Tikrit Journal of Engineering Sciences (2025); 32(Sp1): 2726
DOI: http://doi.org/10.25130/tjes.sp1.2025.33

IRAQI Academic Scientific Journals

ISSN: 1813-162X (Print); 2312-7589 (Online)
Tikrit Journal of Engineering Sciences
available online at: http://www.tj-es.com

TJES Tikrit Journal of Engineering Sciences

SparseBonsai: A Dynamic, Resource-Efficient Classification Model for Edge Computing and Industrial IoT

Neelamadhab Khaya *[a], **Binod Ku Pattanayak** [a], **Bichitrananda Patra** [b], **Pravat Kumar Rautaray** [a], **Bibhuti Bhusan Dash** [c], **Bibhuprasad Mohanty** [d]

a Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar, Odisha, India.
b Department of Computer Application, Institute of Technical Education and Research, Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar, Odisha, India.
c School of Computer Applications, KIIT Deemed to be University, Bhubaneswar, Odisha, India.
d Department of Electronics and Communication Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar, Odisha, India.

Keywords:
Bonsai Algorithm; Dynamic Regularization; Edge Computing; IIoT; Projection Techniques; Resource Optimization; SparseBonsai.

Highlights:

- SparseBonsai: Adaptively regulates sparsity and regularization parameters during training.
- Achieves 86.91 % accuracy with 0.048 MB model size and 35 % faster inference.
- Accuracy improves by 8 % and size by 33 %.
- Shows an AUC of nearly 0.95.
- Provides a real-time fault detection on IIoT edge nodes.

ARTICLE INFO

Article history:

Received	18 Aug. 2025
Received in revised form	29 Aug. 2025
Accepted	10 Oct. 2025
Final Proofreading	20 Dec. 2025
Available online	26 Dec. 2025

© THIS IS AN OPEN ACCESS ARTICLE UNDER THE CC BY LICENSE. http://creativecommons.org/licenses/by/4.0/

Citation: Khaya N, Pattanayak BKu, Patra B, Rautaray PK, Dash BB, Mohanty B. **SparseBonsai: A Dynamic, Resource-Efficient Classification Model for Edge Computing and Industrial IoT**. *Tikrit Journal of Engineering Sciences* 2025; **32**(Sp1): 2726.
http://doi.org/10.25130/tjes.sp1.2025.33

*Corresponding author:
Neelamadhab Khaya
Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar, Odisha, India.

Abstract: The rapid evolution of edge computing and IIoT ecosystems demands lightweight machine learning models that deliver accurate predictions under resource constraints. Traditional classifiers, such as deep neural networks and decision trees, often struggle to balance accuracy, interpretability, and computational efficiency in such environments. This work introduces SparseBonsai, an enhanced variant of the Bonsai Tree algorithm that includes projection techniques, dynamic sparsity parameters, and adaptive regularization. Bonsai Tree models work with fixed parameters, whereas SparseBonsai dynamically adjusts sparsity and regularization during training. It improves adaptability and generalization. SparseBonsai achieves 86.91% classification accuracy with a model size of 0.048 MB, and inference time is less than 35% as of neural networks with a competitive accuracy. Model's robustness and efficiency is evaluated using precision, recall, F1-score, and ROC-AUC. These results show that the SparseBonsai can be a practical solution for a real-time and resource-efficient fault detection system as an IIoT edge computing platform. The novelty of SparseBonsai is its dynamic adjustment of sparsity and regularization in training in comparison to the conventional Bonsai Tree algorithm. SparseBonsai reduces inference time by 35% and memory footprint by 38% compared with existing lightweight classifiers.

1.INTRODUCTION

The integration of Artificial Intelligence in IIOT reshapes the modern industrial control system and automation with resource-efficient operations. Integrating AI in industry optimization and neural network-based control systems is a major application for enhancing precision, stability, and self-learning ability in resource-constrained systems [1]. Many studies emphasize AI–IIoT fusion for smart automation and scalable industrial control with self-adaptive edge intelligence [2,3]. Recent research highlights that edge computing platforms and real-time inference systems use machine learning models that can operate efficiently in resource-constrained environments [4]. Conventional machine learning models often have limited computational capacity, memory and power availability. It is impractical to deploy conventional machine learning models, such as deep neural networks (DNNs) or ensemble models, which require substantial computational resources [5,6]. Neural networks require significant memory and computational resources, which are unsuitable for edge computing scenarios [7]. DNNs, DTs, and SVMs fail to achieve optimal efficiency and accuracy in IIoT fault detections. DNNs provide high accuracy but are computationally expensive. SVMs are efficient but do not generalize well. The base Bonsai Tree algorithm performs well in terms of the feasibility of resource-efficient models, but uses static regularization with fixed sparsity. Which is not adaptable to diverse datasets. To address this gap, we propose SparseBonsai, an improved variant of Bonsai that introduces dynamic sparsity adjustment and adaptive regularization with optimized projection techniques. These features of SparseBonsai enable achieving high classification accuracy while optimizing memory footprint and inference time, suitable for IIoT edge deployment. Traditional Bonsai applies uniform sparsity penalties in the training, whereas SparseBonsai dynamically tunes these parameters while ensuring efficient computation without compromising accuracy.

