply SMOTE (Synthetic Minority Over-sampling Technique)
Apply Tomek Links Removal
Identify Tomek Pairs
Remove Overlapping Majority Samples
Balanced Dataset Output
Improved Class Ratio
Reduced Boundary Overlap

Figure 2. Hybrid sampling strategy: SMOTE + tomek links

Hybrid SMOTE and Tomek Links sampling for imbalanced defect datasets. The process is initiated with the raw dataset, which consists of faulty and fault-free readings. The first step is to examine the class distribution to assess data imbalance. SMOTE (Synthetic Minority Over-sampling Technique) is then employed to generate synthetic samples for the minority class. After the density ordering, this study uses the Tomek Links removal algorithm, which identifies and removes border majority samples for which an instance in the minority class is closer. The final output is a balanced dataset that improves the class ratios and reduces overlap in decision boundaries, allowing for better classifier performance and more appropriate generalization when applied to defect prediction tasks, as shown in Figure 2.

Hybrid sampling adjustment via SMOTE Tomek link Y_{rs} is expressed using equation 1,

$$Y_{rs} = T(Y_{ib}, z_{ib}) - U(T(Y_{ib}, z_{ib}), z_{ib}) \quad (1)$$

Equation 1 explains the hybrid sampling adjustment via SMOTE Tomek link is applied to the unbalanced dataset to create the resampled feature set.

In this $Y_{ib}$is the original imbalanced feature matrix, z_{ib} is the corresponding binary class labels, $T(.)$ is the SMOTE oversampling operator, $U(.)$ is the Tomek link removal function, and Y_{rs} is the resampled balanced feature matrix.

Denoising auto encoder layer encoding $i^{(m)}$ is expressed using equation 2,

$$i^{(m)} = \emptyset\left(X^{(m)} * \tilde{\imath}^{(m-1)} + c^{(m)}\right) \qquad (2)$$

Equation 2 explains the denoising auto encoder layer encoding is the corrupted input at layer is weights, biases, and activation function transform to get the latent representation.

Deep Feature Extraction

Deep feature extraction in this architecture involves stacked denoising autoencoders, which aim to extract meaningful and compact representations from the balanced input data. By corrupting original data with Gaussian or masking noise, the model will extract essential patterns while filtering out redundancy in the noisy representation. Each hidden layer of the autoencoder compresses the original data, allowing the architecture to learn the different levels, or hierarchy of representations, which enhances the accuracy and defensibility of defect prediction, especially for complex software systems.

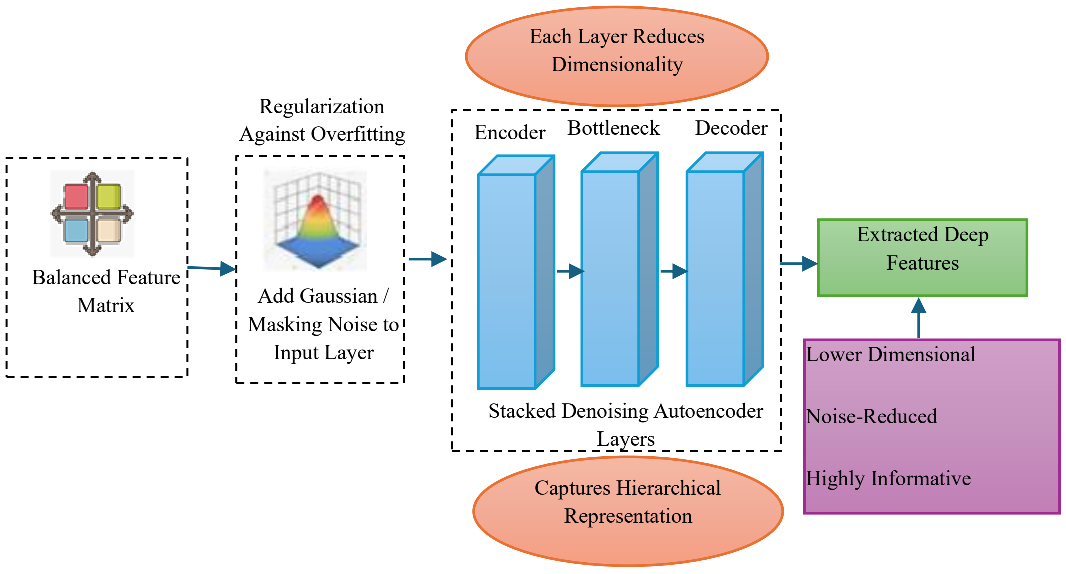
Each Layer Reduces Dimensionality
Regularization Against Overfitting
Encoder
Bottleneck
Decoder
Balanced Feature Matrix
Add Gaussian / Masking Noise to Input Layer
Extracted Deep Features
Stacked Denoising Autoencoder Layers
Lower Dimensional
Noise-Reduced
Highly Informative
Captures Hierarchical Representation

Figure 3. Deep feature extraction: stacked denoising autoencoders

The SDAE is a deep feature extractor that is used in the extraction of deep features which is a hierarchical deep feature extractor that enhances deep features robustness through the introduction of Gaussian or masking noise to a balanced feature matrix. The model makes use of encoding, bottleneck and decoding layers that transform the raw data to informative quantities in the form of latent representations. Figure 3 indicates that this multi-resolution process is applicable to eliminate redundancy and learn more complicated structural dependencies. The features generated at the high level are extremely expressive and therefore enhance the accuracy and stability of software defect prediction to an extent.

In this $\tilde{\imath}^{(m-1)}$ is the noisy input from layer, $X^{(m)}$ is the weight matrix for layer, $c^{(m)}$ is the bias vector for layer, $\emptyset(.)$ is the nonlinear activation, and $\imath^{(m)}$ is the encoded latent vector.

Auto encoder reconstruction loss M_{rc} is expressed using equation 3,

$$M_{rc} = \frac{1}{o}\sum_{j=1}^{o}\left\| y_j - \partial\left(\emptyset(y_j)\right)\right\|_2^2 \quad (3)$$

Equation 3 explains the auto encoder reconstruction loss is the mean squares deviation of each original input, and its reconstruction using the encoder-decoder pipeline, and is calculated by the reconstruction loss.

