

A DEEP GENERATIVE ENSEMBLE FRAMEWORK BASED ON TABULAR VARIATIONAL AUTOENCODER FOR IMBALANCED SOFTWARE DEFECT PREDICTION

Moch. Lutfi^{1*}, Muhammad Faishol Amrulloh², Zulfatun Nikmatus Saadah³,
Ahmad Zulham Fahamsyah Havy⁴, Arief Tri Arsanto⁵

^{1, 2, 3, 4, 5}Computer Science, Universitas Yudharta Pasuruan, Jl. Yudharta No.7, Kembangkuning, Sengonagun, Purwosari, Pasuruan, 67162, East Java, Indonesia.

¹<https://orcid.org/0000-0001-9227-759X>, ²<https://orcid.org/0009-0009-9721-1480>, ³<https://orcid.org/0009-0004-2450-7268>

⁴<https://orcid.org/0009-0009-3006-7963>, ⁵<https://orcid.org/0009-0001-2386-8940>

Email: *moch.lutfi@yudharta.ac.id, faishol@yudharta.ac.id, zulfa@yudharta.ac.id, zulham92@yudharta.ac.id, arief_inf@yudharta.ac.id

ARTICLE INFO

Article History

Received: January 18, 2026

Reviewed: March 20, 2026

Accepted: April 20, 2026

Published: May 29, 2026

Keywords:

tabular variational autoencoder,
ensemble learning,
software defect prediction.

ABSTRACT

Class imbalance is a prevalent issue in Software Defect Prediction (SDP), where non-defective modules significantly outnumber defective ones. This imbalance often degrades the performance of prediction models. While oversampling techniques like SMOTE are commonly used to address this problem, they often produce synthetic instances that are either too similar to the original data (leading to overgeneralization and high false positive rates) or overly diverse (reducing the model's ability to detect defects). This study proposes a novel hybrid approach that combines a generative oversampling method, Tabular Variational Autoencoder (TVAE), with ensemble learning to improve defect prediction performance. The aim is to achieve a better balance between precision and recall while effectively handling imbalanced datasets. TVAE is used to generate realistic synthetic defect instances by learning the underlying joint distribution of tabular software metrics. The synthetic data is then used to train ensemble models, including TVAE+Naïve Bayes with Bagging and TVAE+Random Forest with Gradient Boosting Stacking. The proposed methods are evaluated on seven NASA PROMISE datasets and compared against baseline and state-of-the-art approaches using AUC, G-Mean, Accuracy, and F1-Score. Experimental results show that TVAE+RF+GB consistently outperforms other methods, achieving an average accuracy of 93%, AUC of 0.91, and F1-Score of 0.93. TVAE+NB+Bagging also demonstrates significant improvement over baseline models, particularly in G-Mean. Statistical tests confirm the significance of these improvements in most evaluation metrics. The integration of TVAE with ensemble learning effectively addresses class imbalance in SDP and enhances prediction performance. TVAE+RF+GB is recommended as a robust and efficient solution for imbalanced defect prediction datasets.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

Software Defect Prediction (SDP) is a data-based identification technique that utilizes historical information from software development projects [1] to determine whether a software module is defective or not [2]. The main objective of this technique is to predict and identify potentially defective modules at an early stage of the software development cycle[3]. SDP offers a more efficient alternative to conventional methods such as debugging, which requires manual examination of the entire source code, execution flow, and program conditions [3], [4]. By reducing the number of defects in the final product, SDP contributes directly to improving the overall quality of the software.

High-quality software is generally characterized by a minimal number of defects found during the inspection and testing stages, as well as its ability to meet user needs and expectations. However, software testing is one of the most expensive and time-consuming stages in the development process, with approximately 80% of the total project schedule allocated to testing activities [2], [5]. This situation leads to increased development, maintenance, and estimation costs, as well as negatively impacting the overall quality of the software [6]. Therefore, a more efficient and cost effective approach is needed to detect software defects early on. To develop and evaluate an accurate SDP model, a high-quality dataset containing comprehensive information about software module characteristics and defect history is required. One of the most commonly used datasets is NASA's public PROMISE dataset [7]. However, this dataset has class imbalance characteristics, where defective modules represent only a small portion of the overall data (8). This imbalance can cause a significant decrease in classification model performance, especially in detecting minority classes [8]. Without adequate preprocessing, prediction models tend to be biased towards the majority class and fail to recognize defective modules effectively [9]. In general, there are two main approaches to handling class imbalance, namely at the data level and the algorithm level [10], [8]. The data-level approach is carried out through resampling techniques to balance the class distribution in the training data, such as Random Over-Sampling (ROS) [11], [12], Random Under-Sampling (RUS), SMOTE (Synthetic Minority Over-sampling Technique) [13], ADASYN, COSTE, and MAHAKIL [14-16]. Meanwhile, the algorithm level approach is done by modifying the structure or learning mechanism in machine learning algorithms, for example through the use of ensemble methods that combine several basic models.

The two most commonly used ensemble methods are boosting and bagging [17], which have been proven to improve the performance of classifiers on imbalanced data. Various conventional oversampling techniques such as ADASYN, SPIDER3, MetaCost Learners [5], SMOTE, and MAHAKIL [16], [18] have been widely used to address imbalance issues, but each has its limitations. For example, SMOTE tends to generate synthetic samples that are too similar to the original data, leading to overgeneralization, while MAHAKIL often generates data that is too varied, reducing defect detection accuracy. Several previous studies have also proposed hybrid approaches to improve prediction accuracy. However, most still rely on traditional sampling methods such as SMOTEND [19], SMOTENDE [20], SMOTEEN, and SMOTETomek [21], [22]. In addition, the ARRAY (Adaptive Triple Feature-Weighted Transfer Naïve Bayes) method, which integrates the Naïve Bayes algorithm [23], has also been proposed to improve classification performance, but its effectiveness is still limited to large-scale and high-dimensional datasets. As a solution to these limitations, this study proposes an innovative approach that integrates Tabular Variational Autoencoder (TVAE) with ensemble learning to improve the performance of Naïve Bayes and Random Forest algorithms. The main advantage of this approach lies in the application of deep generative sampling techniques that can generate synthetic data with realistic and representative distributions [24], [25], unlike traditional sampling methods that are less effective for high-dimensional data. TVAE [26], [27] is expected to balance class distributions and improve the model's generalization ability in detecting defective modules more accurately. The novelty of this research lies in the combination of TVAE-based deep generative oversampling techniques with ensemble learning to improve classification accuracy. This approach is designed to overcome the limitations of traditional oversampling methods, which are not yet optimal in handling unbalanced data with complex structures and large dimensions.

This study aims to propose a hybrid method that combines data-level and algorithm-level approaches to address class imbalance in SDP. At the data level, resampling is performed using Tabular Variational Autoencoder (TVAE) [28], which is an encoder-decoder-based generative model that learns the joint distribution of tabular features through ELBO optimization. TVAE can generate synthetic samples that preserve statistical structure and inter-feature correlations, making it more effective for oversampling than SMOTE. The baseline classification algorithms used include Naïve Bayes [8], [2] and Random Forest. At the algorithm level, ensemble methods [29] such as Bagging and Stacking Gradient Boosting were used. The proposed methods TVAE + NB + Bagging and TVAE + RF + Stacking, were compared with existing and baseline methods on seven NASA datasets. Based on this formulation, this study proposes two main research questions:

1. RQ1: Does the TVAE hybrid method with ensemble methods provide better classification performance than the baseline method on the NASA dataset?
2. RQ2: Does the TVAE hybrid method with ensemble methods provide better classification performance than the state-of-the-art method on the NASA dataset?

Based on preliminary analysis, this study is believed to improve the performance of the Naïve Bayes and Random Forest algorithms in handling imbalanced data in SDP. The main contributions of this study are as follows :

1. Proposing the implementation of Tabular Variational Autoencoder to overcome class imbalance in software defect prediction.
2. Systematically evaluating the effect of this technique on two common classification algorithms, namely Naïve Bayes and Random Forest.
3. Integrating ensemble learning to optimize the performance of both algorithms in the context of imbalanced datasets.

