Enhancing Predictive Maintenance Method Using Machine Learning to Improve IoT-Embedded Machinery Efficiency and Performance

Abiinesh Nadarajan, Iskandar Ishak, Noridayu Manshor, Raihani Mohamed, Mohamad Yusnisyahmi Yusof
Department of Computer Science-Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia

***Abstract*—Predictive maintenance plays a crucial role in minimizing unplanned downtimes, reducing maintenance costs, and optimizing the operational efficiency of IoT-embedded industrial machinery. Despite its transformative potential, traditional predictive maintenance methods often face challenges such as limited accuracy, high latency, and inefficiencies in processing large and imbalanced datasets. This study proposes an enhanced predictive maintenance method using the Sliding Window Method with XGB model (E.XGB), incorporating advanced data preprocessing, permutation importance, and hyperparameter optimization to address these limitations. The proposed method was evaluated on two datasets, which are the synthetic AI4I 2020 Predictive Maintenance Dataset and the real-world CNC Milling Dataset. A comparative analysis with a predictive maintenance method using E.AB from prior research as a benchmark, along with several baseline models, DT, RF, and SVM, revealed that the E.XGB model consistently outperformed other methods in accuracy, precision, recall, and F1-scores. On the AI4I2020 dataset, the E.XGB model achieved an accuracy of 99.05%, while on the CNC Milling dataset, it attained an accuracy of 99.01%. Additionally, the E.XGB model also demonstrated reduced training and prediction times, meeting the real-time requirements of industrial applications. The proposed model demonstrated training speed of approximately 94% and prediction speeds of approximately 99.8% improvement over the E.AB model, making it highly suitable for real-time industrial applications. By improving accuracy, training speed, and prediction latency, the predictive maintenance method offers a robust, scalable, and reliable solution for predictive maintenance across diverse industrial contexts.**

Keywords*—Internet of Things; machine learning; predictive maintenance

I. Introduction

Industrial maintenance has evolved from reactive strategies to predictive maintenance, a proactive approach utilizing machine learning and IoT technologies. Traditional methods are often costly, inefficient, and prone to unplanned downtimes due to manual feature extraction and inaccurate degradation modelling [1, 2]. Predictive maintenance, however, predicts potential failures and optimizes schedules, allowing for timely interventions without detailed knowledge of degradation processes [3]. This approach offers significant benefits, such as reduced downtime, extended equipment life, and enhanced sustainability [4]. However, implementing predictive maintenance remains challenging due to the complexity of industrial systems and the intricate data patterns requiring analysis [5]. These challenges highlight the need for advanced methods that enhance accuracy and processing efficiency. By leveraging IoT and machine learning technologies, this study addresses these limitations, focusing on adaptability and scalability to meet the demands of diverse industrial applications.

The main problem that motivates this research is that current predictive maintenance methods still face challenges in achieving fast training speeds and prediction speeds while maintaining high accuracy, which is essential for real-time industrial applications [1, 6]. Due to massive sensor data generated by IoT-embedded machines, many existing methods rely on complex machine learning models to predict faults. Although this is effective and accurate, it leads to prolonged training times and higher prediction [7]. Having long training time can lead to delayed model updates, causing adaptability and scalability issues, while long prediction time causes delayed fault detection, which is important for real-time industrial machines. This study addresses these limitations by proposing an Enhanced Predictive Maintenance Method using the Sliding Window Method with XGB model (E.XGB) on sensor data, which can optimize training speeds and lower prediction latency while still maintaining a high accuracy [5]. By addressing these gaps, the study aims to enhance current predictive maintenance methods to satisfy the real-time requirements of industrial machinery.

The goal is to improve fault prediction accuracy, minimize false alarms, and reduce unnecessary maintenance and downtime. Faster training speeds and real-time predictions are also prioritized to support scalability and timely decision-making in industrial operations. The research focuses on IoT-driven machinery, using the AI4I 2020 Predictive Maintenance Dataset [8] and CNC Milling Dataset [9]. The proposed E.XGB model is evaluated against other machine learning methods like AdaBoost, Decision Tree, Random Forest, and Support Vector Machine. Performance is measured using metrics like accuracy, precision, recall, F1 score, ROC AUC, and prediction latency, with results benchmarked against existing approaches. This study's significance lies in improving fault detection accuracy, reducing training times, and enabling real-time predictions, which together lower downtime and maintenance costs. By bridging the gap between machine learning advancements and practical industrial applications, the research provides a robust foundation for future predictive maintenance innovations.

The rest of this study is organized as follows: Section II reviews related work on predictive maintenance, including algorithms, and its performance for IoT-based industrial systems. Section III details the methodology, including data preparation, model training, and optimization techniques. Section IV explains the proposed E.XGB method, supported by flowcharts, formulas, and calculations. Section V presents experimental results, comparing the proposed model's performance to previous methods. Section VI summarizes the study's contributions and provides suggestions for future research.

II. Related Works

In recent years, predictive maintenance for IoT-embedded industrial machinery has garnered significant attention. In [1], the authors proposed a predictive maintenance method that utilizes machine learning algorithms, particularly AdaBoost, to classify machine stops in knitting machines. Their system achieved 92% accuracy, enabling timely maintenance and improved efficiency in the textile industry. In terms of hyperparameter tuning, they employed grid search cross-validation (GridSearchCV) for hyperparameter optimization. However, the resulting output possessed high latency in processing and may hinder its application in real-time systems.

The challenge of imbalanced datasets is tackled by enhancing the KNN algorithm [10]. By employing techniques like feature engineering, standardization, and under-sampling, their model achieved a 97.1% accuracy. The ability to handle imbalanced data is crucial in predictive maintenance, although KNN's performance can vary based on dataset characteristics. The approach also explicitly mentions hyperparameter tuning and states that Grid Search tuning was applied to optimize the K-Nearest Neighbours (KNN) model. However, it does use an automated hyperparameter tuning method.

