

Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models

Md. Ariful Islam Arif¹, Md Hasibul Islam¹, Hosney Jahan², , Md Akhtaruzzaman^{3,4}, Md Mahbubur Rahman¹

1. Department of Computer Science and Engineering, Military Institute of Science and Technology, Dhaka, BGD

2. Department of Computer Science and Engineering, East West University, Dhaka, BGD

3. Department of Robotics and Mechatronics Engineering, Daffodil International University, Dhaka, BGD

4. Research and Development, Quantum Robotics and Automation Research Group (QRARG), Dhaka, BGD

Received: March 07, 2026 | Review began: May 11, 2026 | Review ended: June 08, 2026 | Published: June 16, 2026

© Copyright 2026

This is an open access article distributed under the terms of the Creative Commons Attribution License CC-BY 4.0., which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

Software defect prediction is a crucial aspect of software quality assurance that facilitates early detection and prevention of potential defects, minimizing the cost and adverse consequences associated with software failures. However, it always suffers from the class imbalance problem, which makes it difficult for the model to generalize from minority defect instances. As a result, models may become biased toward the majority class. In this paper, we propose a software defect prediction framework that integrates generative adversarial network-based data augmentation with semi-supervised machine learning (SSML) techniques. To predict software defects, both traditional machine learning and SSML algorithms are applied in this study. The implemented techniques include k-nearest neighbors, support vector machine, decision tree, random forest, label propagation, and artificial neural network, with performance evaluated using accuracy, precision, recall, and F1-score. A comparative analysis has been performed between conventional machine learning and SSML approaches based on six benchmark datasets: CM1, Linux-driver net, Eclipse 3.0, JM1, KC1, and Prop-1. The results show that random forest achieved the highest accuracy (96.67%) and F1-score (94.38%) on the Linux-driver net dataset. These findings confirm the potential of SSML models combined with tabular generative adversarial network-based augmentation in enhancing defect prediction performance.

Categories: Software Quality Assurance (SQA), Software Testing, Machine Learning (ML)

Keywords: tgan, semi-supervised machine learning, software defect, data augmentation, prediction model

Introduction

Software faults and defects are critical issues that software developers and engineers must address to ensure the reliability, functionality and usability of their applications. A software fault or defect can arise from mistakes in the code, design flaws, or issues in the software requirements. They can lead to system crashes, data corruption, or security vulnerabilities, impacting the software's reliability and performance. It is imperative to discover and correct such errors so that the software functions accordingly and meets the expectations of users [1]. It is beneficial for software companies and engineering teams to detect software errors early because it saves cost and time by avoiding expensive repairs and extended rework. Predictive analysis is a significant tool in modern industries, analyzing trends and patterns in data to facilitate proactive decision-making [2]. With the integration of artificial intelligence (AI) and machine learning (ML) into software programs, workflows are optimized, tasks are automated and decision-making is improved [3]. All of these contribute to enhancing the industry.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

ML model development for software defect prediction (SDP) often uses supervised and unsupervised learning approaches. While labeled historical data is used in supervised learning to train models for predicting defects based on patterns, unsupervised learning can be used to identify unknown patterns and outliers without the availability of known labels, providing valuable knowledge of new kinds of defects. Sources of data for SDP are code metrics, change history and defect logs. The common problem in ML for defect prediction is the class imbalance problem. This is a case where there is a large number of samples of non-defective software compared to defective software. It becomes challenging for the model to generalize from minority defect examples [4]. This may lead to majority-biased models, which require the use of data augmentation techniques for consistent defect prediction. To overcome the issue of imbalanced datasets in SDP, several data generation approaches are used. The approaches include synthetic minority over sampling (SMOTE) [5], generative adversarial network (GAN) [5] and variational autoencoder (VAE) [6] that can create synthetic samples for the minority class. Data augmentation and creating realistic situations of defects can also enhance the accuracy of the model. Although many supervised ML models have been applied in SDP, the potential of applying GAN and semi-supervised machine learning (SSML) together in this field is yet to be explored.

This study proposes a framework for SDP by integrating GAN-based augmentation with SSML. It aims to investigate the performance of semi-supervised learning for SDP. In the proposed framework, tabular GAN (TGAN) is used to augment the training dataset and make it more balanced. This study uses multiple publicly available software defect datasets collected from different repositories, i.e., Prop-1, KC1, JM1, CM1, Eclipse 3.0 and Linux Driver Net. Firstly, a number of ML algorithms, Random forest (RF), support vector machine (SVM), K-nearest neighbors (KNN), Naive bayes (NB), decision tree (DT) and artificial neural networks (ANN), were used as classifiers. For semi-supervised learning, these models were integrated with the label propagation (LP) approach. In the supervised setting, fully labeled datasets were used for model training. In contrast, for semi-supervised learning, a small portion of the labeled data was intentionally withheld and treated as unlabeled. This enables the models to iteratively learn from both the original labeled data and the LP-generated pseudo-labels during training. Finally, the results of both the ML and SSML classifiers are assessed using various evaluation metrics, i.e., accuracy, precision, recall and F1-score. The contributions of this research are described below:

- a) A framework for efficient SDP by incorporating TGAN data augmentation method with SSML.
- b) Extensive ablation studies evaluating the impact of SSML algorithms compared to supervised ML algorithms across several benchmark datasets.
- c) A comprehensive performance comparison with state-of-the-art approaches, denoting the effectiveness of the proposed framework.

Materials And Methods

Background study

SDP is used to predict potential defects during software development. By finding problems early, debugging becomes less costly and helps developers to improve the overall quality and reliability of the software. Predictive models are trained with both labeled datasets where the defects are known, and also with unlabeled datasets. This approach also allows proper resource allocation and helps in reducing the post deployment defects. Figure 7 shows the systematic diagram for SDP. It takes input queries and returns predictions through an expert system. This diagram for SDP begins with collecting user feedback and system performance data, which is passed through a data pipeline. This data is processed by an inference engine that interacts with a pretrained expert system and a prediction model to identify potential defects. The insights are reserved in knowledge base and used to collect feedback, supporting the continuous learning procedure and improvement of the performance.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. *Cureus J Comput Sci* 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