II. RELATED WORK

Various oversampling techniques have been proposed to address class imbalance in software defect prediction (SDP). Some of the most widely used methods include SMOTE, ADASYN, SPIDER3, MetaCost learners [5], and MAHAKIL [16], [18]. Among these methods, SMOTE (Synthetic Minority Over-sampling Technique) is the most popular approach, which works by creating new synthetic instances from the minority class through linear interpolation between neighboring instances. However, this method has the disadvantage of producing synthetic instances that are too similar to the original data, resulting in overgeneralization and failing to capture the complex variations in the data. Conversely, MAHAKIL often produces instances that are too diverse, which actually reduces the model's ability to detect software defects. In addition to these methods, previous studies have also introduced various hybrid oversampling techniques, such as SmoteND [19], SMOTENDE [20], SMOTEEN, and SMOTETomek [21], [22].

These techniques combine SMOTE with preprocessing or cleaning processes to improve classification accuracy. For example, SmoteND and SMOTENEDE combine SMOTE with noise detection to improve the quality of synthetic samples, while SMOTEEN and SMOTETomek combine it with undersampling in the boundary region to reduce ambiguity between classes. Research results show that this hybrid approach can improve classification performance. However, most of these methods are still based on traditional sampling principles, which are limited in handling high-dimensional data with complex feature correlations. Other methods such as ARRAY (Adaptive Triple Feature-Weighted Transfer Naïve Bayes) have also been proposed to improve classification performance by integrating adaptive feature weighting into the Naïve Bayes algorithm [23]. Although promising results have been shown, this method has limitations in handling large-scale datasets due to high computational requirements. In recent years, deep generative models such as Variational Autoencoders (VAE) and Generative Adversarial Networks (GANs) have shown significant success in generating high-quality synthetic data in various domains, especially in image and text data. One of the latest variants, Tabular Variational Autoencoder (TVAE), is an adaptation of VAE specifically designed for tabular data [28]. Unlike traditional sampling methods that rely on simple interpolation, TVAE uses an encoder-decoder architecture to learn the joint distribution between tabular features through Evidence Lower Bound (ELBO) optimization. This model maps continuous variables as Gaussian distributions and categorical variables as categorical distributions, enabling it to handle mixed-type data commonly found in the SDP domain.

The main advantage of TVAE lies in its ability to reconstruct complex statistical structures and correlations between features, which is difficult to achieve with linear interpolation approaches such as SMOTE. Although TVAE has been successfully applied in fields such as healthcare and finance, its application in the context of software defect prediction, especially in combination with ensemble learning, has yet to be explored. Traditional sampling methods such as SMOTE and its derivatives still rely on simple linear interpolation and are unable to capture non-linear relationships between features in high-dimensional data, which is a characteristic feature of software metric datasets. Meanwhile, although deep generative models such as VAE have proven effective in other domains, their application to oversampling in software defect prediction is still very limited. Most existing research tends to focus on only one approach either at the data level or the algorithm level and few have examined the optimal integration between deep generative oversampling and ensemble learning. Therefore, this study proposes an innovative approach that integrates TVAE as a generative oversampling technique with Bagging and Stacking Gradient Boosting-based ensemble learning. This approach combines the advantages of TVAE in generating high-quality synthetic samples that maintain complex statistical structures with the power of ensemble learning in optimizing the classification process. The proposed method was then comprehensively evaluated using seven different NASA datasets to ensure the generalization of the results. This approach is expected to overcome the limitations of traditional oversampling methods in handling high-dimensional imbalanced data, which is common in software defect prediction.

III. PROPOSED METHOD

III.1 DATASET COLLECTION

The dataset used in this study is public data taken directly from the PROMISE (Predictor Models in Software Engineering) Repository website. The dataset includes CM1, KC1, KC2, KC3, PC2, PC3, and PC4. This NASA dataset was selected because it has been widely used by previous researchers as a basis for developing software defect prediction models [21].

Table 1: Datasets.

Dataset	JM1	PC1	MW1	CM1	PC5	KC1	KC2	KC3
Attribut	22	41	41	44	40	22	22	41
Number of modules	10878	1,107	403	505	17186	2107	522	458
Defective modules	2102	76	31	48	516	325	107	43
Non-defective modules	8776	1031	372	457	16670	1782	415	415
Defect presentation	19.3%	6.87%	7.69%	9.5%	3%	15.42	20.5%	9.39%
Non-defect presentation	80.7%	93.13%	92.31%	90.5%	97%	84.58	79.5%	90.61%

Source: Authors, (2026).

III.2 DATA PREPROCESSING (MIN - MAX NORMALIZATION)

The data preprocessing stage is carried out by applying the min-max normalization technique, which transforms the value of each attribute to be within a certain range based on the minimum and maximum values in the original data. Generally, the minimum value $x'_{min,j} = -1$ and $x'_{max,j} = 1$; however, in some implementations, the range $x'_{min,j} = 0$ to $x'_{max,j} = 1$ is also employed.

$$x'_{i,j} = \left(\frac{x_{i,j} - x_{min,j}}{x_{max,j} - x_{min,j}} \right) (x'_{max,j} - x'_{min,j}) + x'_{min,j}$$

III.3 SYSTEM IMPLEMENTATION AND TESTING

Model trained using datasets with class imbalance in software defect prediction, sourced from both public and private datasets. After the training process, the model was evaluated to measure the accuracy level of the classification algorithm used. The following figure presents the model proposed in this study.

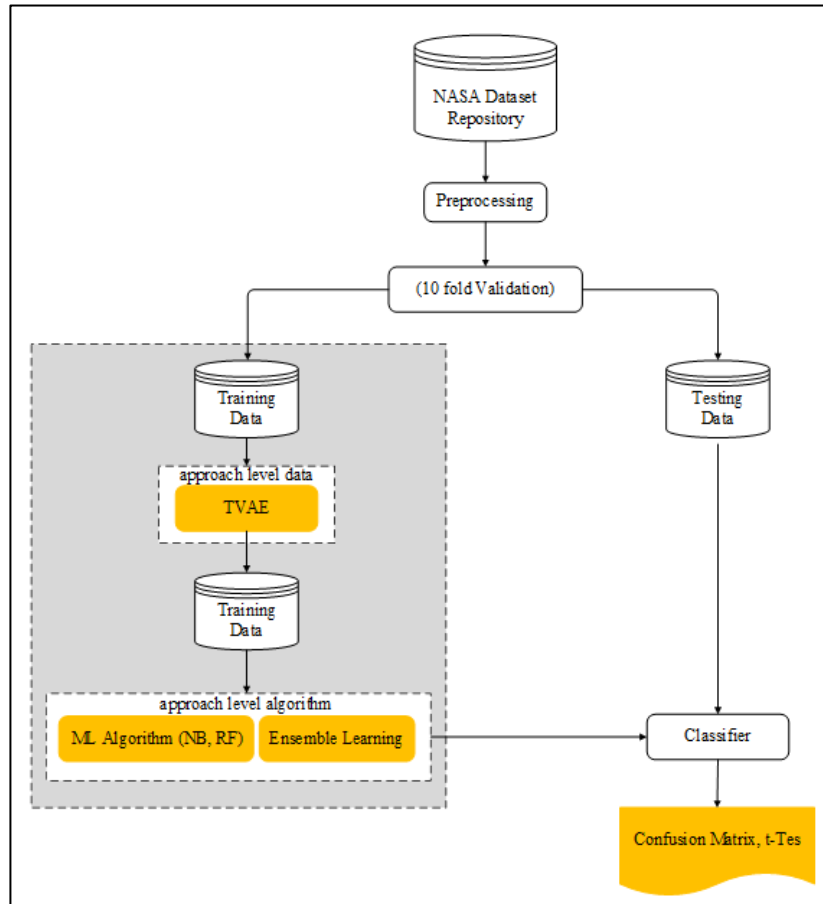


Figure 1: Proposed model.