Similarly, in [11], the authors proposed a framework combining XGB and LOF models with XAI to address imbalanced datasets. Their method achieved 96% accuracy with XGB and 91% with LOF. The dual approach helps by focusing on predictive classification with XGB and anomaly detection with LOF. However, the study highlighted potential issues with the synthetic noise introduced by SMOTE and scalability challenges in real-time applications. However, no explicit mention of hyperparameter tuning was found in this study.

Multiple machine learning models including LR, KNN, and ANN, for failure prediction [12]. The ANN model outperformed others with an accuracy of 96.85%, but it struggled with false positives, indicating a need to balance sensitivity and specificity in complex environments. In this approach, structured hyperparameter tuning methods that include GridSearchCV and RandomizedSearchCV were used in their work.

LSTM networks combined with Auto Encoder is employed to predict tool wear in CNC milling machines [13]. Their model achieved 98% accuracy, showing improved performance over previous methods. However, long datasets could reduce the model's accuracy, and the authors recommended prioritizing relevant data through a weighting mechanism. Also, there was no hyperparameter tuning approach mentioned in the approach.

In another deep learning approach, reinforcement learning techniques like Q-learning and SARSA alongside LSTM networks is applied to monitor CNC tool conditions [14]. The SARSA algorithm achieved 98.66% accuracy. While reinforcement learning optimizes decision-making, the complexity of the model posed challenges in training, requiring large datasets and still misclassifying certain instances. Again, in this approach, no mention of hyperparameter tuning is involved.

Machine learning and deep learning methods for tool wear prediction were compared, showing that CNN and AE-LSTM outperformed KNN [15]. While deep learning models are more accurate, they are computationally intensive, raising concerns about their scalability. In this approach, there was no mention about the usage of hyperparameter tuning.

The use of SVM for detecting machine failures in predictive maintenance is compared in [16], achieving up to 88% accuracy in testing data. In terms of hyperparameter tuning, this approach employs grid search for selecting optimal hyperparameters (C and Gamma). However, the SVM's reliance on parameter tuning, such as C and gamma, can slow optimization, making it unsuitable for real-time applications.

LSTM models are used for predictive degradation modelling in milling machines, achieving 80% accuracy [17]. The advantage of this method lies in its ability to predict degradation by considering sequential dependencies. However, it struggled with imbalanced data and may not account for all failure modes, especially in more complex systems. On real-time systems, [2] integrated IoT and machine learning for predictive maintenance, using algorithms like SVM, RNN, and CNN. Their method achieved 87% accuracy with SVM on a milling dataset and 98% with CNN on a bearing dataset. While the models performed well, processing large datasets and extracting meaningful features for real-time predictive maintenance posed challenges.

A hybrid deep learning model combining CNN and LSTM for fault detection is proposed in [18], achieving 97.9% accuracy. The model's ability to capture both spatial and temporal patterns in multivariate sensor data made it effective for fault detection. However, its complexity and need for substantial annotated data slow down its real-time applicability.

A PM-C-LSTM model is used for wind turbines, achieving a 96.77% accuracy [19]. While promising for wind energy applications, the model's effectiveness across different environments requires further validation. Despite the advancements, the reviewed works reveal several gaps in predictive maintenance research. A significant issue is the trade-off between accuracy and processing speed. Deep learning models, like CNN-LSTM and hybrid models, achieve high accuracy but come with high computational costs, which may limit their use in real-time applications.

Models like AdaBoost [1] and SVM [16] also face challenges in terms of processing speed, as they require substantial parameter tuning, slowing down optimization. Managing imbalanced datasets remains another challenge.

While techniques like SMOTE and under-sampling improve model performance [10, 11], they may introduce synthetic noise or fail to generalize effectively. Furthermore, models may perform poorly when failure events are rare, highlighting the need for better methods to manage imbalanced data without sacrificing accuracy.

In summary, the research on predictive maintenance for IoT-embedded industrial machinery has made significant strides, but challenges in accuracy, processing speed, and handling imbalanced data persist. Moving forward, research should focus on optimizing the balance between accuracy and speed, improving the generalization of models across diverse datasets, and finding better methods to handle imbalanced data while avoiding the introduction of noise. Addressing these issues will pave the way for more efficient, dependable, and scalable predictive maintenance systems that can meet the demands of real-world industrial environments.

III. Methodology

A. Experimental Setup

This study evaluates the proposed Enhanced XGBoost (E.XGB) model for predictive maintenance by comparing it with the method by [1] and several baseline models. The E.XGB model builds upon the approach in [1] for IoT industrial applications, improving against the AB model with the proposed XGB model by incorporating minor enhancements. The method proposed by [1] is replicated in this study using the same configurations as the original study to be used as a comparison. The replicated Elkateb method, referred to as Enhanced AdaBoost (E.AB), is used as the primary benchmark. Additionally, three baseline models, DT, RF, and SVM, will serve as secondary benchmarks, using default parameters without optimization. All models will be trained and tested on both imbalanced and balanced datasets to ensure comprehensive evaluation. Performance is measured using metrics such as accuracy, precision, recall, F1-score, confusion matrices, ROC-AUC, training times, and prediction times. This study aims to address the challenges in the method in [1] and demonstrate the effectiveness of the proposed E.XGB model.

B. System Infrastructure

The predictive maintenance system for IoT-embedded machinery integrates real-time data collection, cloud-based processing, and machine learning to predict and prevent equipment failures. Fig. 1 illustrates the system's workflow in an industrial setting.

Sensors embedded in machines monitor critical parameters like vibration, temperature, speed, torque, and tool wear during operation. These sensors transmit real-time data to the cloud via secure, high-speed connections. The cloud provides the necessary computational resources to process large volumes of data. In the cloud, data undergoes preprocessing to clean and organize it, ensuring it is accurate and ready for analysis. Machine learning models then analyze this data to detect patterns, trends, and anomalies. Trained on historical data, these models predict potential faults by identifying early warning signs, such as abnormal temperature increases or irregular torque patterns. When a potential issue is detected, the system generates alerts for maintenance teams through dashboards, notifications, or emails. Alerts specify the affected machine or component, the type of fault, and its urgency, enabling technicians to address problems proactively. By preventing unexpected breakdowns, this system reduces downtime, cuts repair costs, and extends machine life. As more data is collected, the system improves its accuracy, making predictive maintenance an efficient and reliable alternative to traditional maintenance practices.