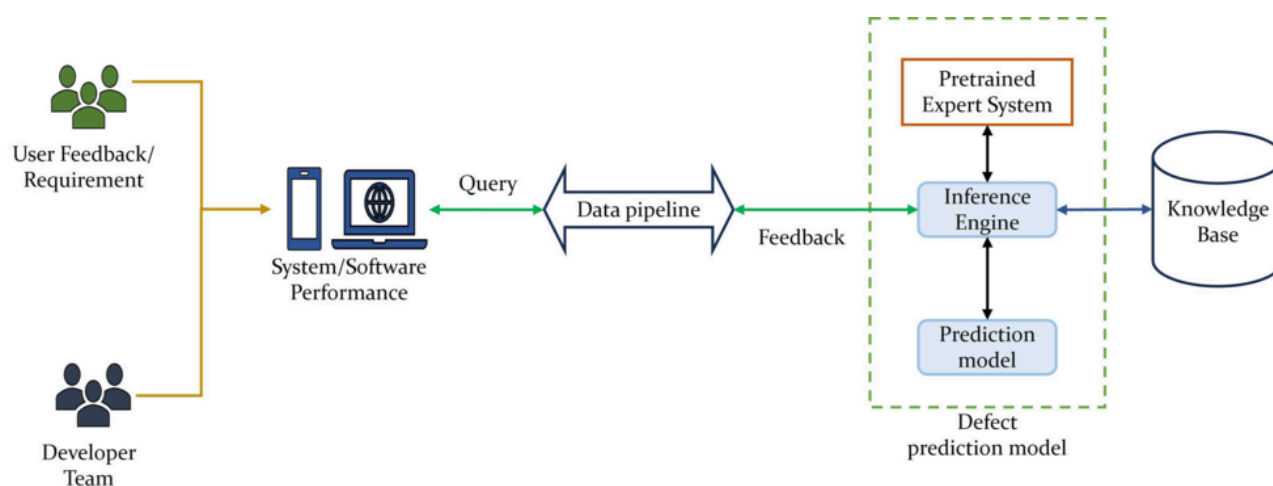


FIGURE 1: Systematic diagram for software defect prediction

It explores previous research work on SDP and highlights the methodologies applied for supervised and semi-supervised learning models. Also, current research work on SDP has been discussed with distinct techniques in this section. Jain et al. [7] reviewed the use of generative models in heterogeneous software systems for defect prediction in software environments. They focused on the transformation of generative models for software quality assurance and addressing challenges of heterogeneous software architecture. GAN was used to detect defects at the code change level. This method achieved superior performance in detecting semantically preserved changes on diverse project datasets, demonstrating its usefulness for SDP [8]. Song et al. [9] introduce DP-GANPT as a new model that integrates GANs with pre-trained code models to predict software defects. They used a semi-supervised learning model and bi-model code prompt input representation to address data limitations. DP-GANPT was shown to surpass existing baselines in within-project and cross-project defect prediction on the PROMISE dataset. To handle the issue of data imbalance, Kumar and Venkatesan [10] used GANs to generate synthetic data for improving the prediction accuracy of defective modules. They demonstrated marginally significant improvements in classification metrics with precision, recall and F1-score. Their score indicated the effectiveness of GAN-based data augmentation techniques for SDP. Zou and Wang [11] worked on research on three-stage cross-project defect prediction framework. The aim was to mitigate insufficient feature representation and inadequate alignment of data distributions between source and target projects. Eivazpour and Keyvanpour [12] proposed an approach using VAE for generating synthetic samples of minority classes. By augmenting the training data with synthetic examples, their method significantly boosts model accuracy as demonstrated on a number of datasets from the PROMISE repository.

Hu et al. [13] proposed a federated oversampling learning framework for imbalanced SDP with privacy preservation. They used PROMISE and NASA repositories datasets. VAE methodologies have worked well in improving the prediction of software faults on NASA datasets [14]. The efficiency of these techniques was determined by the quantity and quality of data availability. Employing GANs for data preprocessing and balancing in SDP has recorded improvements in model performance. SeC-GAN has been shown to outperform standard classifiers in SDP and showed utility in the unbalanced dataset [15]. Another research work comparing results with PROMISE data has revealed that the predictive accuracy varied from task to task. It showed the improvement with GAN-generated data in software engineering. These articles typically quantify performance metrics using Area Under the Curve (AUC). It illustrated the performance of GAN with conventional methods such as SMOTE and ADASYN [16]. Authors have explored GANs, SMOTE and AdaBoost to solve data imbalance in SDP. Matthews Correlation Coefficient (MCC) and the classification accuracy were calculated for software fault prediction [17]. This study highlights the ongoing evolution towards more robust predictive models in

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. *Cureus J Comput Sci* 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

software engineering. Extended RF methodologies were applied with GANs for software fault prediction, particularly in datasets such as Eclipse [18]. These studies advocate hybrid approaches that leverage both supervised and semi-supervised learning paradigms to enhance predictive performance.

Zhang et al. [19] explored LP-based semi-supervised model for SDP. They applied Laplacian score sampling technology for labeled defect-free components. Sun et al. [20] contribute a comparative study of VAE and GAN approaches in software fault prediction. Their analysis provides insights about the strengths and limitations of each method. They were working on handling dataset imbalances and improved prediction accuracy. In predicting software aging-related bugs (ARBs), GAN-based approaches have demonstrated significant advancements by generating synthetic data from imbalanced datasets, thereby enhancing prediction accuracy in long-running systems [21]. Similarly, optimizations using Bayesian hyperparameter techniques have underscored the effectiveness of oversampling strategies in improving recall rates for security vulnerability prediction models [22]. Other studies have introduced novel GAN-based generative oversampling methods to balance fault and non-fault modules in software fault prediction tasks [23]. Islam et al. [24] proposed a two-layer stacking ensemble model for SDP. They applied SMOTE and used multiple datasets of Apache corporation. Another study utilized CatBoost gradient boosting algorithm on SDP [25]. They integrated Local Interpretable Model-agnostic Explanations, SHapley Additive Explanations and Explainable Artificial Intelligence to enhance model transparency with JM1 dataset. These methods have shown superior performance in diverse evaluation metrics, including precision, recall, F1-score and AUC, signaling their potential to address inherent challenges in class imbalance.

The previous research highlights the use of GANs and VAEs in improving SDP through semi-supervised learning and bi-modal input representations. They have explored the use of generative models like SMOTE, GAN and VAE for improving prediction accuracy. There is limited focus on utilizing TGAN for synthetic data generation to balance class distributions. Data augmentation across diverse SDP datasets may increase model performance. Some studies depended on labeled data and did not use the semi-supervised learning algorithms to enhance performance by incorporating both labeled and unlabeled data. This research bridges these gaps by applying TGAN and semi-supervised approaches to enhance defect prediction accuracy and improve model robustness with limited labeled data.

Proposed methodology

This portion presents a detailed outline of the methodology used in the proposed SDP model, incorporating TGAN-generated data and a semi-supervised learning approach. It describes each step in the data processing workflow, including essential preprocessing techniques such as data cleaning and normalization. Figure 2 illustrates the SDP process. It begins with data collection, where various types of datasets are gathered from different repositories. Data preprocessing is then applied to ensure consistency and eliminate anomalies or irrelevant data in the dataset. To address class imbalance, data augmentation is performed using TGAN, which helps enhance the representation of minority classes. Both ML and SSML algorithms are applied to each dataset to analyze the data. Evaluation metrics such as accuracy, precision, recall, and F1-score are used to assess model performance.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. *Cureus J Comput Sci* 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

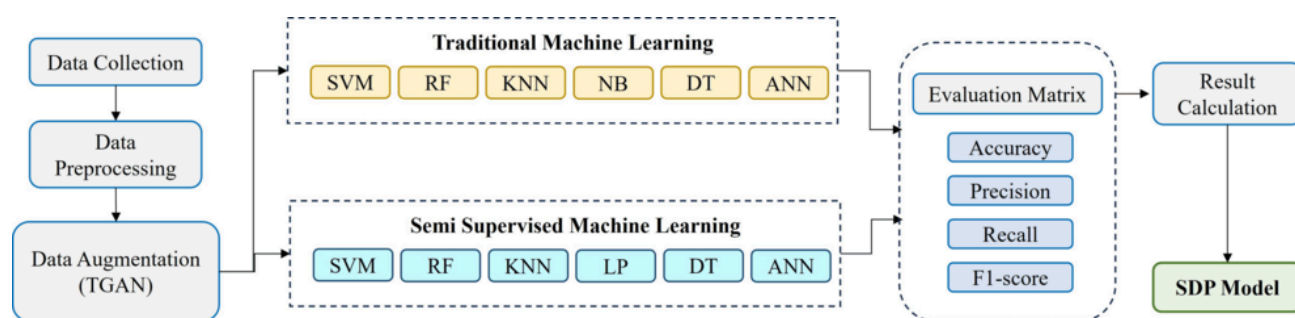


FIGURE 2: Methodology of the proposed SSML-based model for SDP

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; NB, Naive Bayes; RF, Random Forest; SDP, Software Defect Prediction; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine; TGAN, Tabular Generative Adversarial Network

Data collection

SDP datasets are essential for identifying bugs and defects in software modules. These datasets have labeled code modules with different features to help train models for defect prediction. Using different datasets helps the model work well on various software projects. Table 1 provides an overview of the six datasets collected for the SDP model. The datasets originate from various repositories, including PROMISE [26], ARB [27] and Eclipse [28]. This table contains dataset name, repository source, size and the number of features.

Serial No.	Dataset	Repository	Size	No. of Features
1	Prop-1	PROMISE	6,510	22
2	KC1	PROMISE	998	22
3	JM1	PROMISE	6,893	22
4	Eclipse_3.0	Eclipse	10,593	202
5	Linux driver net	ARB	2,307	83
6	CM1	PROMISE	703	22

TABLE 1: Overview of datasets used for the software defect prediction model

Data preprocessing

To maintain the quality and consistency of any dataset, several preprocessing techniques have been applied, which help the model to achieve better performance. The preprocessing transformed the unstructured data into a clean and structured format. Afterwards, the formatted data are fed to the ML models for further processing. The preprocessing steps are checking for null values, replacing null values with NaN, sorting special characters, label encoding and outliers detection. Outliers are found and removed using interquartile range methodology. It shows the values that fall outside

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

from the normal range. Figure 3 illustrates this for a single feature, where extreme values beyond the upper and lower limits are identified and excluded. A similar method is applied in each dataset to ensure the cleanliness and reliability of data.