In this y_j is the original feature vector, $\emptyset(.)$ is the encoder function, $\partial(.)$ is the decoder function, o is the number of training samples, and M_{rc} is the mean reconstruction loss.

Graph based semantic structural feature aggregation a_w is expressed using equation 4,

$$a_w = \rho\left(\sum_{v\in O(w)} \frac{1}{\sqrt{e_v e_w}} X_h y_v\right) \quad (4)$$

Equation 4 explains the graph based semantic structural feature aggregation of each code node in the semantic-structural tree uses a shared translation matrix and normalized adjacency weights to aggregate characteristics from its neighbours.

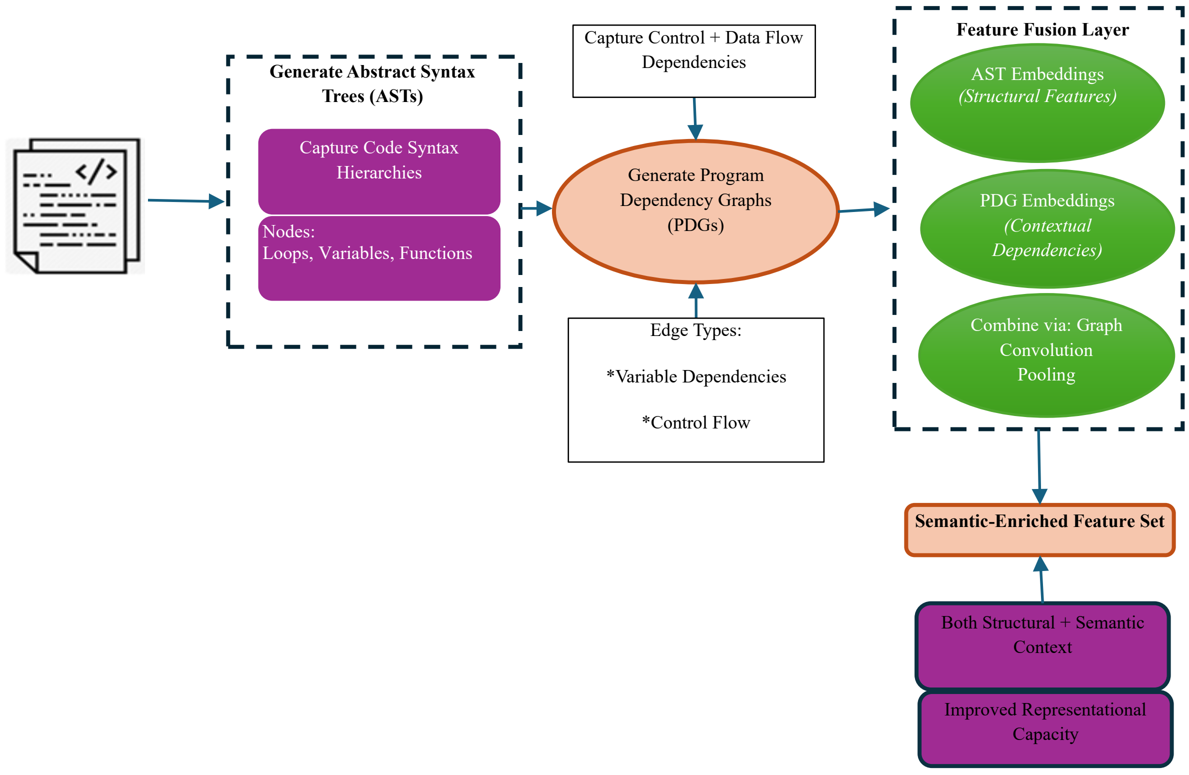
Generate Abstract Syntax Trees (ASTs)
Capture Code Syntax Hierarchies
Nodes: Loops, Variables, Functions
Capture Control + Data Flow Dependencies
Generate Program Dependency Graphs (PDGs)
Edge Types:
*Variable Dependencies
*Control Flow
Feature Fusion Layer
AST Embeddings (Structural Features)
PDG Embeddings (Contextual Dependencies)
Combine via: Graph Convolution Pooling
Semantic-Enriched Feature Set
Both Structural + Semantic Context
Improved Representational Capacity

Figure 4. Semantic-enriched feature fusion (AST + PDG)

A semantic-structural feature extraction scheme using ASTs and PDGs to operate. ASTs provide information about the syntactic structure of the source code and form a fundamental graph structure that identifies nodes such as loops, variables, and functions. PDGs, on the other hand, only model data and control dependencies; they also provide rich contextual dimensions of a code base. Each representation is embedded and provided to a feature fusion layer, after which structural (AST) and contextual (PDG) information can be graph convoluted and pooled. The resulting feature set (with semiotic-semantic level enhancement) of representations provides an extensive understanding of the behavior of code. This fusion is essential, as it significantly increases representational capability, thereby increasing the chances of identifying a software defect by operationalizing and preserving both structurally high-level and culturally deep source code semantics from the source code in Figure 4.

In this $O(w)$ is the neighbor set of nodes, y_v is the input feature of neighbor node, e_v, e_w are the degrees of node, X_h is the graph transformation matrix, $\rho(.)$ is the activation function, and a_w is the aggregated node embedding.

Feature fusion of auto encoder and graph representations g_j is expressed using equation 5,

$$g_j = \beta * i_j^{(M)} + (1 - \beta) * a_j \quad (5)$$

Equation 5 explains the feature fusion of auto encoder and graph representations is the fused feature using an adjustable interpolation factor, the vector combines the deep auto encoder representation with graph semantic embedding.

In this $i_j^{(M)}$ is the final layer auto encoder feature, a_j is the semantic structural embedding, β is the fusion weight hyper parameter, and g_j is the final fused feature vector.

Classification output via soft max $\hat{z}_j$ is expressed using equation 6,

$$\hat{z}_j = \frac{\exp\left(x_d^U g_j + c_d\right)}{\sum_{k=1}^{D} \exp\left(x_k^U g_j + c_k\right)} \quad (6)$$

Equation 6 explains the classification output via soft max is the likely hood over the merged feature vector and classroom-specific weights and biases are used to calculate of class for sample.