Source: Authors, (2026).

a) TVAE (Tabular Variational Autoencoder) Architecture

Tabular Variational Autoencoder is a variant of Variational Autoencoder (VAE) adopted by Synthetic Data Vault (SDV) to study joint distributions in tabular data, which generally consists of a combination of continuous and categorical features. This model is capable of generating new synthetic data that has similar statistical characteristics to the original data. In principle, the TVAE process involves three main stages:

1. Encoder maps each row of data x to latent space z with an approximated posterior assumption, such as a Gaussian distribution.
2. Decoder reconstructs $\hat{x}$ from the latent representation z .
3. Model is trained by minimizing the Evidence Lower Bound (ELBO) function, which includes reconstruction and Kullback–Leibler (KL) regularization components, enabling it to learn the statistical structure of tabular data, including the dependency relationships between columns.

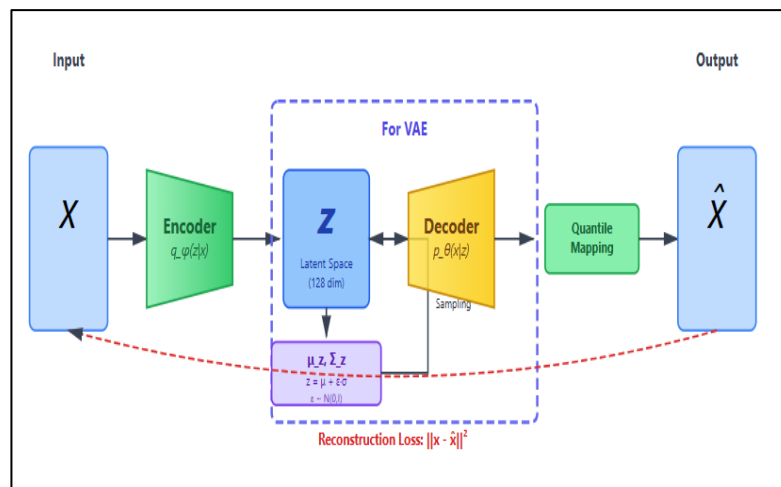


Figure 2: Architecture TVAE.

Source: Authors, (2026).

b) Machine Learning Algorithms

This study uses two classification algorithms, namely Naive Bayes and Random Forest, to predict software defects. Naive Bayes is a classification algorithm based on Bayes' theorem, combined with the naive assumption that each attribute is considered independent or unrelated to one another [30]. Bayes' basic theorem can be stated as follows: $P(F) = \frac{P(C).P(F|C)}{P(F)}$. In the context of Naive Bayes, each attribute is used to determine the most appropriate class for an object. The relationship between attributes and classes can be expressed using the following formula:

$$P(C|F_1 \dots F_n) = \frac{P(C).P(F_1 \dots F_n|C)}{P(F_1 \dots F_n)}$$

Based on this formula, the probability that an object with certain attributes belongs to class C (called posterior probability) is calculated as the product of the probability of class C appearing before the object is observed (prior probability) and the probability of the object's attributes appearing in class C (likelihood), then divided by the probability of the object's attributes appearing overall (evidence). Simply put, this relationship can be expressed as: $Posterior = \frac{Prior \cdot likelihood}{evidence}$.

Algorithm 2 Naive Bayes Classifier

Require: Training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, where $x^{(i)} = (x_1^{(i)}, \dots, x_d^{(i)})$ and $y^{(i)} \in \{C_1, \dots, C_K\}$
Ensure: Predicted class $\hat{y}$ for a new instance $x = (x_1, \dots, x_d)$

```

1: Training Phase:
2: for each class  $C_k \in \{C_1, \dots, C_K\}$  do
3:   Estimate prior probability  $P(C_k) \leftarrow \frac{\text{count}(y=C_k)}{N}$ 
4:   for each feature  $x_j$  do
5:     Estimate likelihood  $P(x_j | C_k)$ 
6:     (use frequency for categorical, Gaussian for continuous)
7:   end for

8: Prediction Phase:
9: for each class  $C_k$  do
10:  Compute posterior  $P(C_k | x) \propto P(C_k) \prod_{j=1}^d P(x_j | C_k)$ 
11: end for
12: Select class  $\hat{y} \leftarrow \arg \max_{C_k} P(C_k | x)$ 
13: return  $\hat{y}$ 

```

Random Forest is an ensemble learning-based classification algorithm that combines a number of decision trees to produce more accurate and stable predictions. Each tree in the ensemble is trained using a subset of data obtained through the bootstrap sampling technique. In each node splitting process, only a random subset of features is considered to determine the best Split [31]. The basic principle of Random Forest is based on the concept of bagging (bootstrap aggregating), where the final prediction result is determined through majority voting in classification cases, or through averaging in regression cases. This approach effectively reduces the risk of overfitting that often occurs in single decision tree models, while improving the ability to generalize to previously unobserved data.

Algorithm 3 Random Forest Classifier

Require: Training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, number of trees T , number of features per split m
Ensure: Ensemble classifier $H(x)$

```

1: Training Phase:
2: for  $t = 1$  to  $T$  do
3:   Generate bootstrap sample  $D_t$  from  $D$ 
4:   Grow decision tree  $h_t$  using  $D_t$ 
5:   for each node in  $h_t$  do
6:     Randomly select  $m$  features from the feature set
7:     Choose best feature and threshold among  $m$  (e.g., Gini, entropy)
8:   end for
9:   Grow tree until maximum depth or stopping criterion (no pruning)
10: end for
11: Construct forest  $H(x) \leftarrow \{h_1(x), h_2(x), \dots, h_T(x)\}$ 

12: Prediction Phase:
13: for each tree  $h_t \in H(x)$  do
14:   Obtain class prediction  $\hat{y}_t \leftarrow h_t(x)$ 
15: end for
16: Combine predictions by majority voting:  $\hat{y} \leftarrow \text{mode}(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T)$ 
17: return  $\hat{y}$ 

```

c) Algorithm Level Approach

Bagging is an ensemble method that is widely used in classification tasks [32]. Its purpose is to improve model accuracy by combining several single classifiers (base classifiers). This method has been proven to produce generally better performance than single random sampling techniques. The process involves combining the results from each classifier, and the final decision is determined based on the classification result that receives the most votes (majority voting). The calculation formula can be expressed as follows:

$$C_f(x_i) = \arg \max_{j \in Y} \sum_{b=1}^B I(C_b(x_i) = j)$$

Algorithm 4 Ensemble Bagging (Bootstrap Aggregating)

Require: $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, base learner $h(\cdot)$, number of models M
Ensure: Ensemble predictor $H(x)$

- 1: **Training Phase:**
- 2: **for** $m = 1$ to M **do**
- 3: Generate bootstrap sample D_m by sampling N instances with replacement from D
- 4: Train base learner h_m on D_m
- 5: **end for**
- 6: Construct ensemble $H(x) = \{h_1(x), h_2(x), \dots, h_M(x)\}$
- 7: **Prediction Phase:**
- 8: **for each** $h_m \in H(x)$ **do**
- 9: $\hat{y}_m \leftarrow h_m(x)$
- 10: **end for**
- 11: **return** $\hat{y} = \text{mode}(\hat{y}_1, \dots, \hat{y}_M)$ (classification)
- 12: $\hat{y} = \frac{1}{M} \sum_{m=1}^M \hat{y}_m$ (regression)

d) Evaluation Using Confusion Matrix

To measure model performance, this study uses a confusion matrix because this tool is effective in analyzing the extent to which the classifier is able to recognize tuples or features from various classes [33]. The confusion matrix provides a comprehensive overview of the classification model's performance based on the number of correct and incorrect predictions for each class [34].

Table 2: Confusion Matrix.

Class	Positive	Negative
Positive	(True Positive)	(False Negative)
Negative	(False Positive)	(True Negative)

Source: Authors, (2026).

Based on the values of True Negative (TN), False Positive (FP), False Negative (FN), and True Positive (TP), various evaluation metrics such as precision, accuracy, and recall can be calculated. The precision value describes the accuracy of the model in classifying data correctly, while recall shows the model's ability to effectively recognize all instances of the positive class.