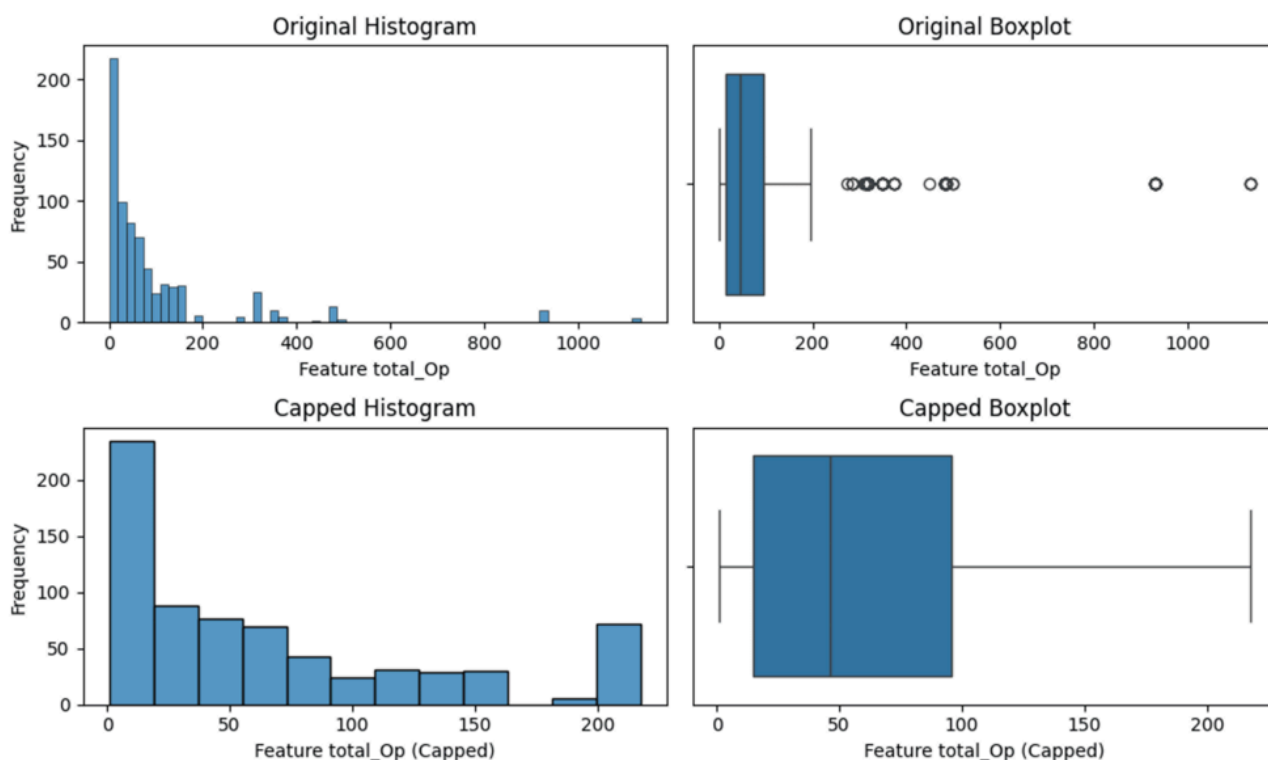


FIGURE 3: Box plot showing outlier detection and removal for a selected feature (total_op)

Data augmentation

In this research, TGAN has been used to generate synthetic tabular data. Figure 4 describes the architecture of a TGAN. It consists of two elements: generator and discriminator. The generator generates synthetic data from noise vectors and the discriminator labels the data as real or fake. The two networks are trained simultaneously and automated via backpropagation. The generator's task is to produce realistic samples such that its loss is minimized. The discriminator, on the other hand, learns how to tell apart real from fake samples.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. *Cureus J Comput Sci* 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

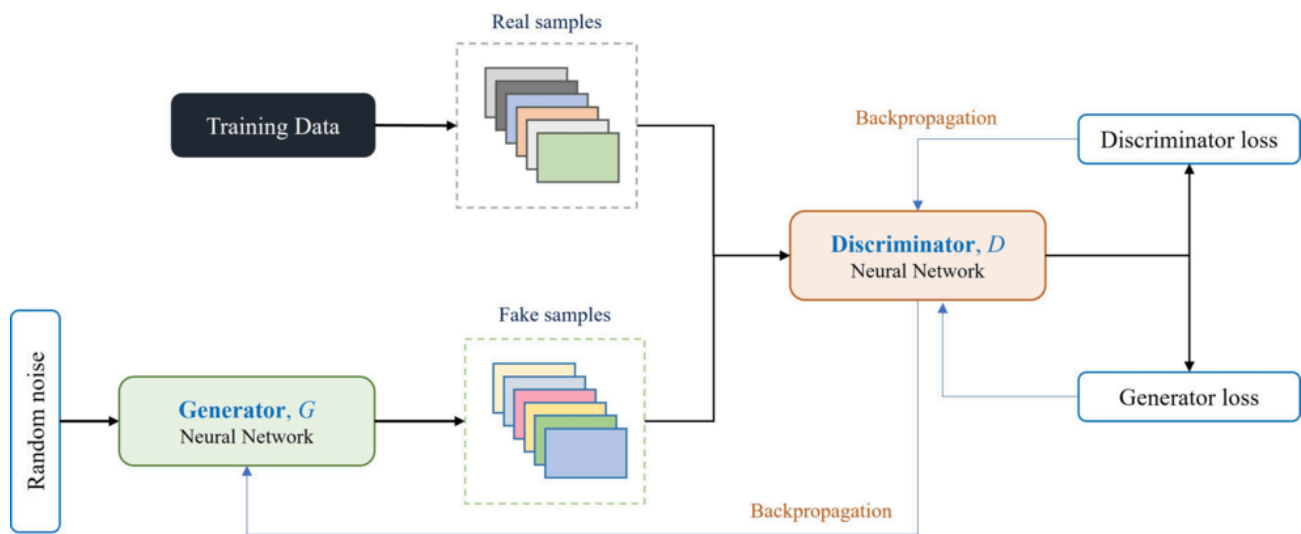


FIGURE 4: The overview of TGAN data generation process

TGAN, Tabular Generative Adversarial Network

The data generation process is used to handle data imbalance problems in different datasets. It learns from the distribution of real data and generates synthetic data containing similar statistical properties. TGAN works by training two neural networks: generator and discriminator [29]. The equation of TGAN training represents the process between generator (G) and discriminator (D). Mathematically, this can be expressed as follows:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim P_r(x)} [\log D(x)] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - DGz)]$$

The discriminator (D) maximizes the probability of correctly classifying real data x as real and generated data $G(z)$ as fake. The generator (G) aims to minimize the discriminator's ability to distinguish synthetic data from real data, by generating data $G(z)$ that appears as real as possible. Here, x is a real data sample, z is random noise and $G(z)$ is the generated output. $E_{x \sim P_r(x)}$ represents the expectation (average) over all real data samples x that are drawn from the real data distribution $P_r(x)$. Similarly, $E_{z \sim P_z(z)}$ represents the expectation over all noise vectors z sampled from a noise distribution $P_z(z)$. TGAN uses adversarial training to generate realistic synthetic tabular data. This helps improve ML models by adding more training samples, especially in cases like SDP where real data is limited [30].