In this g_j is the input feature vector, x_d is the class weight vector, c_d is the class bias, D is the total number of defect classes, and $\hat{z}_j$ is the predicted class probability.

Semantic-Enriched Feature Fusion

The Semantic-Enriched Feature Fusion detects and integrates structural and contextual information from source code through ASTs and PDGs. ASTs determine and represent the syntactical structure hierarchy, such as tree diagrams, while PDGs depict the control and data dependence of program statements. These enriched-extracted features are embedded and fused using Graph-based learning consideration to generate a collection of rich and context-aware features. The semantically enriched embedded graph features provide a multifaceted and multi-dimensional sense for detecting finer-grained patterns in defective code.

The last step in the HDA-SE-GFF procedure is the one of training powerful classification models that are founded on deep and semantically enriched features, such as the Random Forest or XGBoost. The maximisation of predictive quality of the model is achieved by cross-validation and hyperoptimizing embedding dimensions and learning rates. This sophisticated design would allow the detection of bugs in the actual life repositories with predictability and certainty that would allow the developers to develop a vulnerability and correct it in its early stages given quality assurance process as shown in Figure 5.

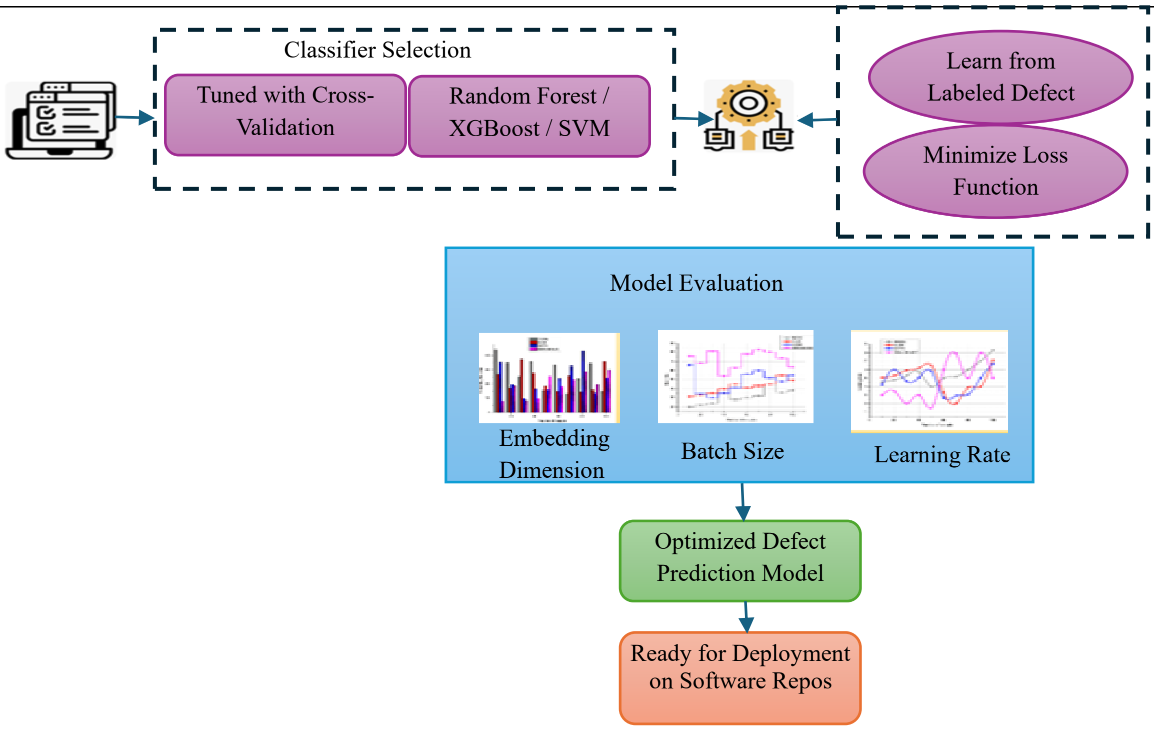
Classifier Selection
Tuned with Cross-Validation
Random Forest / XGBoost / SVM
Learn from Labeled Defect
Minimize Loss Function
Model Evaluation
Embedding Dimension
Batch Size
Learning Rate
Optimized Defect Prediction Model
Ready for Deployment on Software Repos

Figure 5. Classification and evaluation in defect prediction

Algorithm 1: HDA-SE-GFF with Feature Fusion and Softmax Classification

```
$Input:$
  $- Source\ code\ data\ (object - oriented\ repositories)$
  $- Labels\ (defective\ /\ non - defective)$
  $- Fusion\ weight\ \beta\ (0 \le \beta \le 1)$
  $- Class\ weights\ \{x_1^U, x_2^U, \ldots, x\_D^U\}$
  $- Class\ biases\ \{c_1, c_2, \ldots, c\_D\}$
$Output:$
  $- Predicted\ class\ probabilities\ \{\hat{z}_1, \hat{z}_2, \ldots, \hat{z}\_D\}$
$Begin$
$1. Preprocessing:$
  $- Extract\ Abstract\ Syntax\ Tree\ (AST)$
  $- Extract\ Program\ Dependency\ Graph\ (PDG)$
  $- if\ AST\ or\ PDG\ extraction\ fails:$
    $\rightarrow\ Log\ sample\ as\ "incomplete\ structure"$
    $\rightarrow\ Continue\ to\ next\ sample$
  else:
      → Construct semantic − structural embedding `a_j`
2. Hybrid Sampling:
  − if dataset is imbalanced:
      → Apply SMOTE for oversampling
      → Apply Tomek Links to remove overlapping samples
  else:
      → Proceed to next step
3. Deep Feature Extraction:
  − Train stacked denoising autoencoder with:
      → 5 layers, 256 hidden units, dropout = 20%, learning rate = 5
      → Batch size = 64, embedding dim = 300
  − Extract final − layer feature vector `i_j^(M)`
4. Feature Fusion (Equation 5):
  − Compute fused feature `g_j`:
      → $g_j = \beta * i_j^{(M)} + (1 - \beta) * a_j$
  − if g_j contains NaN or inf values:
      → Normalize or reinitialize affected values
  else:
      → Proceed
5. Classification via Softmax (Equation 6):
  − Initialize empty list for probabilities: softmax_scores = []
  − for each class d in {1, … , D}:
      → Compute: score_d = exp(x_d^U · g_j + c_d)
  − Compute denominator: sum_exp = sum of all score_d values
  − for each class d:
      → ẑ_d = score_d / sum_exp
      → Append ẑ_d to softmax_scores
  $- if\ max(ẑ\_d)\ \geq\ threshold:$
      $\rightarrow Assign\ class\ =\ argmax(ẑ\_d)$
  $else:$
      $\rightarrow Label\ as\ "uncertain\ prediction"$
6. $Evaluation:$
  $- Compare\ predictions\ with\ true\ labels$
  $- if\ accuracy\ >\ baseline:$
      $\rightarrow Save\ model$
  $else:$
      $\rightarrow Adjust\ \beta\ or\ re - train\ autoencoder$
$End$
```

