A RELIABILITY-AWARE EDGE AI APPLICATION FOR INDUSTRIAL MACHINE FAILURE PREDICTION UNDER DEGRADED SENSOR INPUTS

Andrei Loghin[1] [0009-0006-9623-462X], Florentina Badea[2] [0000-0001-8910-6060] and Eugen Adrian Tamaș[3,] [0009-0005-1574-7638]*
[1]Robodictive SRL, Gheorghe Doja Street, Pitești, Romania
[2]National Institute of Research and Development in Mechatronics and Measurement Technique, 6-8 Pantelimon Road, Bucharest, Romania
[3]National University of Science and Technology POLITEHNICA Bucharest, 313 Splaiul Independentei, Bucharest, Romania
E-mail(s): eugen_adrian.tamas@upb.ro (✉)

Abstract - Industrial systems use many sensors to monitor machines and to detect possible failures. These data are often used by machine learning models for predictive maintenance. However, sensor data are not always clean. Some values can be missing, noisy, frozen, affected by drift, or affected by random spikes. These problems can reduce prediction quality and can lead to wrong maintenance decisions. This paper presents a reliability-aware Edge AI application for industrial machine failure prediction under degraded sensor inputs. The application was developed in Python and tested on the AI4I 2020 Predictive Maintenance Dataset, which contains 10000 instances, including 9661 no-failure cases and 339 failure cases. The application includes dataset loading, preprocessing, generation of degraded sensor inputs, model training, failure prediction, reliability score calculation, and automatic export of results. Five machine learning models were tested: Logistic Regression, Random Forest, Gradient Boosting, Linear Support Vector Machine, and Multilayer Perceptron Classifier. On clean data, Gradient Boosting obtained the best F1-score, with 0.7534. Under combined degradation, its F1-score decreased to 0.3493. Random Forest was more stable in some degraded cases, especially under random spikes and combined degradation. The mean reliability score decreased from 99.55 for clean data to 71.45 for combined degradation. The results show that clean-data accuracy is not enough for predictive maintenance model evaluation. The proposed application can support early Edge AI testing by comparing models under both clean and degraded sensor input conditions.

Keywords: Edge AI; Machine learning; Predictive maintenance; Industrial Internet of Things; sensor degradation; Reliability score; Machine failure prediction; Python application.

1. Introduction

Industrial systems use more sensors, software tools, and intelligent devices than before. Many machines collect data during normal work. These data can describe temperature, rotational speed, torque, tool wear, vibration, pressure, or other technical values. When these data are processed with artificial intelligence, they can help users detect machine problems earlier [1,2]. This is important because maintenance should not start only after the machine has already failed.

Predictive maintenance is based on this idea. It uses data to estimate the condition of a machine and to support maintenance decisions. In many modern systems, sensor data are collected through Industrial Internet of Things devices. The data are then processed by software tools and machine learning models [3,4]. Similar applied AI ideas were also used for detecting imperfections generated by changing 3D printer parameters [5]. This shows that image data, sensor data, and machine data can be useful when they are connected with intelligent analysis methods.

Industrial Internet of Things systems can support better monitoring, but they also create practical problems. A system must collect data, process them, and give a useful result in a short time. Edge and fog computing are often used because part

dx.doi.org/10.17683/ijomam/issue25/v1.12

of the processing can be done closer to the machine [6,7]. This can reduce delays and can support local decisions. Software tools for traffic monitoring and network supervision also show that applied software can process technical data and support local monitoring tasks [8].

Machine learning models are often used in predictive maintenance. Common models include Logistic Regression, Random Forest, Support Vector Machine, Gradient Boosting, and neural networks. These models can learn relations between input variables and machine failure states [9,10]. Related signal-processing applications also show that technical signals can be transformed into useful outputs with intelligent software [11]. In all these cases, the quality of the input data is very important.

A major problem is that sensor data are not always clean. Some values can be missing. Some values can include noise. Other values can be blocked, delayed, or affected by sudden spikes. Sensor drift can also appear when a sensor slowly changes its behavior over time [12,13]. Practical embedded systems, such as Raspberry Pi-based applications, also show that real systems must work with hardware limits and imperfect signals [14]. For this reason, data quality should be checked before the prediction is used.

Sensor degradation can affect the final result of a machine learning model. A model can work well on clean data, but it can become less stable when the input data are degraded. This problem is important in industrial environments because wrong predictions can lead to wrong maintenance decisions [15,16]. Real-time automation systems also show that the system must react correctly when the input changes during operation [17]. In predictive maintenance, this means that the model should not only give a class, but should also support a reliable decision.

Many predictive maintenance studies focus mainly on model accuracy. They compare algorithms, tune parameters, or use more complex models. This is useful, but it does not fully solve the problem of degraded input data [18,19]. Previous studies on software tools for learning and personalization also show that application design and processed information can influence the final result [20]. In industrial systems, this idea becomes even more important because the input data come from sensors and machines.

The AI4I 2020 Predictive Maintenance Dataset is used in this paper as the experimental input. It is a public dataset from the UCI Machine Learning Repository. The dataset contains 10000 instances and includes variables such as air temperature, process temperature, rotational speed, torque, tool wear, and machine failure [21]. The dataset is useful because it can be used for machine failure prediction and for testing different machine learning models. Recent predictive maintenance studies also show that the selected dataset, data structure, and model type can strongly influence the final results [22,23].

Python is a good option for this type of applied research. It can be used for data loading, preprocessing, model training, result export, and figure generation. Python also supports many machine learning libraries and can be used to create simple applications for experimental analysis [24,25]. Previous work on interpolation and extrapolation methods also shows that Python-based or software-oriented analysis can support numerical comparison and model evaluation [26]. In the present paper, the same practical direction is used for industrial machine failure prediction.

The proposed application follows an Edge AI direction. It does not try to replace a complete industrial platform. Its purpose is to test if a lightweight application can support machine failure prediction and reliability checking close to the data source. Recent studies show that Edge AI can be useful in industrial automation, predictive maintenance, and intelligent monitoring [27], [28]. Related robotic studies also show that local computation and software analysis can influence the quality of technical system behavior [29].

The main idea of this paper is different from a simple predictive maintenance classifier. The proposed application does not only predict if a machine failure may appear. It also tests if the input sensor data are reliable. For this purpose, several degraded input cases are generated. These cases include missing values, Gaussian noise, frozen sensor readings, sensor drift, random spikes, and combined degradation. This makes it possible to compare the behavior of the models on clean data and on degraded data.

The application also calculates a reliability score. This score is used to show if the input data and the final prediction can still be trusted. This is useful because high accuracy on clean data does not always mean that the model is stable under degraded input conditions. Recent studies in predictive maintenance and industrial monitoring show that practical systems need not only accurate models, but also reliable decision support [30], [31]. Educational machine learning studies also show that prediction results should be interpreted carefully when they are used for guiding decisions [32].

The need for reliable prediction is also connected with explainable and uncertainty-aware artificial intelligence. Users should understand not only the predicted class, but also the limits of that prediction [33], [34]. This is important for edge applications because a wrong local decision can affect the maintenance process. Local AI tools and gesture-based offline systems also show that practical applications can combine local processing, AI models, and interactive decision support [35]. In the present paper, this idea is moved toward industrial IoT monitoring.