1.1.Motivation and Problem Statement

Traditional classification models are unable to balance computational efficiency and predictive accuracy. Neural networks overfit when applied to small datasets. It requires extensive training time and consumes significant computational resources. Decision trees lead to poor generalization and are unable to capture complex feature relationships [8,9]. This work presents an adaptive, resource-efficient intelligent system for edge-computing IIoT subsystems, inspired by intelligent control systems in smart microgrids [10]. SparseBonsai gives a tree-based neural network architecture that optimizes input dimensions and passes data through non-linear branching functions, which improves classification accuracy. SparseBonsai preserves important feature relationships while highly reducing computational complexity by using learnable projection matrices.

1.2.Key Characteristics of SparseBonsai

The architecture of SparseBonsai consists of three main components.

Projection Layer: This layer reduces and transforms high-dimensional input data into a lower-dimensional space. It holds crucial feature relationships. SparseBonsai optimizes projection dimensions.

Tree Structure with Nonlinear Branching: Nonlinear branching functions at each inner node, which dynamically route samples based on parameters. SparseBonsai improves branch-level accuracy by optimizing node-level parameters, which introduces adaptive branching thresholds.

Leaf Nodes for Classification: The leaf nodes classify by aggregating information from the routed data. SparseBonsai uses L2 regularization with sparsity constraints on the leaf nodes, which supports efficient parameter usage and helps avoid overfitting to the dataset.

1.3.Optimization Techniques and Training Framework

SparseBonsai has a novel optimization technique to enhance the model's efficiency.

Dynamic Regularization: The sparsity penalties associated with L2 regularization coefficients are dynamically adjusted during training. It prevents overfitting. The model holds an optimal balance between model complexity and generalization.

Gradient-Based Updates and Adam Optimizer: A gradient-based optimization method is used to minimize classification loss, with the Adam optimizer, which gives faster convergence and prevents gradient explosion.

Hyperparameter Tuning: SparseBonsai tunes the projection dimensions, tree depth, learning rate, and sparsity coefficients. These are fine-tuned to optimize classification accuracy and computational efficiency.

1.4.Dataset Description

In this work, the Case Western Reserve University (CWRU) Bearing Fault Dataset has been applied to the SparseBonsai architecture. This dataset is widely used for predictive maintenance and fault diagnosis in industrial machines [11]. The dataset was collected from a motor with accelerometer sensors fitted on the drive-end and fan-end of the motor housing. These signals were sampled at 12 kHz and 48 kHz under various load conditions (0–3 HP) and different fault diameters (0.007–0.021 inches). The dataset contains three fault categories:

- Outer Race Fault (OR),
- Inner Race Fault (IR),
- Ball Fault (B).

1.4.1.Preprocessing

The following preprocessing steps were applied for consistent training: Segmentation: The vibration signals were segmented into fixed-length windows of equal duration, generating multiple training and testing samples. Normalization: The samples were normalized to have a mean of zero and unit variance. This improves the stability of training.

Label Encoding: The fault types were encoded into integer labels {OR=0, IR=1, B=2}.

Stratified Splitting: The dataset was split into 80% training and 20% test sets, with balanced class proportions.

2.RELATED WORK

The increasing adoption and advancement of Internet of Things (IoT) devices in personal, domestic, and industrial environments have raised serious concerns about cybersecurity and the limited computational capacity of these devices [12,13]. Amgbara et al. [7] introduced lightweight ML models for securing personal IoT devices. They highlighted trade-offs between accuracy and complexity using models such as decision trees and SVMs. Abdul Wahab et al. [14] developed a lightweight host-based Intrusion Detection System (HIDS) using N-gram features and a Multilayer Perceptron model, which is optimized for fog-based IoT devices. Their architecture reduced computational overhead using sparse matrices and feature selection techniques. Our work employs the concept of lightweight modeling by deploying a SparseBonsai-based classifier with projection-based optimization and dynamic regularization. Similarly, Li and Dou [15] proposed a smart healthcare framework combining IoT and AI for cardiac disease diagnosis using a CuSO-optimized MuLSTM model. Their focus is on sequential learning for diagnosis in a resource-constrained environment. Tanveer et al. [16] introduced Light Ensemble Guard, which uses an ensemble model combining LightGBM, XGBoost, and Extra Trees for detecting IoT attacks in real time. It has been designed for devices with limited memory and computing power. It balances detection accuracy with efficiency using majority voting and AUC-based validation. Our SparseBonsai model achieves similar goals through a compact tree structure and sparse projection layers, which offer fast, interpretable classification. In another study, Daghero et al. [17] demonstrated dynamic decision tree ensembles that adapt the number of executed trees based on input complexity. Their approach optimizes ensemble inference. Qiu et al. [18] proposed a directed-edge-based weight-prediction model that combines multiple decision tree ensembles to predict relationships in dynamic decision neural networks. This method improves edge-prediction accuracy by using similarity-based features. Their work focused on classification tasks with a single tree structure optimization. Al Smadi et al. [19] presented energy-efficient storage and management frameworks for microgrids. They have highlighted the growing relevance of adaptive, low-power algorithms that align with the efficiency goals of IIoT and edge computing. Al Smadi et al. [20] introduced a fault localization method developed for the Samarra Power Station in Iraq which shows the effectiveness of ANN GA integration to identify complex fault types and improve system reliability. This method underscores the ongoing need for adaptive, computationally efficient AI models. The Bonsai algorithm by Kumar et al. [21] demonstrated accurate classification within just 2KB of RAM by combining sparse projection with nonlinear tree-based decision paths. SparseBonsai builds directly on this principle. It introduced dynamic regularization, adaptive sparsity with optimized projection adjustments to improve generalization. Naveen and Kounte [22] improved the Bonsai algorithm by reinforcing the feasibility of projection-based tree models for edge classification tasks. While existing studies on lightweight IIoT models have largely focused on intrusion detection and healthcare monitoring, a few works have addressed lightweight intelligent models. Decision trees, SVMs, and Bonsai-based classifiers have been applied to the CWRU dataset, but they show limited accuracy. SparseBonsai directly addresses this gap by combining compact tree-based structures with adaptive sparsity, with better efficiency and competitive accuracy in industrial fault detection.