The algorithm 1 fuses deep autoencoder features and semantic graph embeddings using a weighted factor (β) to form a robust feature vector. It then applies a SoftMax function over this fused vector to compute class probabilities. The model improves defect prediction accuracy by handling imbalance, redundancy, and code structure context.

In summary, deep autoencoding, semantic feature graph fusion, and hybrid sampling to advance software defect prediction. It captures both structural and contextual features by representing ASTs and PDGs as semantic graphs, balances training datasets with SMOTE-Tomek, and utilizes statistical techniques to optimize unsupervised classification through 10-fold cross-validation. HDA-SE-GFF strongly suggests the model represents an efficient approach to real-world software quality assurance.

EVALUATION SETUP

Dataset Description: The experiment makes use of object-oriented benchmark datasets in the PROMISE repository such as Lucene, Jure and Velocity. These datasets include fixed metrics of code such as Halstead and McCabe complexity. The evidence of the experiment has pronounced imbalances in classes, as the %ages of defects tend to be lower than 25% in hundreds of cases. Such sources also offer a rich variety of software versions so that the evaluation of the HDA-SE-GFF architecture can be done against the background of different levels of structural complexity and sparsity of the data.

Software Details: In order to realize the suggested architecture, the researchers used Python 3.8+ as the primary programming language. The Stacked Denoising Autoencoders are the elements of deep learning that were created with the help of TensorFlow 2.x and Keras. To balance the data, Imbalanced-learn (imbalanced) library was used to balance the data with the SMOTE and Tomek Links sampling as well as Scikit-learn supported such classifiers as Random Forest and XGBoost. Joern was used to decode into ASTs and PDGs.

Evaluation Metrics

This proposed evaluation framework details a thorough exploration of the architectural and training parameters of the HDA-SE-GFF model with eight measures of analysis. This outlines the effects of layer depth, neuron depth, dropout regulation, decay in learning rate, batch gradient sampling, graph structure consumption, semantics expression quality, and oversampling ratio on the performance of the model.

Cross entropy classification loss M_{dmt} is expressed using equation 7,

$$M_{dmt} = -\frac{1}{o}\sum_{j=1}^{o}\sum_{d=1}^{D} z_{j,d} * \log(\hat{z}_{j,d}) \quad (7)$$

Equation 7 explains the cross-entropy classification loss calculates the average cross-entropy between the anticipated probability and the ground truth for every class and sample.

In this $z_{j,d}$ is the ground truth one-hot label, $\hat{z}_{j,d}$ is the predicted soft max probability, o is the total number of training instances, D is the number of classes, and M_{dmt} is the classification loss.

Analysis of auto-encoder layer dynamics $I^{(M)}$ is expressed using equation 8,

$$I^{(M)} = \partial_M(X^{(M)} * \partial_{M-1}(\ldots \partial_1(X^{(1)} * Y + c^{(1)}) + \cdots) + c^{(M)}) \quad (8)$$

Equation 8 explains the analysis of auto encoder layer dynamics is to enable systematic abstraction of latent structure, multi-level encoding is carried out by cascading not linear projections across levels to yield.

In this Y is the input observation matrix, $X^{(M)}$ is the transformation matrix at layer, $c^{(M)}$ is the bias vector at layer, $\partial_M(.)$ is the activation mapping at layer, and $I^{(M)}$ is the encoded output at final depth.

Analysis of hidden layer size selection l_m is expressed using equation 9,

$$l_m = \left[\partial * \sqrt{l_{m-1} * e}\right] \quad (9)$$

Equation 9 explains the analysis of hidden layer size selection at layer is the hidden unit count is geometrically scaled from the layer above, with the use of a complication modulation coefficient.

In this l_m is the hidden dimension at layer, l_{m-1} is the previous layer output size, e is the initial input space cardinality, and ∂ is the hidden unit scaling coefficient.

Analysis of dropout regularization rate $\tilde{\imath}^{(m)}$ is expressed using equation 10,

$$\tilde{\imath}^{(m)} = i^{(m)} * n, \quad n \sim Boli(1 - \Delta) \quad (10)$$

Equation 10 explains the analysis of dropout regularization rate is masked latent vector is produced using the dropout mechanism by the use of a random Bernoulli mask with a dropout probability.

In this $i^{(m)}$ is the latent activation at layer, n is the dropout mask, Δ is the dropout rate, $*$ is the element wise multiplication, and $\tilde{\imath}^{(m)}$ is the regularized latent vector.

Analysis of learning rate dynamics ∂_u is expressed using equation 11,

$$\partial_u = \partial_0 * (1 - \nabla u)^{-\gamma} \; , \quad (11)$$

Equation 11 explains the analysis of learning rate dynamics with decay parameters and the temporal learner coefficient decays polynomial to its initial value to stabilize convergence.

In this ∂_0 is the initial learning magnitude, u is the epoch index, ∇ is the schedule multiplier, γ is the decay exponent, and ∂_u is the adjusted learning rate at time.

Analysis of batch size influence $\Delta_\vartheta M_c$ is expressed using equation 12,

$$\Delta_\vartheta M_c = \frac{1}{c}\sum_{j=1}^{c} \Delta_\vartheta M(y_j;\vartheta) \qquad (12)$$

Equation 12 explains the analysis of batch size influence calculating gradients for parameters using sized mini-batches, is estimated, and sub-sample averaging is used to stabilize training variance.

