A Project Report On

FAILGUARD AI: MAINTENANCE PREDICTION AND REPORTING

In The Partial Fulfillment for the Award of The Degree

Of

Bachelor of Technology
In
Information Technology

By

121B1F069	**CHETAN MAHALE**
121B1F072	**SANCHALEE MESHRAM**
121B1F080	**PRANAV NARKHEDE**

Under The Guidance of

MRS. BABITA R. JANE

Department of Information Technology

Pimpri Chinchwad College of Engineering, Pune

2024- 2025

CERTIFICATE

This is to certify that the project report entitled

FailGuardAI: Maintenance Prediction and Reporting

Submitted by

121B1F069	Chetan Mahale
121B1F072	Sanchalee Meshram
121B1F080	Pranav Narkhede

is a bonafide work carried out by them under the supervision of Mrs. Babita R. Jane and it is approved for the partial fulfillment of the requirement for the award of the Degree of Bachelor of Technology in Information Technology.

This project report has not been earlier submitted to any other Institute or University for the award of any degree or diploma.

Mrs. Babita R. Jane
Internal Guide
Department of Information Technology

Dr. Jayashree V. Katti
Head of Department
Department of Information Technology

Dr. Govind N. Kulkarni
Director
Pimpri Chinchwad College of Engineering

External Examiner
Date:
Place:

ACKNOWLEDGEMENT

We express our sincere thanks to our Guide **Mrs. Babita R. Jane** for her constant encouragement and support throughout our project, especially for the useful suggestions given during the project and for having laid down the foundation for the success of this work.

We would also like to thank our Project Coordinator, **Dr. Maya S. Bembde** for her assistance, genuine support, and guidance from the early stages of the project. We would like to thank **Dr. Jayashree V. Katti, HoD IT** for her unwavering support during the entire course of this project work. We are very grateful to our Honorable Director, **Dr. Govind N. Kulkarni** for providing us with an environment to complete our project successfully. We also thank all the staff members of our college and technicians for their help in making this project a success.

Finally, we take this opportunity to extend our deep appreciation to our family and friends, for all that they meant to us during the crucial times of the completion of our project.

Name of the Student	**Sign**
Chetan Mahale	
Sanchalee Meshram	
Pranav Narkhede	

Abstract

Predictive Maintenance is one of the recent strategies in Industry 4.0 which aims to reduce the losses and unnecessary expenses of the organization. The reviewed literature includes methods in which a state of machine is classified as failure or no failure. This classification can be further used for actionable steps with the help of a diagnostic report. The proposed framework FailGuard AI aims to achieve this objective. The system classifies the machine parameters and also provides a comprehensive diagnostic report. This report is generated with the help of the Llama 3.1 Nemotron 70B model by Nvidia. The challenge of data imbalance is addressed with statistical and GenAI-based techniques like SMOTE and CTGAN. Various data-driven approaches are used to classify the machine parameters. Experimentation on imbalanced data is also done. Considering the use case a new class called "About to Fail" is also added to the dataset. In retrospect, the project offers a comprehensive and modern solution to Predictive Maintenance which can be easily deployed in the industry.

List of Figures

List of Tables

List of Abbreviations

AI: Artificial Intelligence

ML: Machine Learning

CNN: Convolutional Neural Network

RF: Random Forest

API: Application Programming Interface

UI: User Interface

PDM: Predictive Maintenance

LLM: Large Language Model

OSF: Overstrain Failure

GAN: Generative Adversarial Network

SMOTE: Synthetic Minority Over-sampling Technique

AI4I: Artificial Intelligence for Industry dataset

SVM: Support Vector Machine

UID: Unique Identifier

CTGAN: Conditional Tabular Generative Adversarial Network

KNN: K-Nearest Neighbors

XAI: Explainable Artificial Intelligence

SHAP: Shapley Additive exPlanations

LIME: Local Interpretable Model-agnostic Explanation

RAG: Retrieval Augmented Generation

SDG: Sustainable Development Goals

UML: Unified Modeling Language

SME: Subject Matter Expert

COCOMO: Constructive Cost Model

LLaMA: Large Language Model Meta AI

XGBoost: Extreme Gradient Boosting

AdaBoost: Adaptive Boosting

PostgreSQL: Open-source Relational Database Management System

Contents

Chapter 1

Introduction

Predictive maintenance is a technique in Industry 4.0 that allows machine operators and workers to predict when their machines are going to stop operating in advance simply by putting some machine parameters like air temperature, process temperature, rotational speed, torque, and tool wear. This allows operators to reduce downtimes and decrease their financial losses, enabling them to leverage their finances on other useful tasks.

1.1 Background

The traditional methods that operators were using were preventive maintenance and reactive maintenance [1]. In preventive maintenance safety measures are taken prior to the occurrence of machine failure such as regular and scheduled inspections. In reactive maintenance, corrective actions are taken after the failure has occurred. New technology includes predictive maintenance where we can predict machine failure in advance before the failure has even occurred.

1.2 Motivation

The primary motivation behind the execution of this project is to help industries by letting them save funds that are wasted in unexpected downtimes and machine failures. Industrialists would now be able to predict failures at the click of a button, just by entering certain parameters and

that's it - the results would be in front of them.

1.3 Aims and Objectives

- To develop an ML model capable of predicting machine failures.
- To develop an LLM that can generate diagnostic reports.
- A user-friendly website for easy and seamless predictions.
- To help machine operators detect failures easily.

1.4 Sustainable Development Goals (SDG) addressed

Amongst the 17 UN sustainable development goals, our project aligns with the following 3 goals:

- SDG 7: (Affordable and Clean Energy)
  By finding inefficiencies in machinery operations PDM helps to optimize energy usage. Early fault detection prevents excessive energy consumption, which contributes towards more cleaner and more sustainable energy footprint.
- SDG 9: (Industry, Innovation and Infrastructure)
  Our model improves industrial efficiency by reducing downtimes and minimizing equipment failures.
- SDG 12: (Responsible Production and Consumption)
  Our project ensures the consumption of resources responsibly by preventing unexpected equipment failures and increasing its lifespan.

1.5 Organization of Project Report

The project is divided into 11 chapters namely:

- Chapter 1: Introduction: Introduction of problem statement.
- Chapter 2: Literature review: Reviewed existing research done on PDM.
- Chapter 3: Requirements: Noted down the functional, non-functional requirements as well as the tools and technologies required.
- Chapter 4: Project plan: Identified the project phases and timelines of the project, also calculated the cost of the overall project.
- Chapter 5: Methodology: Description of the system architecture and algorithms used for the project.
- Chapter 6: Design: Drawn various UML diagrams.
- Chapter 7: Implementation: Contains code snippets and database implementation information.
- Chapter 8: Testing: Performed various types of testing such as functional, performance testing etc.
- Chapter 9: Results and discussions: Contains results obtained and graphs and figures.
- Chapter 10: Conclusion and future work: Includes future work, advantages etc.
- Chapter 11: References: Contains references.

Chapter 2

Literature Review

To understand the existing work on Predictive Maintenance and to find areas of improvement, an extensive review of the existing literature was done. The reviewed articles include various methods of implementing Predictive Maintenance.

The authors of [2] implemented predictive maintenance for cutting machines on the shop floor. The dataset was collected from the sensors attached to the machine. Random Forest for multi-label classification was used to predict the failure. They achieved an overall accuracy of 95%. The study in[3] proposed two multi-class predictive maintenance training algorithms that use Monte Carlo simulations and can be adapted to any classification algorithm. The authors used SVM and KNN in the article. The experiment carried out in [4] involves predictive maintenance for railway defect inspection using a multimodal model. The authors used an LLM-based AI pipeline to analyze seen and unseen visual defects in railways. They achieved a precision of 92% on image and 76% on video. The article [5] introduces a real-time predictive maintenance system based on motion current which can detect abnormal electric conditions. A precision of 97.59% was achieved on MLP with softmax activation function and scaled conjugate gradient optimization technique. The article [6] proposed a system which can answer questions based on context. It uses unlabelled data to do the task. Another approach [7] used logs for failure detection. The authors achieved a precision of 90% and a recall of 86%. The authors of article [8] gave a hypothesis of using Generative AI in the IoT industry to detect real-time anomalies. Proposed metrics include precision, recall, and f1 score. The study by authors in [9] used tempo-

ral association mining, predictive maintenance, temporal filter, machine learning, and semantic similarity on textual data for facility management requests. The English RoBERTa model has the highest acceptance rates for the minimum similarity thresholds of 0.65 and 0.7. In [10] predictive maintenance has been used in the manufacturing industry for process automation and quality control. Other methodologies include transfer learning, federated learning, edge computing, and model compression. The study demonstrated that using LLMs in manufacturing leads to more accurate predictions for maintenance, reducing machine downtime and repair costs. It also enhances quality control by automating inspections which results in fewer defects.

In [11] data collection, LLM integration, solution recipe generation, automation framework, and evaluation have been done for industrial asset management. The results of the paper show that using LLMs to automatically generate maintenance solutions significantly improves the efficiency of asset management. The model successfully provides clear, step-by-step instructions that reduce equipment downtime and optimize performance. The study in the paper [12] is used for detecting faults in robots and to estimate their remaining life duration. Algorithms like extremely randomized trees, k-nearest neighbors, and CNN have been used for the same. Using extremely randomized trees, the model performed well but improved even more after applying PCA. Without PCA, it had a survival analysis score of 0.76, an accuracy of 0.95, a precision of 0.98, and a recall of 0.96. After applying PCA, the survival analysis score increased to 0.80, and the accuracy went up to 0.98, while the precision and recall stayed the same at 0.98 and 0.96. In short, PCA helped the model become more accurate, especially in survival analysis, while maintaining its precision and recall levels. Another study [13] uses predictive maintenance for aircraft components. It was proved that proportional hazard models significantly improved predictive maintenance for aircraft maintenance. Their model reduced unscheduled failures by up to 90% without increasing costs. The authors of [14] also conducted research for aircraft components when subjected to various wear profiles. They used an optimal predictive maintenance strategy that reduced the overall maintenance cost of aircraft components. Another study [15] used unsupervised machine learning to detect faults early in predictive maintenance. Exhaust fan vibrational data was collected and features like peak acceleration and velocity were extracted. PCA was also used to reduce the dataset. The results showed that PCA was effective in de-

tecting the faults early. In [16] the methodology that was used is data collection, digital twin creation, fault detection, condition precondition, maintenance planning, and framework validation. A digital twin was created for air handling units for fault detection. When the cooling valve is closed during the cooling phase, the system achieves an accuracy of 87.5%, with an error rate of 12.5%. If the dampers are closed, accuracy drops to 80%, with a higher error rate of 20%. When there are no faults, the system performs almost perfectly, with an accuracy of 99.8% and a minimal error of just 0.2%. The authors of the paper [17] highlight the critical role of continuously tracking sensor data from tubing machines in predictive maintenance using machine learning. The study emphasizes data cleaning and feature extraction as essential steps to uncover patterns that inform maintenance strategies. By utilizing the LSTM networks, the research aims to improve failure predictions, set up maintenance schedules, and improve overall productivity. The authors of the paper [18] present advanced predictive maintenance techniques which include multiple methodologies such as Calibration Verification, Empirical and Physical Modeling, Time-Domain Reflectometry (TDR), Neural Network Modeling, and Predictive Maintenance Data Analysis. These approaches have effectively reduced calibration issues to 10% and improved sensor reliability by 90%, thereby enhancing decision-making processes in maintenance strategies. The study in the paper [19] proposes a deep attention-based method for predictive maintenance in IoT applications. This approach leverages positional encoding to retain the sequence of time steps and an attention module to prioritize key data points, assigning weighted significance to improve focus on relevant information. A feed-forward network processes these insights, combining regression with advanced neural techniques to boost predictive precision. Results show RMSE scores of 18.92 ± 0.26 at 10 cycles, 14.40 ± 0.21 at 20 cycles, and 13.50 ± 0.30 at 30 cycles, with a training speed that is 20% faster than traditional LSTMs over 300 epochs. This model demonstrates significant gains in predictive maintenance accuracy and efficiency in IoT environments. The authors of the paper [20] present a predictive maintenance system specifically for industry robots, aiming to improve quality production in the manufacturing process. This approach integrates three modules: a fault prediction module using a KNN-LSTM model for complex, non-linear time series data; a knowledge graph (KG) module to store and interpret robot maintenance data; and a maintenance decision support module that

draws on the KG for precise maintenance recommendations. The KNN-LSTM model achieved an F1 Score of 0.9109, Recall of 0.9388, Precision of 0.8846, and Accuracy of 0.9091, surpassing other tested models like ResNet18, SVM, XGBoost, LSTM, and standard neural networks.