The proposed application is developed as a Python software tool. It loads the dataset, prepares the data, trains machine learning models, creates degraded input cases, computes performance metrics, calculates reliability scores, and exports numerical and graphical results. Recent studies show that software-oriented AI applications can support industrial analysis and technical decision-making [36], [37]. Previous work on obstacle-avoiding path planning in RoboDK also shows that Python applications can be used for practical technical validation [38].

Lightweight classifiers are important for this study. Edge applications should not always depend on very large models. Simple models can be easier to train, easier to explain, and faster to run. Predictive maintenance studies still use common machine learning models because they offer a good balance between performance and simplicity [39]. Similar comparative ideas were also used in robotic object sorting, where lightweight classifiers were evaluated in a Python-based application [40].

The proposed work is also related to the wider use of AI-based analysis in technical and decision-support systems. Industrial monitoring, robotic applications, learning analytics, and sustainability studies show that data-driven tools can support ranking, classification, prediction, and practical decisions [41], [42]. Work on educational data governance also shows that prediction and data use should be connected with trust and responsible decision-making [43]. In the present paper, this principle is applied to machine failure prediction under degraded sensor input conditions.

The main contributions of this paper are the following. First, a Python-based Edge AI application is proposed for industrial machine failure prediction. Second, the AI4I 2020 Predictive Maintenance Dataset is used as the experimental input. Third, several degraded sensor input cases are generated and tested. Fourth, different machine learning models are compared under clean and degraded data conditions. Fifth, a reliability score is calculated in order to support safer maintenance decisions. Sixth, the application automatically exports numerical and graphical results for analysis and reporting.

2. Materials and Methods

This section presents the dataset, the proposed Python application, the machine learning models, the degraded sensor input cases, and the evaluation metrics. The purpose of this section is to explain the experimental workflow in a clear and reproducible way.

2.1. Dataset Description

The experiments were carried out using the AI4I 2020 Predictive Maintenance Dataset from the UCI Machine Learning Repository. The dataset is public and synthetic, but it was created to reproduce predictive maintenance data that can appear in industrial applications. It contains 10000 instances and can be used for classification, regression, and causal analysis tasks [21].

Each row in the dataset represents one operating case of a machine. The original file contains identification information, product information, operating variables, a binary failure label, and a failure type label. In this study, the binary target Machine failure was used. The value 0 indicates normal operation, while the value 1 indicates the presence of a machine failure.

The input variables used in the application were Type, Air temperature [K], Process temperature [K], Rotational speed [rpm], Torque [Nm], and Tool wear [min]. The variable Type is categorical and describes the product quality category. The other variables are numerical and describe machine or process conditions. The Failure Type column was not used for model training, because the objective of this study was binary failure prediction and not multi-class failure classification.

The selected variables are suitable for this study because they can be interpreted as sensor or machine-operation inputs. They also allow controlled degradation to be introduced during testing. For example, temperature, rotational speed, torque, and tool wear can be modified with missing values, Gaussian noise, frozen readings, drift, and spikes. This makes the dataset useful for testing how machine learning models react when sensor input quality decreases.

2.2. Application Design

The proposed application was implemented in Python as a desktop software tool with a PySimpleGUI interface (see Figure 1). The implementation follows a procedural and modular programming style. The application is not organized as a complex object-oriented system, because the purpose was to keep the workflow simple, reproducible, and easy to inspect. Instead, the source code is divided into clear functions, and each function has one main role in the analysis.

The graphical interface is separated from the processing logic. The interface collects the user inputs, such as the CSV file path, the output folder, and the selected test size. The processing functions then load the dataset, validate the columns, prepare the data, train the models, generate degraded input cases, compute metrics, calculate reliability scores, and export results. This separation makes the

application easier to modify and also reduces the risk of mixing interface code with experimental logic.

The main script contains several functional blocks. The first block defines the required column names, the selected input variables, and the target variable. The second block contains the dataset loading and validation function. The third block defines preprocessing and model creation. The fourth block generates degraded versions of the test data. The fifth block computes data quality indicators and reliability scores. The sixth block evaluates the trained models and exports the results. The last block creates the PySimpleGUI window and connects the interface buttons with the analysis function.

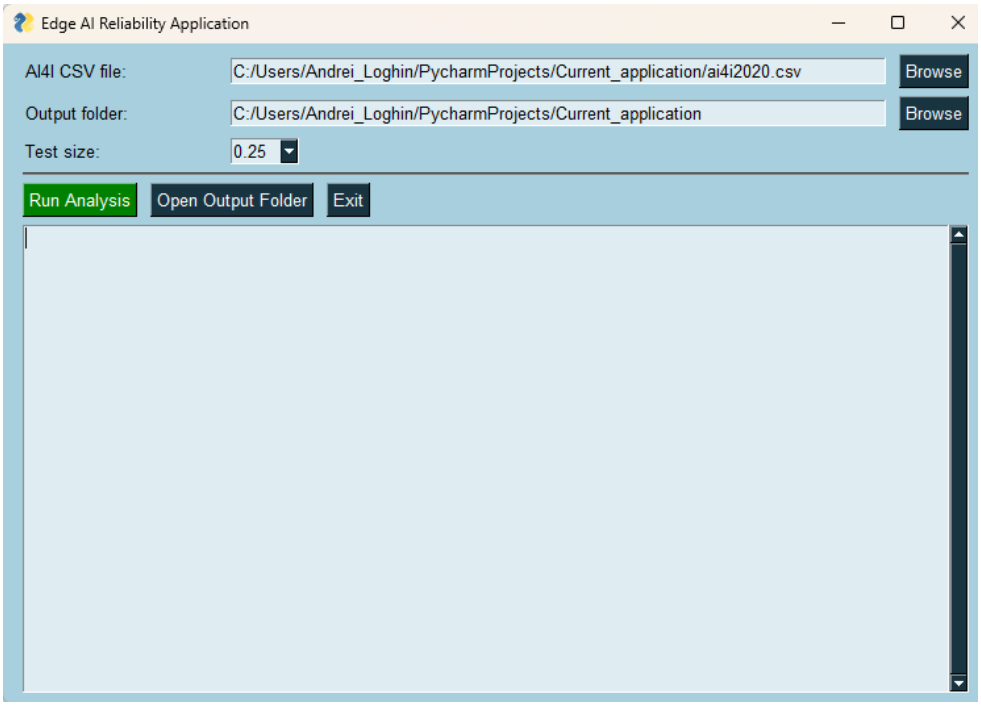

Figure 1: Application GUI

This design was selected because it fits the purpose of an experimental Edge AI application.

The code can be run by a user without changing the internal implementation, but the logic remains visible and reproducible. The application also creates automatic output files.