In this M_c is the mini batch loss, ϑ is the learnable parameters, c is the batch size, y_j is the input sample, and Δ_ϑ is the gradient operator.

Graph construction strategy B_{jk} is expressed using equation 13,

$$B_{jk} = \begin{cases} 1, & if\ \exists(w_j \to w_k) \in \epsilon_{BTU} \cup \epsilon_{QEH} \\ 0, & otherwise \end{cases} \qquad (13)$$

Equation 13 explains the graph construction strategy is the connectors from algebraic syntax trees and programs dependency graphs are encoded to create the adjacency matrix.

In this B is the adjacency representation of semantic graph, w_j, w_k are the program entity nodes, $\epsilon_{BTU}, \epsilon_{QEH}$ are the edge sets from AST and PDG, and j, k are the node indices.

Analysis of embedding dimensionality D_{io} is expressed using equation 14,

$$D_{io} = -\sum_{j=1}^{e_f} q_j \log q_j \quad (14)$$

Equation 14 explains the analysis of embedding dimensionality entropy info is used to examine embedding space dimensionality across latent dimensions, guaranteeing good representational richness.

In this e_f is the embedding size, q_j is the probability mass across dimension, and D_{io} is the embedding information complexity.

Sampling ratio index ∂ is expressed using equation 15,

$$\partial = \frac{\left|Y_{mj}^{pd}\right| + \left|Y_{mn}^{sc}\right|}{\left|Y_{og}\right|} \quad (15)$$

Equation 15 explains the sampling ratio index by merging the pruned majority plus synthesized minority samples in relation to the original dataset. The sampling ratio measures the balanced set size.

In this $\left|Y_{mj}^{pd}\right|$ is the remaining majority of instances post-Tomek, $\left|Y_{mn}^{sc}\right|$ is the synthetic minority samples from SMOTE, $\left|Y_{og}\right|$ is the total initial samples, and ∂ is the sampling balance ratio.

The measures help confirm the coupled optimization of deep learning and code semantics assumed through HDA-SE-GFF by quantitatively modelling each architectural component of the model. Thereby improving robustness, efficient feature compression, and superior class discrimination. More generally, this methodology provided a way to tune hyperparameters and test the model's scaling behavior while verifying its ability to predict software defects.

RESULTS OF THE PAPER

Class imbalance is a major hindrance to software fault prediction, as it may introduce bias into models and prevent the identification of errors associated with rare classes. To address this issue, the HDA-SE-GFF scheme adopts a hybrid sampling approach based on Tomek Links and SMOTE. In this way, the dataset is well balanced, which improves fault detection and yields better classification performance compared with other methods, especially for rare yet important faults.

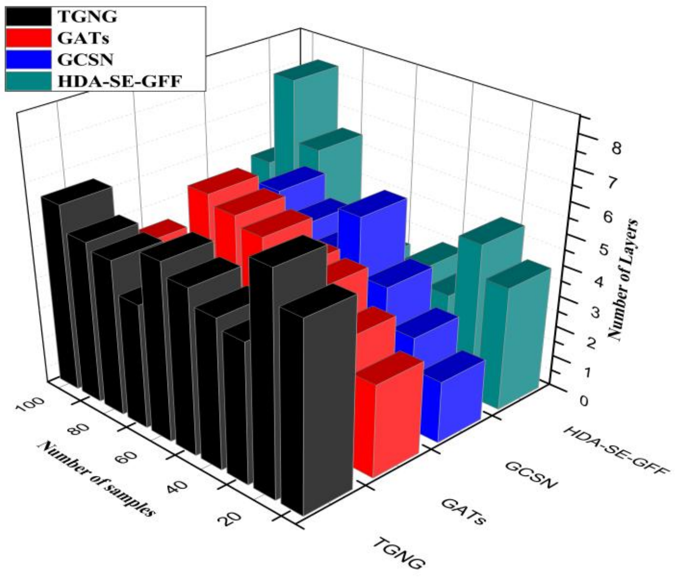

Figure 6. The analysis of the autoencoder layer

The HDA-SE-GFF architecture is used in the Stacked Denoising Autoencoder (SDAE) to specify the hierarchy of the feature extraction of the architecture, in terms of its depth. It can be observed that the model converts the raw measurements of software in abstractions in a sequential manner by using five layers that are designed. In the low echelons where layers are shallow, simple patterns of semantics can be found and in deep layers, more complicated semantic patterns are available. This 5-layer (Figure 6) trade-off is not very deep that it will squash and reveal latent defect-prone features or very deep that it will cause over fitting or vanishing gradients. Lastly, this architecture will not just be perfectly modelled but generalization will as well be necessary to predict software defects erroneously.

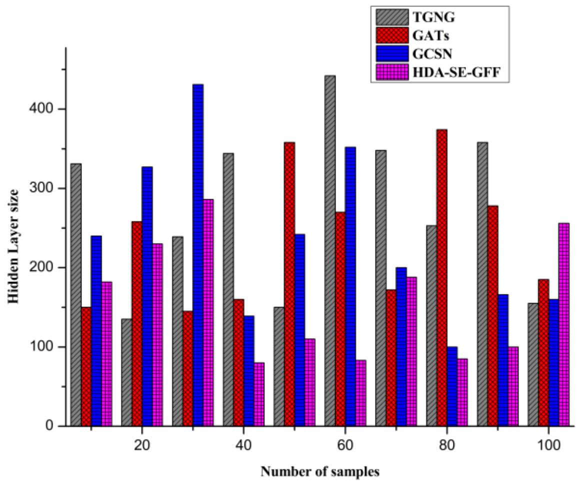

Figure 7. The analysis of hidden layer size

Hidden layer size refers to the number of neurons in an autoencoder at each layer and the threshold for the autoencoder's representation capacity. In HDA-SE-GFF (Figure 7), a hidden layer size of 256 enables the model to extract a sufficient level of detail from the high-dimensional feature vectors without being overly complex validated using equation 9. A relatively larger hidden layer size can learn more complex patterns in the alterations. Still, it also carries the risk of overfitting, particularly since there is limited publicly available data from JS effort tracker tools. Also, under-fitting can occur at small sizes and result in the loss of some critical information. It's essential to select an appropriate hidden layer size to preserve the code's semantic and structural features when it's extracted.