The researchers of the paper [21] tested multiple models, including Random Forests and Support Vector Machines, to develop reliable maintenance strategies. A financial analysis shows a notable 40% reduction in maintenance costs and downtime post-implementation. Among the models, Logistic Regression reached an accuracy of 94.2%, precision of 92.5%, recall of 95.8%, and an F1-Score of 94.1%. Meanwhile, Decision Trees achieved 91.7% accuracy, 90.2% precision, 92.5% recall, and an F1-Score of 91.3%. Random Forests performed best, with 96.5% accuracy, 95.8% precision, 97.2% recall, and an F1-Score of 96.5%. The authors of the paper [22] explore the time series data to model machines Remaining Useful Life (RUL). Using deep learning models like DNN and RNN, it forecasts maintenance needs and connects these with production schedules for better operational efficiency. The study reports high accuracy rates, 95.01% for Random Forest, 92.2% for CNN, and 91.08% for LSTM, highlighting the models' effectiveness for predictive maintenance. The researchers of pthe aper [23] used both Support Vector Machines (SVM) and Artificial Neural Networks (ANN) with RNN and CNN as basic algorithms for predictive maintenance. Results show an accuracy of 87% on detected anomalies after getting relevant features using datasets obtained from the milling and bearing experiments. This illuminates the capability of AI methods to improve maintenance plans for machine tool systems. In this research paper [24], the authors explore the use of LSTM-based encoder-decoder models to detect anomalies in multi-sensor data. It uses density-based clustering algorithms (for example DBSCAN) to find clusters in a large dataset, even if the noise is present. The study also employs active learning methods to improve model performance with labeled data. Using datasets like NASA Turbofan Engine Degradation and PHM 2012 Bearing Fault Detection the results exhibit an AUROC of 0.98, AUPRC of 0.87, and an F1-Score of 0.93 for the LSTM model, which outperforms other algorithms among Autoencoder and One-Class SVM. The article [25] highlights challenges in traditional strategies of predictive maintenance. The authors have proposed Probabilistic and Possibilistic neuro-fuzzy clustering methods. The system integrates filtering and tracking for real-time processing. The losses for Probabilistic

and Possibilistic approaches are 15.6% and 15.2% respectively. The authors of [26] have experimented on fault detection by vibration and pV diagram analysis. Feature extraction was done using spectrogram differences and correlation for fault detection in compressor valves. For plastic valves, two class SVM outperformed achieving an accuracy of 99.97%. Authors of [27] used a data-driven approach to predict railway track conditions and got a success rate of 91.1%. Authors of [28] used unsupervised ensemble learning models and got an accuracy of 95.1%, a precision of 24.5%, and an F1 score of 38.5%. The authors in the paper [29] suggested a Fault Detection and Diagnosis system coupled with LLM. Retrieved data such as text, images, audio, and vibration signals through web scraping and APIs and saved preprocessed data in a knowledge base. The framework uses models such as LangChain, LlamaIndex, and GPT-4-Preview to enhance accuracy. Analysis revealed GPT-4-Preview to be 96.3% accurate, outperforming Random Forest (90.36%) and Neural Networks (86.07%). The authors in [30] presented a sensor-based predictive maintenance system based on MQTT and MongoDB to collect and store data such as CPU usage, memory, temperature, and voltage. They used XGBoost, LSTM, and ARIMA for time-series analysis and used LSTM Autoencoder, Isolation Forest, and DBSCAN for fault detection. LSTM Autoencoder attained 100% accuracy, whereas DBSCAN recorded the maximum ROC-AUC of 0.9992.

Despite the various approaches used for Predictive Maintenance in the literature reviewed there remain some gaps. The first issue was the narrow scope of many of the papers which often concentrated on a specific type (i.e. railway systems or shop floor machines) and thus made their findings harder to generalize across all types of machines in the industry. Focusing on individual machine types highlights a deficiency in universal strategies that could be leveraged between industries, and across different machines. A broader approach that includes different machines of all kinds would further improve the stability and flexibility of predictive maintenance. Second, one widely underestimated critical part in today's research are the insights which will be used to reprogram the machines. None offer guidelines or clear support for machine operators based on the predictive maintenance analysis carried out. It is critical that operators can make use of the actionable insights from predictive models, not just to perform their daily tasks more efficiently or predict a potential machine failure. Finally, very few of the descriptions give a thorough

explanation for why the machines failed.

Chapter 3

Requirements

The FailGuard AI system can predict the failure of machines and suggest actionable steps. The requirements of the system are as follows.

3.1 Functional Requirements

- The system should allow only authorized users to log in to the system.
- The system must allow users to input machine parameters.
- The system should be able to predict the mode of failure precisely. Mode of failure includes “No Failure”, “About to Fail”, “Power Failure”, “Heat Dissipation Failure”, “Overstrain Failure”, “Random Failure” and “Tool Wear Failure”.
- Users should be able to hear a warning sound after any failure prediction occurs.
- The system should be able to generate a detailed diagnostic report which includes recommended actions for the machine operator.
- The user should be able to download the generated report.

3.2 Non-Functional Requirements

- The system should be able to classify the given conditions with high precision.
- The report generated should be easy to understand by a human and should not contain any technical jargon.
- The report should be generated within 30 seconds.
- The system should be robust to all kinds of data.
- The User Interface should be easy to use and intuitive.

3.3 Tools and Technologies Used

3.3.1 Programming Language

- **Python:** Used for data preprocessing, developing machine learning models, and website development.

3.3.2 Libraries

- **Numpy and Pandas:** Used for data manipulation and for performing complex numerical calculations.
- **Matplotlib and Seaborn:** Used for data visualization and plotting graphs.
- **Sklearn:** Used for implementing machine learning algorithms.
- **XGBoost:** Used for implementing ensemble methods.
- **CTGAN:** To augment the imbalanced data.

3.3.3 Large Language Model

- **LLaMA 3.1 Nemotron 70B by NVIDIA:** For diagnostic report generation.

3.3.4 UI Development

- **Streamlit:** Used for developing a website for easy access to the system.

3.3.5 Integrated Development Environment

- **Google Colab**
- **Visual Studio Code**

3.3.6 Application Programming Interface

- **Vext App API:** For accessing the Large Language Model.

Chapter 4

Project Plan

4.1 Project Phases and Timeline

The project's phases and timelines are illustrated with the help of 2 project management tools: Gantt chart and Pert chart. Both these charts help us visualize the scheduling and planning tasks.

- Gantt Chart

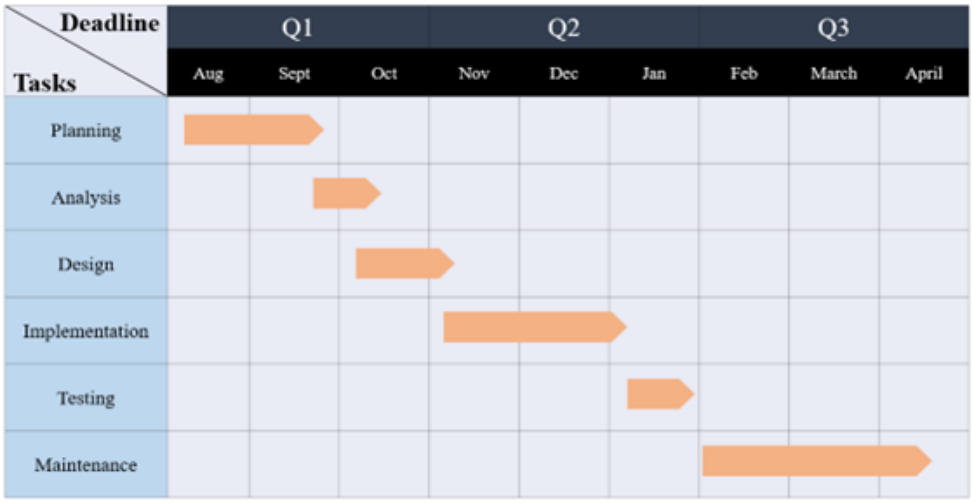


Figure 4.1: Gantt chart

Above is the Gantt chart which basically represents a horizontal bar chart with the bars being the duration of a particular project phase. We can clearly see which phase is going to take how much time.

- Pert Chart

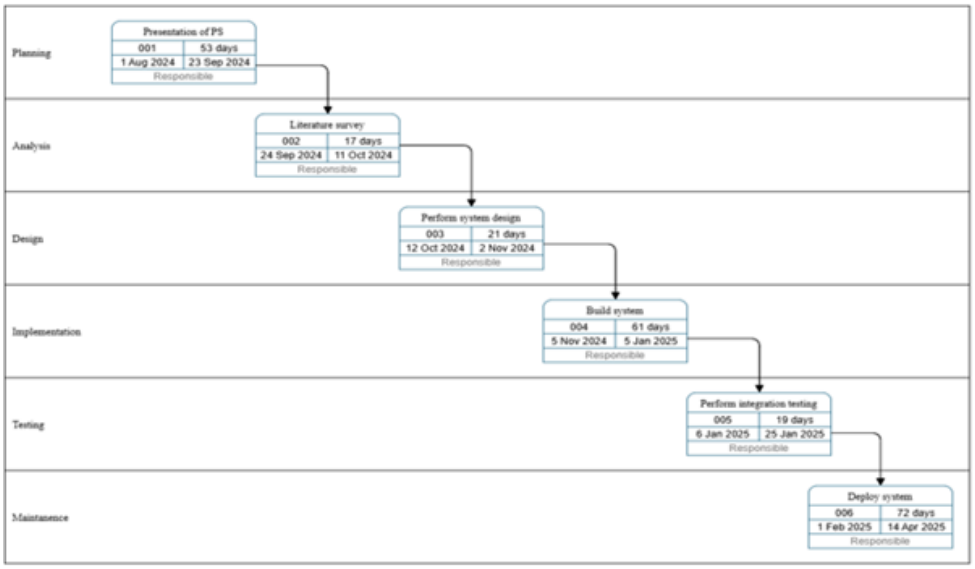


Figure 4.2: PERT Chart

Above is the pert chart which is a more detailed version of gantt chart. It contains Task IDs, start date and end date and the total duration of each phase. One can clearly see when the project has started and when it will end.

4.2 Cost Estimation

The type of COCOMO model and development mode applicable to our project is “Intermediate” and “Semi-detached” respectively.

- Effort: E = a (KLOC)b × EM

  Where:

  E = Effort in Person-Months (PM)

  KLOC = Kilo lines of code

  a, b = Constants

  EM = Effort Multipliers (cost drivers)

  For Semi-Detached Mode, the constants are:

a = 3.0, b = 1.12

Effort multiplier:

Factor	Rating	EM
Software reliablity	High	1.15
Database size	Medium	1.08
Product complexity	High	1.30
Analyst Capability	High	0.86
Development experience	Nominal	1.00
Use of tools	good	0.90

Table 4.1: Common effort multipliers

EM (total)=1.15×1.08×1.30×0.86×1.00×0.90 = 1.18

Putting all the required values:

E = 3.0 (4.7)1.12 × 1.18 = 20.03 = Approx 20 person - month.

- Development Time: DT = c × Ed = 2.5 × 200.35 = 7.13 = Approx 7 months.
- Development Time: Team size = E/DT = 20/7 = 2.8 = Approx 3 People

Chapter 5

Methodology

5.1 Introduction

The FailGuard AI demonstrates a straightforward method to develop a predictive maintenance system. It employs cutting-edge machine learning methods and a Large Language Model (LLM) to reduce industrial downtime and maintenance expenses.

Our strategy starts with the dataset we chose: the "AI4I 2020 Predictive Maintenance Dataset" from the UC Irvine Machine Learning Repository. It consists of 10,000 data points, capturing real-world machine data such as air temperature, process temperature, rotational speed, torque, and tool wear. It also includes six types of failures: Machine Failure, Tool Wear Failure, Heat Dissipation Failure, Power Failure, Overstrain Failure, and Random Failure. As with most maintenance datasets, it is imbalanced by design. Machines rarely fail, so the "failure" data points are significantly fewer than the "no failure" data points. To address this, we used data augmentation techniques—SMOTE and CTGAN—to balance the dataset. We then experimented with a variety of machine learning methods, ranging from basic algorithms to advanced techniques such as bagging and boosting, and even a stacking classifier to combine them. Additionally, we incorporated a Large Language Model—Llama 3.1 Nemotron 70B—to translate predictions into simple English diagnosis reports. We also developed a user-friendly web app, allowing operators to input machine information and receive reports quickly.