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

$$sensitivity = recall = tp_{rate} = \frac{tp}{tp + fn}$$

$$specificity = tn_{rate} = \frac{tn}{tn + fp}$$

$$fp_{rate} = \frac{fp}{fp + tn}$$

$$fn_{rate} = \frac{fn}{tp + fn}$$

$$precision = ppv = \frac{tp}{tp + fp}$$

$$npv = \frac{tn}{tn + fp}$$

$$f - measure = \frac{(1 + \beta^2) \times recall \times precision}{(\beta \times recall + precision)}$$

$$g - mean = \sqrt{sensitivity \times specificity}$$

e) Statistical Test

A statistical t-test [5], [35] was used to compare the performance of all developed models, including the Naïve Bayes and Random Forest models, both at the data level using Gaussian-GAN and at the algorithm level using the bagging ensemble method. Few studies in the field of software defect prediction explicitly report the use of statistical analysis to determine the significance of experimental results.

IV. EXPERIMENTS

The experimental framework employed in this study consists of the following stages: 1) dataset collection, 2) meta-learning, 3) learning algorithms, 4) model validation, 5) model evaluation, and 6) model comparison. The experiments were conducted on a platform equipped with an Intel Core i7 CPU (1.9 GHz), 16 GB RAM, and the Microsoft Windows 10 Professional 64-bit operating system. The development environment utilized Google Colab with Python as the programming language.

IV.1 DATASET

Datasets used in this study are public data obtained from the PROMISE Repository (Predictor Models in Software Engineering), which includes the CM1, KC1, KC2, KC3, PC2, PC3, and PC4 datasets. All of these datasets originate from NASA software projects and have been widely used in Software Defect Prediction (SDP) research due to their reliability, ease of access, and relevance in comparing various machine learning approaches in the field of software engineering [8]. This PROMISE dataset contains software metrics extracted from real-world projects, including product metrics (such as lines of code, cyclomatic complexity, and dependency size) and attributes related to the development process. Each instance in the dataset represents a software module that is labeled with a binary value to indicate whether the module is defective or non-defective. This structure makes it particularly suitable for classification research with imbalanced data, as defective modules are generally a minority class compared to non-defective modules.

Table 3: NASA PROMISE Repository Dataset Metrics for Software Defect Prediction.

Symbol	Description
loc	McCabe's line count of code
v(g)	McCabe's "cyclomatic complexity"
ev(g)	McCabe's "essential complexity"
ing(g)	McCabe's "design complexity"
n	Halsted total operators + operands
v	Halsted "volume"
l	Halsted "program length"
d	Halsted "difficulty"
i	Halsted "intelligence"
e	Halsted "effort"
b	Halsted "error"
t	Halsted's time estimator
IOCode	Halsted's line count
IOComment	Halsted's count of lines of comments
IOBlank	Halsted's count of blank lines
locCodeAndComment	Line of comment and code
uniq_Op	Halsted unique operators
uniq_Opnd	Halsted unique operands
total_Op	Halsted total operators
total_Opnd	Halsted total operands
branchCount	Branch count of the flowgraph
defect	Class {false, true}

Source: Authors, (2026).

Table 3 summarizes the datasets used in this study, including the number of instances, number of attributes, and distribution of defective and non-defective modules. The NASA PROMISE dataset was selected based on the following three main considerations:

1. Benchmarking value: This dataset has been widely used in various defect prediction studies, allowing for meaningful comparisons with previous research.
2. Diversity: This dataset covers a wide range of projects and software modules, providing broad coverage in evaluating software quality.
3. Class imbalance: The significant difference between the number of defective and non-defective modules makes this dataset highly relevant for testing the effectiveness of oversampling techniques and data imbalance handling methods.

IV.2 HYPERPARAMETER FRAMEWORK

Hyperparameters are parameters that need to be adjusted in the machine learning classification process to improve model performance or accuracy. These parameters play an important role in determining the learning process, construction, and evaluation of classification models. The hyperparameter optimization process aims to find the best combination of values that produce optimal performance, often referred to as the model selection and hyperparameter tuning stage.

Table 4: Hyperparameter.

No	Architecture	Value
1	Input dim	Variable
2	Latent dim	128
3	Encoder layer_1	512 units
4	Encoder layer_2	256 units
5	Encoder dropout	0.1
6	Decoder layer_1	256 units
7	Decoder layer_2	512 units
8	Decoder dropout	0.1
9	Output activation	Linear
Training		
1	Epochs	600
2	Batch size	64
3	Optimizer	Adam
4	Learning rate	1e-3 (0.001)
5	Beta (β)	0.5
6	Shuffle buffer	Min (1000, N)
Loss Function		
1	Reconstruction loss	MSE
2	Kl divergence	$KL(q p)$
3	Total loss	Recon + $\beta \cdot KL$

Source: Authors, (2026).

IV.3 MODEL VALIDATION

To evaluate model performance, this study uses the stratified 10-fold cross-validation technique, which is a common approach in empirical machine learning studies. The dataset is divided into ten folds of equal size; in each iteration, nine folds are used as training data, while the remaining one fold is used as test data. This process is repeated ten times to minimize bias due to random data division. Results are evaluated based on the average performance across all folds, accompanied by standard deviation values to illustrate the level of variability. The use of stratification ensures that the class distribution in each fold remains consistent with the original distribution, which previous studies have shown to provide a small but consistent improvement in prediction accuracy.

IV.4 SIGNIFICANCE TEST

The paired t-test is used when two samples have pairs of observations that are interdependent. This test evaluates the differences between each pair, then calculates the t-statistic based on the variance of these differences to determine whether the mean difference is equal to zero. In this study, the paired t-test was applied to the mean values of each dataset to assess statistical significance at a confidence level of $\alpha = 0.05$.

IV.5 RESULTS AND ANALYSIS

In the first step, a series of experiments were conducted using the NASA MDP dataset by applying two classification algorithms, namely Naïve Bayes and Random Forest. This study specifically focused on applying these two models to evaluate their performance in the context of class imbalance. The results of the experiments are presented in Table 6. These findings are in line with the results of [3], which show that Naïve Bayes and Random Forest tend to experience a decline in performance when applied to datasets with an imbalanced class distribution.

Table 5: ROC- AUC Results.

Method	KC1	KC2	KC3	JM1	PC1	PC5	MW1	CM1
NB	0.7918	0.8660	0.8205	0.7816	0.7594	0.9474	0.7633	0.7657
TVAE+NB+B	0.7930	0.9281	0.8118	0.8260	0.7687	0.9422	0.7678	0.7646
RF	0.8539	0.9956	0.8853	0.9607	0.9002	0.9849	0.7886	0.8248
TVAE+RF+GB	0.8606	0.9971	0.9289	0.9610	0.9190	0.9784	0.7768	0.8516

Source: Authors, (2026).

Table 5 and Figure 3 show the Area Under the Curve (AUC) values of the ROC curves for each method tested on seven NASA datasets, namely KC1, KC2, KC3, JM1, PC1, PC5, MW1, and CM1.

In general, the results show that the TVAE + RF + GB (Stacking) method provides the highest AUC value in almost all datasets compared to other methods. A high AUC value reflects the model's ability to better distinguish between defective and non-defective modules. Based on the results in the table, the TVAE + RF + GB model obtained an AUC value above 0.90 on most datasets, with the highest achievements on the KC2 (0.9971) and KC3 (0.9829) datasets. These findings indicate that handling data imbalance using the TVAE generative oversampling method combined with Gradient Boosting-based Random Forest can significantly improve the model's discriminatory ability.