3.METHODOLOGY

SparseBonsai is an optimized tree-based machine learning architecture which is designed to improve classification accuracy while balancing computational efficiency. SparseBonsai involves three key components: dimensionality reduction by projection matrices, hierarchical decision-making using tree structures and an adaptive regularization method. The training algorithm uses dynamic sparsity constraints across diverse datasets. Tables 1-2 summarize the three bearing fault categories (Outer Race, Inner Race, and Ball Fault) that are considered in this study. The dataset contains the respective class labels and the distribution of training and test samples [11].

3.1.Algorithm Architecture

The core architecture of SparseBonsai builds on the Bonsai algorithm by incorporating several enhancements that improve classification performance and computational efficiency.

Fig.1 describes the block diagram of SparseBonsai architecture.

Table 1 Bearing Fault Types and Corresponding Class Labels Used for Model Training and Evaluation.

Bearing Type	Class Label	Classes
Outer Ring Fault	OR	Class 0
Inner Ring Fault	IR	Class 1
Ball Fault	B	Class 2

Table 2 Distribution of Samples in Training and Testing Sets [11].

Data Set	Total Samples	Class 0 Samples	Class 1 Samples	Class 2 Samples
Training Set	11,673	3,100	3,119	5,454
Testing Set	2,919	786	768	1,365

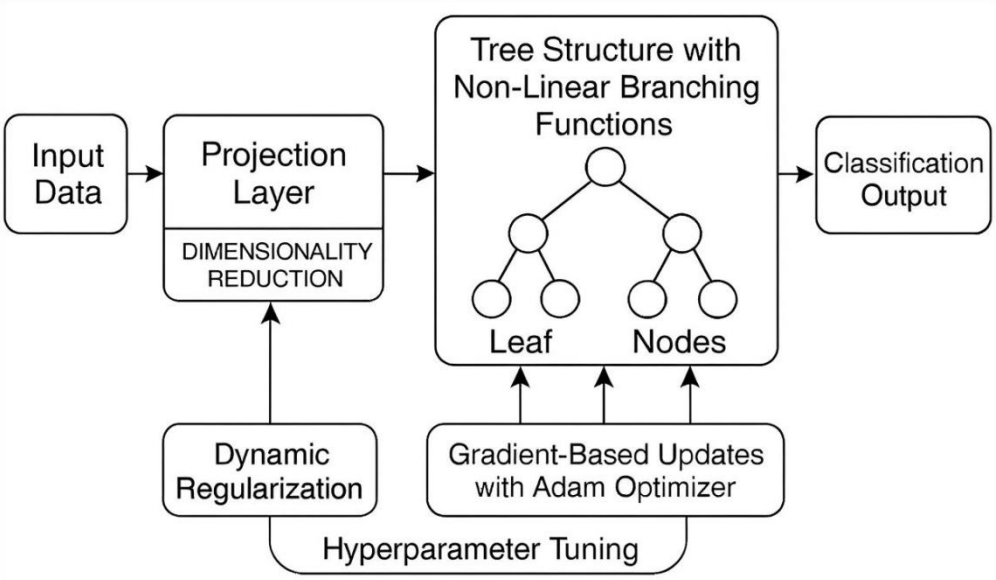

Fig. 1 Block Diagram of the Proposed SparseBonsai Model for Efficient and Accurate Classification in Resource-Constrained Environments.

Input Layer: In the input layer, raw data (features) are used for classification.

Pre-processing: Preprocessing involves data normalization, feature extraction, and scaling.

Bonsai Model:

- Bonsai Tree Structure
- Sparse Projection
- Node Assignment and Prediction

Decision Layer: Decision layers bring predicted class labels (Class 0, Class 1, Class 2).

Evaluation Metrics: Precision, Recall, and F1-Score are key metrics for comparative evaluation.

3.1.1.Projection Layer

The projection layer is the first step in the SparseBonsai architecture, which reduces the dimensionality of the input feature space to a more compact representation. Given an input vector $x \in \mathbb{R}^d$, the projection matrix $Z \in \mathbb{R}^{m \times d}$ transforms the input to a lower-dimensional feature space (Eq. (1)):

$$\tilde{x} = Zx, \ \tilde{x} \in \mathbb{R}^m \quad \textbf{(1)}$$

In Eq. (1):

m is taken as the projection dimension, which is set to 32 in the current implementation

Z is learned during training to facilitate feature preservation and dimensionality reduction. SparseBonsai dynamically adjusts the projection dimensions based on the dataset's complexity. It is ensured that the most important features are left while reducing computational challenges. The projection layer keeps relevant feature relationships, which are essential for classification tasks.

3.1.2.Tree Structure with Non-Linear Branching

The tree structure in SparseBonsai routes data samples through a sequence of non-linear branching functions at internal node levels. Each internal node gives a branching function that computes a nonlinear decision boundary to separate the feature space. The branching function is defined as:

$$B(x) = \tanh\left(\sigma^{-1} \times \tilde{x}^T T_i \tilde{x}\right) \quad \textbf{(2)}$$

In Eq. (2):

- $T_i \in \mathbb{R}^{m \times m}$ is the learned parameter matrix for the node i.
- σ is the branching parameter that controls the sharpness of decision boundaries, set to 1.0 in our work.
- $\tilde{x}$ is the projected input

The tree depth, set to 3 in our work, balances model capacity and computational efficiency. SparseBonsai dynamically adjusts branching thresholds and node parameters.