5.2 System Architecture

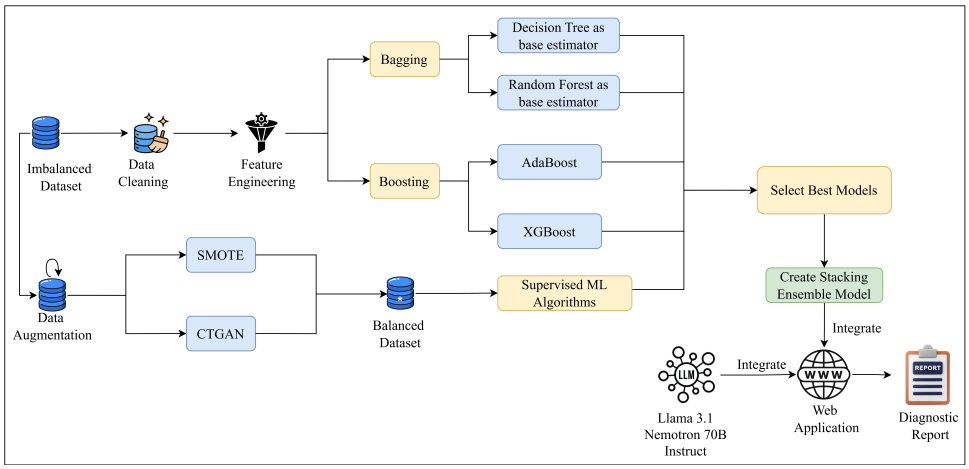


Figure 5.1: System Architecture

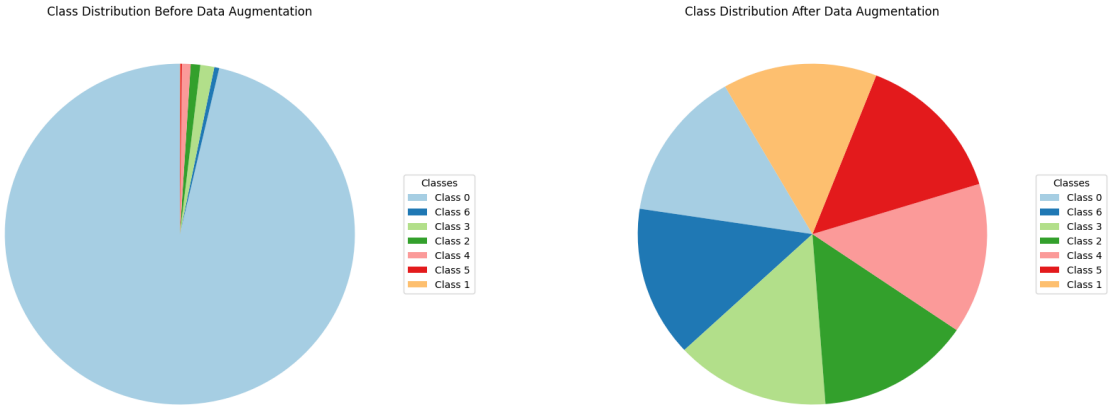
The FailGuard AI system is designed for early machine fault detection. We begin with the AI4I dataset, containing 10,000 records of parameters such as air temperature and torque. Redundant fields like "UID" and "Product ID" are removed, and the data is cleaned. An "About to Fail" class is introduced based on thresholds to identify issues proactively. Since failures are rare, we apply SMOTE and CTGAN to balance the data. Advanced algorithms, including Bagging and Boosting, are run on the imbalanced data, prioritizing Precision, Recall, and F1 metrics alongside accuracy. On the balanced datasets, we apply standard methods such as Naive Bayes, Logistic Regression, KNN, Decision Tree, Random Forest, and SVM. The best-performing models are combined into a Stacking Classifier with an SVM meta-model, trained on SMOTE and CTGAN datasets for a final prediction. The Llama 3.1 Nemotron 70B language model translates these predictions into clear reports using a specialized prompt. Users interact through a Streamlit web app, where they input machine data and receive a downloadable, easy-to-read report.

5.3 Algorithms

Below is an explanation of the algorithms applied in this project:

SMOTE (Synthetic Minority Over-sampling Technique) SMOTE enriches data by creating

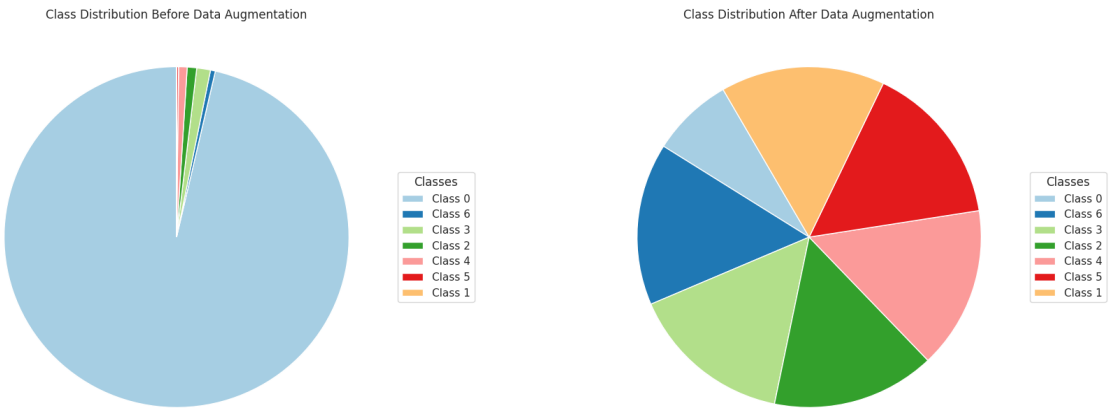
synthetic examples for the minority class. It selects nearby points, draws a line between them, and generates new examples along this line.



(a) Before SMOTE (b) After SMOTE

Figure 5.2: Class Distribution before and after applying SMOTE

CTGAN (Conditional Tabular Generative Adversarial Network) CTGAN is a GAN variant for tabular data, comprising a generator and a discriminator. The generator creates synthetic data, while the discriminator distinguishes between real and synthetic data. CTGAN captures complex patterns, enabling realistic synthetic samples.



(a) Before CTGAN (b) After CTGAN

Figure 5.3: Class Distribution before and after applying CTGAN

Several supervised machine learning algorithms were applied to the balanced datasets to

evaluate their performance:

1. **Logistic Regression**: A statistical method for binary classification, used to model the relationship between features and the target variable.

2. **Decision Tree**: A tree-like structure that splits data into subsets based on feature values, aiming to create pure nodes representing a class.

3. **Random Forest**: An ensemble of decision trees where results are aggregated to improve accuracy and reduce overfitting.

4. **Support Vector Machine (SVM)**: A model that finds the hyperplane maximizing the separation between classes in the feature space.

5. **K Nearest Neighbours (KNN)**: A simple algorithm that classifies objects based on the majority class among their k nearest neighbors.

6. **Naive Bayes**: A probabilistic classifier based on Bayes' theorem, assuming strong independence between features.

Ensemble Algorithms

Ensemble methods combine predictions from multiple models to enhance performance:

1. **Bagging**: Trains models in parallel on different data subsets, reducing variance and regularizing the model. We used decision trees and random forests as base estimators.

2. **Boosting**: A sequential technique where each model learns from the errors of its predecessor, focusing on misclassified instances. AdaBoost adjusts weights to emphasize difficult cases, while XGBoost uses regularization to prevent overfitting and improve performance.

3. **Ensemble Stacking**: A stacking classifier combines top-performing models. It trains a meta-model (SVM) on the predictions of base models, leveraging their strengths for a robust predictive model. The stacking ensemble achieved high accuracy and performance scores, demonstrating its effectiveness in predictive maintenance.

Prompt for LLM Usage

The prompt used in this research ensures the system delivers concise, relevant information to machine operators.

```
Generate a diagnostic report based on the following machine parameters:
Type: {Type}
Air Temperature [K]: {Air_temp}
Process Temperature [K]: {Process_temp}
Rotational Speed [rpm]: {Rot_speed}
Torque [Nm]: {Torque}
Tool Wear [min]: {Tool_wear}
and the predicted failure class: {predicted_class} (one of the following: TWF - Tool Wear Failure,
HDF - Heat Dissipation Failure, OSF - Overstrain Failure, PWF - Power Failure,
RNF - Random Failure, About to Fail, or No Failure).
Include in the report:
A brief summary of the operational parameters.
Actionable recommendations tailored only to the specific failure class {predicted_class} provided.
Avoid listing or referring to other failure classes.
If the failure class is "No Failure," confirm that the machine is operating normally and
requires no immediate maintenance,
while suggesting continued monitoring and scheduled maintenance.
Ensure the output is concise, actionable, and relevant to the provided parameters.
```

Figure 5.4: Prompt used

Chapter 6

Design

6.1 Use Case Diagram

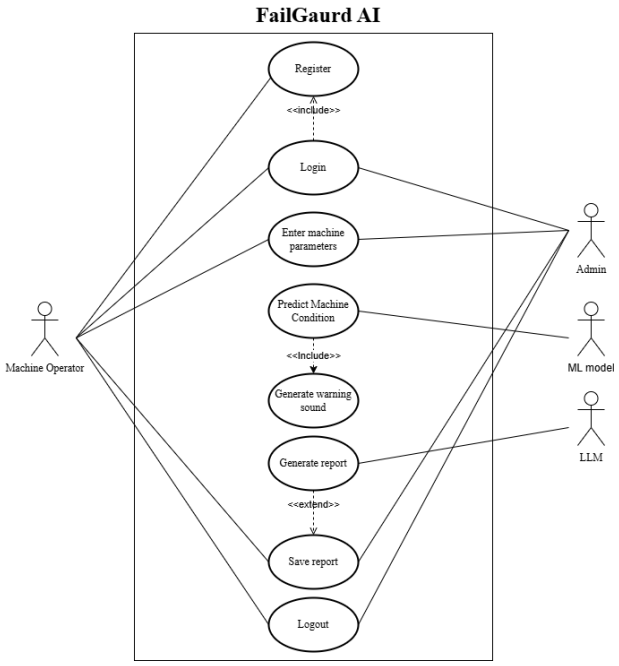


Figure 6.1: Use Case Diagram

Actors

Machine Operator Interacts with the system to monitor machine conditions.

Admin Manages user accounts and additional system administration tasks.

ML Model Used to predict the machine conditions based on the parameters.

LLM Used to generate reports and also a system component.

Use Cases

Login Allows users to access the system. Essential for security and personalized access.

Enter Machine Parameters The machine Operator inputs data about the machine. Triggers the prediction process.

Predict Machine Condition Uses the ML Model to analyze input parameters. Outputs a prediction about the machine's state.

Generate Warning Sound The system alerts the Machine Operator if a potential failure is detected.

Generate Report Creates a detailed report about the machine's condition. Uses the LLM to generate human-readable text.

Save Report Stores the generated report for future reference.

Logout Allows users to exit the system securely. Important for session management.

Relationships

Relationships in UML use case diagrams include Associations, which connect actors to use cases they interact with. `<<include>>` represents mandatory behavior, while `<<extend>>` indicates optional behavior that occurs under certain conditions.