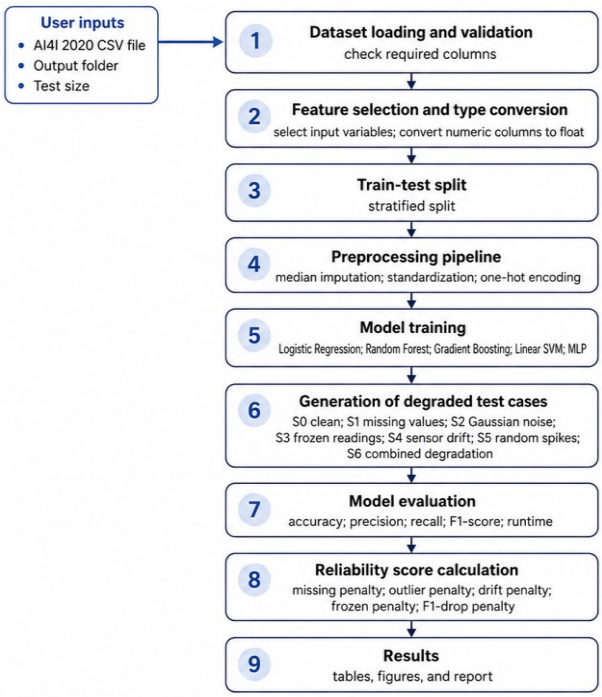

Figure 2: Software diagram of the proposed application

2.3. Application Workflow

The application starts by asking the user to select the AI4I 2020 CSV file and an output folder. The user can also choose the test size, with available values of 20%, 25%, and 30%. After the Run Analysis button is pressed, the application starts the full workflow and writes execution messages in the interface log.

The first internal step is dataset validation. The application checks if the selected file contains the required columns: Type, Air temperature [K], Process temperature [K], Rotational speed [rpm], Torque [Nm], Tool wear [min], and Machine failure. If one of these columns is missing, the application stops and shows an error message. This prevents the analysis from being run on a wrong input file.

After validation, the application keeps only the variables needed for the experiment. The numeric columns are converted to floating-point format. This step is important because some columns in the original dataset are stored as integer values, while the degraded input cases introduce decimal values through noise, drift, and spikes. Without this conversion, the application could not safely modify the numeric sensor values.

The dataset is then divided into training and testing subsets by using a stratified split. Stratification is used because the target variable is imbalanced. The failure class is much smaller than the no-failure class, and the split must keep a similar class distribution in both subsets. The models are trained only on the clean training data. The test subset is used in two forms: the original clean form and several degraded forms.

2.4. Data Preprocessing

The preprocessing stage is implemented with scikit-learn pipelines. In this workflow, the preprocessing transformations and the machine learning model are kept together, and the transformations learned from the training data are applied in the same way to all test cases.

The numerical variables are processed by median imputation and standardization. Median imputation is used because it is simple and less affected by extreme values than mean imputation. Standardization is then applied so that numerical variables with different units and scales can be used by the models in a consistent way.

The categorical variable Type is processed by most-frequent imputation and one-hot encoding. Most-frequent imputation is used in case the categorical value is missing. One-hot encoding transforms the product type into numerical columns that can be used by the machine learning models.

The preprocessing is fitted only on the training subset. The same fitted preprocessing structure is then applied to clean and degraded test subsets. This avoids data leakage and keeps the evaluation procedure correct.

2.5. Machine Learning Models

Five machine learning models were included in the application. The selected models were Logistic Regression, Random Forest, Gradient Boosting, Linear Support Vector Machine, and Multilayer Perceptron Classifier. These models were selected because they represent different families of classifiers and can be reproduced easily with standard Python libraries.

Logistic Regression was used as a simple linear baseline. Random Forest and Gradient Boosting were used as ensemble tree-based models. Linear Support Vector Machine was used as a margin-based classifier. Multilayer Perceptron Classifier was used as a simple neural model. For Logistic Regression, Random Forest, and Linear Support Vector Machine, class balancing was used to reduce the effect of the imbalanced target variable.

All models were trained using the same training subset. After training, each model was tested on the clean test data and on all degraded test data. This made it possible to compare both the baseline prediction performance and the stability of each model under degraded input conditions.

2.6. Generation of Degraded Sensor Input Cases

The degraded input cases were generated only for the test subset. The training subset was kept clean. This choice simulates a realistic situation where a model is trained on normal historical data, but later receives lower-quality sensor inputs during use.

Seven test cases were used in the experiment. The clean case, named S0_clean, uses the original test data without any modification. The missing-value case, named S1_missing_10pct, replaces 10% of the numeric sensor values with missing values. The Gaussian-noise case, named S2_gaussian_noise, adds random noise to numeric variables based on the training standard deviation. The frozen-reading case, named S3_frozen_readings, replaces part of the torque and rotational speed values with fixed median values. The drift case, named S4_sensor_drift, adds progressive changes to selected variables. The spike case, named S5_random_spikes, introduces high-amplitude abnormal values. The combined case, named S6_combined_degradation, applies several degradation types together.

These cases were not designed to reproduce one exact industrial defect. Their role was to create controlled and repeatable input-quality problems. In this way, the application can show how much the tested models are affected by different types of sensor degradation.

2.7. The Proposed Reliability Score

The application calculates a reliability score for each model and each test case. The score has values between 0 and 100. A higher score means that the input data and the prediction are more reliable. A lower score means that the input data are more degraded or that model performance decreased more strongly.

The score starts from 100 and subtracts penalties for five elements: missing values, outliers, mean drift, frozen readings, and F1-score drop compared with the clean case. The general form of the score is:

$$\text{Reliability\_Score} = 100 - \text{missing\_penalty} - \text{outlier\_penalty} - \text{drift\_penalty} - \text{frozen\_penalty} - \text{F1-drop\_penalty} \quad (1)$$

The five penalty terms used in equation (1) were calculated with fixed weights in the Python application. They are defined as follows:

$$P_{missing} = r_{missing} \cdot 100 \cdot 1.20 \quad (2)$$

$$P_{outlier} = r_{outlier} \cdot 100 \cdot 1.10 \quad (3)$$

$$P_{drift} = \min(d_z, 3.0) / 3.0 \cdot 25 \quad (4)$$

$$P_{frozen} = r_{frozen} \cdot 100 \cdot 0.60 \quad (5)$$

$$P_{F1} = \max(0, F1_{clean} - F1_{degraded}) \cdot 100 \cdot 0.80 \quad (6)$$

where r_missing is the ratio between missing numeric cells and all numeric cells, r_outlier is the ratio of outlier values, d_z is the average standardized mean shift of the degraded numeric variables compared with the training data, r_frozen is the average reduction in the number of unique rounded numeric values compared with the clean test data, F1_clean is the F1-score obtained on clean data, and F1_degraded is the F1-score obtained on the degraded input case.

An outlier was defined as a value with an absolute z-score higher than 3, computed using the training mean and standard deviation. The final reliability score was limited to the interval 0–100 after all penalties were subtracted.

The reliability score is not proposed as a universal industrial safety indicator. It is an experimental score used in this application to compare input cases. Its purpose is to show that a model output should not be interpreted only through accuracy or F1-score, but also through the quality of the input data.