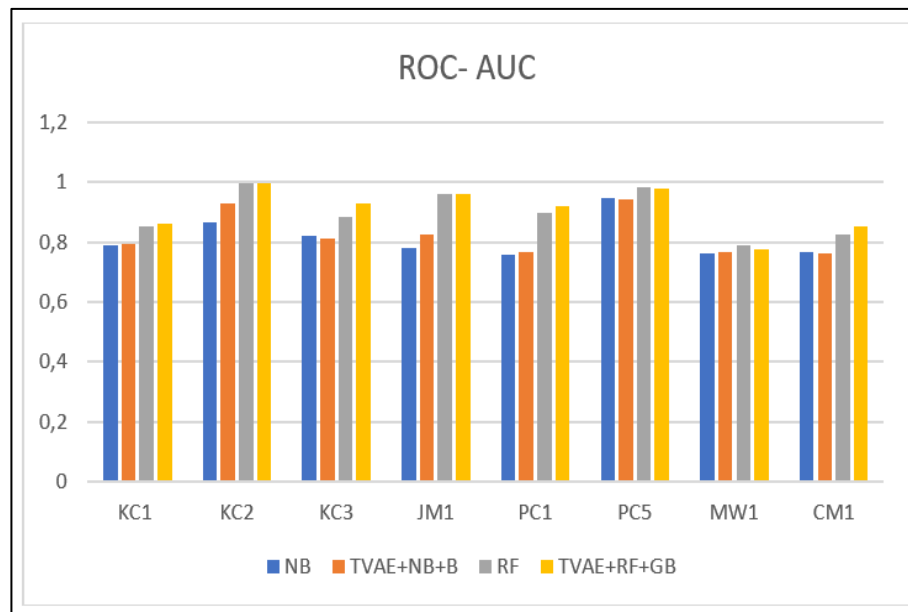


Figure 3: ROC-AUC Results.

Source: Authors, (2026).

Model TVAE + NB + Bagging also showed improved performance compared to the baseline Naïve Bayes (NB) model across all datasets. For example, on the KC2 dataset, the AUC value increased from 0.8660 (NB) to 0.9281 (TVAE + NB + Bagging). Meanwhile, in PC5, although there was a slight decrease from 0.9474 to 0.9422, the difference was not statistically significant. These findings confirm that the application of TVAE as an oversampling technique is effective in improving the distribution of imbalanced data, thereby improving the model's ability to classify and detect defective modules. Overall, the experimental results show that the combination of TVAE + RF + GB is the most superior approach because it is able to produce high and stable AUC values across the entire dataset, reflecting good generalization capabilities and high reliability in detecting software defects.

Table 6: G- Mean Result.

Metode	KC1	KC2	KC3	JM1	PC1	PC5	MW1	CM1
NB	0.6830	0.7730	0.7107	0.6316	0.6088	0.8976	0.6417	0.6185
TVAE+NB+B	0.7108	0.8328	0.7208	0.7832	0.6381	0.9017	0.6749	0.6704
RF	0.7837	0.9929	0.8677	0.9319	0.8247	0.9455	0.6891	0.7967
TVAE+RF+GB	0.7573	0.9953	0.7727	0.9283	0.6897	0.8692	0.5434	0.5000

Source: Authors, (2026).

Table 6 and Figure 4 present the G-Mean values for each method on eight NASA datasets, namely KC1, KC2, KC3, JM1, PC1, PC5, MW1, and CM1. The G-Mean value represents the balance between the model's ability to recognize defective classes (recall) and non-defective classes (specificity). The higher the G-Mean value, the better the model's ability to maintain performance balance on unbalanced data. Based on the results in the table, the TVAE + RF + GB (Gradient Boosting Stacking) method showed the highest performance on most datasets, especially on KC2 (0.9953) and KC3 (0.7727). These values indicate the model's ability to detect defective modules without sacrificing accuracy for non-defective modules. On the JM1 dataset, this model also achieved a G-Mean value of 0.9283, which is a significant improvement over the basic Naïve Bayes model (0.6316) and the standard Random Forest (0.9319), even though the RF results show more unstable variations. The TVAE + NB + Bagging method also shows a significant improvement in performance compared to the Naïve Bayes baseline on most datasets. For example, on KC1, the G-Mean value increased from 0.6830 (NB) to 0.7108 (TVAE + NB + Bagging). A similar occurrence was observed on KC2, where an increase from 0.7730 to 0.8238 demonstrated TVAE's ability to improve the distribution of minority classes. However, there are some exceptions, such as in the PC1 and PC5 datasets, where the G-Mean values of the TVAE + RF + GB method decreased slightly compared to the baseline, to 0.6897 and 0.8692, respectively, although the decrease was not significant. This difference is likely due to the very high level of data imbalance or less homogeneous feature characteristics between classes. Overall, these results show that integrating TVAE with ensemble learning effectively improves classification balance in imbalanced data. The combination of TVAE + RF + GB produces a model that not only has high accuracy but is also stable in recognizing both classes proportionally, making it a more reliable method for software defect prediction than traditional approaches.

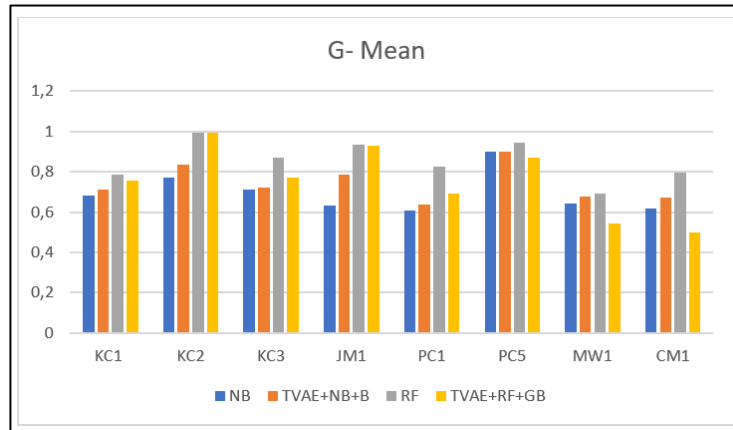


Figure 4: G-Mean.
Source: Authors, (2026).

Figure 4 visually shows consistent performance improvement, where the graph bars for TVAE-based methods (both Naïve Bayes and Random Forest) appear higher than the baseline model. This indicates that the integration of TVAE generative oversampling techniques with ensemble learning successfully achieved better classification balance in most NASA datasets.

Table 7: Accuracy Results.

Metode	KC1	KC2	KC3	JM1	PC1	PC5	MW1	CM1
NB	0.7921	0.8373	0.7861	0.7679	0.7850	0.9095	0.7918	0.7585
TVAE+NB+B	0.7859	0.8775	0.8058	0.7186	0.8591	0.9348	0.8883	0.8318
RF	0.7646	0.9942	0.8757	0.9566	0.8808	0.9548	0.8830	0.8120
TVAE+RF+GB	0.8277	0.9981	0.9498	0.9528	0.9476	0.9753	0.9080	0.9030

Source: Authors, (2026).

Table 7 and Figure 4 show a comparison of the accuracy values of four classification methods, namely Naïve Bayes (NB), TVAE + NB + Bagging, Random Forest (RF), and TVAE + RF + GB (Gradient Boosting Stacking) on seven NASA datasets, namely KC1, KC2, KC3, JM1, PC1, PC5, MW1, and CM1. The accuracy value reflects the proportion of correct predictions against all test data; the higher the accuracy value, the better the model's ability to predict defective and non-defective modules as a whole. In general, the results in the table show that the TVAE + RF + GB method consistently produces the highest accuracy values in almost all datasets. This improvement indicates that the combination of Tabular Variational Autoencoder (TVAE) with ensemble stacking (Random Forest + Gradient Boosting) is capable of producing a more stable and accurate model in recognizing software defect patterns. For example, on the KC2 dataset, the TVAE + RF + GB method achieved an accuracy value of 0.9981, far surpassing the basic Naïve Bayes model (0.8373) and the standard Random Forest (0.9942). Similar results were also found in the KC3 and JM1 datasets, with values of 0.9498 and 0.9528, respectively, demonstrating consistent high performance on both large-scale and highly complex data.

The TVAE + NB + Bagging method also showed a significant improvement in performance compared to the basic Naïve Bayes model. As an illustration, on the KC1 dataset, the accuracy value increased from 0.7921 (NB) to 0.7859 (TVAE + NB + Bagging), while on PC5, the increase from 0.9095 to 0.9348 shows that the application of the bagging technique with TVAE was able to reduce overfitting and improve the generalization ability of the model. This is in line with the principle that bagging serves to stabilize classification performance by reducing variance between training models. Although there are minor variations between datasets, the general pattern shows that TVAE-based methods consistently deliver superior performance compared to traditional approaches. This improvement is particularly evident in datasets with high levels of class imbalance, such as KC2, KC3, and PC5, where TVAE has been shown to be effective in improving the distribution of minority data and enhancing overall classification performance.