3.1.3.Leaf Nodes and Classification Vectors

Once the input vector traverses the tree, it reaches a leaf node that computes class predictions. The leaf nodes generate classification output decisions based on learned classification vectors with probability:

$$y = \sum_j p_j(x) W_j \quad (3)$$

In Eq. (3):

- $p_j(x)$ is the probability of reaching leaf node j
- W_j is the classification vector for the corresponding leaf node.

While preventing overfitting, SparseBonsai applies L2 regularization and sparsity constraints to leaf node parameters.

3.1.4.Algorithm

```
Algorithm: Step-by-step Training Framework of SparseBonsai
Input: Dataset D = {X, y}, projection dimension m, tree depth d, and learning rate η are taken as input.
Output: Trained phase of SparseBonsai model starts as
    1. Preprocessing stage of data:
        a. Input features are normalized (zero mean, unit variance).
        b. Class labels are encoded into integers (0, 1, 2).
        c. Dataset is split into training (80%) and testing (20%) sets using stratified sampling.
    2. Projection matrix is initialized Z ∈ R^(m×d_in).
    3. A bonsai tree with depth d is constructed.
    4. For each epoch:
        a. Each minibatch (X_batch, y_batch:
            i. Project inputs: X_proj = Z * X_batch
            ii. Projected samples routed through the tree using branching functions
            iii. Classification at leaf nodes is computed
            iv. Loss calculation: L = L_classification + λ1 L_reg + λ2 L_sparse
            v. λ and ρ are dynamically adjusted based on validation performance
            vi. Parameters are updated using the Adam optimizer
    5. Trained model is returned.
```

3.2.Mathematical Formulation

The SparseBonsai algorithm is mathematically combines classification loss, regularization penalties, and sparsity constraints.

3.2.1.Objective Function

The objective function minimizes classification loss while imposing regularization and sparsity penalties the:

$$L = L_{classification} + \lambda_1 L_{reg} + \lambda_2 L_{sparse} \quad (4)$$

Where:

- $L_{classification}$ Is the cross-entropy loss function defined as:

$$L_{classification} = -\sum_{i=1}^{c} y_i \log(\hat{y}_i) \quad (5)$$

- L_{reg} Is the L2 regularisation term that controls model complexity:

$$L_{reg} = \lambda_1 ||Z||_2^2 + \lambda_2 \sum_i ||T_i||_2^2 + \lambda_3 \sum_j ||W_j||_2^2 \quad (6)$$

- L_{sparse} Is the sparsity constraint that enforces efficient computation:

$$L_{sparse} = \rho_1 ||Z||_1 + \rho_2 \sum_i ||T_i||_1 + \rho_3 \sum_j ||W_j||_1 \quad (7)$$

SparseBonsai dynamically adjusts the regularization and sparsity coefficients λ and ρ during training to adapt the model to different datasets Eq. (4-7).

3.3.Training Configuration and Optimization

The training of SparseBonsai follows a structured optimization process. Below are the steps.

3.3.1.Data Preprocessing

In data preprocessing, feature normalization and label encoding should be performed to ensure consistency during training. Feature normalization is to be accomplished using the mean and standard deviation of the training data.

3.3.2.Hyperparameter Configuration

SparseBonsai's hyperparameter selection was done by a systematic tuning process rather than manual trial and error. Search Strategy: A grid search strategy combined with random search was carried out over candidate ranges for key hyperparameters (projection dimension is {16, 32, 64}, tree depth is {2, 3, 4}, learning rate is {0.001, 0.01, 0.05}, batch size is {32, 64, 128}). Random search was used initially to identify a pattern, followed by a fine-grained grid search for the final selection. Evaluation Protocol: For each candidate setting, 5-fold cross-validation was performed on the training dataset. The hyperparameter set that performs the highest macro-averaged F1-score on validation folds was chosen. F1-score balances both precision and recall across fault classes. The optimal parameters are projection dimension = 32, tree

depth = 3, learning rate = 0.01, batch size = 64 and training epochs=100. These parameters set gives the best tradeoff between accuracy and computational efficiency. Dynamic Adjustment parameters like regularization (λ) and sparsity coefficients (ρ) were adapted to prevent overfitting.

3.3.3.Regularization and Sparsity Penalties

SparseBonsai applies a combination of L2 regularization and sparsity penalties to the projection matrix, internal nodes, and leaf nodes. Regularization prevents overfitting, while sparsity constraints ensure efficient parameter usage.