6.2 Class Diagram

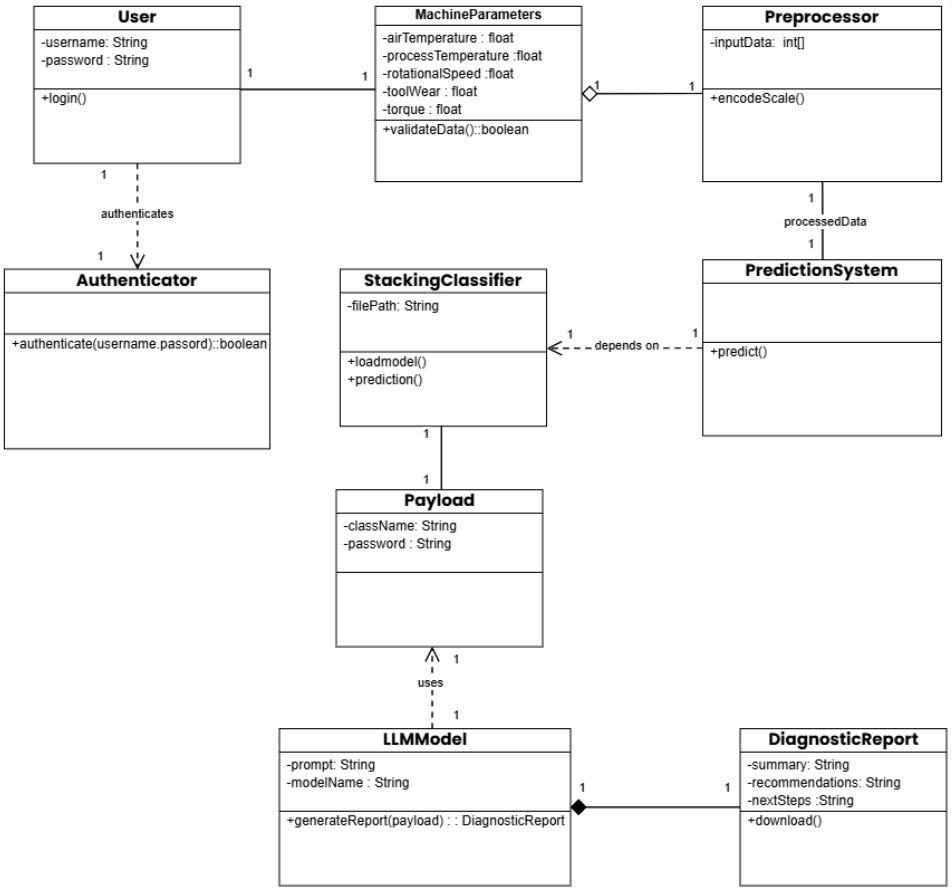


Figure 6.2: Class Diagram

Classes

User Represents a system user; has a username, password, and `login()` method.

Authenticator Authenticates users; has `authenticate(username, password)` to validate credentials.

MachineParameters Holds machine operation data; includes air temperature, process temperature, rotational speed, tool wear, and torque (all float).

Preprocessor Processes machine parameter data; has `inputData` (number array) and `encodeScale()`

to encode/scale data.

StackingClassifier Loads and uses a machine learning model; has `filePath` (model file path), `loadmodel()`, and `prediction()`.

PredictionSystem Performs prediction logic; uses `predict()` from StackingClassifier.

Payload Container for data needed for report generation.

LLMModel Generates reports using a Large Language Model; has `prompt` and `modelName`, and uses `generateReport(payload)`.

DiagnosticReport Represents the generated report; includes summary, recommendations, next steps, and a `download()` function.

6.3 Sequence Diagram

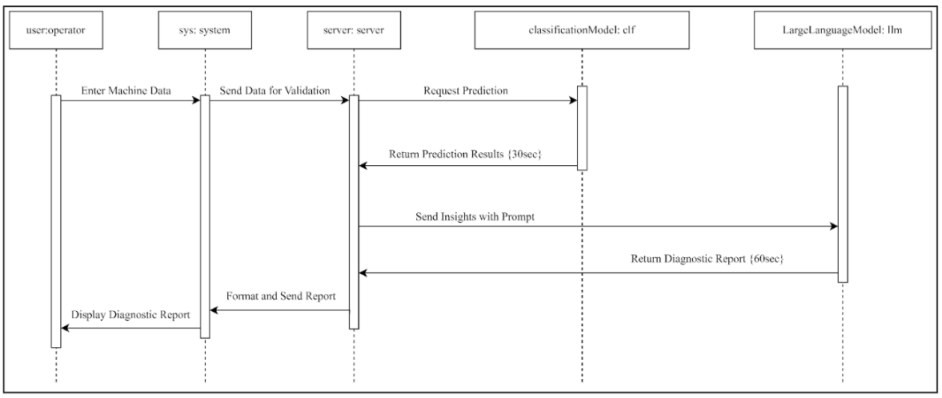


Figure 6.3: Sequence Diagram

Diagnostic Workflow Overview

User Input

1. **Operator**: Enters machine data into the system.

2. **WebApp System**: Submits data to the server for validation (transparent to the user).

Machine Learning Prediction

1. **Server**: Requests prediction from the classification model (clf).

2. **Classification Model**: Processes data $\rightarrow$ Returns prediction ($\approx$30 sec delay).

Report Generation

1. **Server**: Sends prediction + context to the LLM (LLM).

2. **LLM**: Generates a diagnostic report and returns the report to the server.

Final Output

1. **System**: Receives and formats the report $\rightarrow$ Displays results to the operator.

6.4 Activity Diagram

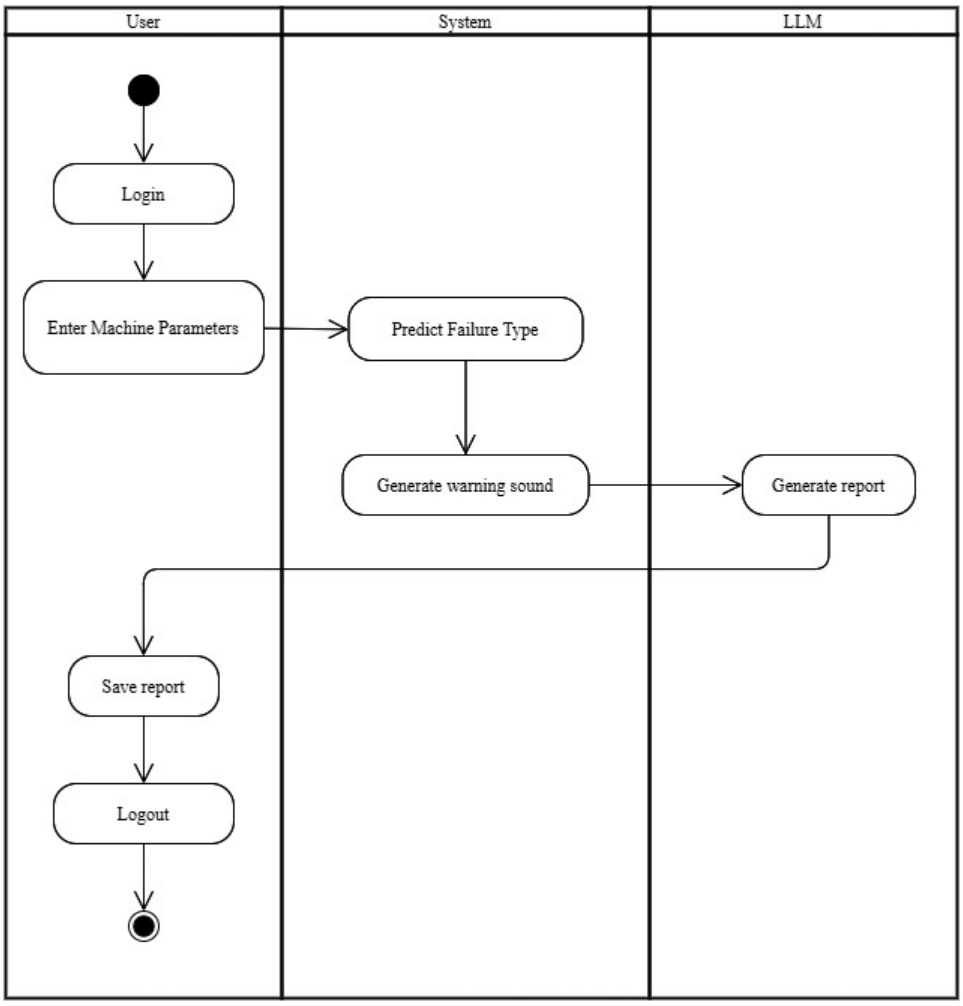


Figure 6.4: Activity Diagram

Partitions (Swimlanes)

User Represents the actions taken by either a user interacting with the system.

ML Model This partition represents actions or processes performed by the Machine Learning Model.

LLM This partition represents actions or processes performed by the Large Language Model.

Explanation of the Flow

1. **Start (Initial Node)**: The process begins.
2. **Login (Activity)**: The user or admin initiates a login to the system.
3. **Enter Machine Parameters (Activity)**: The user enters the required machine parameters.
4. **Predict Machine Condition (Activity)**: The ML Model analyzes the input parameters and makes a prediction.
5. **Generate Warning Sound (Activity)**: If the prediction indicates a potential issue, the system generates a warning sound.
6. **Generate Report (Activity)**: The LLM generates a report based on the machine's condition.
7. **Save Report (Activity)**: The user has the option to save the generated report.
8. **Logout (Activity)**: The user logs out of the system.
9. **End (Final Node)**: The process concludes.

6.5 ER Diagram

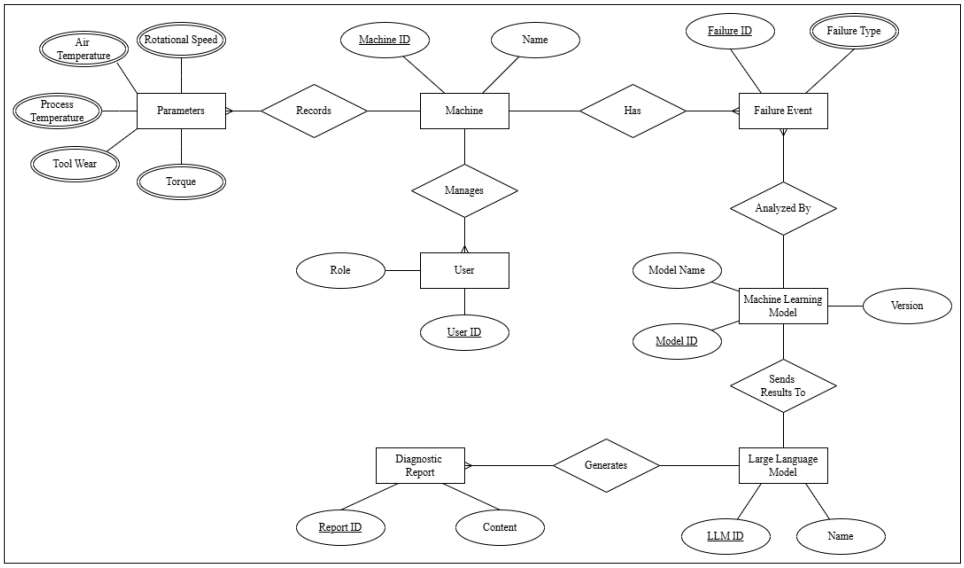


Figure 6.5: ER Model

Entities

Machine Represents the physical machine being monitored.

User Represents the human user interacting with the system.

Machine Learning Model Represents the machine learning model used for predictions.

Large Language Model Represents the language model used for generating reports.

Diagnostic Report Represents the report generated by the language model.

Records Represents records of machine data.

Failure Event Represents a recorded failure event of the machine.

Attributes

Machine Machine ID, Name.

User User ID, Role.

Machine Learning Model Model ID, Model Name, Version.

Large Language Model LLM ID, Name.

Diagnostic Report Report ID, Content.

Records Parameters (which are further broken down into Air Temperature, Process Temperature, Tool Wear, Rotational Speed, Torque).

Failure Event Failure ID, Failure Type.