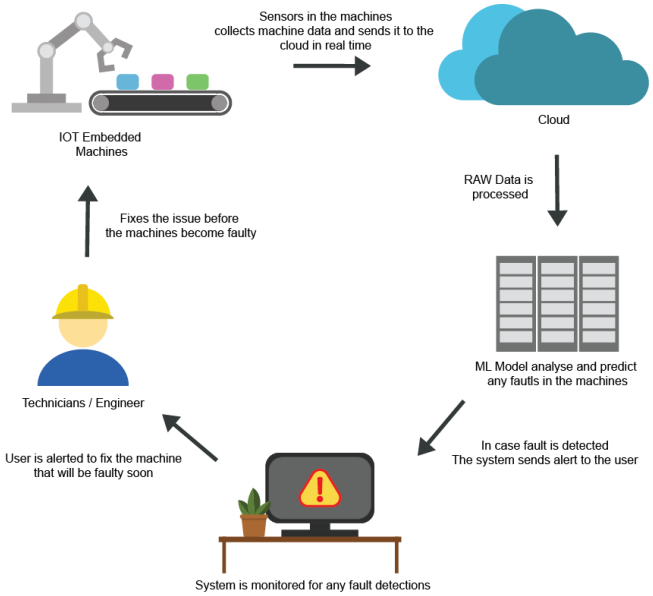

Fig. 1. System overview.

C. Methodology

This study follows the CRISP-DM methodology, a structured and iterative framework ideal for developing predictive maintenance systems [18]. Fig. 2 illustrates the entire methodology overview. It begins with data collection, using existing datasets to ensure high-quality and representative data. In data preprocessing, key steps include data cleansing, addressing imbalances, data smoothing, standardization, and splitting data into training (80%) and testing (20%) sets.

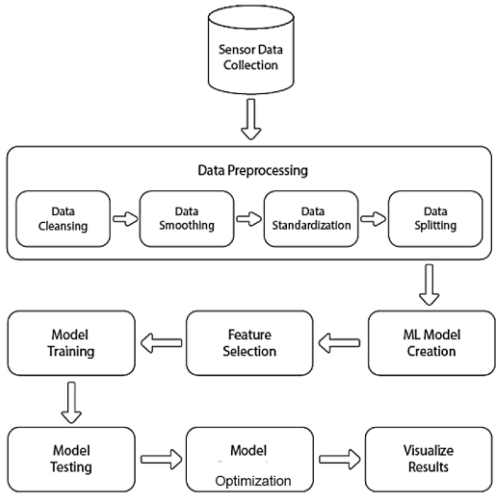

Fig. 2. Methodology overview.

During the model creation phase, the study replicates a prior model and develops three additional benchmark models. Feature importance is analyzed before training begins. The model testing phase evaluates performance on the test set using metrics like accuracy, precision, and F1 score to assess generalization capability. The proposed E.XGB model is then optimized through techniques like hyperparameter tuning and cross-validation to enhance accuracy and efficiency. Finally, visualizing results with tools such as confusion matrices and ROC curves provides insights into model performance and supports transparent communication with stakeholders.

D. Technical Setup and Development Environment

The simulation for this study was developed in Visual Studio Code (VS Code), a popular, free, and open-source code editor known for its versatility and extensive support for plugins [20]. The programming language used is Python, widely recognized for its effectiveness in data science and machine learning due to its powerful libraries [21].

Several Python libraries were employed for simulation. Pandas was used for data cleaning, preparation, and transformation [22]. Scikit-learn facilitated data pre-processing, machine learning model imports, feature selection, and performance metrics calculations. The Time library helped track training and processing time, while Matplotlib and Seaborn were used for visualizations.

The simulations were run on a system with the following specifications:

- Processor: Intel(R) Core (TM) i7-8750H CPU
- Memory: 32 GB RAM
- Operating System: Microsoft Windows 11 Home Single Language.

E. Dataset

Datasets are a structured collection of data used for analysis, modelling, and decision-making in various fields such as machine learning, statistics, and research. One of the key limitations of the previous approach is its reliance on a single dataset for experimentation [1, 2, 10, 11, 12, 13, 14, 15, 16, 19]. This restricts the generalizability of the model, as its performance has not been validated across diverse IoT environments with varying sensor data and operational conditions. In the context of IoT-based predictive maintenance, different datasets may exhibit variations in data distribution, sensor noise, and failure patterns, which a single dataset cannot fully capture. Without evaluation on multiple datasets, the robustness of the method remains uncertain, increasing the risk of overfitting to dataset-specific characteristics. Therefore, in our experiment, we use two datasets to enhance the generalizability and robustness of our proposed approach, ensuring its applicability across different IoT environments. The datasets used are the AI4I 2020 Dataset [8] and the CNC Milling Dataset [9], with each representing synthetic and real-world data, respectively. Additionally, these datasets exhibit both imbalanced and balanced characteristics, allowing us to assess the model's ability to handle class imbalance issues, which are common in real-world predictive maintenance applications.

1) AI4I2020 dataset: The AI4I 2020 Predictive Maintenance Dataset [8] is a synthetic dataset designed to replicate real-world industrial predictive maintenance data. It is widely used as a benchmark for machine learning algorithms, particularly for handling imbalanced datasets in predictive maintenance, quality assurance, and anomaly detection tasks. The dataset combines time-series and static data, including sensor readings, control signals, and process parameters, making it a valuable resource for developing predictive maintenance methods. The dataset contains 10,000 rows with two IDs, which are UID and Product ID, six features, and six failure modes. The features include product type, air temperature [K], process temperature [K], rotational speed [rpm], torque [Nm], and tool wear [min]. The six failure modes are machine failure, tool wear failure (TWF), heat dissipation failure (HDF), power failure (PWF), overstrain failure (OSF), and random failures (RNF).