3.3.4.Training Algorithm and Optimization

The SparseBonsai model was trained over multiple epochs, with each epoch consisting of a forward and backward pass through the data. In the forward pass, the architecture generated logits and evaluated the classification loss. The loss function was augmented with L2 regularization and sparsity penalties, which constrained both the projection space and the tree parameters for compactness and generalization. Thus, the combined objective function balanced the predictive accuracy with model efficiency. Adam optimizer was used for optimization with adaptive learning rates for different parameters. Backpropagation evaluate gradients with respect to the loss function and model weights were updated iteratively until convergence.

$$\boldsymbol{\theta \leftarrow \theta - \eta \nabla L(\theta)} \qquad \textbf{(8)}$$

In Eq. (8):

- θ represents the model parameters.
- η is the learning rate.
- $\nabla L(\theta)$ is the gradient of the objective function

3.4.Regularization Framework and Sparsity Constraints

Traditional Bonsai and many lightweight classifiers like Bonsai tree classifier employ static regularization where the coefficients (λ, ρ) remain fixed throughout training with sparsity penalties. This method is simple but not optimal. If λ and ρ are too high, the model constrained high, which generates underfitting and reduced the accuracy. Overfitting occurs if they are too low. Here the model retains too many parameters causing overfitting and increased computation resources. But SparseBonsai utilizes dynamic adjustment of λ and ρ during training, based on validation loss and accuracy patterns. This gives two benefits and one of them improves the accuracy. This is achieved with weaker penalties and gradually increasing them. The model first explores high feature representations, then gradually becomes compact. Premature pruning prevented for essential features, which gives higher final accuracy. As sparsity increases on each epoch, redundant weights are pruned. Here inactive branches are suppressed which reduces both memory footprint and inference time.

3.5.Model Evaluation and Performance Metrics

SparseBonsai's performance is evaluated using some classification metrics as following. Classification Accuracy: Under classification accuracy Precision, Recall, and F1-Score evaluated for predictive performance.

$$\boldsymbol{Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}} \qquad \textbf{(9)}$$

$$\boldsymbol{Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}} \qquad \textbf{(10)}$$

$$\boldsymbol{F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}} \qquad \textbf{(11)}$$

Confusion Matrix: This ensures the analysis of model errors and class-wise predictions in a matrix. ROC Curves and AUC: This is the next level evaluation of classification quality at different thresholds.

3.6.Implementation Details

The SparseBonsai model was implemented in PyTorch. The training was executed on an Intel i5 processor with 16 GB of RAM, running in a Python environment. All input features were normalized before the training for the data consistency. The dataset was divided into training (80%) and testing (20%) splits using stratified sampling, which makes the dataset balanced. Training was done using a batch size 64, a learning rate 0.01 with 100 epochs. Hyperparameters like projection dimension (32) and tree depth (3) were taken. To avoid randomness of performance each training experiment was repeated five times using different random seeds. The final classification accuracy of 86.91% is observed as the average performance across these five runs. This averaging ensured it is not influenced by a particular random initialization.

4.RESULTS AND DISCUSSION

The performance of SparseBonsai is evaluated by analyzing classification accuracy and computational efficiency with training in diverse datasets. The evaluation was conducted by comparing SparseBonsai with classical ML and DL models. Precision, recall, F1-score, confusion matrix, and ROC curves are evaluation metrics used to validate the performance of the proposed algorithm. SparseBonsai balances between model complexity and classification performance by dynamically adjusting sparsity constraints, projection dimensions and branching thresholds. The results show the proposed model will be ideal for deployment in resource-constrained IIoT systems.

4.1.Classification Accuracy and Model Performance

SparseBonsai shows an overall classification accuracy of 86.91% in comparison to classical

ML, DL models in terms of performance. The accuracy remains consistent across every epoch. The model also generalizes well to different types of input data. The classification performance is examined using the cross-entropy loss function. The optimized projection layer with dynamic sparsity penalties gives high classification accuracy. It preserved critical feature relationships with minimal computational challenges.

4.1.1.Comparative Accuracy Analysis

The classification accuracy of SparseBonsai is compared with traditional models like:

Decision Trees which are known for interpretability but mostly lack representational power for complex datasets. Support Vector Machines is effective in high dimensional spaces but uses more computational power. Neural Networks (NNs) is highly accurate but resource intensive. The results shows that SparseBonsai outperform than other classical models with a higher accuracy. It offers best performance significantly lower computational resources.

4.2.Evaluation Metrics

SparseBonsai's performance is evaluated using different classification metrics to observe its effectiveness.

4.2.1.Precision, Recall, and F1-Score

Model's performance was evaluated using precision, recall and F1-score and presented in Table 3 using Eq. (9-11). Table 3 presents the per-class precision, recall, and F1 Scores. For the Class 0, the model achieved 86.45% precision, and 83.59% recall, and an F1-score of 84.99%. These results indicate that SparseBonsai maintains balanced precision and recall across all classes.

Table 3 Per-Class Classification Performance of the SparseBonsai Model in Terms of Precision, Recall, and F1-Score.

Metrics	Class 0	Class 1	Class 2
Precision	86.45%	80.59%	86.10%
Recall	83.59%	77.86%	89.38%
F1-Score	84.99%	79.21%	87.71%

4.2.2.Confusion Matrix Analysis

A confusion matrix provides detailed classification results, highlighting the true positives, false positives, false negatives, and true negatives for each class. SparseBonsai's confusion matrix reveals that most of the predictions are attained within the correct class with low classification errors. SparseBonsai minimizes false positives and false negatives by optimized branching functions and regularization penalties. Figure 2 presents the confusion matrix for SparseBonsai on the IIoT test set. Most predictions fall on the diagonal, which gives high classification accuracy. Misclassifications are minimal between Class 0, Class 1, and Class 2, reflecting their similar vibration patterns.

4.2.3.ROC Curve and AUC Score

Receiver Operating Characteristic (ROC) curves are used to evaluate the classification quality for different decision thresholds [23]. ROC-AUC (Receiver Operating Characteristic – Area Under Curve) is achieved by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at varying thresholds using Eq. (12-14).