6.6 Component Diagram

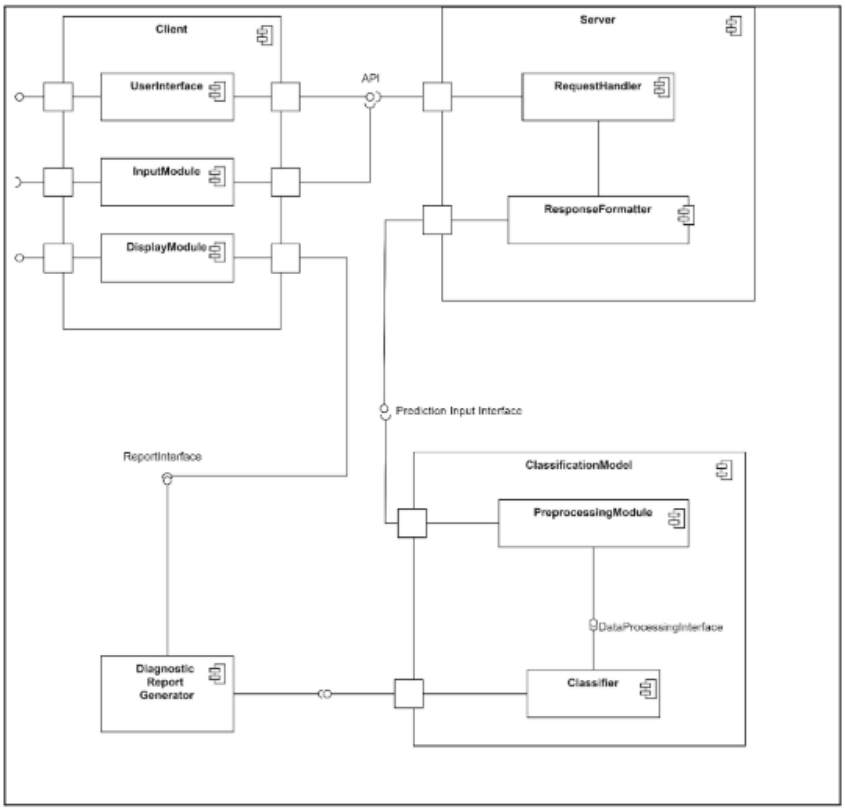


Figure 6.6: Component Diagram

Components

Client This is the component of the system where the user communicates.

UserInterface Handles the user's interaction with the system.

InputModule Determines how the data is entered by the user.

DisplayModule Used for presenting data to the user.

Server This is the part of the system that executes data and logic.

RequestHandler Handles requests from the client.

ResponseFormatter Modifies the server's responses so they can be returned to the client.

Diagnostic Report Generator Generates the diagnostic reports.

ClassificationModel Uses the machine learning classification model.

PreprocessingModule Prepares the input data for the classification model.

Classifier Performs the actual classification.

Interfaces

API Facilitating the communication between Client and Server.

Prediction Input Interface This is the way to input data into the classification model.

DataProcessingInterface Specifies the data preprocessing interface of the ClassificationModel.

ReportInterface Facilitates users in working with report generation functionality.

Dependencies

Client - API - Server The Server parts are depended upon by the Client parts via the API interface.

ReportInterface - Diagnostic Report Generator ReportInterface communicates with the Diagnostic Report Generator.

Server - ClassificationModel The Server classes utilize the ClassificationModel.

ClassificationModel - PreprocessingModule - Classifier ClassificationModel uses the PreprocessingModule and Classifier.

Chapter 7

Implementation

7.1 Module - Wise Implementation

7.1.1 Data Preprocessing

To ensure the data was useful for model training, data preprocessing steps were applied. Initially, the irrelevant feature “UID” was dropped. The redundant feature “Machine Failure” was also removed. Typecasting was performed to convert categorical features into numerical representations. Standard Scaling was applied on numerical features to bring all of them on the same scale. For feature engineering, a new class, “About to Fail,” was introduced based on thresholds to indicate when a machine component was nearing failure. This class is the most useful in real-world scenarios since the operator can take corrective action immediately. Further, the seven sparse failure mode features (No Failure, About to Fail, Tool Wear Failure, Heat Dissipation Failure, Power Failure, Overstrain Failure, and Random Failures) were merged into one dense target feature. This converted the problem from a multi-label to a multi-class classification problem. Label encoding was utilized to convert categorical target labels into numerical values. This step was essential for training machine learning models, as they require numerical input for classification tasks. Finally, the dataset was split into training and testing sets using a 70-30 ratio.

7.1.2 Bagging and Boosting

Classical machine learning algorithms were sure to give poor results on imbalanced datasets. Hence advanced machine learning algorithms were implemented to check the performance. First, a Bagging Classifier was implemented using Decision Trees as the base classifier. This method trained multiple weak Decision Tree models on bootstrapped subsets of the dataset and aggregated their predictions. Another bagging approach was applied using a Random Forest as the base classifier. This method is called the Bagging of Bagging model. This model enhanced model stability and handled imbalanced data. Among Boosting techniques, AdaBoost (Adaptive Boosting) and XGBoost were implemented. In Boosting techniques, the models are trained sequentially where each successor model gives more weightage to the misclassified points by its predecessor model. This helps focus on misclassified instances and increases accuracy.

7.1.3 Data Augmentation using SMOTE

The Synthetic Minority Oversampling Technique (SMOTE) is a statistical data augmentation technique. It takes into account the nearest neighbors while creating new points. It internally uses the K Nearest Neighbours algorithm to balance the data points of the minority class. It reduces the bias towards the majority class. Before augmentation, the number of samples of the "No Failure" class was 6747 which was the highest. After augmentation all the classes have 6747 samples increasing the size of the dataset from 10000 to 47229. To assess the quality of the data generated, cumulative distribution functions of all the features were plotted. From 7.1, it can be observed that SMOTE has done a decent job of capturing the patterns in data. Even if there are fewer samples of the failure modes, SMOTE was able to augment the datasets properly. The disadvantage of SMOTE is that it relies on statistical methods to augment the data. Due to this reason some gaps in real and fake data of the features "Torque" and "Tool Wear" are seen. To overcome this disadvantage, experimentation was done with the Gen-AI-based technique.

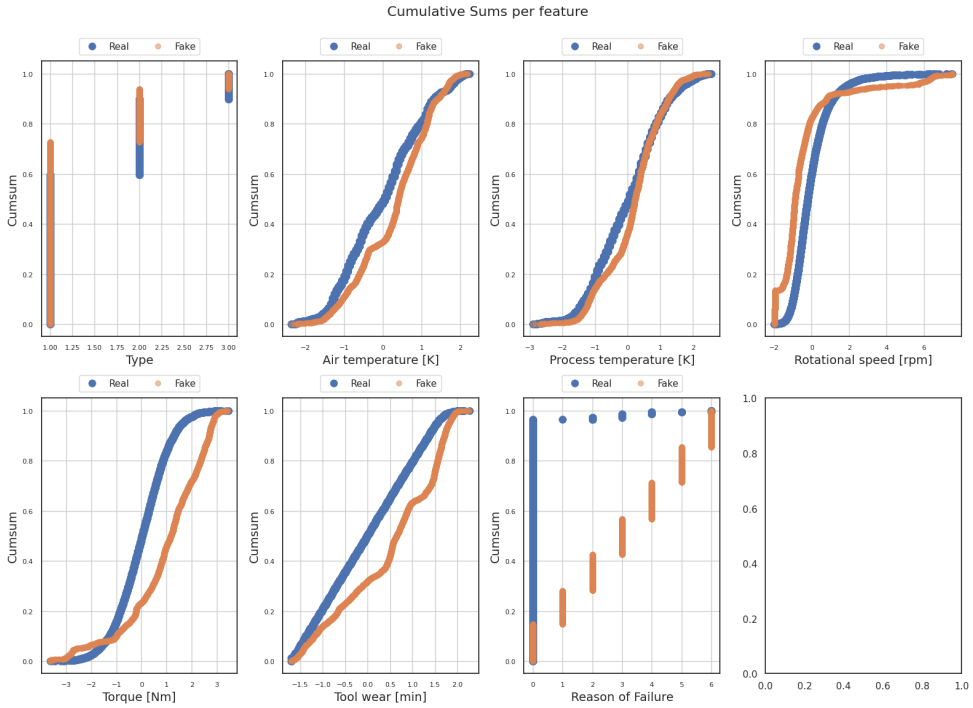


Figure 7.1: Quality of data generated by SMOTE.

7.1.4 Data Augmentation using CTGAN

Conditional Tabular Generative Adversarial Network (CTGAN) is a GenAI-based technique to generate consistent tabular data. A Generative Adversarial Network (GAN) is trained on the dataset. The Generator part of the GAN generates the fake data and the Discriminator tries to discriminate between the real and fake data. The GAN was fine-tuned to increase performance. The used hyperparameters are displayed in Table. By using this Deep Learning based approach it is ensured that the GAN will capture complex patterns in the data. The size of the dataset after augmentation is 47229. To assess the quality of the generated dataset, the cumulative distribution function of each feature was plotted. From 7.2, we can notice that GAN has done a much better job at capturing patterns in the data than SMOTE. This was possible because of the Universal Approximation Theorem for Deep Neural Networks which make the building block of GANs.

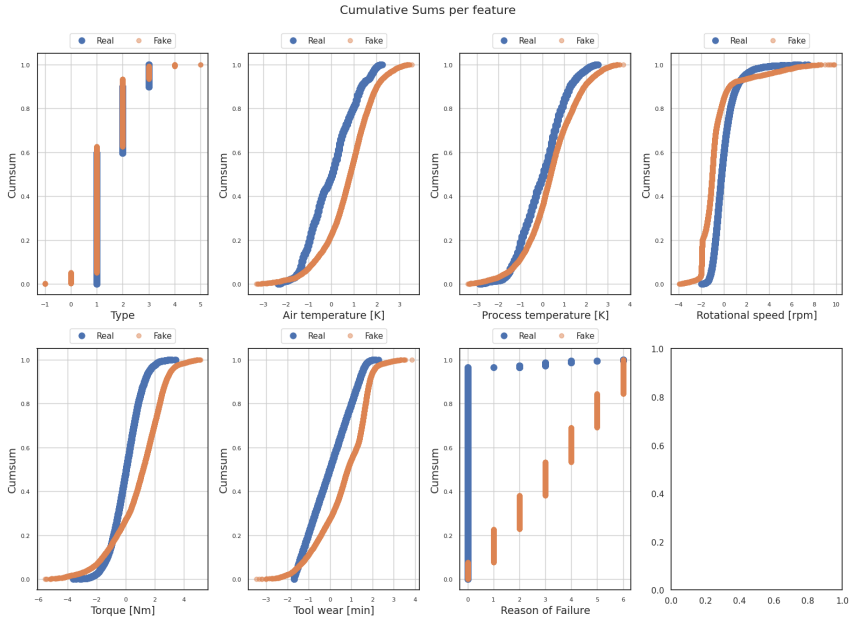


Figure 7.2: Quality of data generated by CTGAN.

Hyperparameter	Value
generator_lr	0.001
discriminator_lr	0.001
batch_size	200
epochs	500

Table 7.1: Hyperparameters used to finetune CTGAN

7.1.5 Building Stacking Model

To make a robust and generalized model, a stacking classifier was built. Stacking Classifier is an ensemble technique in which a model learns in two levels. The first level consists of various base models. The second level consists of a single model, called the meta-model. The Stacking Classifier learns from the predictions of the base models and makes the final prediction. This ensemble method helps in building a model with low bias and low variance which increases the performance of the model. In our case, the base models include the Decision Tree, Random Forest model, and Support Vector Machine trained on the CTGAN augmented dataset and Logistic Regression model, K Nearest Neighbour model, and Random Forest model trained on the SMOTE augmented dataset. The Support Vector Machine trained on the SMOTE augmented dataset was selected as a metamodel. To train the Stacking Classifier, both the datasets - CT-

GAN and SMOTE augmented datasets were aggregated. This aggregated dataset was split into a 7:3 ratio for training and testing respectively. Since the aggregated dataset is also balanced, the metrics will remain the same i.e. Precision, Recall, F1 Score, and Accuracy.