2) CNC milling dataset: The CNC Milling Dataset from the University of Michigan SMART Lab [9] is a real-world dataset based on machining experiments conducted on a CNC milling machine. Data was collected from four machine motors, X, Y, Z axes, and spindle at a sampling rate of 100ms during 18 experiments on wax blocks. The dataset includes 25,286 rows with 55 features, including 1 ID and 3 failure modes as tool_condition, machining_finalized, and passed_visual_inspection. This study will use data from all 18 experiments but focus on selected features to challenge the model with limited input data. The tool_condition feature will be the target variable. Unlike the synthetic and imbalanced AI4I 2020 dataset, this balanced dataset provides an opportunity to evaluate the model's performance in a real-world environment, enhancing the robustness of the experiment.

IV. Enhanced Predictive Maintenance Method Using XGB

The proposed enhanced predictive maintenance method that is used in this study is designed based on the [1] approach to predictive maintenance for IOT industrial applications. However, the core model is modified along with several other minor changes in this study to further enhance the method proposed by [1]. This study's proposed enhanced predictive maintenance method aims to improve by eliminating some of the challenges present in that method.

Fig. 3 illustrates the flowchart of the entire enhanced predictive maintenance method. The flowchart provides a detailed pipeline for predictive fault detection using sensor data and an E.XGB model. This end-to-end process is designed to systematically prepare the data, refine feature selection, optimize the model, and deliver accurate predictions for identifying faults. By integrating feature importance methods, data preprocessing, and hyperparameter optimization, the pipeline aims to achieve reliable and efficient results, particularly for predictive maintenance applications.

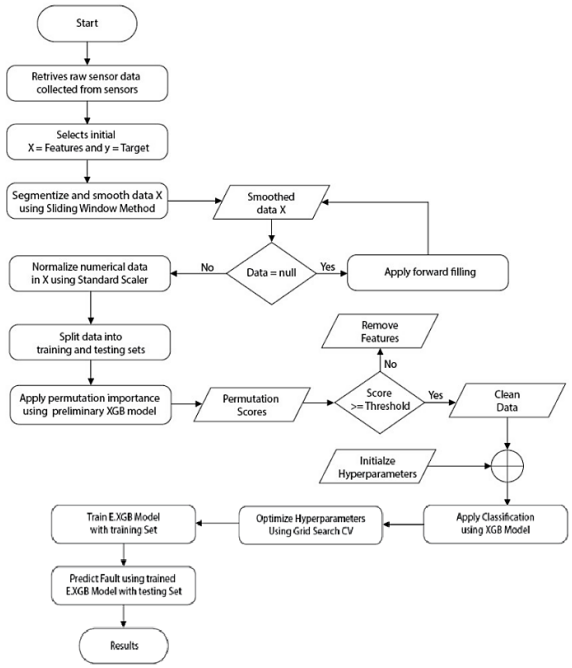

Fig. 3. Enhanced predictive method overview.

A. Data Pre-Processing

Pre-processing is the first and critical step in the machine learning workflow to ensure that the data is clean, consistent, and suitable for analysis. Raw data is often messy, incomplete, or unstructured, which can lead to inaccurate or inefficient model training. Preprocessing helps address these issues by transforming the data into a format that models can understand and learn effectively. After receiving the data collected from the sensors and the initial features x and target y selection is done, several steps are gone through to pre-process the data. Detailed information on how each pre-processing step is done is described in the sections below.

B. Sliding Window Method

One of the key enhancements in our proposed approach is the introduction of a data smoothing process using the Sliding Window Method. This technique is particularly well-suited for preprocessing IoT data, including streaming data, as demonstrated in [23]. By applying the Sliding Window Method to raw sensor data, we effectively smooth fluctuations and manage missing values, improving data consistency. The method segments the data into fixed-sized windows and computes the mean within each window, ensuring a more stable and reliable representation of the underlying patterns in the dataset.

The mean within each window starting at index t is computed as:

$$\mu_t = \frac{1}{K} \sum_{i=t}^{t+K-1} x_i$$

where,

μt is the mean of the window starting at position t.

Xi represents the data points in the window.

k is the number of elements in each window.

The summation runs from i=t to t+k−1

The missing values in the dataset are fixed using the forward filling method.

TABLE I. SLIDING WINDOW PARAMETERS

Parameter	Method	Inplace	Rolling	Window	Minimum required data points
Range	Forward filling	true	mean	3	1

Table I summarizes the parameters used for the Sliding Window Method in the proposed E.XGB approach. The forward fill (ffill) method replaces missing values with the last valid entry, ensuring data continuity. With *inplace* set to true, changes are made directly to the dataset. A rolling window of size 3 computes the mean over three consecutive values, moving forward one step at a time. The minimum required data points parameter is set to 1, ensuring that the mean is calculated even if some values are missing, as long as at least one valid value exists. This method smooths fluctuations while effectively handling missing data.

C. Data Normalization

The next step for data preprocessing in the proposed method is data scaling. During the data scaling stage, numerical data in the dataset are selected to undergo normalization. For this, the StandardScaler method is imported from Scikit-Learn's preprocessing library. The StandardScaler uses the z-score normalization technique to normalize the data. The z-score normalization technique is applied to each data point Di in category j, which calculates the number of standard deviations each point is away from the mean of its category. The z-score normalization is defined as:

$$D_j^{i'} = \frac{D_j^i - \mu_j}{\sigma_j}$$

where,

$D_j^{i'}$ is the normalized data point for feature i in category j,

D_j^i is the original data point in category j,

μ_j is the mean of data category j,

σ_j is the standard deviation of data category j.