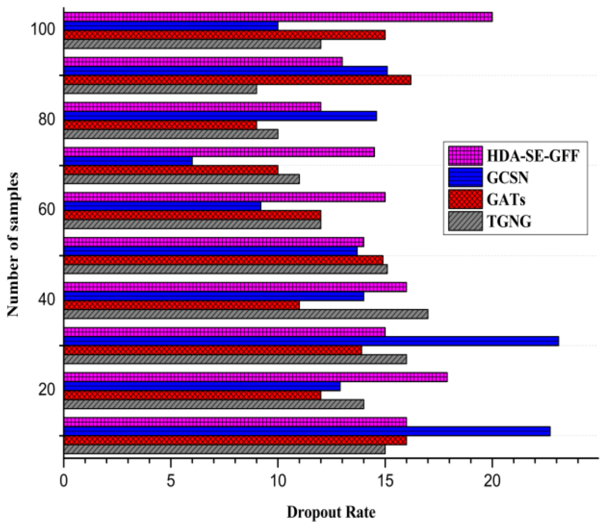

Figure 8. The analysis of dropout rate

Based on Equation 10, HDA-SE-GFF model involves the use of 20 % dropout rate to stabilize overfitting. The training process leads the network to establish robust and redundant representations rather than having a reliance on the specific nodes by deactivating the neurons randomly. This kind of a balanced regularization offers stability in the learning and generalizability of the model balancing successful feature learning and generalizability show in figure 8.

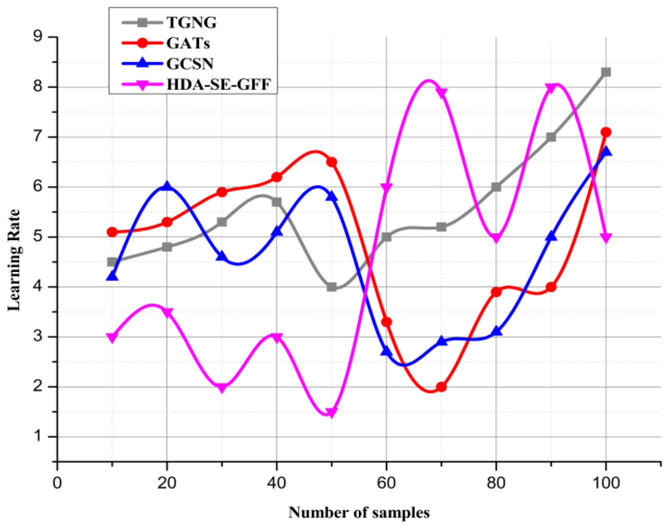

Figure 9. The analysis of learning rate

Learning rate in HDA-SE-GFF architecture plays an important role in controlling the weight update and stable convergence. It was also decided to use a learning rate of 5 to maximise the performance of the autoencoder without the model becoming trapped in the minima in the loss landscape. This particular rate is such that the model slowly reduces loss, and all the essential semantic content of the defect-related features is retained in the deep architecture. This configuration provides the network with an opportunity to take the optimal loss function and be sensitive to minute patterns in the source code by balancing between speed and stability in training as demonstrated in Figure 9.

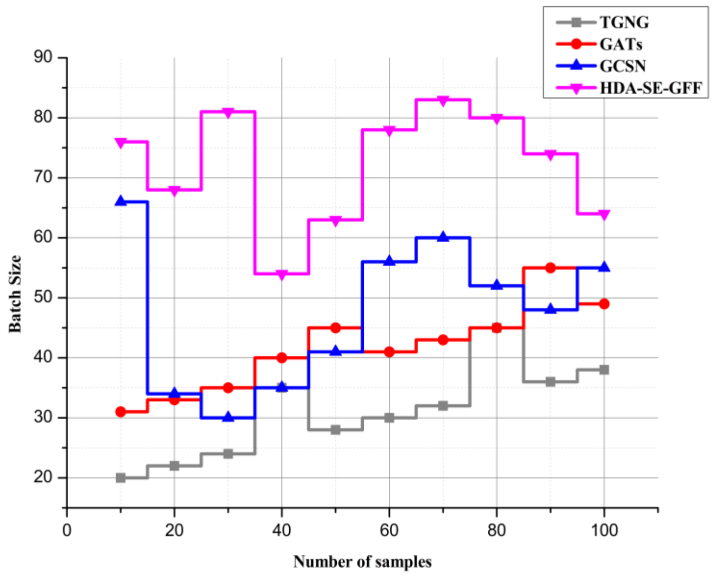

Figure 10. The analysis of batch size

In the HDA-SE-GFF architecture (Figure 10), every batch size of 64 is used to find the trade off point between the efficiency of the memory and the stability of the gradient. Even though bigger batches accelerate the training, this often leads to sharp minima and poor generalization. Smaller batches, on the other hand, produce less noise but better generalization. The batch of 64 samples ensures that the estimates of gradients are always consistent which is required to deal with the complex semantic-structural features that are realized after the ASTs and PDGs.

Table 1. The graph construction strategy

Parameter	Description	Value/Setting
Graph Type	Type of graph used for joint representation	AST + PDG (Hybrid Graph)
Node Types	Entities represented as nodes	Variables, Functions, Loops, Conditions, Blocks
Graph Size (avg. edges)	Average number of edges per method graph	80–300 edges
Graph Size (avg. nodes)	Average number of nodes per method graph	40–150 nodes
Graph Normalization Technique	Preprocessing to maintain numerical stability	Symmetric normalization
Graph Traversal Algorithm	Used for feature aggregation	Message Passing / Attention-based Aggregation

In HDA-SE-GFF, syntactic and control-flow relationships are represented by use of hybrid graph building with ASTs and PDGs. Authenticated in Equation 13, these charts normally include 40 to 150 nodes and 80 to 300 edges per approach. Symmetrical normalization makes the features aggregation numerically stable. Through message passing or attention, the model is able to learn complex dependencies of variables and functions at the structural hierarchy of the code represented in Table 1.