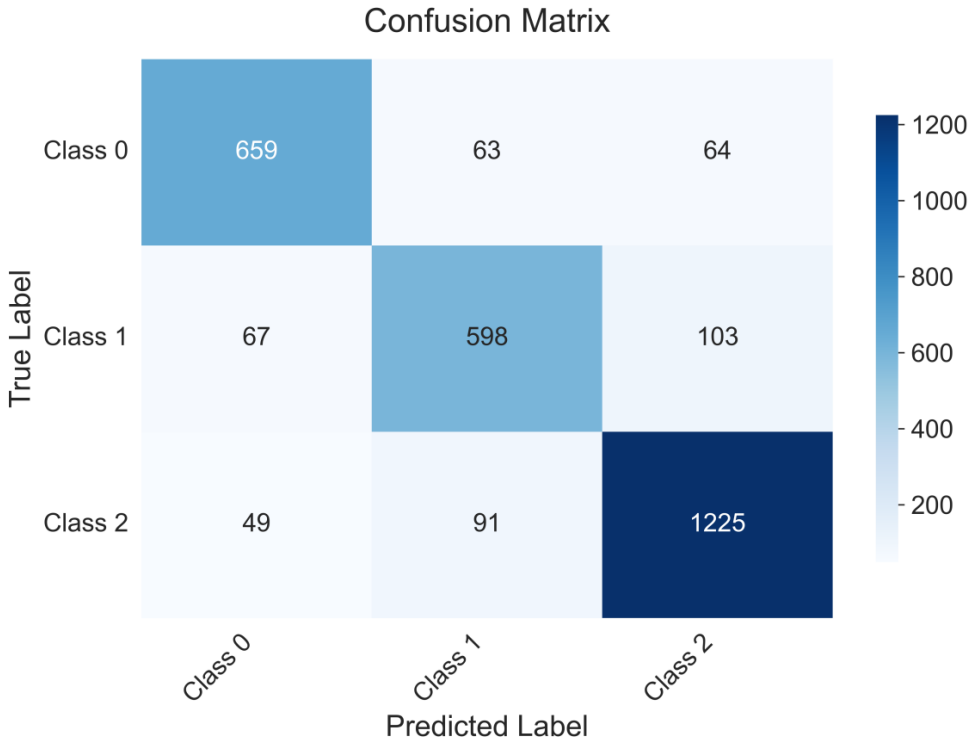

Fig. 2 Confusion Matrix for SparseBonsai: Diagonal Blocks Show High True-Positive Rates Throughout All Classes with Minimal Misclassifications.

$$TPR = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (12)$$

$$FPR = \frac{False\ Positive}{False\ Positive + True\ Negative} \quad (13)$$

$$ROC - AUC = \int_0^1 TPR(FPR)d(FPR) \quad (14)$$

The Area Under the Curve (AUC) shows the overall classification performance, with values closer to 1 indicating higher discrimination between classes. Figure 3 demonstrates ROC curves for the three fault classes. SparseBonsai achieves AUC scores of 0.96 for Class 0, 0.93 for Class 1, and 0.96 for Class 2, with a macro-average AUC of 0.95. These results show that the model performs well across all fault types.

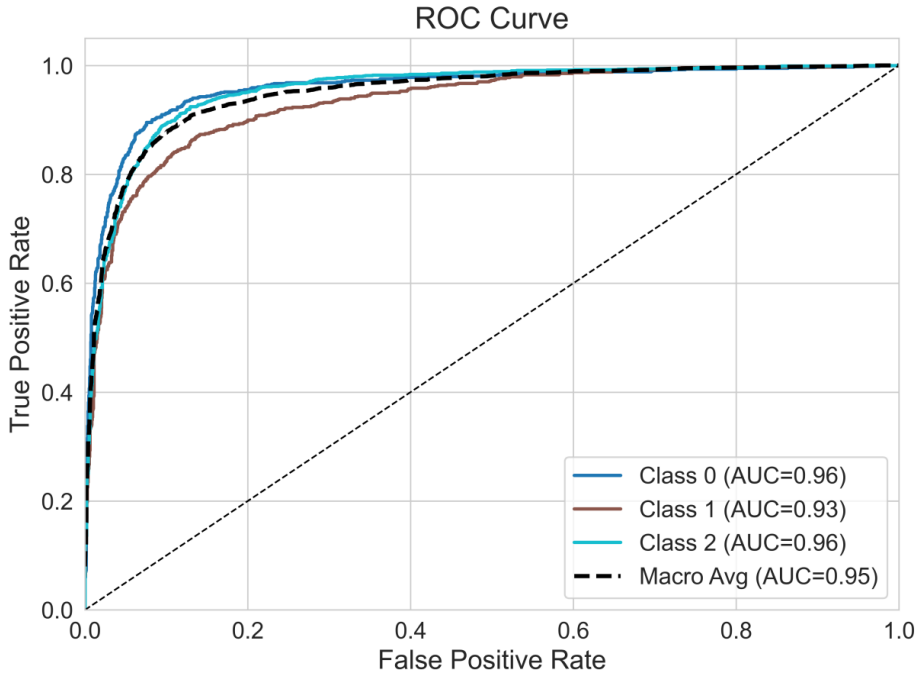

Fig. 3 ROC Curves Illustrating High True-Positive Rates and Low False-Positive Rates Achieved by SparseBonsai Across All Classes.

4.3.Computational Efficiency and Resource Utilization

SparseBonsai's performance is evaluated based on training time, memory consumption, and inference speed.