Table 2 presents a detailed overview of the key parameters used in TGAN. It also outlines the process of generating balanced synthetic tabular data to address the issue of class imbalance. Initially, it started with few data in each set. In the TGAN model, it learns the patterns and connections among different features and data points. After training, TGAN produces artificial data to balance the counts of minority class with majority class. The produced synthetic data is subsequently mixed with the original dataset and labeled. In the balanced dataset, the ratio of 'YES' to 'NO' labels is very close.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Parameters	Values
Latent Dimension	100
Output Dimension	Data.shape [1]
Generator Layer 1 Units	128
Generator Layer 2 Units	64
Generator Activation	LeakyReLU (alpha=0.2)
Generator Output Activation	Tanh
Discriminator Layer 1 Units	128
Discriminator Layer 2 Units	64
Discriminator Activation	LeakyReLU (alpha=0.2)
Discriminator Output Activation	Sigmoid
Discriminator Loss Function	Binary cross-entropy
Generator Loss Function	Binary cross-entropy
Optimizer	Adam (lr=0.002, $\beta_1=0.5$)
Noise Distribution	Normal (mean=0, std=1)

TABLE 2: Key parameters and values for the TGAN architecture

TGAN, Tabular Generative Adversarial Network

Table [3](#) shows the generated data, declaring the true and false properties of software defects. It highlights an overview of the datasets used for the SDP model both before and after TGAN was applied. It shows the number of data instances for each dataset and the number of ‘YES’ and ‘NO’ software defect values. The table highlights the distribution of defects across six different datasets, reflecting the balance between true and false classifications.

Serial No.	Dataset	Before TGAN	After TGAN	Yes	No
1	cm1	703	1,353	649	704
2	Linux driver net	2,307	4,357	2,220	2,137
3	eclipse_3.0	10,593	20,653	10,946	9,707
4	jm1	6,893	13,493	6,881	6,612
5	kc1	998	1,898	873	1,025
6	prop-1	6,510	13,010	6,114	6,896

TABLE 3: Summary of data distribution before and after applying TGAN

TGAN, Tabular Generative Adversarial Network

Figure 5 demonstrates the comparison of original data distribution and synthetic data generated using TGAN for data augmentation. The original data exhibits a clustered distribution with clear patterns. The synthetic data appears uniformly spread within a normalized range between -1 and 1. This highlights how TGAN effectively generates new samples and the distribution may differ significantly from the original dataset.

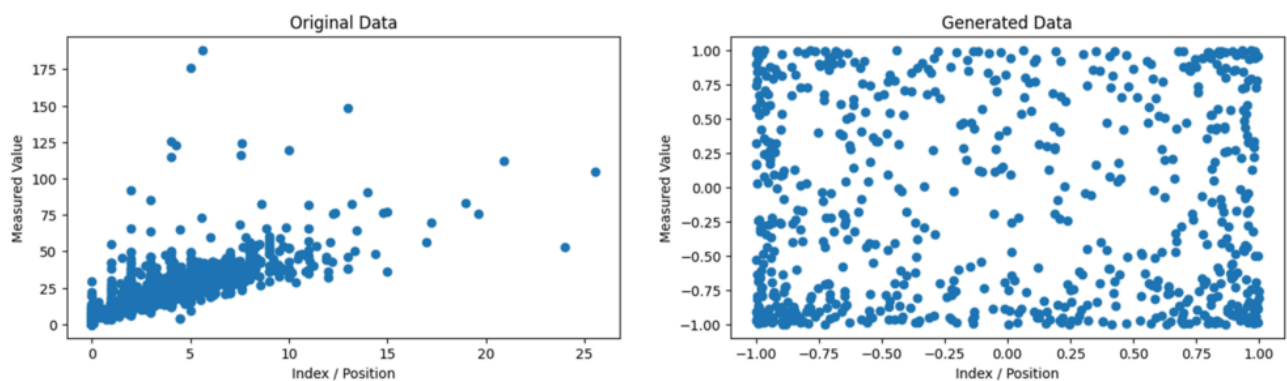


FIGURE 5: Comparison of original data distribution and synthetic data generated using TGAN for data augmentation

TGAN, Tabular Generative Adversarial Network

Figure 6 displays the correlation coefficient matrices of the 'KC1' dataset. It shows the difference between synthetic data generated with TGAN and original data. The heatmaps illustrate relationships between variables, where darker shades represent stronger correlations. The synthetic data's correlation patterns differ from the original. The relationships between features have changed while TGAN captures some structural similarities. Coefficient matrix represents the pairwise correlation between variables in a dataset, where values range from -1 (perfect negative correlation) to +1 (perfect positive correlation). It is used to identify the strength and direction of linear relationships between variables.

How to cite this article:

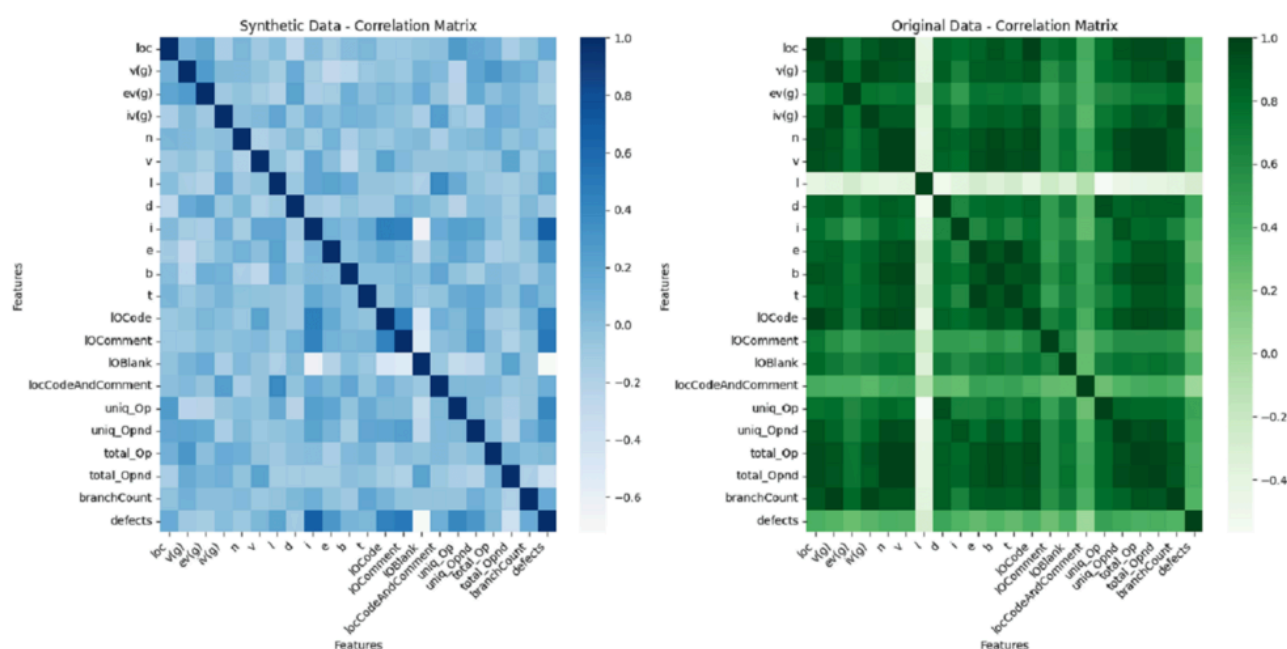


FIGURE 6: Correlation coefficient matrices of original and TGAN-generated synthetic data for the "KC1" dataset

TGAN, Tabular Generative Adversarial Network

ML and SSML classifiers

In this research, ML algorithms were first applied and after that SSML algorithms were used. Some algorithms were common in both cases. ML uses only labeled data to train the model, while SSML uses both labeled and a large amount of unlabeled data. This makes SSML more useful when labeled data is limited, as it can still learn from the unlabeled part. SSML is highly valuable in SDP because labeled defect data is often limited, while unlabeled data is abundant. SSML helps leverage this unlabeled data to enhance prediction models, leading to more accurate and early defect detection. This results in cost savings and improved software quality by identifying issues during the development phase.

ML algorithms include RF, SVM, KNN, NB, DT, and ANN

SSML algorithms include RF, SVM, KNN, LP, DT, and ANN

- **RF:** RF is an ensemble learning method that creates multiple decision trees and merges their outputs to improve accuracy and reduce overfitting. It is effective in both classification and regression tasks and handles large datasets with higher dimensionality. The RF equation is shown below.

$$\hat{y}_{\text{RF}} = \text{mode}(\{\hat{y}_{\text{tree}_1}, \hat{y}_{\text{tree}_2}, \dots, \hat{y}_{\text{tree}_N}\})$$

Here, it combines predictions y_{tree_j} from decision trees N [31].