Normalization ensures that all features are on a similar scale, centered at the origin, with a peak of 1. This improves the stability of training by reducing variance and the impact of outliers. This method ensures that features have a mean of 0 and a standard deviation of 1, making the model more robust to outliers.

D. Data Splitting

After normalization, the dataset is split into training and testing sets to ensure the model trains on one subset and is tested on unseen data, reducing bias. Without splitting, testing on the same dataset used for training could lead to misleading results.

TABLE II. DATA SPLIT PARAMETERS

Parameter	Train size	Test size	Random state
Value	0.8	0.2	42

The train_test_split function from Scikit-Learn's model selection library is used for this purpose, with parameters specified in Table II. For this study, 80% of the data is allocated for training and 20% for testing. The data is shuffled randomly before splitting, with a random state of 42 set to ensure reproducibility.

E. Feature Selection

After splitting the data, feature selection is performed using permutation importance, a technique that evaluates the significance of each feature by shuffling its values and observing the impact on model performance, with a significant drop in accuracy indicating importance. The permutation importance function from Scikit-Learn's inspection library is applied to the preliminary XGB model and dataset to calculate a reference score. The importance i_j for feature f_j is calculated as follows:

$$i_j = s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j}$$

where,

K is the total number of data points in the dataset,

k is the k^{th} datapoint.

For each feature, the values are shuffled to create a distorted dataset, and the model is re-scored to determine feature importance based on the performance drop. The process uses the number of repeats parameter to specify the number of shuffles, where higher values provide more reliable scores but increase computation time.

TABLE III. PERMUTATION IMPORTANCE PARAMETERS

Parameter	Number of estimators	Maximum depth	Learning rate	Number of repeats
Value	100	10	0.05	10

Table III shows the importance of the permutation parameters used. The initial XGB model is configured with key hyperparameters of number of estimators to 100 trees, maximum depth to 10 to balance complexity and overfitting, and learning rate of 0.05, to control tree contribution. Features with low importance scores, based on a predefined threshold, are removed to streamline the dataset, retaining only the most relevant features and improving model efficiency and performance.

F. XGB Classification

Once data preprocessing is complete, the core XGB model is initialized using the XGBoost classifier from the *xgboost* library. XGB is a highly efficient implementation of the gradient boosting framework, known for its speed, accuracy, and flexibility in handling structured data. It builds an ensemble of decision trees sequentially, optimizing each tree to correct errors from the previous ones using gradient descent to minimize the loss function. The training process starts with an initial prediction, often the mean of the target values. The objective function combines a loss function to measure prediction errors and a regularization term to prevent overfitting. The function is:

$$Obj(\theta) = \sum_{i=1}^{m} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where,

$L\ (y_i, \hat{y}_i)$ is the loss function, which is the difference between true and predicted values,

$\Omega\ (f_k)$ is the regularization term for the weak learner f_k,

m is the number of data points.

K is the number of trees.

At each iteration, the model adds a new tree, updating the prediction. The final prediction is the sum of all tree predictions. XGB excels at handling large datasets and complex relationships efficiently. Its combination of speed, accuracy, and regularization makes it a top choice for structured data and a frequent winner in machine learning competitions.

G. Hyperparameter Optimization Using GridSearchCV

GridSearchCV is used to fine-tune the hyperparameters of the XGB model by testing all possible combinations of predefined values to find the best set. It evaluates each combination using 3-fold cross-validation, ensuring the model performs well on unseen data while avoiding overfitting. This process improves both accuracy and robustness. Table IV shows the ranges for key hyperparameters set and Table V shows the GridSearchCV parameters used.

TABLE IV. XGB HYPERPARAMETER SEARCH RANGE

Parameter	Number of estimators	Maximum depth	Learning rate	Sub sample
Range	50 - 200	1 - 10	0.01 - 0.07	0.01 - 0.07

TABLE V. GRIDSEARCHCV PARAMETERS

Parameter	Model	Scoring	CV
Value	XGBClassifier (eval_metric='logloss')	accuracy	3

For this study, GridSearchCV is imported from Scikit-Learn's model selection library and applied to the XGB classifier, initialized with an evaluation metric set based on the Logarithmic Loss value. The search grid includes ranges for key hyperparameters such as the number of estimators at 50 to 200, maximum depth at 1 to 10, learning rate at 0.01 to 0.07, and subsample at 0.01 to 0.07. These parameters influence the model's performance, and GridSearchCV compares average accuracy scores across all combinations to select the best one. The optimal hyperparameters are then used to build the final XGB model.

V. RESULTS AND DISCUSSION

This section summarizes the experimental results on the AI4I2020 dataset and CNC Milling dataset, comparing various

machine learning models across key metrics such as accuracy, precision, recall, F1-score, ROC-AUC, confusion matrices, training time, and prediction time. The results are organized into tables and visualized with figures to provide a comprehensive view of model performance and to facilitate comparisons.

A. Pre-processing on AI4I2020 Dataset

The AI4I2020 dataset underwent a series of preprocessing steps to ensure quality and consistency. Initially, the dataset was checked for missing values, and no null entries were found. Irrelevant features such as UDI, Product ID, and Type were removed, as they did not contribute to machine failure predictions, leaving air temperature, process temperature, rotational speed, torque, and tool wear as the selected features. The study also focused on predicting overall machine failures rather than specific failure types, so failure mode features like TWF, HDF, and PWF were excluded.

Fig. 4. Data balance in AI4I2020 dataset.

As shown in Fig. 4, the dataset revealed a significant class imbalance, with only 3.4% of instances labelled as machine failures. The data was then smoothed and segmented using a sliding window method. To address the varying scales of features, standardization was applied, ensuring all features had a mean of 0 and a standard deviation of 1, making them comparable and enhancing model performance. Finally, the data was split into 80% training and 20% testing subsets, preparing it for machine learning tasks.