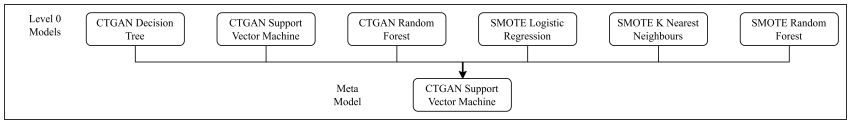


Figure 7.3: Stacking Classifier

7.1.6 Large Language Model Selection

For better interpretation of predictive maintenance results, an LLM was integrated into the system to generate the diagnostic report. The model used for this purpose is Llama 3.1 Nemotron 70B Instruct by Nvidia. This model is an improved version of the Llama 3.1 model by Meta. Nvidia improved this model to make the responses more human-like which makes this model helpful in real-world applications. To compare these two models the benchmark used was Arena Hard. This benchmark was chosen due to its features, one of which includes measuring agreement with human preference. The Arena Hard score of Llama 3.1 is 51.6 and that of Llama 3.1 Nemotron 70B Instruct is 70.9. The network architecture is the same as that of Llama 3.1. The architecture type used in this LLM is Transformer. Transformer architecture uses self-attention mechanisms to process long-range dependencies in textual data. It consists of encoder-decoder layers. The Encoder processes the data and the Decoder generates the response. A detailed prompt is crafted to generate the diagnostic report. The prompt was fine-tuned by experimentation. The final prompt used is shown in figure 7.4 This prompt ensures that the report generated is well-structured and easily interpretable making the maintenance process easier. The predictions from the Stacking Classifier and the above-detailed prompt from the payload. This payload is parsed to the LLM to generate the response.

```
Generate a diagnostic report based on the following machine parameters:
Type: {Type}
Air Temperature [K]: {Air_temp}
Process Temperature [K]: {Process_temp}
Rotational Speed [rpm]: {Rot_speed}
Torque [Nm]: {Torque}
Tool Wear [min]: {Tool_wear}
and the predicted failure class: {predicted_class} (one of the following: TWF - Tool Wear Failure,
HDF - Heat Dissipation Failure, OSF - Overstrain Failure, PWF - Power Failure,
RNF - Random Failure, About to Fail, or No Failure).
Include in the report:
A brief summary of the operational parameters.
Actionable recommendations tailored only to the specific failure class {predicted_class} provided.
Avoid listing or referring to other failure classes.
If the failure class is "No Failure," confirm that the machine is operating normally and
requires no immediate maintenance,
while suggesting continued monitoring and scheduled maintenance.
Ensure the output is concise, actionable, and relevant to the provided parameters.
```

Figure 7.4: Detailed prompt used

7.1.7 Website Development

A web application was developed to facilitate the predictive maintenance of the machines. The system follows a structured workflow. This workflow is shown in figure 7.5. The process starts with authenticating users. Upon successful login, the user is granted access to input machine parameters. These parameters are passed to the Stacking Classifier to predict failure class. This prediction along with the prompt is sent to the LLM to generate a report. The report is displayed on the user interface and further users can also download the report.

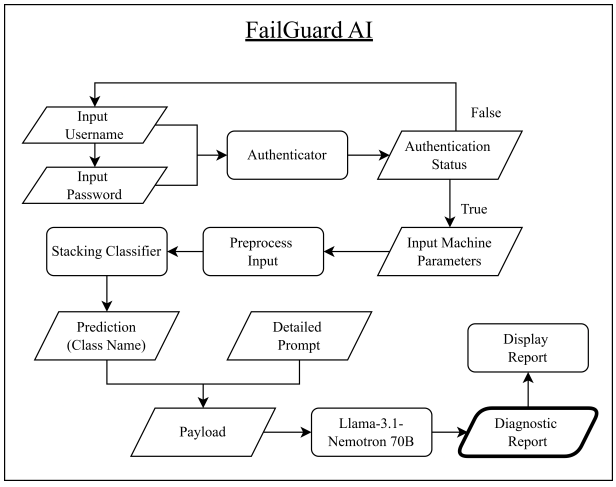


Figure 7.5: Web application workflow

7.2 Code Snippets and Explanation

7.2.1 Data Preprocessing

- Dropping irrelevant columns

```
features_to_drop = ["UID", "Product ID", "Machine failure"]
df = df.drop(features_to_drop, axis = True)
df.head()
```

Figure 7.6: Dropping irrelevant features

- A new class called “About to Fail” was added to the dataset according to thresholds. This class helps machine operators to implement corrective actions immediately.

```
df['About to Fail'] = df.apply(lambda row: 1 if (275 <= row['Air temperature [K]'] <= 295) or
                                                (275 <= row['Process temperature [K]'] <= 305) or
                                                (900 <= row['Rotational speed [rpm]'] <= 1200) or
                                                (2 <= row['Torque [Nm]'] <= 3) or
                                                (row['Tool wear [min]'] > 250)
                                                else 0, axis=1)
```

Figure 7.7: Adding a new class

7.2.2 Bagging and Boosting

- Bagging with Decision tree as base estimator: 50 Decision Tree stumps were aggregated together to form this classifier.

```
base_estimator = DecisionTreeClassifier(max_depth=10, random_state=42)
bagging_clf = BaggingClassifier(
    estimator=base_estimator,
    n_estimators=50,
    max_samples=0.8,
    max_features=0.8,
    bootstrap=True,
    random_state=42
)
```

Figure 7.8: Bagging Classifier with Decision tree as base estimator

- Bagging of Bagging Model: 100 Random Forest estimators were put sequentially to make this model.

```
base_estimator = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
bagging_clf_rf = BaggingClassifier(
    estimator=base_estimator,
    n_estimators=50,
    max_samples=0.8,
    max_features=0.8,
    bootstrap=True,
    random_state=42
)
```

Figure 7.9: Bagging Classifier with Random Forest as base estimator

- AdaBoost

```
estimator = DecisionTreeClassifier(max_depth=1,
                                   random_state=42)

adaboost_model = AdaBoostClassifier(estimator=estimator,
                                    n_estimators=50,
                                    learning_rate=0.001)
```

Figure 7.10: AdaBoost Classifier

- XGBoost

```
xgboost_model = XGBClassifier(
    n_estimators = 100,
    objective='multi:softmax',
    num_class=7,
    eval_metric='mlogloss',
)
```

Figure 7.11: XGBoost Classifier

```
smote = SMOTE(random_state=42, k_neighbors=183)

X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

Figure 7.12: Data Augmentation using SMOTE

7.2.3 Data Augmentation

- SMOTE Data Augmentation: KNN SMOTE was used with 183 nearest neighbors.
- CTGAN Data Augmentation: The GAN was finetuned with the parameters shown in figure 7.13. The GAN was trained for 500 epochs. This prolonged training helped to capture the patterns in data.

```
ctgan = CTGAN(generator_lr = 0.001,
              discriminator_lr = 0.001,
              batch_size = 200,
              epochs = 500,
              verbose = True
              )
```

Figure 7.13: Data Augmentation using CTGAN

7.3 Database Implementation

The FailGuard AI system employs a PostgreSQL database to manage user logins and store prediction outputs as well as machine data. It is hosted via the psycopg2 library in Python, which aids in saving crucial information securely. This enables us to monitor and analyze machine conditions over time.

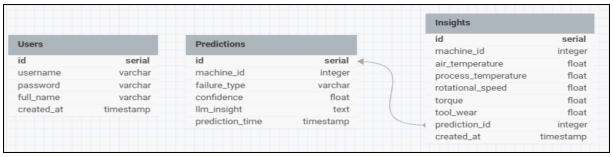


Figure 7.14: Database Schema

```
postgres=# \d users;
                                    Table "public.users"
   Column   |            Type             | Collation | Nullable |              Default
------------+-----------------------------+-----------+----------+-----------------------------------
 id         | integer                     |           | not null | nextval('users_id_seq'::regclass)
 username   | character varying(50)       |           | not null |
 password   | text                        |           | not null |
 created_at | timestamp without time zone |           |          | CURRENT_TIMESTAMP
 full_name  | character varying(100)      |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
    "users_username_key" UNIQUE CONSTRAINT, btree (username)
```

Figure 7.15: User Table

```
postgres=# \d insights;
                                    Table "public.insights"
       Column        |            Type             | Collation | Nullable |              Default
---------------------+-----------------------------+-----------+----------+-----------------------------------
---
 id                  | integer                     |           | not null | nextval('insights_id_seq'::regclas
s)
 machine_id          | integer                     |           | not null |
 air_temperature     | double precision            |           | not null |
 process_temperature | double precision            |           | not null |
 rotational_speed    | double precision            |           | not null |
 torque              | double precision            |           | not null |
 tool_wear           | double precision            |           | not null |
 prediction_id       | integer                     |           |          |
 created_at          | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "insights_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "insights_prediction_id_fkey" FOREIGN KEY (prediction_id) REFERENCES predictions(id) ON DELETE CASCADE
```

Figure 7.16: Insights Table

```
postgres=# \d predictions;
                                   Table "public.predictions"
     Column      |            Type             | Collation | Nullable |                Default
-----------------+-----------------------------+-----------+----------+---------------------------------------
--
 id              | integer                     |           | not null | nextval('predictions_id_seq'::regclass
)
 machine_id      | character varying(50)       |           | not null |
 failure_type    | character varying(50)       |           | not null |
 confidence      | double precision            |           | not null |
 llm_insight     | text                        |           |          |
 prediction_time | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "predictions_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "insights" CONSTRAINT "insights_prediction_id_fkey" FOREIGN KEY (prediction_id) REFERENCES predictio
ns(id) ON DELETE CASCADE
```

Figure 7.17: Predictions Table

Chapter 8

Testing

8.1 Functionality Testing

Functionality testing was performed to ensure that the FailGuard AI system performs as expected in all aspects. This testing confirms that data preparation, machine learning model prediction, Large Language Model (LLM) report generation, and web application interface are accurately integrated to provide a reliable predictive maintenance solution.

Test Case ID	Test Scenario	Test Steps	Input Data	Expected Output	Actual Output
TC001	Verify User Login	1. Open WebApp 2. Enter username/password 3. Click “Login”	Username: admin, Password: admin123	Redirects to dashboard	Redirected successfully
TC002	Validate Input Fields	1. Enter invalid values 2. Click submit	(Empty fields)	Error “Fields cannot be empty”	Error displayed
TC003	Machine Prediction	1. Enter machine parameters 2. Click “Predict”	[1, 300, 297.99, 2000, 85, 250]	Returns Overstrain Failure (OSF)	Overstrain Failure (OSF) detected
TC004	LLM Report Generation	1. Predict failure 2. Generate report	Failure Type: OSF	OSF-specific recommendations	OSF Recommendations
TC005	Report Download	1. Click “Save Report”	N/A	File download Successful	File downloaded

Table 8.1: Test Case Table

8.2 Performance Testing

Performance testing was used to measure the efficiency, speed, and scalability of the FailGuard AI system. Data preprocessing, failure prediction with a stacking classifier, generation of a diagnostic report with the Llama 3.1 Nemotron 70B Instruct LLM, and a Streamlit web application are part of the system. Response time for prediction and report generation, system performance under load (Locust), and resource usage are some of the important metrics captured. For FailGuard AI, they are:

Response Time Time taken for prediction, report generation, and web app interactions.

Load Testing System performance under simulated user load (using Locust).

Resource Usage CPU and memory utilization.

Key Details Provided:

- Locust Load Testing: 15 users, spawn rate of 3 seconds, tested on localhost.
- Diagnostic Report Generation Time: 15–20 seconds (due to the Llama 3.1 Nemotron 70B Instruct LLM, likely hosted on a cloud server).

8.3 Testing Methodologies Used

The approaches employed are unit Testing, Integration Testing, Functional Testing, and Performance Testing. It helped in verifying how the system functions at different levels, from individual components to the entire system under stress. Each approach examined specific components of the system to ensure everything was thoroughly tested.

Unit Testing

Unit tests were run to verify each part of the system independently. For instance, the `preprocess_input` function was checked to ensure that it correctly processes input data. The prediction logic of the stacking classifier was run on a single input to ensure the model was executed correctly.

```python
class TestWebApp(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """Setup test database connection before all tests"""
        cls.db = Database(DB_CONFIG)
        print("\n✅ Database connection setup successfully for testing.")

    def test_authenticate_user_success(self):
        """Test successful user authentication"""
        correct_password = "pln123"
        hashed_password = bcrypt.hashpw(correct_password.encode("utf-8"), bcrypt.gensalt()).decode("utf-8")

        with patch.object(Database, "get_user", return_value=(3, "nilesh", hashed_password)):
            is_authenticated, username = authenticate_user("nilesh", correct_password)
            self.assertTrue(is_authenticated)
            self.assertEqual(username, "nilesh")
            print("✅ Authentication success test passed.")

    def test_authenticate_user_failure(self):
        """Test user authentication with invalid credentials"""
        with patch.object(Database, "get_user", return_value=None):
            is_authenticated, username = authenticate_user("wronguser", "wrongpass")
            self.assertFalse(is_authenticated)
            self.assertIsNone(username)
            print("✅ Authentication failure test passed.")
```

```python
@patch("requests.post")
def test_send_to_vext_success(self, mock_post):
    """Test successful response from Vext API"""
    mock_response = MagicMock()
    mock_response.status_code = 200
    mock_response.json.return_value = {"text": "Sample diagnostic report"}
    mock_post.return_value = mock_response

    payload = "Test payload data"
    response = send_to_vext(payload)

    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json()["text"], "Sample diagnostic report")
    print("✅ Vext API request test passed.")
```

Figure 8.1: Unit Tests

Integration Testing

Integration testing was all about making sure the parts of the system were communicating well. We tested the `preprocess_input` function and the stacking classifier to make sure how data was being processed. We tested the predicted class with the LLM to produce reports, and it showed that while the LLM produced reports, the data showed the correct predicted class.

Functional Testing

Functional testing confirmed that the system met all the requirements defined. The testing took into account user login, input validation, failure predictions, report generation, and downloading reports. Login and input checks were seamless. The web application was capable of displaying and downloading reports.

Performance Testing

Performance testing verifies the system's performance and its scalability (Sections 8.2 and 8.7). The prediction time and reporting time were also measured to verify system efficacy. Load testing using Locust with 15 users and a 3-second start rate had an average response time of 1.5 seconds with no failures, which indicates it's suitable for small-scale use.

8.4 Database Independence Testing

The system relies on the database for user authentication (users table), storage of prediction results (predictions table), and logging machine parameters (insights table). Used unit test to check database connection (`setUpClass`), mock user data, and checked storage of predictions (`machine_id=1`, `failure_type="TWF"`, `confidence=0.95`) and insights ([300.0, 297.99, 2000.0, 85.0, 250.0]).