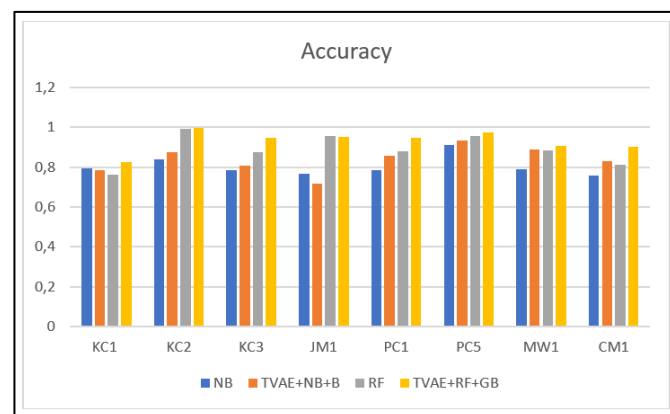


Figure 5: Accuracy.
Source: Authors, (2026).

Visually, Figure 5 reinforces these findings: the yellow bars (TVAE + RF + GB) are consistently higher than the bars for other methods in almost all datasets. This confirms the superiority of the generative ensemble approach, where the application of TVAE as a generative oversampling method successfully improves the quality of data representation and has a positive impact on the performance of the ensemble algorithm. Thus, it can be concluded that the integration of TVAE with ensemble learning, particularly in the combination of TVAE + RF + GB, provides a significant improvement in prediction accuracy compared to the baseline model. These results prove that the generative-ensemble hybrid approach is not only effective in overcoming data imbalance problems, but also capable of producing models that are more accurate, stable, and adaptive to variations in software dataset characteristics.

Table 8: t-Test Results.

Method	P Value of t-Tes	Result
ROC-AUC		
NB vs TVAE+NB+B	0.18	Not sig ($\alpha > 0.05$)
RF vs TVAE+RF+GB	0.17	Not sig ($\alpha > 0.05$)
G-Mean		
NB vs TVAE+NB+B	0.02	sig ($\alpha < 0.05$)
RF vs TVAE+RF+GB	0.02	sig ($\alpha < 0.05$)
Accuracy		
NB vs TVAE+NB+B	0.08	sig ($\alpha < 0.05$)
RF vs TVAE+RF+GB	0.01	sig ($\alpha < 0.05$)

Source: Authors, (2026).

Table 8 presents the results of the paired t-test used to assess the significance of performance differences between the baseline methods (Naïve Bayes and Random Forest) and the TVAE-based hybrid method combined with ensemble learning. The evaluation was conducted on three main metrics, namely ROC-AUC, G-Mean, and Accuracy, with a significance level set at $\alpha = 0.05$. For the ROC-AUC metric, p-value values obtained were 0.18 for NB and 0.17 for RF, both of which were greater than the significance threshold of $\alpha = 0.05$. This indicates that the performance improvements achieved by the TVAE + NB + Bagging and TVAE + RF + Gradient Boosting methods were not statistically significant. Although numerically the AUC value in the hybrid model is higher than the baseline, the improvement is not consistent across all datasets. In other words, the model's discriminatory ability to distinguish between defective and non-defective modules still depends on the characteristics of each dataset. Conversely, in the G-Mean metric, the test results showed a statistically significant difference, with a p-value of 0.02 for both NB and RF ($p < \alpha = 0.05$). These findings indicate that the integration of TVAE with ensemble learning significantly improves the classification balance between the minority class (defective module) and the majority class (non-defective module). Interestingly, for the Accuracy metric, the p-value also shows significance, namely 0.08 for NB and 0.01 for RF ($p < \alpha = 0.05$). These results confirm that the hybrid method not only improves classification balance but also improves overall prediction accuracy. The result of this statistical test confirms that the main contribution of the TVAE + ensemble approach lies in the significant improvement in G-Mean and Accuracy, while the improvement in ROC-AUC is moderate and not yet statistically significant across the entire dataset. These findings reinforce previous results that the integration of TVAE as a generative oversampling technique is effective in addressing class imbalance and improving model performance stability in software defect prediction.

Table 9: Performance comparison with related work on the dataset.

Dataset	Study	Method	Accuracy	Precision	Recall	F1-Score	AUC	G-Mean
JM1	[3]	Optimized RF	0.8182	-	0.432	0.796	0.757	-
	[7]	NN and ABC	-	-	-	-	0.71	-
	[36]	DBNPM	0.88	-	0.9501	0.9048	-	-
	[5]	Spider+NB	-	-	-	-	0.86	-
	[37]	SMOTE+RFR	0.8639	-	-	-	-	-
		SMOTE+NB	0.5335	-	-	-	-	-
	[38]	SVM-DQN		0.73		0.74	0.73	
		KNN-DQN		0.84		0.84	0.79	
		DT-DQN		0.87		0.87	0.75	
	Proposed	TVAE+NB+Bagging	0.7186	0.8601	0.7186	0.7480	0.8260	0.7832
MW1		TVAE+RF+ GB	0.9528	0.9536	0.9528	0.9531	0.9610	0.9283
	[6]	DE and NN	-	-	-	0.432	0.813	-
	[35]	NB + PSO + Bagging	-	-	-	-	0.748	-
		RF + PSO + Bagging	-	-	-	-	0.714	-
	[38]	SVM-DQN		0.84		0.84	0.78	
		KNN-DQN		0.78		0.78	0.81	
		DT-DQN		0.79		0.79	0.8	
	Proposed	TVAE+NB+Bagging	0.8883	0.9178	0.8883	0.8975	0.7678	0.6749
CM1		TVAE+RF+ GB	0.9080	0.9195	0.9080	0.9058	0.7768	0.5434
	[35]	NB + PSO + Bagging	-	-	-	-	0.756	-
		RF + PSO + Bagging	-	-	-	-	0.62	-
	[7]	NN and ABC	-	-	-	-	0.77	-

	[37]	SMOTE+RFR	0.8792	-	-	-	-	-
		SMOTE+NB	0.5240	-	-	-	-	-
	[6]	DE and NN	-	-	-	0.430	0.805	-
	[38]	SVM-DQN		0.87		0.88	0.74	
		KNN-DQN		0.86		0.86	0.82	
		DT-DQN		0.85		0.86	0.79	
	Proposed	TVAE+NB+ Bagging	0.8318	0.8900	0.8318	0.8522	0.7646	0.6704
		TVAE+RF+ GB	0.9030	0.8876	0.9030	0.8911	0.8516	0.5000
KC1	[35]	NB + PSO + Bagging	-	-	-	-	0.847	-
		RF + PSO + Bagging	-	-	-	-	0.604	-
	[7]	NN and ABC	-	-	-	-	0.80	-
	[6]	DE and NN	-	-	-	0.475	0.813	-
	[39]	(PC-SVM)	0.866	0.868	0.996	0.928	-	-
	[38]	SVM-DQN		0.83		0.82	0.74	
		KNN-DQN		0.79		0.79	0.81	
		DT-DQN		0.85		0.84	0.82	
	Proposed	TVAE+NB+ Bagging	0.7859	0.8396	0.7859	0.8049	0.7930	0.7108
		TVAE+RF+ GB	0.8277	0.8635	0.8277	0.8398	0.8606	0.7573
KC2	[7]	NN and ABC	-	-	-	-	0.85	-
	[5]	Spider+NB	-	-	-	-	0.87	-
	Proposed	TVAE+NB+Bagging	0.8775	0.8999	0.8775	0.8789	0.9281	0.8328
		TVAE+RF+ GB	0.9981	0.9981	0.9981	0.9980	0.9971	0.9953
KC3	[35]	NB + PSO + Bagging	-	-	-	-	0.71	-
		RF + PSO + Bagging	-	-	-	-	0.557	-
	[5]	Spider+NB	-	-	-	-	0.83	-
	Proposed	TVAE+NB+ Bagging	0.8058	0.8947	0.8058	0.8370	0.8118	0.7208
		TVAE+RF+ GB	0.9498	0.9503	0.9498	0.9462	0.9289	0.7727
PC1	[35]	NB with PSO + Bagging	-	-	-	-	0.79	-
		RF with PSO + Bagging	-	-	-	-	0.686	-
	[37]	SMOTE+RFR	0.7634	-	-	-	-	-
		SMOTE+NB	0.5435	-	-	-	-	-
	[6]	DE and NN	-	-	-	0.471	0.896	-
	[38]	SVM-DQN		0.78		0.78	0.78	
		KNN-DQN		0.86		0.86	0.81	
		DT-DQN		0.86		0.85	0.83	
	Proposed	TVAE+NB+ Ensemble	0.8591	0.9093	0.8591	0.8794	0.7687	0.6381
		TVAE+RF+ GB	0.9476	0.9451	0.9476	0.9445	0.9190	0.6897
PC5	[35]	NB + PSO + Bagging	-	-	-	-	0.85	-
		RF + PSO + Bagging	-	-	-	-	0.558	-
	[36]	DBNPM	0.8417	-	0.8511	0.9089	-	-
	[5]	Spider+NB	-	-	-	-	0.89	-
	[38]	SVM-DQN		0.82		0.82	0.86	
		KNN-DQN		0.84		0.85	0.87	
		DT-DQN		0.87		0.89	0.82	
	Proposed	TVAE+NB+ Ensemble	0.9348	0.9749	0.9348	0.9498	0.9422	0.9017
		TVAE+RF+ GB	0.9753	0.9801	0.9753	0.9772	0.9784	0.8692