2.8. Evaluation Metrics and Runtime Measurement

The models were evaluated using accuracy, precision, recall, and F1-score. These metrics were calculated for each model and each input case. Accuracy shows the general percentage of correct predictions. Precision shows how many predicted failure cases were correct. Recall shows how many real failure cases were detected. F1-score combines precision and recall into one value.

F1-score was treated as an important metric because the machine failure class is much smaller than the no-failure class. In this situation, accuracy alone can be misleading. A model can obtain high accuracy by predicting the majority class, but it can still miss failure cases.

The application also records runtime values. Training time, prediction time, and prediction time per sample are exported for each model and input case. These values are useful for discussing whether the tested models can be considered practical for simple Edge AI conditions.

In order to check the robustness of the results, the complete experiment was also repeated using five stratified random train/test splits with different random states. For each model and scenario, the mean and standard deviation of the main metrics were calculated. This additional check was used to support the comparison between clean and degraded input conditions.

3. Results

This section presents the results obtained with the proposed Edge AI application. The application generated numerical tables and graphical outputs for the clean dataset and for the degraded sensor input cases. The results show both the predictive performance of the tested models and the effect of sensor input degradation.

3.1. Dataset and Class Distribution

The AI4I 2020 Predictive Maintenance Dataset contains 10000 instances. In the selected binary classification task, 9661 cases belong to the no-failure class and 339 cases belong to the failure class. The resulting failure rate is 3.39%. The dataset contains no missing values before the degradation cases are generated (see Table 1).

Table 1. Dataset summary

Item	Value
Total instances	10000
No-failure cases	9661
Failure cases	339
Failure rate	3.39%
Input variables	6
Numeric input variables	5
Categorical input variables	1
Original missing values	0

The class distribution is shown in Figure 3. The figure confirms that the dataset is highly imbalanced. The no-failure class is much larger than the failure class. Therefore, accuracy alone can give an incomplete view of model performance. A model can obtain a high accuracy by predicting the majority class, but it may still miss many failure cases.

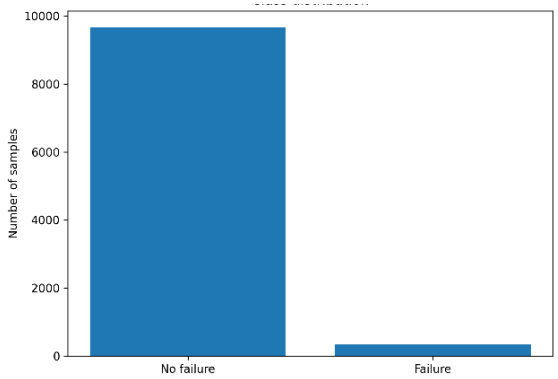

Figure 3: Class distribution of the AI4I 2020 dataset

3.2. Model Performance on Clean Data

The first experiment evaluated the models on the clean test data. The results are shown in Table 2. Gradient Boosting obtained the best clean-data performance, with an accuracy of 0.9856 and an F1-score of 0.7534. Random Forest also performed well, with an accuracy of 0.9808 and an F1-score of 0.7037. The MLP Classifier obtained an F1-score of 0.6433.

Logistic Regression and Linear SVM had much lower F1-scores, although their recall values were high. These two models detected many failure cases, but also produced more false positive predictions.

Table 2. Model performance on clean test data

Model	Accuracy	Precision	Recall	F1-score
Gradient Boosting	0.9856	0.9016	0.6471	0.7534
Random Forest	0.9808	0.7403	0.6706	0.7037
MLP Classifier	0.9756	0.6395	0.6471	0.6433
Logistic Regression	0.8252	0.1393	0.8000	0.2373
Linear SVM	0.8248	0.1376	0.7882	0.2343

The confusion matrix for the best model, Gradient Boosting, is shown in Figure 4. The model correctly classified 2409 no-failure cases and 55 failure cases. It produced 6 false alarms and missed 30 failure cases. This result shows that the model performs well on clean data, but it also confirms that failure detection remains difficult because the failure class is small.

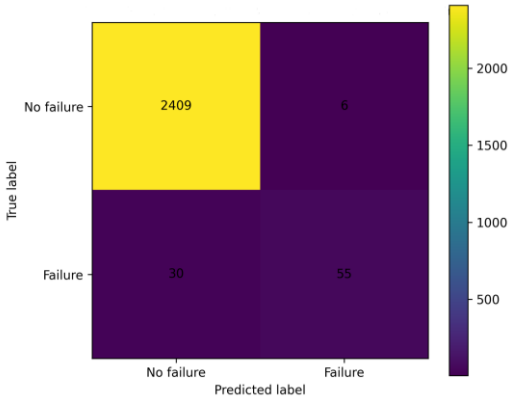

Figure 4: Confusion matrix for Gradient Boosting on clean test data

3.3. Effect of Degraded Sensor Inputs on Accuracy

The accuracy values obtained under all degradation cases are shown in Figure 5. On clean data, Gradient Boosting, Random Forest, and the MLP Classifier obtained high accuracy values. However, the accuracy decreased when stronger degradation cases were applied.

For Gradient Boosting, accuracy decreased from 0.9856 in the clean case to 0.9404 in the combined degradation case. Random Forest decreased from 0.9808 to 0.9588. The MLP Classifier decreased from 0.9756 to 0.9388. The strongest visible drops appeared especially for the random spikes and combined degradation cases.

Accuracy remained high in several cases, but this result must be interpreted carefully. Because the no-failure class is dominant, high accuracy does not always mean that the model detects failures well. For this reason, the F1-score is more useful for understanding the behavior of the models on the minority failure class.