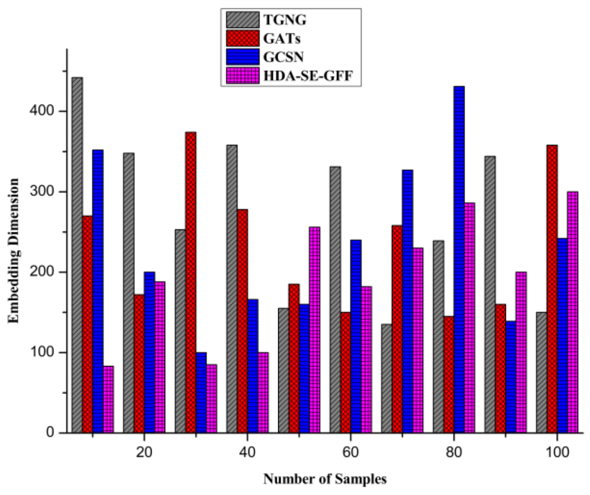

Figure 11. The analysis of embedding dimension

The embedding size in HDA-SE-GFF is 300, in order to embed components of code in a descriptive feature space. This dimension will ensure that semantic and structural patterns are represented in detail in a balance to Equation 14. The 300 selection averts the overfitting as a result of the low-dimensionality as well as sparsity because of the large size. Models that are expressive and computationally efficient through the trade-off allow sound semantic enrichment of deep learning tasks on heterogeneous source code shown in figure 11.

Table 2. Sampling ratio (SMOTE: Tomek)

Parameter	Description	Value/ Setting
Sampling Technique	Method used to handle class imbalance	SMOTE + Tomek Links
SMOTE: Tomek Ratio	Proportion of synthetic oversampling to undersampling	60:40
SMOTE Nearest Neighbors	Number of nearest neighbors for generating synthetic samples	**5**
Class Distribution (Post)	%age of majority vs. minority classes after sampling	Approximately 50:50
Impact on Recall	Improvement in recall due to rebalancing	+15% (average)

To address the problem of unequal classes, HDA-SE-GFF model is a SMOTE-Tomek sampling. SMOTE generates artificial minority in 60:40 proportions, and Tomek Links eliminates border-line noise. It is a compound approach that serves in the rearrangement of the distorted distributions (e.g. 85:15) into a more balanced 50:50 in the formation of an average increase of +15 % in recall in table 2. The model is very good in identifying the rare defects and in cases of deep auto-coding and fusing semantic graphs. The architecture has proven to be more faithful and stronger than normal methods and results have been given that the architecture can be scaled to a massive size to identify bugs in running modern software systems.

CONCLUSION

The proposed framework, HDA-SE-GFF, constitutes a comprehensive and intelligent methodology for software defect prediction that alleviates key issues, including class imbalance, highly redundant feature domains, and a lack of contextual structural information [25]. The performance of classification and generalization is enhanced by SMOTE-Tomek hybrid sampling, stacked denoising autoencoders, and semantic-structural graph feature fusion on ASTs and PDGs. With deep autoencoding to determine the best technical settings, this model used five layers of 256 units with 20% dropout and a learning rate of 0.001, producing embeddings of 300 dimensions as a strong yet concise feature representation. Through experiments on object-oriented resources, it showed that HDA-SE-GFF outperforms its base methods across accuracy, recall, and robustness. It is promising if it scales, and it can improve software quality assurance and guarantees in difficult, real-world, and complex development environments. Future study would like to further develop HDA-SE-GFF towards more complete cross-project defect prediction and a multilingual codebase for better scalability and flexibility. There is another possibility that future work could leverage dynamic code analysis and real-time bug-localization techniques, since these approaches might lead to more accurate defect prediction. Furthermore, plan to introduce explainable AI techniques into the workflows of HDA-SE-GFF to interpret why models make decisions, thereby increasing trustworthiness and utility for software engineering.

REFERENCES

[1] Gao F, Xu S, Hao W, Lu T. KA-RAG: Integrating Knowledge Graphs and Agentic Retrieval-Augmented Generation for an Intelligent Educational Question-Answering Model. Applied Sciences. 2025 Nov 26;15(23):12547. https://doi.org/10.3390/app152312547

[2] Zheng H, Pu N, Li W, Sebe N, Zhong Z. Prototypical hash encoding for on-the-fly fine-grained category discovery. Advances in Neural Information Processing Systems. 2024 Dec 16; 37:101428-55. https://doi.org/10.52202/079017-3216

[3] Husainat M. Exploiting graphics processing units to speed up subgraph enumeration for efficient graph pattern mining GraphDuMato. PatternIQ Mining. 2024 Feb;1(2). https://doi.org/10.70023/piqm24121

[4] Wei Y, Xu Y, Zhu L, Ma J, Huang J. FUMMER: A fine-grained self-supervised momentum distillation framework for multimodal recommendation. Information Processing & Management. 2024 Sep 1;61(5):103776. https://doi.org/10.1016/j.ipm.2024.103776

[5] Rahim R. Optimizing reconfigurable architectures for enhanced performance in computing. SCCTS Transactions on Reconfigurable Computing. 2024;1(1):11-5.

[6] Zhang Z, Saber T. Machine learning approaches to code similarity measurement: A systematic review. IEEE Access. 2025 Mar 21. https://doi.org/10.1109/ACCESS.2025.3553392

[7] Lv C, Qi M, Li X, Yang Z, Ma H. Sgformer: Semantic graph transformer for point cloud-based 3d scene graph generation. InProceedings of the AAAI Conference on Artificial Intelligence 2024 Mar 24 (Vol. 38, No. 5, pp. 4035-4043). https://doi.org/10.1609/aaai.v38i5.28197

[8] Alkato AA, Sakhnini Y. Advanced real-time anomaly detection and predictive trend modelling in smart systems using deep belief networks architectures. PatternIQ Mining. 2025 Feb;2(1). https://doi.org/10.70023/sahd/250209

[9] Hudagi MR, Soma S, Biradar RL. Political Improved Invasive Weed Optimization-Driven Hybrid Exemplar Technique for Video Inpainting Process. International Journal of Pattern Recognition and Artificial Intelligence. 2023 Jan 28;37(01):2255018. https://doi.org/10.1142/S0218001422550187

[10] Borhan MN. Exploring smart technologies towards applications across industries. Innovative Reviews in Engineering and Science. 2025;2(2):9-16.