- **SVM:** SVM is a classification algorithm that finds the best hyperplane separating data into different classes. It maximizes the margin between support vectors, making it ideal for high dimensional spaces and binary classification tasks. The hyperplane of SVM is expressed below:

$$\min_{(w,b)} \frac{1}{2} \|w\|^2 \text{ where } y_i(wx_i + b) \geq 1$$

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Here, w represents weight vector and b represents bias [32]. It finds the optimal hyperplane that maximizes the margin between $y_i = \pm 1$.

- **KNN:** KNN is a simple, non-parametric algorithm that classifies data points by looking at the 'k' nearest data points in the feature space. It is commonly used for classification and regression, relying heavily on distance metrics. The classification rule of voting for KNN is shown below:

$$\hat{y}_{\text{test}} = \text{mode}(\{y_i : x_i \in N_k(x_{\text{test}})\})$$

where, $N_k(x_{\text{test}})$ is the set of k nearest neighbors to x_{test} [32].

- **NB:** NB is a probabilistic classifier. It works based on Bayes' theorem. It assumes independence between features. It produces good results with large datasets. It also works fast and efficiently in tabular data by making probability computation simple. The basic equation for Bayes' theorem is as follows:

$$P(y | X) = \frac{P(X | y)P(y)}{P(X)}$$

where, X is the feature vector and y is class label [32].

- **DT:** DT generally splits the data into several nodes based on features. It builds a tree for making decisions. By using pruning, the overfitting issue is mitigated. At this time, the model becomes easier to interpret. The information gain calculated with entropy is shown below.

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Here, to achieve maximum information gain, it recursively splits dataset S on feature A [32].

- **ANN:** ANN models are built following the human brain. It has interconnections among the layers of neurons. Those neurons are very flexible and powerful for solving complex calculations. They are very efficient in classification and detection models.

$$a^{(l)} = \sigma(W^l a^{(l-1)} + b^l)$$

Here, for layer l , activation $a^{(l)}$ is calculated by applying a non-linear activation function σ with the previous layer $a^{(l-1)}$ [32].

- **LP:** LP is a graph-based SSML classifier. In this algorithm, labels are propagated from labeled to unlabeled data. It maintains a similar network for each data point. The data points are represented in a graph structure and perform well.

$$F^{(t+1)} = \alpha S F^{(t)} + (1 - \alpha) Y$$

Here, propagate labels F in a graph over time (t) using similarity matrix S . It blends with initial labels Y with clamping factor α [33].

Figure 7 describes the algorithm of SSML procedure followed in this study. This algorithm has been prepared by the authors at their own initiative for the proposed SSML model. The dataset was partitioned into two segments: labeled portion and unlabeled portion. The labeled data was first used to train the model. Later on, the model was applied to assign labels to the unlabeled data. Following this step, 80% of the labeled data was used for training, while the remaining 20% was applied for evaluating the model's performance. Different models were tested using the

How to cite this article:

aforementioned classifiers. Each model is then evaluated using metrics like accuracy, precision, recall, and F1-score. After evaluation, the trained models are used to predict the missing labels in the dataset. This process repeats in a loop until all the unlabeled examples in the dataset are labeled. It helps improve model performance when labeled data is limited by learning from both labeled and unlabeled samples.

Algorithm 1: Semi-supervised Learning

Input:

- N training examples with partial labels: $\text{dataset} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- Number of folds for cross-validation: $cv = 10$
- Classifier models: $\text{models} = [SVM, KNN, DT, RF, LP, ANN]$

Output:

- Trained models for each classifier

Initialize: $i \leftarrow 0$;

do

for each i **in** cv **do**

$\text{data_samples} \leftarrow \text{dataset} / cv$

$\text{training_set} \leftarrow \text{dataset} - \text{data_samples}[i]$

for each model **in** models **do**

$M_i \leftarrow \text{model.train}(\text{training_set}, k)$

$\text{Accuracy}_i \leftarrow \text{model.evaluate}(\text{data_samples}[i])$

$\text{Precision}_i \leftarrow \text{model.evaluate}(\text{data_samples}[i])$

$\text{Recall}_i \leftarrow \text{model.evaluate}(\text{data_samples}[i])$

$\text{F1-score}_i \leftarrow \text{model.evaluate}(\text{data_samples}[i])$

end for

end for

for each model **in** models **do**

$\text{dataset} \leftarrow \text{model.predict}(\text{dataset})$

end for

until not all examples in dataset have labels

FIGURE 7: Algorithm of semi-supervised learning

Results And Discussion

In this section, we present the experimental results of our proposed semi-supervised models. The performance of each classifier is evaluated across six different datasets. The results are analyzed to determine the strengths and limitations of each algorithm. Tables 4-9 present the performance comparison of ML and SSML models across various evaluation metrics. They show the comparison of the performance of ML and SSML algorithms across accuracy, precision, recall, and F1-score on six datasets: CM1, Linux Driver, Eclipse 3.0, JM1, KC1 and Prop-1. The equations of the evaluation metrics are given below [34] (TP = true positive, TN = true negative, FP = false positive, FN = false negative).

How to cite this article:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	85.79	88.14	85.23	93.51	73.95	77.25	79.19	84.61
SVM	59.83	86.96	45.03	88.53	44.75	79.39	44.89	83.71
KNN	87.4	83.0	82.2	84.24	85.09	73.47	83.62	78.49
DT	79.08	88.07	78.03	84.99	65.53	87.11	71.24	86.04
ANN	80.95	90.27	92.19	90.35	50.86	86.15	65.56	88.2
NB	73.34	0	87.18	0	33.55	0	48.46	0
LP	0	85.05	0	83.11	0	81.07	0	82.08

TABLE 4: Comprehensive evaluation of ML and SSML algorithms for Prop-1 dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine;

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	72.5	85.11	46.15	81.48	34.62	70.97	39.56	75.86
SVM	68	74.67	40.63	68.35	50	43.2	44.83	52.94
KNN	71.5	72.49	41.94	57.81	65	59.68	31.33	58.73
DT	75.5	79.42	60	67.67	57.31	72	56.87	69.77
ANN	73.5	82.85	59.32	79.41	54.69	64.8	56.91	71.37
NB	74	0	48.45	0	38.65	0	44.16	0
LP	0	77.66	0	72.73	0	51.61	0	60.38

TABLE 5: Comprehensive evaluation of ML and SSML algorithms for KC1 dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	77.3	79.05	57.14	79.25	30.24	54.19	39.55	64.37
SVM	70.27	70.42	58.85	64.24	33.64	34.19	40.39	44.63
KNN	71.14	74.07	52.5	70.88	42.56	59.48	46.79	64.07
DT	73.89	73.34	61.57	60.2	36.24	59.68	42.49	60.96
ANN	72.7	72.37	58.93	59.74	36.41	58.31	44.64	58.99
NB	0	73	0	0	0	47.1	0	54.89
LP	0	73	0	0	0	47.1	0	54.89

TABLE 6: Comprehensive evaluation of ML and SSML algorithms for JM1 dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	77.3	83.32	71.72	88.21	34.92	70.2	46.97	78.18
SVM	71.21	82.11	70.24	91.03	32.02	57.96	43.25	70.82
KNN	71.14	79.37	56.21	73.17	42.16	71.08	48.15	72.06
DT	77.68	75.96	74.91	67.64	33.77	68.93	46.55	68.28
ANN	71.9	81.04	69.18	75.97	34.02	72.24	47.42	74.06
NB	71.21	0	69.24	0	32.25	0	46.76	0
LP	0	61.08	0	45.36	0	38.32	0	46.1

TABLE 7: Comprehensive evaluation of ML and SSML algorithms for Eclipse 3.0 dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	71.43	96.67	71.43	95.45	67.25	93.33	63.16	94.38
SVM	70.78	76.71	100	86.79	62.12	25.7	54.26	39.66
KNN	66.02	75.72	58.27	65.06	52.46	40.15	53.81	49.64
DT	72.29	71.9	91.67	53.33	67.97	46.43	64.67	49.64
ANN	69.05	72.71	41.27	56.3	42.46	37.43	43.81	44.97
NB	70.13	0	68.46	0	32.56	0	49.66	0
LP	0	65.33	0	37.93	0	44.44	0	49.73