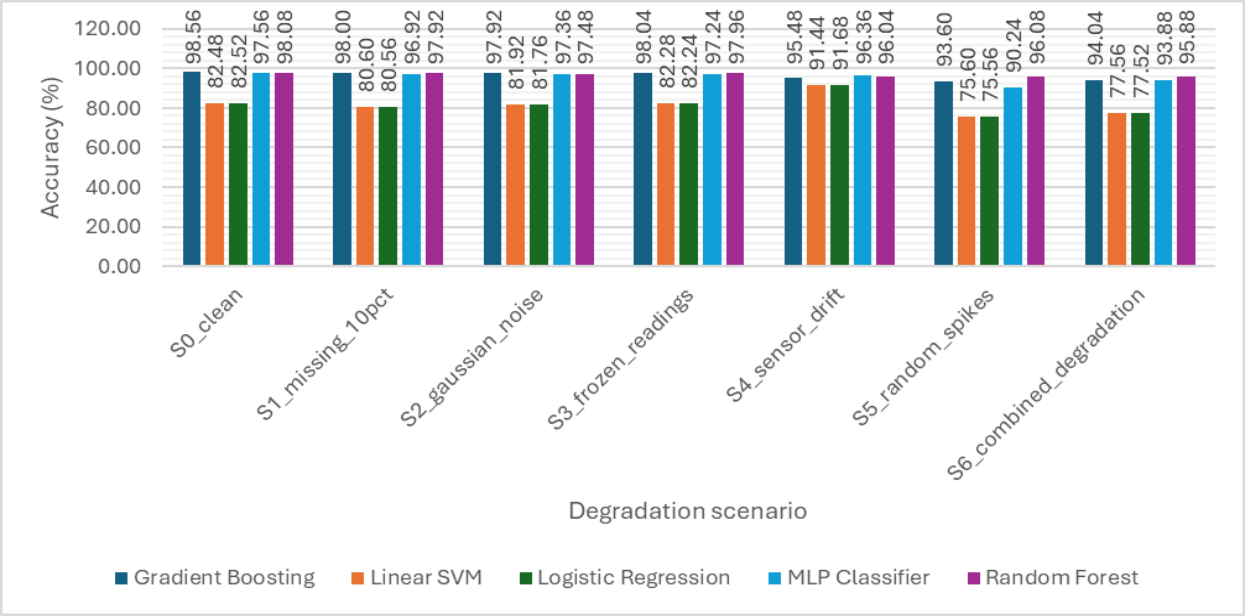

Figure 5: Accuracy by degradation scenario

3.4. Effect of Degraded Sensor Inputs on F1-Score

The F1-score values are shown in Figure 6. The results show that degraded sensor inputs have a stronger effect on F1-score than on accuracy. This confirms that F1-score is more sensitive to the quality of failure detection.

Gradient Boosting had the highest F1-score on clean data, with 0.7534. However, its F1-score decreased to 0.6479 under missing values, 0.6579 under Gaussian noise, 0.6621 under frozen readings, 0.4264 under sensor drift, 0.4203 under random spikes, and 0.3493 under combined degradation.

Random Forest was the second-best model on clean data, with an F1-score of 0.7037. It remained relatively stable under missing values and frozen readings, but it also decreased under sensor drift and combined degradation. The MLP Classifier showed moderate performance on clean data and was strongly affected by random spikes and combined degradation.

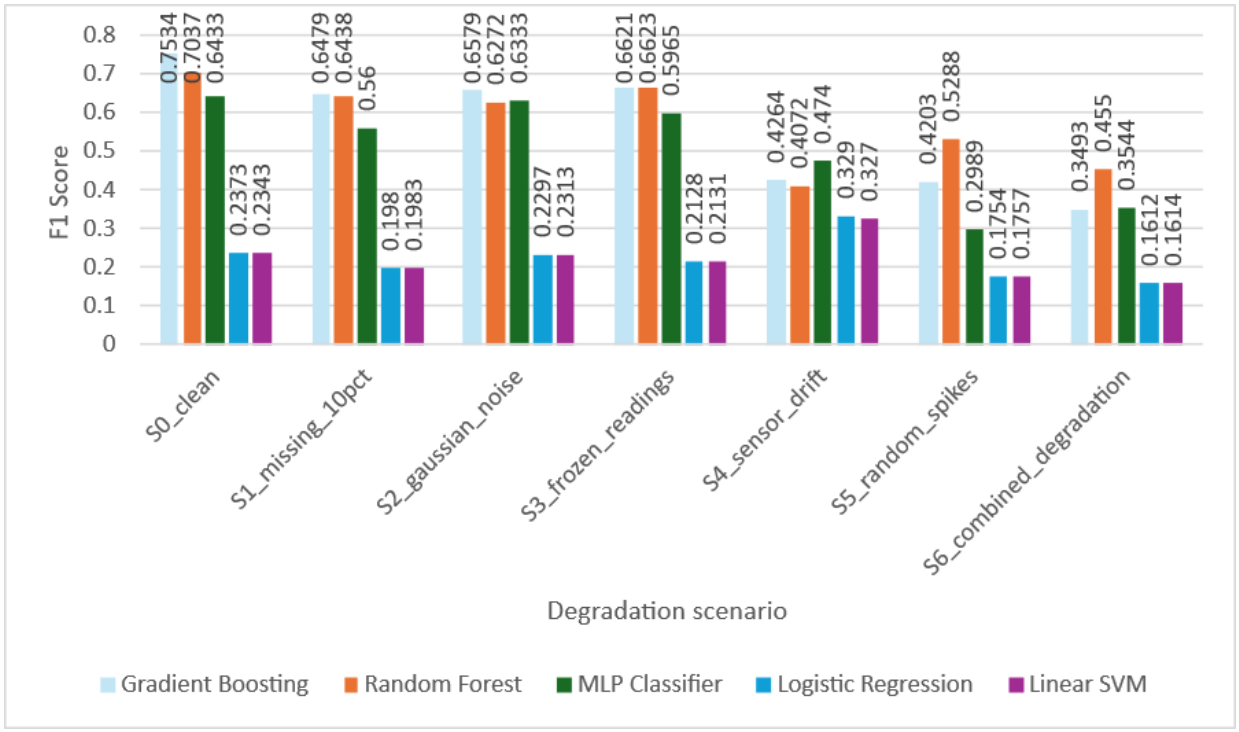

Figure 6. F1-score by degradation scenario

3.5. Reliability Score Analysis

The reliability score gives an additional view of the results. It combines input degradation indicators with the F1-score drop from the clean case. The mean reliability score by scenario is shown in Figure 7. The clean case had the highest mean reliability score, with 99.55.

Gaussian noise and frozen readings also had high mean reliability scores, with 96.48 and 94.81. The missing-value case had a lower score of 82.09. Sensor drift and random spikes produced similar reliability levels, with 79.86 and 79.49. The lowest score was obtained for combined degradation, with 71.45.

These results show that the reliability score captures the progressive loss of input quality. The combined degradation case produced the lowest reliability score, which is expected because it includes several types of sensor problems at the same time. This supports the purpose of the application: the model output should be evaluated together with the quality of the input data.

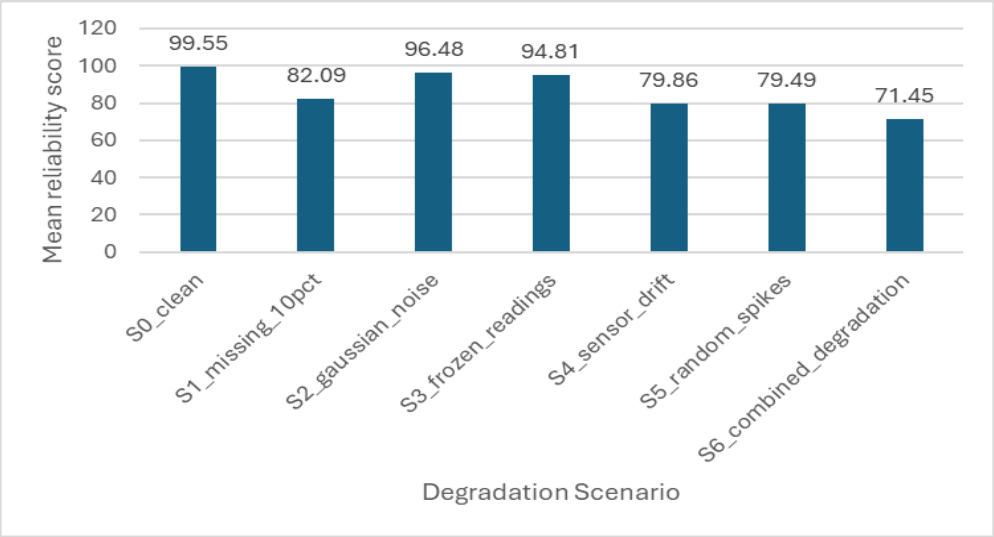

Figure 7: Mean reliability score obtained for each clean and degraded sensor input scenario

3.6. Runtime Results

The full analysis was completed in 5.72 seconds. This includes dataset loading, training of all models, generation of degraded input cases, evaluation, reliability score calculation, and result export.