4.3.1.Memory Footprint and Computational Complexity

SparseBonsai optimizes computational complexity by using a dimensionality reduction technique. It minimizes memory usage while maintaining high accuracy. The computational complexity of SparseBonsai is given by:

$$O(m \times d + m^2 \times h + m \times c) \quad (15)$$

In Eq. (15):

- m is the projection dimension
- d is the original input dimension
- h is the tree depth
- c is the number of classes

SparseBonsai's memory usage is significantly lower than that of conventional models.

4.3.2.Inference Time and Latency

SparseBonsai reduces inference time by optimizing projection with branching functions. It makes faster decisions. The time complexity for inference is:

$$O(\log h) \quad (16)$$

SparseBonsai optimized inference time by approximately 35% compared to other ML and DL models.

4.3.3.Scalability and Edge Deployment

SparseBonsai is applicable for deployment on edge devices in IIoT. These platforms operate under strict energy and latency constraints for applications. It detects faults without depending on remote servers with minimal hardware costs. Energy limitations in battery-powered nodes demand further efficiency improvements [24]. SparseBonsai's lightweight architecture is a promising candidate for scalable IIoT edge deployment.

4.4.Sensitivity Analysis and Hyperparameter Impact

A sensitivity analysis is conducted to assess the impact of essential hyperparameters on SparseBonsai's performance. The study highlights the influence of Projection Dimension, Tree Depth, and Learning Rate.

4.5.Comparative Analysis with Existing Models

SparseBonsai is compared with classical models with multiple parameters. Table 4 presents a comparative analysis of different classification models on the CWRU dataset. Accuracy, computational complexity, and

model size are the primary parameters for comparing other models. The decision tree shows the lowest accuracy (62.5%), having moderate complexity with a relatively large model size of 0.75 MB. Random Forest achieves an accuracy of 70.6% with very high complexity and a model size of 27.54 MB.

Table 4 Comparative Models on the CWRU Dataset Highlighting Accuracy, Complexity, and Model Size.

Classification Model	Test Accuracy (in %)	Model Complexity	Model Size (in MB)
Decision Tree	62.5	Moderate	0.75
Random Forest	70.6	Complex	27.54
XGBoost	86.74	Complex	0.988
LightGBM	86.02	Complex	0.84
DNN	95.5	Complex	1.95
EdgeML Bonsai Tree	78.93	Moderate	0.045
SparseBonsai (Proposed Model)	**86.91**	**Moderate**	**0.048**

Among the architectures, XGBoost and LightGBM achieve higher accuracy of 86.74% and 86.02%, respectively, but exhibit higher computational complexity and larger model sizes. These models show challenges for deployment on microcontrollers or memory constraint devices. The deep neural network (DNN) achieved the highest accuracy (95.5%) but has the largest model size (1.95 MB) and high complexity. Baseline model Bonsai Tree achieved 78.93% accuracy with a low model size of 0.045 MB, demonstrating good efficiency but with a drop in classification quality. The proposed model balances accuracy and efficiency. It achieves 86.91% accuracy, compared to XGBoost and LightGBM. It keeps model complexity moderate and size extremely small (0.048 MB). This makes SparseBonsai suitable for resource-constrained applications, as it provides nearly the same accuracy as gradient boosting but at a fraction of the memory cost.

5.CONCLUSION

SparseBonsai is a resource-efficient variant of the baseline Bonsai algorithm for lightweight, constrained environments. The model achieves 86.91% classification accuracy on the CWRU bearing fault dataset by integrating the projection technique, adaptive sparsity, and dynamic regularization. It maintains a compact memory footprint of only 0.048 MB. Its inference speed is also 35% faster than that of deep neural networks. SparseBonsai provides a 24.4% improvement in accuracy over traditional decision trees and 7.17% gain over the baseline Bonsai algorithm by keeping the model size nearly the same. SparseBonsai achieves a superior balance between accuracy, model size, and inference speed, adaptable in resource-limited environments. Future work may explore automated hyperparameter tuning with incremental learning and dynamic tree growth to enhance adaptability and robustness in diverse IIoT applications. Another direction is to enhance SparseBonsai to expand or contract its tree structure adaptively during training, rather than fixing the depth a priori.

ACKNOWLEDGEMENTS

The authors are grateful to the Institute of Technical Education and Research (ITER), Siksha 'O' Anusandhan University, Bhubaneswar, India, for providing technical assistance and research facilities. This study is part of the research plan under the Department of Computer Science and Engineering.

NOMENCLATURE

A	Accuracy, %
B	Ball fault class
D	Dataset (input data)
IR	Inner Race fault
OR	Outer Race fault
m	Projection dimension
d	Tree depth
Z	Projection matrix
λ	Regularization coefficient
ρ	Sparsity coefficient
η	Learning rate
μ	Mean of features
σ	Standard deviation of features
θ	Model parameters
L	Loss function
N	Number of training samples
t	Epoch number (iteration index)
F1	F1-score metric
P	Precision
R	Recall
AUC	Area Under the Curve
TPR	True Positive Rate
FPR	False Positive Rate
NN	Neural Network
DNN	Deep Neural Network
Greek symbols	
λ	Regularization coefficient (controls L2 penalty strength)
ρ	Sparsity coefficient (controls sparsity constraint)
η	Learning rate used in training
μ	Mean value (used in normalization context)
σ	Standard deviation (used in normalization context)

REFERENCES

[1] Obied H, Al-Taleb MKH, Khaleel HZ, AbdulKareem AF. **Implementation and Derivation Kinematics Modelling Analysis of WidowX 250 6 Degree of Freedom Robotic Arm**. *Journal of Engineering and Sustainable Development* 2025; **29**(4):473–484.