TABLE 8: Comprehensive evaluation of ML and SSML algorithms for Linux driver dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Classifier	Accuracy		Precision		Recall		F1-score	
	ML	SSML	ML	SSML	ML	SSML	ML	SSML
RF	71.63	95.45	51.85	93.33	66.67	95.45	58.33	94.3
SVM	68.9	74.09	45.25	69.62	53.23	65.87	48.87	67.65
KNN	73.05	81.82	55.74	75.91	75.56	79.39	64.15	77.33
DT	78.72	85.91	75.86	80.65	48.89	85.23	59.46	82.87
ANN	72.34	76.64	56.52	70.77	60.95	70.37	58.68	70.57
NB	68.09	0	57.89	0	61.2	0	60.32	0
LP	0	87.27	0	86.59	0	80.68	0	83.53

TABLE 9: Comprehensive evaluation of ML and SSML algorithms for CM1 dataset

ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; LP, Label Propagation; ML, Machine Learning; NB, Naive Bayes; RF, Random Forest; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

The evaluated classifiers include RF, SVM, KNN, DT, ANN, NB and LP. In this research, the semi-supervised version of RF (SSML) consistently delivered the best performance across most datasets. With Prop-1 dataset, ANN (SSML) delivered the best results, achieving 90.27% accuracy, 90.35% precision and an 88.20% F1-score. In contrast, RF (ML) recorded 85.79% accuracy. In the KC1 dataset, RF (SSML) again stood out with 85.11% accuracy and 81.48% precision, while DT (ML) managed 75.5% accuracy. For the JM1 dataset, RF (SSML) outperformed all others with 79.05% accuracy and 64.37% F1-score compared to DT (ML), which had 73.89% accuracy. In the Eclipse 3.0 dataset, RF (SSML) maintained its top position, with 83.32% accuracy and 78.18% F1-score. On the Linux Driver dataset, RF (SSML) again led with 96.67% accuracy and 94.38% F1-score, whereas DT (ML) scored 72.29% in accuracy. For the CM1 dataset, RF (SSML) achieved the highest accuracy of 95.45% and an F1-score of 94.3%, while DT (ML) reached only 78.72% accuracy. All experiments were performed in local machine that included an intel core i5 processor, 16 GB RAM, 512 GB SSD, Windows OS and a stable internet connection. Besides this, Python and Google Colab Pro with reproducible setup were used for model development and data augmentation. The training environment included NVIDIA T4 and A100 GPUs with CUDA support, 32 GB RAM and a 24-hour session limit with sufficient compute units. The software stack consisted of Python 3.12, PyTorch 2.8 (CUDA 12.6) and TensorFlow 2.19.

ANN (SSML) consistently excelled, leading in multiple datasets, including the best accuracy in Linux Driver and CM1. RF (SSML) achieved the highest recall and F1-score in Linux Driver and CM1, while ANN (SSML) dominated in Prop-1 with accuracy and F1-score. RF (SSML) on the JM1 dataset stood out, showing the highest AUC score. Comparisons with previous works were also discussed. Table 10 presents a comparison of the proposed SSML model with state-of-the-art techniques reported in the literature.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Ref.	Balancing method	Dataset	Repository	Methodology	Result
This work	TGAN	Prop-1	PROMISE	SSML-ANN	F1-score = 88.20%
		KC1	PROMISE	SSML-RF	Accuracy = 85.11%
		JM1	PROMISE	SSML-RF	AUC = 84.29%
		Eclipse 3.0	Eclipse	SSML-RF	F1-score = 78.18%
		Linux-driver net	ARB	SSML-RF	Accuracy = 96.67%
		CM1	PROMISE	SSML-RF	Accuracy = 95.45%
[7]	GAN	Lucene	PROMISE	DP-GANT	F1-score = 87.4%
[8]	GAN, SMOTE	CM1	PROMISE	CNN	Accuracy = 92%
[12]	VAE	CM1	PROMISE	SVM	Accuracy = 95.3%
[16]	-	Eclipse 3.0	Eclipse	exRF	F1-score = 75%
[17]	Laplacian	JM1	PROMISE	NSGLP	AUC = 72%
[18]	VAE	KC1	PROMISE	RF	F1-score = 92%
[21]	GAN	Prop-1	PROMISE	KNN	F1-score = 84.3%
[25]	SMOTE	JM1	PROMISE	CatBoost	Accuracy = 85%

TABLE 10: Comparative analysis of previous studies and current research findings

ANN, Artificial Neural Network; AUC, Area Under the Curve; CatBoost, Categorical Boosting; CNN, Convolutional Neural Network; DP-GANT, Differential Privacy Generative Adversarial Network with Transformer; exRF, Extreme Random Forest; GAN, Generative Adversarial Network; KNN, K-Nearest Neighbors; RF, Random Forest; SMOTE, Synthetic Minority Over-sampling Technique; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine; TGAN, Tabular Generative Adversarial Network; VAE, Variational Autoencoder

The proposed technique is benchmarked against methods such as TGAN, GAN, VAE, AC-GAN, and Laplacian sampling on similar datasets. The SSML method not only matches but often surpasses these leading methods in terms of accuracy and F1-score; along with this, it highlights robustness to class imbalance. The current work achieves the highest accuracy of 95.45% and 96.67% using TGAN on the CM1 and Linux-driver datasets, outperforming other studies such as [7,8] (accuracy up to 92%) and [11,12,16,17] (accuracy up to 95.3%, AUC up to 72%). The results demonstrate significant advancements in predictive performance compared to prior works. Figure 8 illustrates the highest accuracy and F1-score of applied ML and SSML on six different datasets with a bar graph for better visualization. Figure 9 presents the ROC curves for six datasets, showcasing the performance of different classifiers in a graphical format. ROC curve illustrates the

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

trade-off between the true positive rate and false positive rate across various threshold settings. A curve that rises sharply towards the top-left corner indicates better classifier performance. The closer the curve is to this corner, the higher the model’s ability to distinguish between classes.

In summary, the proposed TGAN-based semi-supervised approach outperformed other approaches and achieved the highest accuracy and F1-score. The data generated by TGAN aid in enhancing the accuracy and generalization of the SDP model. TGAN serves as a valuable asset for effective data augmentation, strengthening the performance and robustness of SSML frameworks.