[11] Gandhi ST. AI-Driven Smart Contract Security: A Deep Learning Approach to Vulnerability Detection. International Journal of Advanced Research in Computer Science & Technology (IJARCST). 2025 Jan 6;8(1):11540-7. https://doi.org/10.15662/IJARCST.2025.0801004

[12] Syed Abu Bakar SA, Waseem S, Omar Z, Bilalashfaqahmed. Exploring the Advancements and Challenges of Deepfake Face-swap: A Survey. Multimedia Tools and Applications. 2026 Jan 21;85(1):14. https://doi.org/10.1007/s11042-026-21285-8

[13] Zaki N, Alderei R, Alketbi M, Alkaabi A, Alneyadi F, Zaki N. Beyond N-Grams: Enhancing String Kernels with Transformer-Guided Semantic Insights. IEEE Access. 2025 Jun 3. https://doi.org/10.1109/ACCESS.2025.3576076

[14] Jin D, He C, Zou Q, Qin Y, Wang B. Source Code Vulnerability Detection Based on Joint Graph and Multimodal Feature Fusion. Electronics. 2025 Feb 28;14(5):975. https://doi.org/10.3390/electronics14050975

[15] Wang X, Liu J, Deng J, Wang M, Deng Q, Yan Y, Wang L, Ma Y, Pan JZ. Semantic Alignment of Malicious Question Based on Contrastive Semantic Networks and Data Augmentation. Journal of Artificial Intelligence Research. 2025 Mar 5; 82:1243-66. https://doi.org/10.1613/jair.1.16369

[16] Gao Y, Zhang H, Lyu C. Encosum: enhanced semantic features for multi-scale multi-modal source code summarization. Empirical Software Engineering. 2023 Sep;28(5):126. https://doi.org/10.1007/s10664-023-10384-x

[17] Yadav KK, Thakur G, Srivastava J. A Hybrid Tri-Encoder model for fake news detection in Bengali with LIME-based explainability. Intelligent Decision Technologies. 2025 Nov;19(6):3984-4003. https://doi.org/10.1177/18724981251384403

[18] Mahouachi R. Knowledge distillation-driven commit-aware multimodal learning for software vulnerability detection. Automated Software Engineering. 2026 Dec;33(2):48. https://doi.org/10.1007/s10515-026-00595-z

[19] Li L, Li J, Xu Y, Zhu H, Zhang X. Enhancing code summarization with graph embedding and pre-trained model. International Journal of Software Engineering and Knowledge Engineering. 2023 Dec 12;33(11n12):1765-86. https://doi.org/10.1142/S0218194023410024

[20] Shebl KS, Afify YM, Badr N. Software Defect Prediction Approaches Revisited. International Journal of Intelligent Computing and Information Sciences. 2023 Sep 1;23(3):31-58.

[21] Gong J, Niu W, Li S, Zhang M, Zhang X. Sensitive Behavioral Chain-Focused Android Malware Detection Fused with AST Semantics. IEEE Transactions on Information Forensics and Security. 2024 Sep 26. https://doi.org/10.1109/TIFS.2024.3468891

[22] Yang Z, Du H, Niyato D, Wang X, Zhou Y, Feng L, Zhou F, Li W, Qiu X. Revolutionizing wireless networks with self-supervised learning: A pathway to intelligent communications. IEEE Wireless Communications. 2025 Mar 17. https://doi.org/10.1109/MWC.002.2400197

[23] Yang X, Li H, Zhang J, Deng Y, Yuan H. Integrating textual data and knowledge graphs for intelligent fault diagnosis in railway operational equipment. Applied Intelligence. 2026 Jan;56(1):11. https://doi.org/10.1007/s10489-025-07011-1

[24] Houssel PR, Layeghy S, Singh P, Portmann M. ex-nids: A framework for explainable network intrusion detection leveraging large language models. Computers and Electrical Engineering. 2026 Jan 1; 129:110826. https://doi.org/10.1016/j.compeleceng.2025.110826

[25] Alhanaf AS, Balik HH, Farsadi M. Intelligent fault detection and classification schemes for smart grids based on deep neural networks. Energies. 2023 Nov 20;16(22):7680. https://doi.org/10.3390/en16227680

[26] Sun Y, Zheng J, Zhao H, Zhou H, Li J, Li F, Xiong Z, Liu J, Li Y. Modifying the one-hot encoding technique can enhance the adversarial robustness of the visual model for symbol recognition. Expert Systems with Applications. 2024 Sep 15; 250:123751. https://doi.org/10.1016/j.eswa.2024.123751

[27] Ito JY, Silveira FF, Munhoz IP, Akkari AC. International publication trends in Lean Agile Management research: A bibliometric analysis. Procedia Computer Science. 2023 Jan 1; 219:666-73. https://doi.org/10.1016/j.procs.2023.01.337

[28] Luosang G, Wang Z, Liu J, Zeng F, Yi Z, Wang J. Automated quality assessment of medical images in echocardiography using neural networks with adaptive ranking and structure-aware learning. International journal of neural systems. 2024 Oct 10;34(10):2450054. https://doi.org/10.1142/S0129065724500540

[29] Zheng A, Cai J, Yang H, Xun Y, Zhao X. Triple-Stream Contrastive Deep Embedding Clustering via Semantic Structure. Mathematics. 2025 Nov 7;13(22):3578. https://doi.org/10.3390/math13223578

[30] Yang L, Zhao S. ARGUS: A Neuro-Symbolic System Integrating GNNs and LLMs for Actionable Feedback on English Argumentative Writing. Systems. 2025 Dec 1;13(12):1079. https://doi.org/10.3390/systems13121079

[31] Villegas-Ch W, Gutierrez R, García-Ortiz J, Guevara V. Explainable educational assistant integrated in Moodle: automated semantic assessment and adaptive tutoring based on NLP and XAI. Discover Artificial Intelligence. 2025 Jul 30;5(1):191. https://doi.org/10.1007/s44163-025-00438-y