Training time was low for all tested models. Linear SVM and Logistic Regression were the fastest, with training times below 0.02 seconds.

Random Forest required 0.23 seconds, Gradient Boosting required 0.63 seconds, and the MLP Classifier required 2.83 seconds. Prediction time per sample was also very low for all models (see Table 3).

Table 3. Training time by model

Model	Training time (s)
Linear SVM	0.0156
Logistic Regression	0.0197
Random Forest	0.2320
Gradient Boosting	0.6309
MLP Classifier	2.8281

These runtime values suggest that the tested models are suitable for simple Edge AI experiments. The results do not prove deployment readiness for all industrial hardware, but they show that the proposed application can run the complete analysis quickly on a standard computer.

3.7. Robustness Check over Five Random Splits

In order to check if the results depend on one train/test split only, the complete experiment was repeated using five stratified random splits. The random states were 0, 1, 2, 3, and 4. For each model, the mean and standard deviation were calculated for the clean case and for the combined degradation case.

The results are shown in Table 4. The same general trend was observed in all repeated runs. The F1-score values were higher on clean data and lower under combined degradation.

This confirms that the degradation effect was not caused only by one random split.

Table 4. Robustness check over five stratified random splits

Model	Clean F1-score mean ± SD	Combined degradation F1-score mean ± SD	Clean reliability mean ± SD	Combined reliability mean ± SD
Gradient Boosting	0.6856 ± 0.0540	0.3934 ± 0.0177	99.39 ± 0.12	65.12 ± 5.03
Linear SVM	0.2340 ± 0.0120	0.1718 ± 0.0157	99.39 ± 0.12	83.53 ± 1.54
Logistic Regression	0.2333 ± 0.0136	0.1718 ± 0.0173	99.39 ± 0.12	83.57 ± 1.54
MLP Classifier	0.6628 ± 0.0668	0.3528 ± 0.0497	99.39 ± 0.12	63.69 ± 3.89
Random Forest	0.6675 ± 0.0326	0.4564 ± 0.0454	99.39 ± 0.12	71.61 ± 4.50

The robustness check supports the results obtained in the initial experiment. Gradient Boosting and Random Forest remained the strongest models on clean data. Under combined degradation, Random Forest obtained the highest mean F1-score among the tested models. The reliability scores also decreased under combined degradation, which supports the use of the proposed reliability-aware evaluation.

4. Discussion

The proposed application supports the comparison of machine learning models under clean and degraded sensor input conditions. On clean data, Gradient Boosting obtained the best F1-score. However, the results changed when degraded input cases were introduced. This aspect is relevant in practice because industrial sensor data are not always clean during real operation.

The additional five-split robustness check confirmed the same general behavior, with lower F1-score and reliability values under combined degradation.

The accuracy values remained high in several cases, but this was partly influenced by the strong class imbalance of the dataset. Most samples belong to the no-failure class. For this reason, accuracy alone was not enough to evaluate the models. The F1-score gave a clearer view of failure detection performance.

The degraded input cases showed that model behavior can change strongly when sensor quality decreases. Sensor drift, random spikes, and combined degradation produced the largest decreases in F1-score. This means that a model that performs well on clean data may become less useful when the input data are affected by realistic sensor problems.

The reliability score added another useful layer of analysis. It showed not only how the model performed, but also how reliable the input and prediction context were. The combined degradation case had the lowest reliability score, which is consistent with the fact that several sensor problems were applied together. This supports the main idea of the paper: prediction results should be interpreted together with input data quality.

The results also show that model selection should not be based only on clean-data performance. Gradient Boosting was the best model on clean data, but Random Forest was more stable in some degraded cases. This suggests that reliability-aware testing can help choose a model that is not only accurate, but also more stable under imperfect input conditions.

4.1. Practical Meaning

The proposed application can be useful as a basic testing tool before deploying a machine learning model in an industrial monitoring context. It can show how a model behaves when sensor data include missing values, noise, frozen readings, drift, spikes, or combined degradation.

In practical terms, this means that the user can compare models not only by accuracy, but also by F1-score and reliability score. A high accuracy value can hide weak failure detection when the dataset is imbalanced.

The application can also support early Edge AI experiments. It does not replace a complete industrial monitoring platform, but it can help users understand which models are more stable before they are moved closer to real sensors or edge devices.

4.2. Limits of this Study

This study used a synthetic dataset. The AI4I 2020 dataset is useful for controlled experiments, but it does not include all problems that can appear in a real factory.

The degraded input cases were also generated artificially. They simulate common sensor problems, but they do not reproduce one exact industrial machine or one exact sensor fault.

Another limit is that the application was tested on a standard computer, not on a real edge device. The runtime values are useful for comparison, but they do not prove that the same performance will be obtained on all embedded or industrial edge hardware.

The reliability score is simple and experimental. It helps compare cases in this study, but it should not be used as a direct industrial safety metric without further validation.

4.3. Future Work

Future work can test the application on real industrial sensor data. This would show if the same model behavior appears when the data come from real machines.

Another direction is to run the application on real edge hardware, such as an industrial computer, Raspberry Pi, or other embedded device. This would make it possible to evaluate memory use, inference time, and practical deployment limits.

The reliability score can also be improved. Future versions can include more detailed weights, sensor-specific thresholds, and warning levels. The application can also be extended with explainable AI methods, so the user can better understand why a prediction was made.

Finally, the application can be adapted for multi-class failure diagnosis by using the Failure Type column from the dataset. This would allow the system to predict not only if a failure appears, but also the possible type of failure.

5. Conclusions

This paper presented a reliability-aware Edge AI application for industrial machine failure prediction under degraded sensor inputs. The application was developed in Python and tested on the AI4I 2020 Predictive Maintenance Dataset.

Clean-data performance is not enough for evaluating predictive maintenance models. Gradient Boosting obtained the best F1-score on clean data, but model behavior changed when degraded sensor input cases were introduced. Sensor drift, random spikes, and combined degradation had the strongest negative effect on F1-score.

The study also showed that accuracy must be interpreted carefully because the dataset is highly imbalanced. F1-score gave a clearer view of failure detection performance. The proposed reliability score added useful information by connecting prediction performance with input data quality.

Overall, the proposed application can support early testing of machine learning models for industrial monitoring and simple Edge AI scenarios. It helps compare models under clean and degraded input conditions and shows when prediction results should be interpreted with caution. Future work will focus on real industrial sensor data, real edge hardware testing, and improved reliability scoring methods.