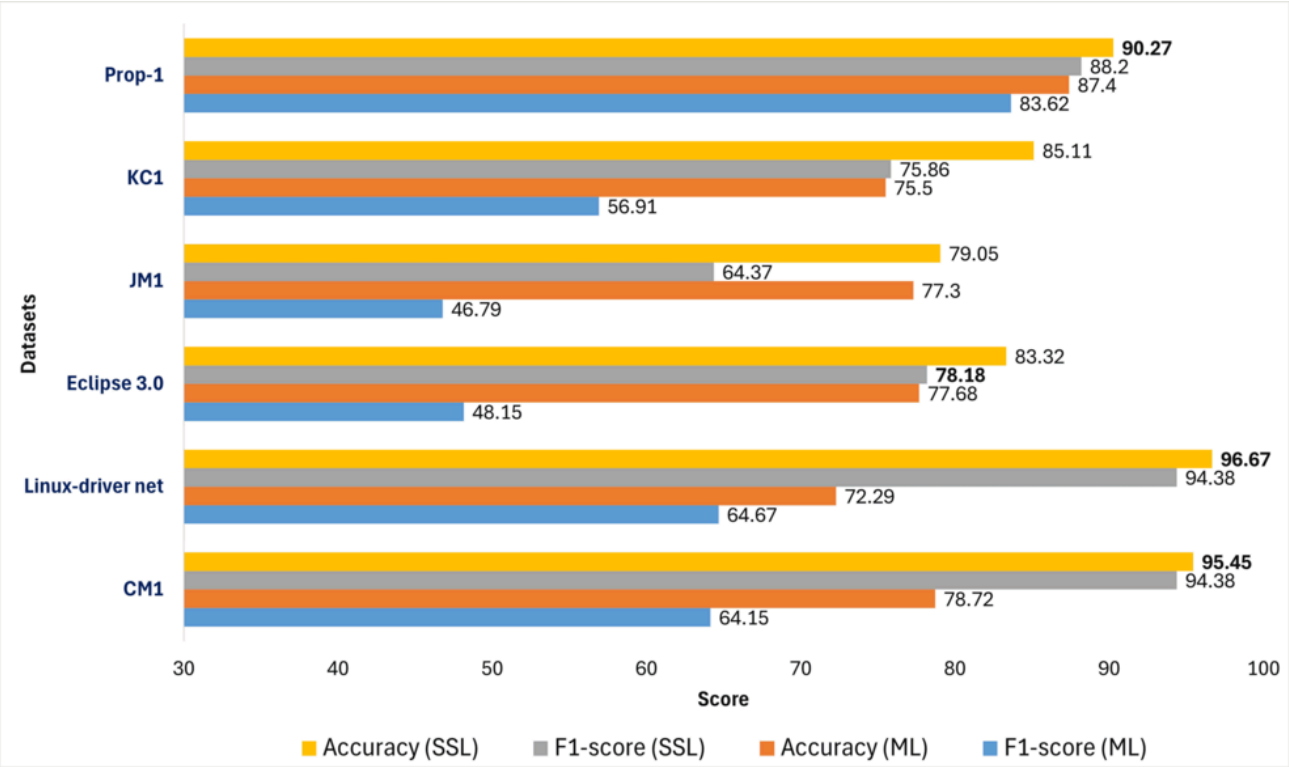


FIGURE 8: Performance metrics of ML and SSML: Accuracy and F1-score visualization

ML, Machine Learning; SSML, Semi-Supervised Machine Learning

How to cite this article:

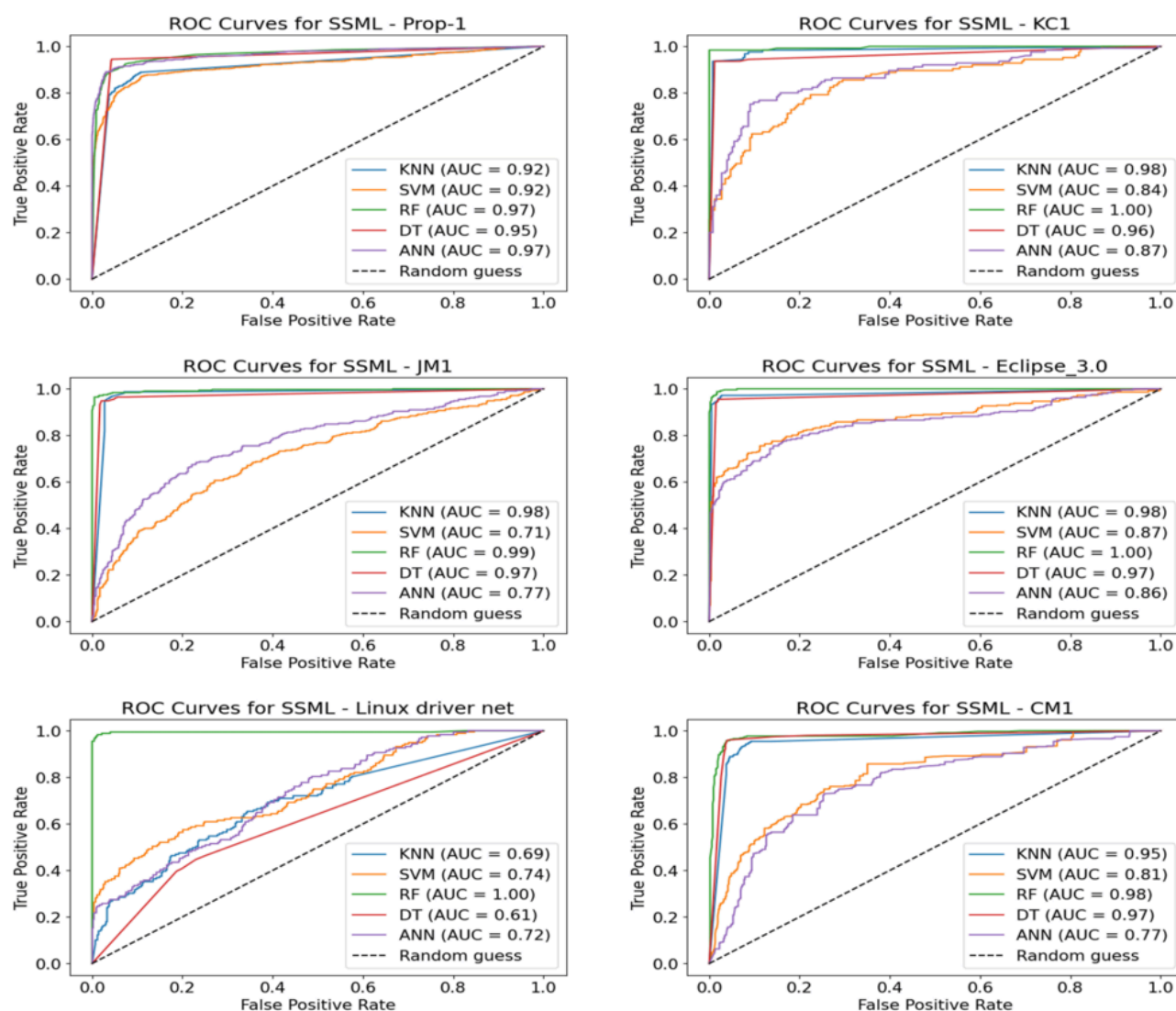


FIGURE 9: ROC curves for six datasets after applying the SSML model

AUC, Area Under the Curve; ANN, Artificial Neural Networks; DT, Decision Tree; KNN, K-Nearest Neighbors; RF, Random Forest; ROC, Receiver Operating Characteristic; SSML, Semi-Supervised Machine Learning; SVM, Support Vector Machine

Conclusions

This study demonstrates the significant impact of using semi-supervised learning techniques with limited labeled data in SDP. SSML models consistently outperformed traditional approaches across key metrics, showcasing their superiority in improving prediction accuracy. The integration of TGAN for data augmentation further enhanced model performance by effectively increasing the size of the training dataset. Future research could explore integrating more advanced data augmentation techniques to further enhance model performance, include software testing and defect data from different testing phases to build a more robust dataset, and investigate the application of other semi-supervised algorithms on larger and more diverse datasets to provide deeper insights into scalability. Additionally, incorporating explainable AI and large language model methods would help in better understanding the decision-making process of these models in defect prediction.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. *Cureus J Comput Sci* 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Md. Ariful Islam Arif, Hosney Jahan, Md Mahbubur Rahman

Acquisition, analysis, or interpretation of data: Md. Ariful Islam Arif, Md Hasibul Islam , Md Akhtaruzzaman

Drafting of the manuscript: Md. Ariful Islam Arif, Md Hasibul Islam , Hosney Jahan

Critical review of the manuscript for important intellectual content: Md Hasibul Islam , Hosney Jahan, Md Akhtaruzzaman, Md Mahbubur Rahman

Supervision: Hosney Jahan, Md Akhtaruzzaman, Md Mahbubur Rahman

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Data Availability Statements

The datasets (and/or code) supporting this study are available from the corresponding author upon reasonable request. The data (and/or code) supporting this study are openly available in NASA REPOSITORY at <https://data.mendeley.com/datasets/923xvkk5mm/1>.

Acknowledgements

The authors acknowledge the use of publicly available software defect prediction datasets from the PROMISE, ARB and Eclipse repositories. The authors would also like to express their sincere gratitude to everyone who contributed to this research.