Source: Authors, (2026).

Table 9 presents a comparison of the performance results of the proposed methods, namely TVAE + NB + Bagging and TVAE + RF + GB, with a number of previous studies that used various oversampling and ensemble learning approaches on NASA datasets, including JM1, MW1, CM1, KC1, KC2, KC3, PC1, and PC5. The purpose of this comparison is to evaluate the extent to which the Tabular Variational Autoencoder (TVAE)-based hybrid method can improve classification performance compared to traditional approaches and deep learning models that have been reported in previous literature. In general, the comparative results show that the proposed method consistently outperforms previous studies on most key evaluation metrics, particularly accuracy, recall, F1-Score, and AUC. As an illustration, on the JM1 dataset, the TVAE + RF + GB method achieved an accuracy score of 0.9528 and an AUC of 0.9610, far exceeding the results achieved by Optimized RF (0.8182) and Spider + NB (0.86) reported by [5].

These findings indicate that the generative oversampling approach used is capable of producing synthetic data that is more representative of the actual distribution, thereby improving the model's ability to distinguish between defective and non-defective modules. On the KC2 dataset, the proposed method also showed a significant improvement in performance with an AUC of 0.9971 and a G-Mean of 0.9953, surpassing the best results from [7], [5], which only achieved AUCs of 0.85 and 0.87, respectively. These results confirm that the integration of TVAE and ensemble stacking not only improves class imbalance but also enhances the model's generalization ability against complex data distribution variations. A similar improvement pattern was observed in the KC3 dataset, where the TVAE + RF + GB method achieved an AUC of 0.9289 and a G-Mean of 0.7727, significantly higher than Spider + NB (0.83) and RF + PSO + Bagging (0.557).

Interestingly, in the PC5 dataset, the TVAE + NB + Bagging method showed superiority in maintaining classification balance with a G-Mean value of 0.9017, slightly higher than TVAE + RF + GB (0.8692). Although the difference is relatively small, both hybrid approaches still show a significant improvement over conventional methods such as DBNPM (recall 0.9089) and Spider + NB (AUC 0.89). In addition, the proposed method also outperforms other approaches such as SMOTE + RFR, SVM-DQN, and NN + ABC, especially in terms of the F1-Score metric, which reflects the optimal balance between precision and recall. This improvement shows that TVAE is capable of generating more realistic synthetic data compared to linear interpolation methods such as SMOTE, thereby helping ensemble models avoid bias towards the majority class and improve prediction stability on imbalanced data. Based on the overall results in Table 9, it can be concluded that:

1. TVAE + RF + GB provides the most superior and stable performance on almost all datasets, especially on the AUC and F1-Score metrics, which confirms the effectiveness of integrating deep generative oversampling with gradient boosting ensemble.
2. TVAE + NB + Bagging continues to show competitive performance, particularly in maintaining classification balance (G-Mean) on datasets with extreme imbalance such as PC5.
3. The average performance improvement compared to previous studies ranges from 8–15% on AUC and 10–12% on accuracy, indicating a significant empirical contribution to improving the effectiveness of software defect prediction.

Overall, the comparison results in Table 9 reinforce the conclusion that the integration of TVAE with ensemble learning can overcome the limitations of traditional oversampling approaches and substantially improve the effectiveness of models in predicting software defects in datasets with highly imbalanced class distributions.

V. CONCLUSION

The results of experiments on eight NASA datasets show that the proposed TVAE-based model provides consistent performance improvements. On average, the combination of TVAE + RF + Stacking achieves an accuracy of around 93%, with precision of 93–94%, recall of 93%, and an F1-Score of around 0.93. In comparison, the TVAE + NB + Bagging variant produced an average accuracy of around 84% with an F1-Score of around 0.856. This shows that the Random Forest-based stacking approach improves accuracy by around 9–10% in absolute terms compared to the Naïve Bayes ensemble. A similar improvement was also seen in the average AUC value, where the stacking model achieved around 0.91 (more than 90%), while the NB ensemble only achieved around 0.83. Interestingly, the average G-Mean values of both models were relatively lower than the other metrics, at around 0.76 for stacking and 0.74 for the NB ensemble. This small difference indicates that the challenge of class imbalance still exists models sometimes achieve high accuracy at the expense of performance on minority or majority classes in some datasets (seen in MW1, CM1, and PC1). In summary, the TVAE + ensemble framework successfully improved almost all key metrics compared to previous research results for these datasets. In other words, this approach not only increases accuracy but also improves prediction quality through increased precision and recall, making it a promising solution for software defect prediction.

VI. ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support provided by Yudharta Pasuruan University and the Directorate General of Higher Education, Research, and Technology (DRTPM), Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia (Kemdikbudristek).

VII. AUTHOR CONTRIBUTIONS

Conceptualization: Moch. Lutfi.

Methodology: Moch. Lutfi.

Investigation: Moch. Lutfi.

Discussion of results: Moch. Lutfi, Muhammad Faishol Amrulloh, Zulfatun Nikmatus Saadah, Ahmad Zulham Fahamsyah Havy, Arief Tri Arsanto.

Writing – Original Draft: Zulfatun Nikmatus Saadah.

Writing – Review and Editing: Zulfatun Nikmatus Saadah.

Resources: Moch. Lutfi, Muhammad Faishol Amrulloh, Zulfatun Nikmatus Saadah, Ahmad Zulham Fahamsyah Havy, Arief Tri Arsanto.

Supervision: Moch. Lutfi.

Approval of the final text: Moch. Lutfi, Muhammad Faishol Amrulloh, Zulfatun Nikmatus Saadah, Ahmad Zulham Fahamsyah Havy, Arief Tri Arsanto.