References

[1] Samatas, G.G.; Moumgiakmas, S.S.; Papakostas, G.A. Predictive Maintenance - Bridging Artificial Intelligence and IoT. In Proceedings of the 2021 IEEE World AI IoT Congress (AIIoT); **2021**, May; pp. 0413–0419. DOI: 10.1109/AIIoT52608.2021.9454173

[2] Hector, I.; Panjanathan, R. Predictive Maintenance in Industry 4.0: A Survey of Planning Models and Machine Learning Techniques. *PeerJ Comput. Sci.* **2024**, 10, e2016, DOI: https://doi.org/10.7717/peerj-cs.2016

[3] Azeema, N.; Nayira, R. Artificial Intelligence in Predictive Maintenance for Industrial IoT. *Proceeding International Conference Of Innovation* **2023**, 3(2):127–130. DOI: https://doi.org/10.62951/icistech.v3i2.125

[4] D'Agostino, P.; Violante, M.; Macario, G. A Scalable Fog Computing Solution for Industrial Predictive Maintenance and Customization. *Electronics* **2025**, 14(1), 24. DOI: https://doi.org/10.3390/electronics14010024

[5] Tălîngă, A.-M.; Hadar, A.; Drăgoi, M.-V.; Ionut, N.; Ali, H.A.; Suciu, C.P. YOLO-v8 IN CAPTURING IMPERFECTIONS GENERATED BY CHANGING 3D PRINTER PARAMETERS. *U.P.B. Sci. Bull., Series C* **2024**, 86(4):253–266.

[6] Kapoor, D.; Gupta, D.; Uppal, M. Analyzing the Impact of Edge, Fog and Cloud Computing on Predictive Maintenance in the Industrial Internet of Things. *Discover Computing* **2025**, 28, 207. DOI: https://doi.org/10.1007/s10791-025-09653-8

[7] Hamasha, M.M.; Qais Albedoor; Hamasha, S.; Ali, H.; Qamar, A.; Berrah, F. A COMPREHENSIVE FRAMEWORK FOR IOT-DRIVEN PREDICTIVE MAINTENANCE: LEVERAGING AI AND EDGE COMPUTING FOR ENHANCED EQUIPMENT RELIABILITY. *Journal of Applied Engineering Science* **2025**, 23,3:471–486. DOI:10.5937/jaes0-57002.

[8] Drăgoi, M.-V.; Maier, A.N.; Alzubaidi, A.A.A.M.H.; Suciu, C.P.; Ali, H.A.; Dobrescu, T.G.; Hadăr, A.; Puiu, R.-A.; Petrea, G.; Avăsiloaie, A.P. MULTITHREADING VPN APPLICATION TO MONITOR WEB TRAFFIC AND UNBLOCK GEOBLOCKING WEBSITES. *U.P.B. Sci. Bull., Series C* **2025**, 87(4):217–231.

[9] Li, W.; Li, T. Comparison of Deep Learning Models for Predictive Maintenance in Industrial Manufacturing Systems Using Sensor Data. *Sci Rep* **2025**, 15, 23545. DOI: https://doi.org/10.1038/s41598-025-08515-z

[10] Chevtchenko, S.F.; Santos, M.C.M. dos; Vieira, D.M.; Mota, R.L.; Rocha, E.; Cruz, B.V.; Araújo, D.; Andrade, E. Predictive Maintenance Model Based on Anomaly Detection in Induction Motors: A Machine Learning Approach Using Real-Time IoT Data. *arXiv* **2023**. DOI:https://doi.org/10.48550/arXiv.2310.14949

[11] Haider, A.A.; Goga, N.; Vasilateanu, A.; Muslim, A.M.; Ali, K.A.; Dragoi, M.-V. English and Romanian Brain-to-Text Brain-Computer Interface Word Prediction System. *International Journal of Advanced Computer Science and*

Applications **2022**, 13(10). doi:10.14569/IJACSA.2022.0131003.
[12] Teh, H.Y.; Kempa-Liehr, A.W.; Wang, K.I.-K. Sensor Data Quality: A Systematic Review. *Journal of Big Data* **2020**, 7, 11, DOI: https://doi.org/10.1186/s40537-020-0285-1
[13] Zheng, H.; Paiva, A. Assessing Machine Learning Approaches to Address IoT Sensor Drift. *arXiv* **2021**. DOI:10.48550/arXiv.2109.04356.
[14] Drăgoi, M.-V.; Hadăr, A.; Goga, N.; Ștefan, A.; Ali, H.A. DESIGN AND IMPLEMENTATION OF AN EEG-BASED BCI PROSTHETIC LOWER LIMB USING RASPBERRY PI 4. *U.P.B. Sci. Bull., Series C* **2023**, 85(3):353–366.
[15] Fuente, R. de la; Radrigan, L.; Morales, A.S. Enhancing Predictive Maintenance in Mining Mobile Machinery through a TinyML-Enabled Hierarchical Inference Network. *arXiv* **2024**, DOI:10.48550/arXiv.2411.07168.
[16] Khemani, K. AI-Driven Predictive Maintenance with Real-Time Contextual Data Fusion for Connected Vehicles. *Research Square* **2025**, doi: https://doi.org/10.21203/rs.3.rs-6680683/v1
[17] Drăgoi, M.-V.; Nisipeanu, I.; Frimu, A.-V.; Tălîngă, A.-M.; Hadăr, A.; Dobrescu, T.G.; Suciu, C.P.; Manea, A.R. Real-Time Home Automation System Using BCI Technology. *Biomimetics* **2024**, 9(10), 594. DOI: https://doi.org/10.3390/biomimetics9100594
[18] Bitam, T.; Yahiaoui, A.; Boubiche, D.E.; Martínez-Peláez, R.; Toral-Cruz, H.; Velarde-Alvarado, P. Artificial Intelligence of Things for Next-Generation Predictive Maintenance. *Sensors* **2025**, 25(24), 7636. DOI: https://doi.org/10.3390/s25247636
[19] Nieminen, W.; Gebreweld, H.; Liuha, A.; Nissinen, M.; Verdugo, M.; Mikkola, A.; Kutvonen, A. Synthetic Data for Predictive Maintenance: A Systematic Review and Framework for Industry 4.0 Applications. *Journal of Intelligent Manufacturing* **2026**. DOI: https://doi.org/10.1007/s10845-026-02795-6
[20] Drăgoi, M.-V.; Puiu, R.-A.; Petrea, G.; Burtea, C.; Spiridon, M.; Suciu, C.P. IMPROVING THE LEARNING PROCESS FOR STUDENTS. PERSONALIZING LEARNING THROUGH SOFTWARE APPLICATIONS. *U.P.B. Sci. Bull., Series C* **2025**, 87(1):157–166
[21] AI4I 2020 Predictive Maintenance Dataset [Dataset]. *UCI Machine Learning Repository* **2020**. DOI:https://doi.org/10.24432/C5HS5C
[22] Bala, A.; Rashid, R. Z. J. A.; Ismail, I.; Oliva, D.; Muhammad, N.; Sait, S. M.; Al-Utaibi, K.A.; Amosa, T.I. & Memon, K. A. Artificial intelligence and edge computing for machine maintenance-review. *Artificial Intelligence Review* **2024**, 57(5), 119. DOI: https://doi.org/10.1007/s10462-024-10748-9
[23] Megdadi, E.; Mohamed, A.; & Shaalan, K. Machine Learning-Driven Best–Worst Method for Predictive Maintenance in Industry 4.0. *Automation* **2025**, 6(4), 91. DOI: https://doi.org/10.3390/automation6040091
[24] Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A.; Gomes, H.M.; Read, J.; Abdessalem, .T. & Bifet, A. River: machine learning for streaming data in python. *Journal of Machine Learning Research* **2021**, 22(110), 1-8.
[25] Gholizadeh, S. Top popular python libraries in research. *Authorea Preprints* **2022**. DOI: https://doi.org/10.22541/au.164580055.55493761/v1
[26] Drăgoi, M.-V.; Afram, R.M.; Alzubaidi, A.A.A.M.H.; Jiga, G.G.; Ali, H.A.; Puiu, R.-A.; Petrea, G.; Hank, A. COMPARATIVE ANALYSIS OF INTERPOLATION AND EXTRAPOLATION METHODS IN REGRESSION MODELS. *U.P.B. Sci. Bull., Series C* **2026**, 88(1):145–160.
[27] Rahman, M. A.; Shahrior, M. F.; Iqbal, K; & Abushaiba, A. A. Enabling intelligent industrial automation: A Review of machine learning applications with digital twin and edge AI Integration. *Automation* **2025**, 6(3), 37. DOI: https://doi.org/10.3390/automation6030037
[28] Halenar, I; Halenarova, L.; Tanuska, P.; & Vazan, P. Machine Condition Monitoring System Based on Edge Computing Technology. *Sensors* **2024**, 25(1), 180. DOI: https://doi.org/10.3390/s25010180
[29] Cristoiu, C. A.; Drăgoi, M.-V.; Nechita, R. M.; Stoian, B. N.; Dudici, C. L.; Puiu, R. A.; & Petrea, G. The Impact of Geometric Continuities (C1, C2, and C3) on the Trajectories of Industrial Robots. *Technologies* **2026**, 14(3), 176. DOI: https://doi.org/10.3390/technologies14030176
[30] Jakubowski, J.; Wojak-Strzelecka, N.; Ribeiro, R. P.; Pashami, S.; Bobek, S.; Gama, J.; & Nalepa, G. J. (2024). Artificial intelligence approaches for predictive maintenance in the steel industry: A survey. *arXiv preprint* **2024**. DOI:https://doi.org/10.48550/arXiv.2405.12785
[31] Subashree, S.; Rajakumaran, M.; Pushpa, G.; & Manivannan, R. Adaptive machine learning models for predictive maintenance in industrial internet of things (IIoT) systems. *Scientific Reports* **2026**, 16. DOI: https://doi.org/10.1038/s41598-026-42666-x
[32] Buzducea, C. A.; Drăgoi, M. V.; Cristoiu, C.; Puiu, R. A.; Puiu, M.; Petrea, G.; & Navligu, B. C. Machine Learning in Education: Predicting Student Performance and Guiding Institutional Decisions. *Educ. Sci.* **2026**, 16(1), 76. DOI: https://doi.org/10.3390/educsci16010076
[33] Arrieta, A. B.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garcia, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; Chatila, R.; & Herrera, F. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* **2020**, 58, 82-115. DOI:10.1016/j.inffus.2019.12.012