References

1. Hryszko J, Madeyski L: [Assessment of the software defect prediction cost effectiveness in an industrial project](#). Software Engineering: Challenges and Solutions. Madeyski L, Śmiałek M, Hnatkowska B, Huzar Z (ed): Springer, Cham; 2017. 504:77-90. [10.1007/978-3-319-43606-7_6](#)
2. Zong Z, Guan Y: [AI-driven intelligent data analytics and predictive analysis in industry 4.0: transforming knowledge, innovation, and efficiency](#). Journal of the Knowledge Economy. 2024, 16:864-903. [10.1007/s13132-024-02001-z](#)
3. Mushtaq W: [20 latest technologies in software industry 2025](#). VisionX. (2024). Accessed: July 8, 2025: <https://visionx.io/blog/latest-technologies-in-software-industry/>.
4. Song Q, Guo Y, Shepperd M: [A comprehensive investigation of the role of imbalanced learning for software defect prediction](#). IEEE Transactions on Software Engineering. 2019, 45:1253-1269. [10.1109/tse.2018.2836442](#)
5. Udu AG, Salman MT, Ghalati MK, Lecchini-Visintini A, Siddle DR, Dong H: [Emerging SMOTE and GAN variants for data augmentation in imbalance machine learning tasks: a review](#). IEEE Access. 2025, 13:113838-113853. [10.1109/access.2025.3584532](#)
6. Alsafadi F, Yaseen M, Wu X: [An investigation on machine learning predictive accuracy improvement and uncertainty reduction using VAE-based data augmentation](#). Nuclear Engineering and Design. 2025, 445:114433. [10.1016/j.nucengdes.2025.114433](#)

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

7. Jain B, Patidar S, Sudershan D: [Heterogeneous software defect prediction using generative models](#). 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT), Indore, India. 2022, 367-372. [10.1109/CSNT54456.2022.9787607](#)
8. Luo N, Ma Y, Li J, Xie Y: [A just-in-time software defect detection method using generative adversarial networks](#). 2023 4th International Conference on Electronic Communication and Artificial Intelligence (ICECAI), Guangzhou, China. 2023, 37-45. [10.1109/ICECAI58670.2023.10176967](#)
9. Song W, Gan L, Bao T: [Software defect prediction via generative adversarial networks and pre-trained model](#). International Journal of Advanced Computer Science and Applications. 2024, 15:1196-1209. [10.14569/ijacsa.2024.01503119](#)
10. Kumar PS, Venkatesan R: [Improving software defect prediction using generative adversarial networks](#). International Journal of Science and Engineering Applications. 2020, 9:117-120. [10.7753/IJSEA0909.1001](#)
11. Zou Y, Wang H: [A three-stage cross-project defect prediction framework based on feature representation and knowledge transfer](#). Complex & Intelligent Systems. 2025, 11:459. [10.1007/s40747-025-02098-y](#)
12. Eivazpour Z, Keyvanpour MR: [Improving performance in software defect prediction using variational autoencoder](#). 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), Tehran, Iran. 2019, 644-649. [10.1109/KBEI.2019.8734915](#)
13. Hu X, Zheng M, Zhu R, Zhang X, Jin Z: [Fed-OLF: federated oversampling learning framework for imbalanced software defect prediction under privacy protection](#). IEEE Transactions on Reliability. 2025, 74:3266-3280. [10.1109/tr.2024.3524064](#)
14. Sun Y, Xu L, Li Y, Guo L, Ma Z, Wang Y: [Utilizing deep architecture networks of VAE in software fault prediction](#). 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, VIC, Australia. 2018, 870-877. [10.1109/BDCloud.2018.00129](#)
15. Cohen S, Elmishali A, Kalech M: [SeC-GAN: generative adversarial network for just-in-time defect prediction](#). 33rd International Workshop on Principle of Diagnosis - DX 2022, LAAS-CNRS-ANITI. 2022, 1-9.
16. Eivazpour Z, Keyvanpour MR: [Adversarial samples for improving performance of software defect prediction models](#). Data Science: From Research to Application. Bohlouli M, Sadeghi Bigham B, Narimani Z, Vasighi M, Ansari E (ed): Springer, Cham; 2020. 45:299-310. [10.1007/978-3-030-37309-2_24](#)
17. Alqarni A, Aljamaan H: [Leveraging ensemble learning with generative adversarial networks for imbalanced software defects prediction](#). Applied Sciences. 2023, 13:13319. [10.3390/app132413319](#)
18. He Q, Shen B, Chen Y: [Software defect prediction using semi-supervised learning with change burst information](#). 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, USA. 2016, 113-122. [10.1109/COMPSAC.2016.193](#)
19. Zhang ZW, Jing XY, Wang TJ: [Label propagation based semi-supervised learning for software defect prediction](#). Automated Software Engineering. 2017, 24:47-69. [10.1007/s10515-016-0194-x](#)
20. Sun Y, Xu L, Guo L, Li Y, Wang Y: [A comparison study of VAE and GAN for software fault prediction](#). Algorithms and Architectures for Parallel Processing. Wen S, Zomaya A, Yang LT (ed): Springer, Cham; 2020. 11945:82-96. [10.1007/978-3-030-38961-1_8](#)
21. Chouhan SS, Rathore SS: [Generative adversarial networks-based imbalance learning in software aging-related bug prediction](#). IEEE Transactions on Reliability. 2021, 70:626-642. [10.1109/tr.2021.3052510](#)
22. Shu R, Xia T, Williams L, Menzies T: [Dazzle: Using optimized generative adversarial networks to address security data class imbalance issue](#). Proceedings of the 19th International Conference on Mining Software Repositories. 2022, 144-155. [10.1145/3524842.3528437](#)
23. Rathore SS, Chouhan SS, Jain DK, Vachhani AG: [Generative oversampling methods for handling imbalanced data in software fault prediction](#). IEEE Transactions on Reliability. 2022, 71:747-762. [10.1109/tr.2022.3158949](#)
24. Islam A, Zaman MS, Jahan H: [A 2-layer ensemble machine learning based system for software defect prediction](#). 2025 International Conference on Electrical, Computer and Communication Engineering (ECCE), Chittagong, Bangladesh. 2025, 1-6. [10.1109/ECCE64574.2025.11013826](#)
25. Keya AJ, Khandaker N, Hasan MJ, Jahan H, Akhtaruzzaman M: [Leveraging CatBoost and XAI for enhanced software defect prediction](#). 2024 27th International Conference on Computer and Information Technology (ICCIT), Cox's Bazar, Bangladesh. 2024, 2998-3003. [10.1109/ICCIT64611.2024.11021971](#)

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>

26. Tong H, Liu B, Wang S: [Benchmark data sets](https://doi.org/10.17632/923xvkk5mm.1). (2017). <https://doi.org/10.17632/923xvkk5mm.1>.
27. Natella R: [Aging-related bugs and software complexity metrics](https://doi.org/10.5281/zenodo.581659). Zenodo. (2017). Accessed: October 10, 2025: <https://doi.org/10.5281/zenodo.581659>.
28. Zimmermann T, Premraj R, Zeller A: [Predicting defects for eclipse](https://doi.org/10.1109/PROMISE.2007.10). Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), Minneapolis, MN, USA. 2007, 1-7. [10.1109/PROMISE.2007.10](https://doi.org/10.1109/PROMISE.2007.10)
29. Liu T, Fan J, Li G, Tang N, Du X: [Tabular data synthesis with generative adversarial networks: design space and optimizations](https://doi.org/10.1007/s00778-023-00807-y). The VLDB Journal. 2024, 33:255-280. [10.1007/s00778-023-00807-y](https://doi.org/10.1007/s00778-023-00807-y)
30. Goodfellow I, Pouget-Abadie J, Mirza M, et al.: [Generative adversarial networks](https://doi.org/10.1145/3422622). Communications of the ACM. 2020, 63:139-144. [10.1145/3422622](https://doi.org/10.1145/3422622)
31. Liaw A, Wiener M: [Classification and regression by randomForest](https://doi.org/10.1201/9781420067194). R News. 2002, 2:18-22.
32. Marsland S: [Machine Learning: An Algorithmic Perspective](https://doi.org/10.1201/9781420067194). Chapman and Hall/CRC, New York; 2009. [10.1201/9781420067194](https://doi.org/10.1201/9781420067194)
33. Zhou D, Bousquet O, Lal T, Weston J, Schölkopf B: [Learning with local and global consistency](https://doi.org/10.1145/3422622). NIPS'03: Proceedings of the 17th International Conference on Neural Information Processing Systems. 2003, 321-328.
34. Russell S, Norvig P: [Artificial Intelligence: A Modern Approach](https://doi.org/10.1145/3422622). 4th ed. Pearson, London; 2020.

How to cite this article:

Arif M, Islam M, Jahan H, et al. (June 16, 2026) Software Defect Prediction Using Generative Adversarial Network: A Comparative Study of Semi-Supervised Machine Learning Models. Cureus J Comput Sci 3 : es44389-026-00120-z. DOI <https://doi.org/10.7759/s44389-026-00120-z>