TABLE VI. PARAMETER OPTIMIZATION RESULT ON AI4I2020 DATASET

Parameter	Number of estimators	Maximum depth	Learning rate	Sub sample
Optimum	200	8	0.07	0.5

Hyperparameter optimization for the XGB model was performed using GridSearchCV, resulting in an enhanced model configuration. The optimal parameters are shown in Table VI, which includes the number of estimators set to 200, maximum depth set to 8, learning rate set to 0.07, and subsample set to 0.5. These parameters were carefully chosen to balance model complexity and generalization, ensuring robust performance. A higher maximum depth allowed the model to capture more complexity without overfitting, while a smaller learning rate enabled gradual and stable learning. The subsample value helped mitigate overfitting by training the model on random subsets of data at each iteration. With these optimized hyperparameters, the E.XGB model achieved significant improvements in metrics such as accuracy, precision, recall, and F1-score, demonstrating its suitability for predicting machine failures in the AI4I2020 dataset.

B. Model Results on AI4I2020 Dataset

The classification performance of various models on the AI4I2020 dataset is summarized in Fig. 5 and Table VII, comparing E.XGB, E.AB, RF, DT, and SVM based on accuracy, precision, recall, F1-score, ROC AUC, training time, and prediction time. Among all models, the proposed E.XGB model outperforms the others, achieving the highest accuracy of 99.05%, precision of 93.62%, F1-score of 82.25%, and ROC AUC of 97.10%.

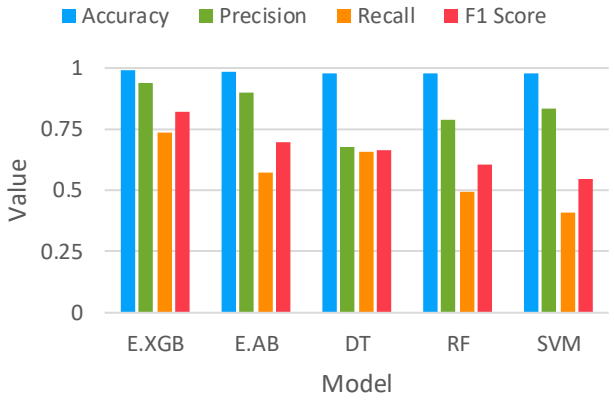

Fig. 5. Efficiency results on CNC AI4I2020 dataset.

Compared to E.AB, RF, DT, and SVM, the E.XGB demonstrates superior predictive capability while maintaining efficiency. Although DT achieves a recall of 65.57%, slightly lower than E.XGB's 73.33%, it lags in precision at only 67.80%, making it less reliable overall.

TABLE VII. PERFORMANCE METRICS OF MODELS ON AI4I2020 DATASET

Details	E.XGB	E.AB	DT	RF	SVM
Accuracy (%)	99.05	98.50	98.00	98.05	97.95
Precision (%)	93.62	89.74	67.80	78.95	83.33
Recall (%)	73.33	57.38	65.57	49.18	40.98
F1-Score (%)	82.25	70.00	66.67	60.61	54.95
ROC AUC (%)	97.10	94.64	87.26	86.63	85.35
Training Time (s)	0.2102	3.9122	0.0312	0.4531	0.4343
Prediction Time (s)	0.0001	0.0544	0.0001	0.0189	0.0937

In terms of computational performance, E.XGB has a training time of 0.2102s, significantly faster than E.AB at 3.9122s and RF at 0.4531s, while being slightly slower than DT at 0.0312s. For prediction time, E.XGB and DT are the fastest at 0.0001s, making them highly efficient for real-time applications. Although DT is computationally fast, it suffers from lower precision and an overall weaker predictive performance. Overall, for the AI4I2020 dataset, E.XGB proves to be the most effective model, offering the best trade-off between accuracy, precision, recall, and efficiency. DT, despite

being computationally efficient, falls short in terms of precision, making E.XGB is the most suitable choice for predictive maintenance applications in IoT-embedded machinery.

C. Pre-processing on CNC Milling Dataset

The CNC Milling Dataset was pre-processed to ensure high-quality data for analysis. Missing values were checked and found to be absent, confirming data completeness. Feature selection was then performed, and a set of relevant features was chosen based on preliminary importance tests. The selected features included x1_actualposition, y1_actualposition, z1_actualposition, s1_actualposition, and m1_current_feedrate, with tool condition as the target feature, which directly relates to machinery degradation.

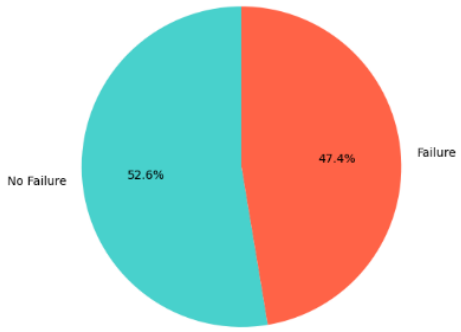

Fig. 6. Data balance in CNC milling dataset.

Unlike the AI4I2020 dataset, the CNC Milling dataset exhibited a balanced class distribution, as shown in Fig. 6, with 47.4% labelled as machine failures. The data is then smoothed and segmented using the sliding window method. The dataset also revealed significant differences in feature scales, with features like X1_ActualPosition spanning wide ranges. To address this, all features were standardized, ensuring equal contribution to model performance. The final step was splitting the data into 80% training and 20% testing sets, preparing it for model training and evaluation.

TABLE VIII. PARAMETER OPTIMIZATION RESULT ON CNC MILLING DATASET

Parameter	Number of estimators	Maximum depth	Learning rate	Sub sample
Optimum	200	10	0.07	0.7

GridSearchCV was applied to the base XGB model to find the optimal hyperparameters. The best combination of parameters found is as shown in Table VIII, where the number of estimators is set to 200, the maximum depth set to 10, the learning rate set to 0.07, and the subsample set to 0.7. These parameters were chosen to strike a balance between model complexity and generalization. The maximum depth of 10 allowed the model to capture sufficient complexity, while the learning rate of 0.07 ensured gradual learning. The subsample value of 0.7 helped reduce overfitting by training the model on a randomly selected subset of the data. These hyperparameters improved the model's performance, as evidenced by better evaluation metrics such as accuracy, precision, recall, and F1-score.