```python
def test_store_prediction_in_database(self):
    """Test if prediction and insights are stored correctly"""
    machine_id = "1"
    failure_type = "TWF"
    confidence = 0.95
    diagnostic_report = "Test diagnostic report"

    prediction_id = self.db.save_prediction_with_insight(machine_id, failure_type, confidence, diagnostic_report)
    self.assertIsInstance(prediction_id, int)
    print("✅ Prediction storage test passed.")

    # Now store insights
    result = self.db.save_insight(machine_id, 300.0, 297.99, 2000.0, 85.0, 250.0, prediction_id)
    self.assertTrue(result)
    print("✅ Insight storage test passed.")
```

Figure 8.2: Unit Test for Database Storage

Test Result:

- Pass: Database connection setup successfully for testing.
- Pass: Authentication failure test passed.
- Pass: Authentication success test passed.
- Pass: Vext API request test passed.
- Pass: Prediction storage test passed.
- Pass: Insight storage test passed.

All tests were completed successfully. Ran 4 tests in 0.586s. OK

8.5 Test Environment & Tools

The following summarizes the hardware and software setup used for testing the FailGuard AI system:

Component	Details
Hardware	Intel i5 processor, 8 GB RAM, 512 GB SSD
Operating System	Windows 10 (64-bit)
Software	Python 3.12 Streamlit (for the web application) Llama 3.1 Nemotron 70B Instruct (cloud-hosted LLM for report generation)
Dataset	AI4I 2020 Predictive Maintenance Dataset (10,000 Data Points)
Network	Localhost is used for web app and load testing; internet connection is used for cloud-hosted LLM

Table 8.2: Hardware/Software Setup

Tool	Purpose	Usage in Testing
Python unittest	Unit testing framework for Python	Used to test individual components, such as `preprocess_input`, login, report generation, `save_report`, etc.
Locust	Open-source load testing tool for web applications	Conducted performance testing with 15 users (spawn rate of 3 seconds) on localhost to evaluate the web app's response time
Python time module	Measure the execution time of specific operations	Measured times for login, prediction, and report generation
Chrome Dev Tools	Measures performance and memory usage of the web app	Check memory usage and availability of web app

Table 8.3: Testing Tools

8.6 Manual Test Cases

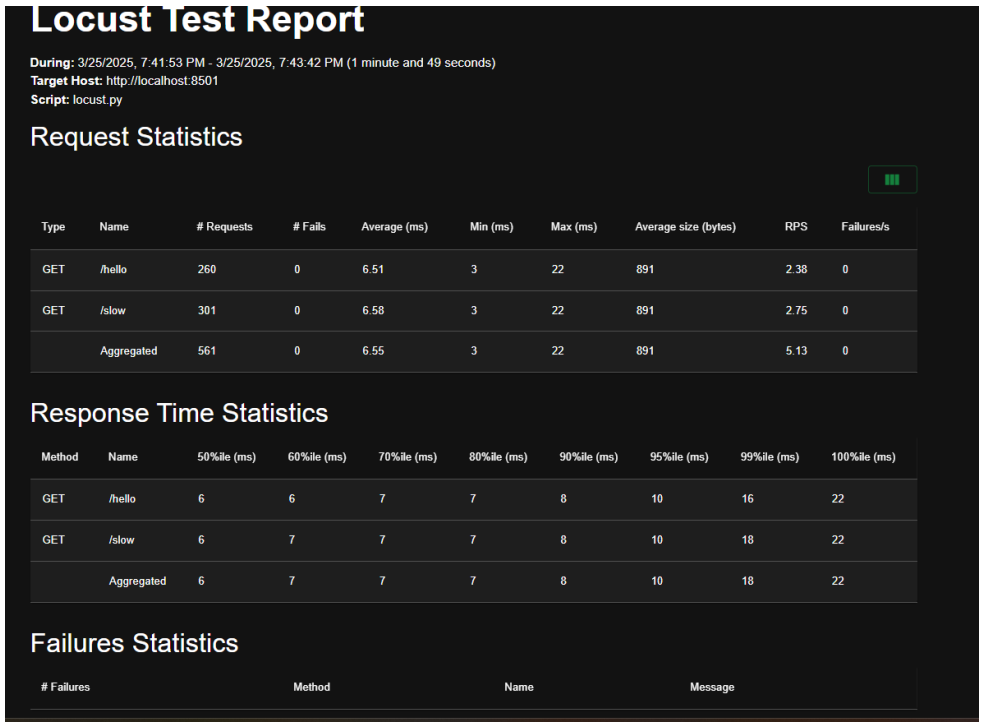
Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Post Conditions	Expected Result	Actual Result	Status	Executed by	Executed Date
TC01	Verify user login functionality	1. Open web app 2. Enter username and password 3. Click "Login"	Web app is accessible	Username: "operator1"Password: "pass123"	User is logged in	User lands on input page after successful login	User landed on input page	Pass	Pranav Narkhede	26/03/2025
TC02	Check login with wrong credentials	1. Open web app 2. Enter wrong username/password 3. Click "Login"	Web app is accessible	Username: "wronguser"Password: "wrongpass"	User remains on login page	Error message: "Invalid username/password" displayed	Error message: "Invalid username/password" displayed	Pass	Pranav Narkhede	26/03/2026
TC03	Test machine parameter input	1. Log in 2. Enter parameters 3. Click "Submit"	User is logged in	Air Temp: 300K Process Temp: 298K RPM: 2000 Torque: 85Nm Wear: 250min	Parameters are submitted	Parameters accepted, system processes them (no error)	Parameters accepted, no error displayed	Pass	Pranav Narkhede	26/03/2027
TC04	Test input with invalid data	1. Log in 2. Enter invalid values 3. Click "Submit"	User is logged in	Air Temp: "abc" Process Temp: -5K RPM: "xyz"	Submission fails	We cannot enter values other than integers	Values cannot be entered	Pass	Pranav Narkhede	26/03/2028
TC05	Verify failure prediction (OSF)	1. Log in 2. Enter OSF-like data 3. Submit	User is logged in	Air Temp: 300K Process Temp: 298K RPM: 2000 Torque: 85Nm Wear: 250min	Prediction is made	Prediction: "Overstrain Failure (OSF)" displayed	Prediction: "Overstrain Failure (OSF)" displayed	Pass	Pranav Narkhede	26/03/2029
TC06	Verify "No Failure" prediction	1. Log in 2. Enter normal data 3. Submit	User is logged in	Air Temp: 290K Process Temp: 300K RPM: 1500 Torque: 40Nm Wear: 100min	Prediction is made	Prediction: "No Failure" displayed	Prediction: "No Failure" displayed	Pass	Pranav Narkhede	26/03/2030

8.7 Manual Test Cases

TC07	Test diagnostic report generation	1. Log in 2. Enter OSF data 3. Submit 4. View report	User is logged in	Air Temp: 300K Process Temp: 298K RPM: 2000 Torque: 85Nm Wear: 250min	Report is generated	Report matches (e.g., "Overstrain Failure" with actionable steps)	Report matches with OSF and steps	Pass	Pranav Narkhede	26/03/2031
TC08	Check report download functionality	1. Log in 2. Enter data 3. Submit 4. Click "Download Report"	Report is generated	Air Temp: 300K Process Temp: 298K RPM: 2000 Torque: 85Nm Wear: 250min	Report is downloaded	PDF file with diagnostic report saved to device	PDF report downloaded successfully	Pass	Pranav Narkhede	26/03/2032
TC09	Test "About to Fail" detection	1. Log in 2. Enter threshold data 3. Submit	User is logged in	Air Temp: 280K Process Temp: 290K RPM: 1100 Torque: 2.5Nm Wear: 300min	Prediction is made	Prediction: "About to Fail" with report suggesting preventive steps	Prediction: "About to Fail" with preventive steps	Pass	Pranav Narkhede	26/03/2033
TC10	Verify UI responsiveness	1. Log in 2. Resize browser 3. Enter data4. Submit	User is logged in	Air Temp: 290K Process Temp: 300KRPM: 1500Torque: 40NmWear: 100min	UI adjusts properly	Interface stays usable, buttons and fields align on all screen sizes	UI adjusted, remained usable on resize	Pass	Pranav Narkhede	26/03/2034
TC11	Test prediction with poor internet	1. Log in 2.Enter data 3. Disconnect internet4. Submit	User is logged in, internet initially on	Air Temp: 300K Process Temp: 298K RPM: 2000 Torque: 85Nm Wear: 250min	Submission fails due to connectivity	Connection lost	No response, system timed out	Fail	Pranav Narkhede	26/03/2035
TC12	Test report with data	1. Log in 2. Enter values 3. Submit 4. View report	User is logged in	Air Temp: 500K Process Temp: 600K RPM: 5000 Torque: 200Nm Wear: 1000min	Report generation attempted	"Unknown Failure"	No meaningful output	Fail	Pranav Narkhede	26/03/2036

Figure 8.3: Manual Test Cases

8.8 Performance Testing Results



Locust Test Report

During: 3/25/2025, 7:41:53 PM - 3/25/2025, 7:43:42 PM (1 minute and 49 seconds)
Target Host: http://localhost:8501
Script: locust.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/hello	260	0	6.51	3	22	891	2.38	0
GET	/slow	301	0	6.58	3	22	891	2.75	0
	Aggregated	561	0	6.55	3	22	891	5.13	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/hello	6	6	7	7	8	10	16	22
GET	/slow	6	7	7	7	8	10	18	22
	Aggregated	6	7	7	7	8	10	18	22

Failures Statistics

# Failures	Method	Name	Message

Figure 8.4: Locust Test Tool Report

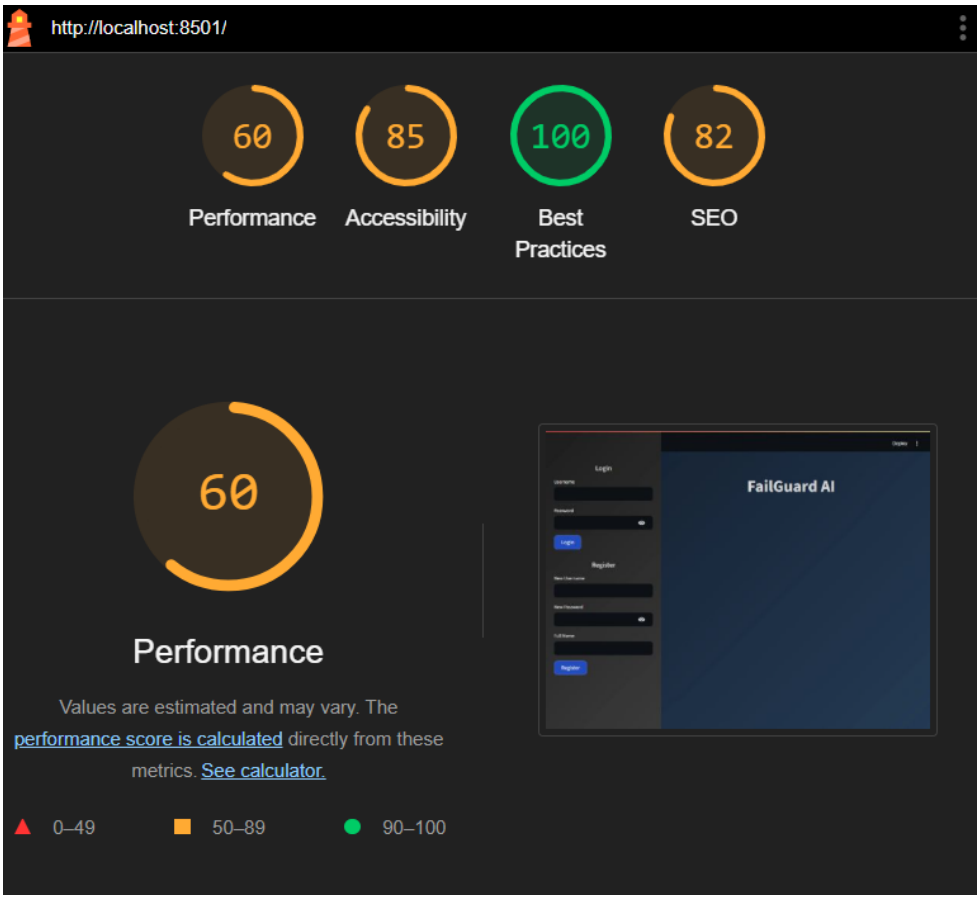


Figure 8.5: Webapp Analytics

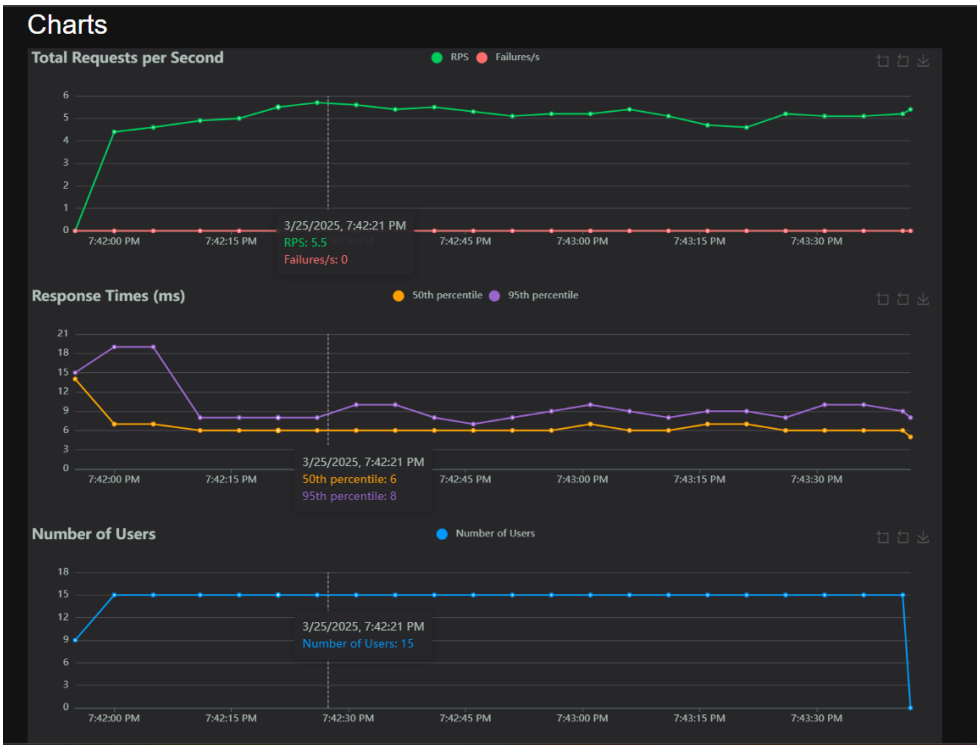