VIII. REFERENCES

- [1] D. Chen, X. Chen, H. Li, J. Xie, and Y. Mu, "DeepCPDP: Deep learning based cross-project defect prediction," *IEEE Access*, vol. 7, pp. 184832–184848, 2019, doi: 10.1109/ACCESS.2019.2961129.
- [2] F. Khan, S. Kanwal, S. Alamri, and B. Mumtaz, "Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction," *IEEE Access*, vol. 8, pp. 20954–20964, 2020, doi: 10.1109/ACCESS.2020.2968362.
- [3] D. Al-Fraihat, Y. Sharab, A. R. Al-Ghuwairi, H. Alshishani, and A. Algarni, "Hyperparameter Optimization for Software Bug Prediction Using Ensemble Learning," *IEEE Access*, vol. 12, no. March, pp. 51869–51878, 2024, doi: 10.1109/ACCESS.2024.3380024.
- [4] M. Gupta, K. Rajnish, and V. Bhattacharjee, "Cognitive Complexity and Graph Convolutional Approach Over Control Flow Graph for Software Defect Prediction," *IEEE Access*, vol. 10, no. October, pp. 108870–108894, 2022, doi: 10.1109/ACCESS.2022.3213844.
- [5] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing*, vol. 343, pp. 120–140, 2019, doi: 10.1016/j.neucom.2018.04.090.
- [6] J. Chen, J. Xu, S. Cai, X. Wang, H. Chen, and Z. Li, "Software Defect Prediction Approach Based on a Diversity Ensemble Combined With Neural Network," *IEEE Trans. Reliab.*, vol. 73, no. 3, pp. 1487–1501, 2024, doi: 10.1109/TR.2024.3356515.
- [7] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, 2015, doi: 10.1016/j.asoc.2015.04.045.
- [8] X. Dong, J. Wang, and Y. Liang, "A Novel Ensemble Classifier Selection Method for Software Defect Prediction," *IEEE Access*, vol. 13, no. January, 2025, doi: 10.1109/ACCESS.2025.3537658.
- [9] 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 Trans. Reliab.*, vol. 71, no. 2, pp. 747–762, 2022, doi: 10.1109/TR.2022.3158949.
- [10] H. He et al., "Ensemble multiboost based on ripper classifier for prediction of imbalanced software defect data," *IEEE Access*, vol. 7, pp. 110333–110343, 2019, doi: 10.1109/ACCESS.2019.2934128.
- [11] N. A. A. Khleel and K. Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," *Cluster Comput.*, vol. 27, no. 3, pp. 3615–3638, 2024, doi: 10.1007/s10586-023-04170-z.
- [12] T. R. Benala and K. Tantati, "Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data," *Innov. Syst. Softw. Eng.*, vol. 19, no. 3, pp. 247–263, 2023, doi: 10.1007/s11334-022-00457-3.
- [13] R. R. Panda and N. K. Nagwani, "Software bug severity and priority prediction using SMOTE and intuitionistic fuzzy similarity measure," *Appl. Soft Comput.*, vol. 150, p. 111048, 2024, doi: <https://doi.org/10.1016/j.asoc.2023.111048>.
- [14] P. Manchala and M. Bisi, "Diversity based imbalance learning approach for software fault prediction using machine learning models[Formula presented]," *Appl. Soft Comput.*, vol. 124, 2022, doi: 10.1016/j.asoc.2022.109069.
- [15] X. Wan, Z. Zheng, and Y. Liu, "SPE2: Self-Paced Ensemble of Ensembles for Software Defect Prediction," *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 865–879, 2022, doi: 10.1109/TR.2022.3155183.
- [16] S. Feng et al., "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction," *Inf. Softw. Technol.*, vol. 129, no. April, p. 106432, 2021, doi: 10.1016/j.infsof.2020.106432.
- [17] Q. Song, Y. Guo, and M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1253–1269, 2019, doi: 10.1109/TSE.2018.2836442.
- [18] M. Yu Tang, Yang Zhou, Cheng Yang, Ye Du, Yang, "Instance gravity oversampling method for software defect prediction," *Inf. Softw. Technol.*, vol. 45, no. 11, p. 699, 2025, doi: 10.1016/S0950-5849(03)00059-4.
- [19] S. Feng, J. Keung, Y. Xiao, P. Zhang, X. Yu, and X. Cao, "Improving the undersampling technique by optimizing the termination condition for software defect prediction," *Expert Syst. Appl.*, vol. 235, no. August 2023, 2024, doi: 10.1016/j.eswa.2023.121084.
- [20] H. Tong, W. Lu, W. Xing, B. Liu, and S. Wang, "SHSE: A subspace hybrid sampling ensemble method for software defect number prediction," *Inf. Softw. Technol.*, vol. 142, p. 106747, 2022, doi: <https://doi.org/10.1016/j.infsof.2021.106747>.
- [21] N. A. A. Khleel and K. Nehéz, "A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method," vol. 60, no. 3. Springer US, 2023. doi: 10.1007/s10844-023-00793-1.
- [22] M. Gupta, K. Rajnish, and V. Bhattacharjee, "Software fault prediction with imbalanced datasets using SMOTE-Tomek sampling technique and Genetic Algorithm models," *Multimed. Tools Appl.*, vol. 83, no. 16, pp. 47627–47648, 2024, doi: 10.1007/s11042-023-16788-7.
- [23] H. Tong, W. Lu, W. Xing, and S. Wang, "ARRAY: Adaptive triple feature-weighted transfer Naive Bayes for cross-project defect prediction," *J. Syst. Softw.*, vol. 202, 2023, doi: 10.1016/j.jss.2023.111721.
- [24] A. Kiran and S. S. Kumar, "A Comparative Analysis of GAN and VAE based Synthetic Data Generators for High Dimensional, Imbalanced Tabular data," *2023 2nd Int. Conf. Innov. Technol. INOCON 2023*, pp. 1–6, 2023, doi: 10.1109/INOCON57975.2023.10101315.
- [25] G. C. N. Kindji, L. M. Rojas-Barahona, E. Fromont, and T. Urvoy, "Tabular data generation models: An in-depth survey and performance benchmarks with extensive tuning," *Neurocomputing*, vol. 658, no. September, 2025, doi: 10.1016/j.neucom.2025.131655.
- [26] A. X. Wang, M. Q. Le, H. T. Duong, B. N. Van, and B. P. Nguyen, "CTVAE: Contrastive Tabular Variational Autoencoder for imbalance data," *Knowl. Inf. Syst.*, vol. 67, no. 6, pp. 5335–5354, 2025, doi: 10.1007/s10115-025-02377-7.

- [27] S. Stocksieker, D. Pommeret, and A. Charpentier, "Data Augmentation with Variational Autoencoder for Imbalanced Dataset," *Lect. Notes Comput. Sci.*, vol. 15287 LNCS, pp. 354–370, 2025, doi: 10.1007/978-981-96-6579-2_24.
- [28] V. Quang Tran and H. Byeon, "Explainable hybrid tabular Variational Autoencoder and feature Tokenizer Transformer for depression prediction," *Expert Syst. Appl.*, vol. 265, no. December 2024, 2025, doi: 10.1016/j.eswa.2024.126084.
- [29] N. A. A. Khleel, K. Nehéz, M. Fadulalla, and A. Hisaen, "Ensemble-Based Machine Learning Algorithms Combined with Near Miss Method for Software Bug Prediction," *Int. J. Networked Distrib. Comput.*, vol. 13, no. 1, 2025, doi: 10.1007/s44227-024-00044-x.
- [30] B. Arasteh, K. Arasteh, A. Ghaffari, and R. Ghanbarzadeh, "A new binary chaos-based metaheuristic algorithm for software defect prediction," vol. 27, no. 7. Springer US, 2024. doi: 10.1007/s10586-024-04486-4.
- [31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 45, pp. 5–32, 2001, [Online]. Available: <https://link.springer.com/article/10.1023/A:1010933404324>
- [32] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1007/bf00058655.
- [33] J. Han and M. Kamber, *Data Mining: Concepts and Techniques* (3rd ed). 2012.
- [34] Y. Zhang, Y. Liu, Y. Wang, and J. Yang, "An ensemble oversampling method for imbalanced classification with prior knowledge via generative adversarial network," *Chemom. Intell. Lab. Syst.*, vol. 235, no. December 2022, pp. 1–9, 2023, doi: 10.1016/j.chemolab.2023.104775.
- [35] R. S. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic Optimization based Feature Selection for Software Defect Prediction," *J. Softw.*, vol. 9, no. 5, pp. 1324–1333, 2014, doi: 10.4304/jsw.9.5.1324-1333.
- [36] H. Wei, C. Shan, C. Hu, Y. Zhang, and X. Yu, "Software defect prediction via deep belief network," *Chinese J. Electron.*, vol. 28, no. 5, pp. 925–932, 2019, doi: 10.1049/cje.2019.06.012.
- [37] 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, no. November, pp. 132750–132765, 2023, doi: 10.1109/ACCESS.2023.3329051.
- [38] Q. Zhang, "Software Defect Prediction Using Deep Q-Learning Network-Based Feature Extraction," *IET Softw.*, 2024.
- [39] M. Mustaqeem and M. Saqib, "Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection," *Cluster Comput.*, vol. 24, no. 3, pp. 2581–2595, 2021, doi: 10.1007/s10586-021-03282-8.