D. Model Results on CNC Milling Dataset

The classification performance of various models on the CNC Milling dataset is summarized in Fig. 7 and Table IX, comparing E.XGB, E.AB, RF, DT, and SVM based on accuracy, precision, recall, F1-score, ROC AUC, training time, and prediction time. Among these models, the proposed E.XGB demonstrates superior performance, achieving the highest accuracy of 99.01%, precision of 99.57%, recall of 98.30%, F1-score of 98.93%, and ROC AUC of 99.98%.

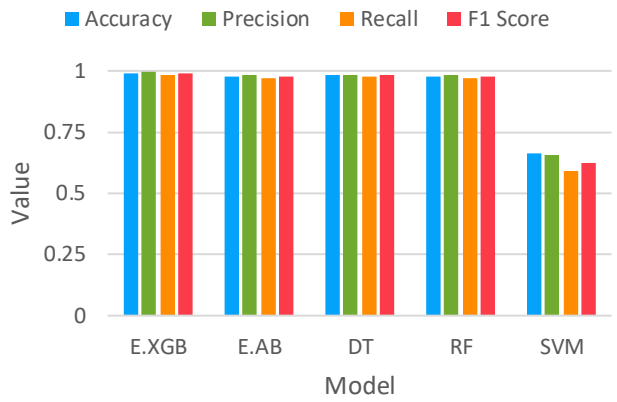

Fig. 7. Efficiency results on CNC milling dataset.

Compared to E.AB, RF, DT, and SVM, E.XGB consistently delivers higher predictive capability while maintaining computational efficiency. In terms of training time, E.XGB completes training in just 0.3126s, significantly outperforming E.AB at 4.6340s, RF at 1.8848s, and SVM at 81.1611s, while being slightly slower than DT at 0.0473s. Similarly, for prediction time, E.XGB and DT are the fastest at 0.0001s, far surpassing the efficiency of other models, particularly SVM, which has the slowest prediction time at 6.8742s.

TABLE IX. PERFORMANCE METRICS OF MODELS ON CNC MILLING DATASET

Details	E.XGB	E.AB	DT	RF	SVM
Accuracy (%)	99.01	97.71	98.16	97.88	66.53
Precision (%)	99.57	98.19	98.38	98.62	65.72
Recall (%)	98.30	96.86	97.67	96.82	58.95
F1-Score (%)	98.93	97.52	98.02	97.71	62.15
ROC AUC (%)	99.98	99.71	99.64	99.87	72.29
Training Time (s)	0.3126	4.6340	0.0473	1.8848	81.1611
Prediction Time (s)	0.0001	0.0696	0.0001	0.0781	6.8742

Despite DT being the fastest to train and predict, it still falls short in terms of accuracy of 98.16%, precision of 98.38%, and recall of 97.67%, making it less reliable for high-accuracy predictive maintenance applications. In contrast, SVM struggles significantly with an accuracy of only 66.53% and the lowest recall of 58.95%, proving unsuitable for this dataset.

Overall, E.XGB once again stands out as the most effective model for predictive maintenance in IoT-embedded machinery, offering the best balance of accuracy, precision, recall, and efficiency.

VI. Conclusion

In conclusion, this study proposes an enhanced predictive maintenance method for IoT-embedded industrial machinery, addressing challenges related to prediction accuracy, training efficiency, and prediction latency. The core of the method is the E.XGB model, which integrates enhanced feature selection, preprocessing, and hyperparameter optimization to overcome limitations of traditional approaches. The model demonstrated superior performance on the AI4I2020 and CNC Milling datasets, excelling in accuracy, precision, recall, and F1-scores, crucial for minimizing false positives and negatives in fault detection. It achieved 99.05% accuracy on the AI4I 2020 dataset and 99.01% accuracy on the CNC Milling dataset. Moreover, the proposed model demonstrated training speed of approximately 94% and prediction speeds of approximately 99.8% improvement than the E.AB model, thus meeting real-time industrial application requirements.

The optimized training and prediction pipelines resulted in significant reductions in computational time, making the model suitable for real-time applications. A comparative analysis with the E.AB method from previous research, as well as baseline models DT, RF, and SVM, highlighted the E.XGB model's consistent superiority. Its combination of advanced optimization techniques and lightweight design enables scalability across diverse operational conditions. The primary goal of the research which is enhancing predictive maintenance was achieved by improving accuracy, training efficiency, and prediction speed. Permutation importance and hyperparameter tuning enabled the identification of critical features for precise fault detection. Optimized preprocessing and grid search tuning reduced training time, while the lightweight architecture ensured low-latency predictions essential for real-time decision-making.

The proposed method minimizes unplanned downtime, reduces maintenance costs, and extends machinery lifespan, with scalability and adaptability for various industries. By enhancing predictive accuracy, training speed, and prediction latency, this study bridges the gap between theoretical machine learning advancements and their practical applications in industrial maintenance, contributing to a more efficient and sustainable industrial ecosystem.

Future research could focus on extending the proposed E.XGB model towards integration with IoT and edge devices, enabling deployment in resource-constrained environments for real-time predictive maintenance. In addition, exploring hybrid approaches that combine E.XGB with deep learning techniques such as CNNs or LSTMs may further enhance the model's ability to capture complex temporal and nonlinear fault patterns. To ensure adoption in real industrial settings, incorporating explainable AI (XAI) methods will be essential for improving model interpretability and building trust among engineers and decision-makers.