[34] Gawlikowski, J.; Tassi, C. R. N.; Ali, M.; Lee, J.; Humt, M.; Feng, J.; Kruspe, A.; Triebel, R.; Jung, P.; Roscher, R.; Shahazad, M.; Yang, W.; Bamler, R.; & Zhu, X. X. A survey of uncertainty in deep neural networks. *Artificial intelligence review* **2023**, 56(Suppl 1), 1513-1589. DOI: https://doi.org/10.1007/s10462-023-10562-9

[35] Drăgoi, M. V.; Nisipeanu, I.; Puiu, R. A.; Tache, F. G.; Spiridon-Mocioacă, T. M.; Hank, A.; & Cristoiu, C. An Inclusive Offline Learning Platform Integrating Gesture Recognition and Local AI Models. *Biomimetics* **2025**, 10(10), 693. DOI:https://doi.org/10.3390/biomimetics10100693

[36] Ucar, A.; Karakose, M.; & Kırımça, N. Artificial intelligence for predictive maintenance applications: key components, trustworthiness, and future trends. *Appl. Sci.* **2024**, 14(2), 898. DOI: https://doi.org/10.3390/app14020898

[37] Windmann, A., Wittenberg, P., Schieseck, M., & Niggemann, O. Artificial intelligence in industry 4.0: A review of integration challenges for industrial systems. In *2024 IEEE 22nd international conference on industrial informatics (INDIN)* **2024**, August. (pp. 1-8). DOI: 10.1109/INDIN58382.2024.10774364

[38] Cristoiu, C. A.; Drăgoi, M. V.; Nechita, R. M.; Verdete, B. M.; Dudici, C. L.; & Cusma, C. N. Python Software Application for Obstacle-Avoiding Path Planning in RoboDK Using Free Space Graph and Robot Level Validation. *Appl. Sci.* **2026**, 16(10), 4786. DOI: https://doi.org/10.3390/app16104786

[39] Samatas, G. G.; Moumgiakmas, S. S.; & Papakostas, G. A. Predictive maintenance-bridging artificial intelligence and IoT. *In 2021 IEEE World AI IoT Congress (AIIoT)* **2021**, May. pp. 0413-0419. DOI: 10.1109/AIIoT52608.2021.9454173

[40] Drăgoi, M. V.; Cristoiu, C. A.; Nechita, R. M.; Navligu, B. C.; & Verdete, B. M. A Python-Based Framework for Learning-from-Demonstration in Robotic Object Sorting: Comparative Evaluation of Lightweight Classifiers. *Appl. Sci.* **2026**, 16(10), 5107. DOI: https://doi.org/10.3390/app16105107

[41] Lawrence, N. P.; Damarla, S. K.; Kim, J. W.; Tulsyan, A.; Amjad, F.; Wang, K.; Chachuat, B.; Lee, J. M.; Huang, B.; & Gopaluni, R. B. Machine learning for industrial sensing and control: A survey and practical perspective. *Control Engineering Practice* **2024**, 145, 105841. DOI: https://doi.org/10.1016/j.conengprac.2024.105841

[42] Ordieres-Meré, J.; Sánchez-Herguedas, A.; & Mena-Nieto, Á. A Data-Driven Monitoring System for a Prescriptive Maintenance Approach: Supporting Reinforcement Learning Strategies. *Appl. Sci.* **2025**, 15(12), 6917. DOI: https://doi.org/10.3390/app15126917

[43] Drăgoi, M. V.; Puiu, R. A.; Petrea, G.; Cristoiu, C. A.; & Dumitrescu, C. I. Willingness to Allow Educational Data Use for Learning Analytics in Higher Education: Trust and Governance Predictors: An Exploratory Study. *Educ. Sci.* **2026**, 16(4), 637. DOI: https://doi.org/10.3390/educsci16040637