[2] Khaleel HZ, Humaidi AJ. **Towards Accuracy Improvement in the Solution of the Inverse Kinematic Problem in a Redundant Robot: A Comparative Analysis**. *International Review of Applied Sciences and Engineering* 2024; **15**(2):242–251.

[3] Khaleel RZ, Khaleel HZ, Al-Hareeri AAA, Al-Obaidi ASM, Humaidi AJ. **Improved Trajectory Planning for a Mobile Robot Based on the Pelican Optimisation Algorithm**. *Journal Européen des Systèmes Automatisés* 2024; **57**(4):1005–1013.

[4] Behera BB, Mohanty RK, Pattanayak BK. **A Synthesised Architecture and Future Research Directions for Industrial IoT in the Mining Industry**. *Journal of East China University of Science and Technology* 2022; **65**(2):511–528.

[5] Behera BB, Pattanayak BK, Mohanty RK. **Deep Ensemble Model for Detecting Attacks in Industrial IoT**. *International Journal of Information Security and Privacy (IJISP)* 2022; **16**(1):1–29.

[6] Rath M, Pattanayak BK. **Technological Advancements in Modern Healthcare Applications Using the Internet of Things (IoT) and the Proposal of a Novel Healthcare Approach**. *International Journal of Human Rights in Healthcare* 2019; **12**(2):148–162.

[7] Amgbara SI, Akwiwu-Uzoma C, David O. **Exploring Lightweight Machine Learning Models for Personal Internet of Things (IoT) Device Security**. *ResearchGate Preprint* 2024; (24).

[8] Hosenkhan MR, Pattanayak BK. **Security Issues in Internet of Things (IoT): A Comprehensive Review**. *Advances in Intelligent Systems and Computing* 2020; (1030):359–369.

[9] Behera BB, Mohanty RK, Pattanayak BK. **An Ensemble Model for Detecting Attacks in the Industrial Internet of Things (IIoT)**. *NeuroQuantology* 2022; **20**(6):1399–1409.

[10] Alsanad HR, Al Mashhadany Y, Algburi S, Abbas AK, Al Smadi T. **Robust Power Management for a Smart Microgrid Based on an Intelligent Controller**. *Journal of Robotics and Control* 2025; **6**(1):166–176.

[11] Case Western Reserve University. **Bearing Data Center Downloadable Files**. https://engineering.case.edu/bearingdatacenter/download-data-file.

[12] Swain S, Mohanty MN, Pattanayak BK. **Precision Medicine in Hepatology: Harnessing IoT and Machine Learning for Personalised Liver Disease Stage Prediction**. *International Journal of Reconfigurable and Embedded Systems* 2024; **13**(3):724–734.

[13] Habboush AK, Elzaghmouri BM, Pattanayak BK, Pattnaik S, Habboush RA. **An End-to-End Security Scheme for Protection from Cyber Attacks on the Internet of Things (IoT) Environment**. *Tikrit Journal of Engineering Sciences* 2023; **30**(4):153–158.

[14] Abdul Wahab AW, Idris MYI, Hussain MA. **Classifier Performance Evaluation for Lightweight IDS Using Fog Computing in IoT Security**. *Electronics* 2021; **10**(14):1633.

[15] Li H, Xia Dou Y. **Resource Optimisation in Smart Electronic Health Systems Using IoT for Heart Disease Prediction via Feedforward Neural Networks**. *Cluster Computing* 2025; **28**:21.

[16] Tanveer MU, Munir K, Amjad M, Alyamani HJ. **LightEnsemble-Guard: An Optimised Ensemble Learning Framework for Securing Resource-Constrained IoT Systems**. *IEEE Access* 2025; **13**:101764–101781.

[17] Daghero F, Burrello A, Macii E. **Dynamic Decision Tree Ensembles for Energy-Efficient Inference on IoT Edge Nodes**. *IEEE Internet of Things Journal* 2023; **11**:742–757.

[18] Qiu T, Zhang M, Liu J, Chen C, Liu X. **A Directed Edge Weight Prediction Model Using Decision Tree Ensembles in Industrial Internet of Things**. *IEEE Transactions on Industrial Informatics* 2021; **17**:2160–2168.

[19] Al Smadi T, Al Sawalha A, Pattanayak BK, Al Smadi K, Habboush AK. **Energy-Efficient Storage System Optimisation and Recent Trends in Enhancing Energy Management and Access Microgrid: A Review**. *Journal of Advanced Sciences and Engineering Technologies* 2024; **7**(1):39–54.

[20] Al Smadi T, Gaeid KS, Mahmood AT, Hussein RJ, Al-Husban Y. **State-Space Modelling and Neural-Network-Based Control for Power-Plant Electrical Faults**. *Results in Engineering* 2025; **25**:104582.

[21] Kumar A, Goyal S, Varma M. **Resource-Efficient Machine Learning in 2 KB RAM for the Internet of Things**. *Proceedings of the 34th International Conference on Machine Learning (ICML)* 2017; **70**:1935–1944.

[22] Naveen S, Kounte MR. **Machine Learning at Resource-Constrained Edge Device Using Bonsai Algorithm**. *Proceedings of the 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)* 2020.

[23] Al-Sharo YM, Al Smadi K, Al Smadi T. **Optimization of Stable Energy PV Systems Using the Internet of Things (IoT)**. *Tikrit Journal of Engineering Sciences* 2024; **31**(2):45–54.

[24] Mohanty MN, Satrusallya S, Al Smadi T. **Antenna Selection Criteria and Parameters for IoT Application**. *Printed Antennas* 2022; **18**:283–295.