References

[1] Elkateb, S., Métwalli, A., Shendy, A., & Abu-Elanien, A. E. (2024). Machine learning and IoT – Based predictive maintenance approach for industrial applications. Alexandria Engineering Journal /Alexandria Engineering Journal, 88, 298–309. https://doi.org/10.1016/j.aej.2023.12.065

[2] Lee, W. J., Wu, H., Yun, H., Kim, H., Jun, M. B., & Sutherland, J. W. (2019). Predictive maintenance of machine tool systems using artificial intelligence techniques applied to machine condition data. Procedia CIRP, 80, 506–511. https://doi.org/10.1016/j.procir.2018.12.019

[3] Wang, L., Zhu, Z., & Zhao, X. (2024). Dynamic predictive maintenance strategy for system remaining useful life prediction via deep learning ensemble method. Reliability Engineering & Systems Safety, 110012. https://doi.org/10.1016/j.ress.2024.110012

[4] Meriem, H., Nora, H., & Samir, O. (2023). Predictive Maintenance for Smart Industrial Systems: a Roadmap. Procedia Computer Science, 220, 645–650. https://doi.org/10.1016/j.procs.2023.03.082

[5] Dalzochio, J., Kunst, R., Pignaton, E., Binotto, A., Sanyal, S., Favilla, J., & Barbosa, J. (2020). Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges. Computers in Industry, 123, 103298. https://doi.org/10.1016/j.compind.2020.103298

[6] Hamaide, V., Joassin, D., Castin, L., & Glineur, F. (2022). A two-level machine learning framework for predictive maintenance: Comparison of learning formulations. arXiv preprint arXiv:2204.10083. Retrieved from https://arxiv.org/abs/2204.10083

[7] Chevtchenko, S. F., dos Santos, M. C. M., Vieira, D. M., Mota, R. L., Rocha, E., Cruz, B. V., Araújo, D., & Andrade, E. (2023). Predictive maintenance model based on anomaly detection in induction motors: A machine learning approach using real-time IoT data. arXiv preprint arXiv:2310.14949. Retrieved from https://arxiv.org/abs/2310.14949

[8] Matzka, S. (2020). UCI machine learning repository, AI4I 2020 predictive maintenance

[9] Sun, S. (2020). University of Michigan System-level Manufacturing and Automation Research Testbed (SMART) Lab, CNC Mill Tool Wear Detection Dataset.

[10] Arisani, M. I., & Muljono, M. (2024). Bagging Nearest Neighbor and its Enhancement for Machinery Predictive Maintenance. Journal of Applied Informatics and Computing, 8(2), 248–256. https://doi.org/10.30871/jaic.v8i2.8158

[11] Ghadekar, P., Manakshe, A., Madhikar, S., Patil, S., Mukadam, M., & Gambhir, T. (2024). Predictive Maintenance for Industrial Equipment: Using XGBoost and Local Outlier Factor with Explainable AI for analysis (pp. 25–30). https://doi.org/10.1109/confluence60223.2024.10463280

[12] Baldovino, R. G., Camacho, K. S. I., Chua-Unsu, M. V. H. Y., Go, J. L. C., Munsayac, F. E. T., & Bugtai, N. T. (2024). Comparative Analysis of Machine Learning Models for Predictive Analysis of Machine Failures. In 2024 9th International Conference on Mechatronics Engineering (ICOM) (Vol. 10, pp. 288–293). https://doi.org/10.1109/icom61675.2024.10652325

[13] Elminir, H. K., El-Brawany, M. A., Ibrahim, D. A., Elattar, H. M., & Ramadan, E. (2024). An efficient deep learning prognostic model for remaining useful life estimation of high-speed CNC milling machine cutters. Results in Engineering, 24, 103420. https://doi.org/10.1016/j.rineng.2024.103420

[14] Kaliyannan, D., Thangamuthu, M., Pradeep, P., Gnansekaran, S., Rakkiyannan, J., & Pramanik, A. (2024). Tool condition monitoring in the milling process using deep learning and reinforcement learning. Journal of Sensor and Actuator Networks, 13(4), 42. https://doi.org/10.3390/jsan13040042

[15] Kamat, P. V., Kumar, S., Patil, S., Sugandhi, R., & Nargund, A. (2022). Tool Wear Prediction in Milling: A comparative analysis based on machine learning and deep learning approaches. International Journal of Computing and Digital Systems, 11(1), 151–165. https://doi.org/10.12785/ijcds/110112

[16] Assagaf, I., Sukandi, A., Abdillah, A. A., Arifin, S., & Ga, J. L. (2023). Machine predictive maintenance by using support vector machines. Recent in Engineering Science and Technology, 1(01), 31–35. https://doi.org/10.59511/riestech.v1i01.6

[17] Misaii, H., Fouladirad, M., Askari, B., & Durupt, A. (2024). Predictive Degradation Modelling Using Artificial Intelligence: Milling Machine Case Study. Advances in Reliability, Safety and Security.

[18] Stow, M. T. (2024). Hybrid Deep learning approach for predictive maintenance of industrial machinery using convolutional LSTM networks. International Journal of Computer Sciences and Engineering, 12(4), 1–11. https://doi.org/10.26438/ijcse/v12i4.111

[19] Gong, L., & Chen, Y. (2024). Machine Learning-enhanced loT and Wireless Sensor Networks for predictive analysis and maintenance in wind turbine systems. International Journal of Intelligent Networks, 5, 133–144. https://doi.org/10.1016/j.ijin.2024.02.002

[20] Richardson, B. (2019). Visual Studio Code as a Lightweight IDE: Features and Flexibility. Journal of Open Source Software Tools and IDEs, 3(1), 87-91.

[21] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. 12th {USENIX} Symposium on Operating Systems Design and Implementation (OSDI} 16), 265-283.

[22] McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56).

[23] Raja Azhan Syah Raja Wahab, Siti Nurulain Mohd Rum, Hamidah Ibrahim, Iskandar Ishak, "Efficient Top-K Continuous Query Processing Over Sliding Window Model (SWM) Method on Uncertain Data Stream," WSEAS Transactions on Systems and Control, vol. 19, pp. 283-308, 2024, DOI:10.37394/23203.2024.19.31.