Figure 8.6: Locust Tool Statistics

Key Performance Metrics and Analysis:

Login Time Super fast (0.07s), ensuring a smooth user experience.

Prediction Time Quick (0.05s), ideal for real-time analysis.

Report Generation Time 13.95s, LLM latency is the bottleneck.

Locust Load Test (15 Users, Spawn Rate: 3s) Stable under load, handled requests efficiently.

Average Response Time Fast (6.65ms), no noticeable delays.

Requests Per Second (RPS) Moderate throughput (5.13 RPS), good for small-scale use.

Failure Rate 0%, perfect reliability across 561 requests.

Response Time Percentiles 95% of requests were completed in ¡10ms, max response time was 22ms.

Web Performance Score 60 (moderate), mainly due to slow report generation.

Accessibility 85 (good), minor UI tweaks needed for better usability.

Best Practices 100 (perfect), adheres to modern web standards.

SEO Score 82, can be improved with better metadata.

Locust Graph Insights Users ramped up to 15 with a 3s spawn rate, and response times remained consistent.

Chapter 9

Results and Discussions

9.1 Experimental Setup and Testing

9.1.1 Metrics For Imbalanced Dataset

Metrics were decided to evaluate model performance on the original imbalanced dataset. These included:

- Precision: To measure the accuracy of failure predictions, critical for minimizing false alarms in maintenance.
- Recall: To ensure most failure instances were captured, vital for predictive maintenance reliability.
- F1 Score: A metric to strike a balance between precision and recall.

These metrics were chosen over accuracy due to the dataset's imbalance, where the majority class (No Failure) was in the majority. The accuracy will always be high but it is not useful for failure prediction.

9.1.2 Metrics For Balanced Dataset

After data augmentation, the dataset is balanced. This allows for accuracy to be a metric of evaluation. Hence the metrics are Precision, Recall, F1 Score, and Accuracy. These metrics will

be used to evaluate the performance of machine learning algorithms on the augmented datasets and also the performance of the Stacking Classifier.

9.2 Observed Results

9.2.1 Bagging And Boosting Results

Among the ensemble methods, Bagging and Boosting algorithms were implemented. From 9.1, it is evident that the XGBoost model has outperformed the imbalanced dataset with an average Precision of 0.69, Recall of 0.63, and F1 Score of 65%. AdaBoost performed very poorly with an average F1 Score of only 22%. Bagging with Decision Tree performed moderately well achieving average precision of 54%, Recall of 31%, and F1 Score of 36%. Bagging with Random Forest led to a lower F1 Score of 27% due to a lower Recall rate of 24%.

Algorithm	**Average Precision**	**Average Recall**	**Average F1 Score**
Bagging with Decision Tree	0.54	0.31	0.36
Bagging with Random Forest	0.55	0.24	0.27
AdaBoost	0.23	0.21	0.22
XGBoost	0.69	0.63	0.65

Table 9.1: Bagging and Boosting Results

9.2.2 Machine Learning Algorithms Results

Classical Supervised Machine Learning algorithms were implemented on the augmented datasets. The implemented algorithms include K Nearest Neighbours, Support Vector Machines, Logistic Regression, Naive Bayes, Decision Tree, and Random Forest. Each model was trained on a preprocessed and balanced dataset ensuring low bias. The performance was evaluated using the metrics Precision, Recall, F1 Score, and Accuracy. The values of metrics are exhibited in Table 9.2.

Dataset	Algorithm	Avg. Precision	Avg. Recall	Avg. F1 Score	Accuracy
SMOTE augmented	K Nearest Neighbours	0.93	0.93	0.92	0.92
	Support Vector Machines	0.98	0.98	0.98	0.98
	Logistic Regression	0.91	0.91	0.91	0.91
	Naive Bayes	0.83	0.83	0.82	0.82
	Decision Tree	1.00	1.00	1.00	1.00
	Random Forest	1.00	1.00	1.00	0.99
CTGAN augmented	K Nearest Neighbours	0.86	0.83	0.83	0.84
	Support Vector Machines	0.89	0.88	0.88	0.89
	Logistic Regression	0.72	0.71	0.71	0.73
	Naive Bayes	0.85	0.84	0.84	0.86
	Decision Tree	0.88	0.88	0.88	0.89
	Random Forest	0.91	0.90	0.90	0.91

Table 9.2: Evaluation Metrics of Supervised Machine Learning Algorithms

9.2.3 Stacking Classifier Results

The Stacking Classifier achieved an overall accuracy of 95% on the aggregated test dataset. Average Precision, Recall, and F1 Score are 95%. The highest F1 Score was gained by the class “About to Fail” which is 98%. Considering real-time environments, this class is the most useful. Repair steps can be taken immediately if the machine conditions are classified as “About to Fail”. Fig.36 shows PR curves for all classes.

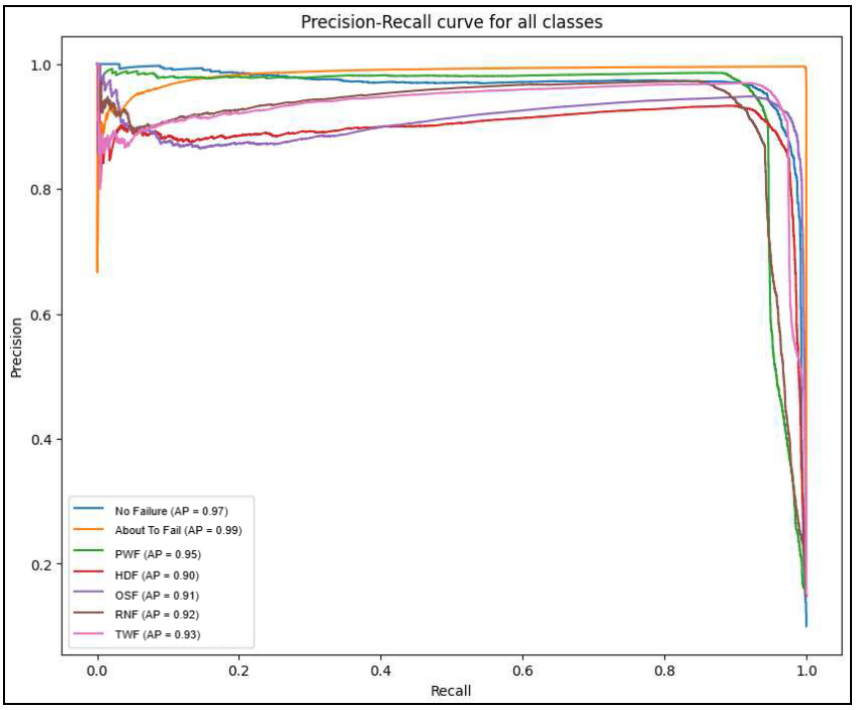


Figure 9.1: Precision-Recall Curves

9.2.4 Sample Report

An example report generated is shown in 9.2. This report was generated after the machine conditions were classified as “Over Strain Failure”. The system is able to give correct recommended actions. Implementing the actions can reduce the downtime in the industry.

Diagnostic Report

Operational Parameters Summary:

Air Temperature: 300.0 K (normal range assumed: 293-308 K)
Process Temperature: 297.99 K (slightly below air temperature, indicating efficient heat dissipation)
Rotational Speed: 2000.0 rpm (within typical operational ranges for many machinery types)
Torque: 85.0 Nm (moderate load)
Tool Wear: 250.0 minutes (cumulative, with no immediate indication of excessive wear based on time alone)

Predicted Failure Class: Overstrain Failure (OSF)

Actionable Recommendations (Tailored to Overstrain Failure):

1. Immediate Load Assessment and Adjustment:
   Review current operational tasks to identify potential overloading.
   Adjust the workload to reduce torque levels, if possible, to prevent further strain.

2. Enhanced Monitoring of Rotational Speed and Torque:
   Increase the frequency of checks on rotational speed and torque to catch any sudden spikes.
   Consider implementing automated alerts for predefined threshold exceedances.

3. Structural Integrity Check:
   Schedule a thorough inspection of the machine’s structural components to ensure they are within design specifications for handling the current operational loads.

4. Operational Parameter Fine-Tuning:
   Collaborate with engineering teams to optimize air and process temperature settings, which might help in reducing the overall strain on the machine during operation.

Next Steps:

Implement the above recommendations within the next 48 hours.
Continue monitoring machine parameters closely for any signs of deterioration.
Schedule a follow-up diagnostic check after implementing the recommendations to assess their effectiveness in mitigating the risk of Overstrain Failure.

Figure 9.2: Sample Report

9.2.5 Website

Streamlit was used to build a web application. The Stacking Classifier and the Large Language Model was integrated into this website. Some screenshots of the web app can be seen in figures 9.3, 9.4, 9.5.

Figure 9.3: User Authentication

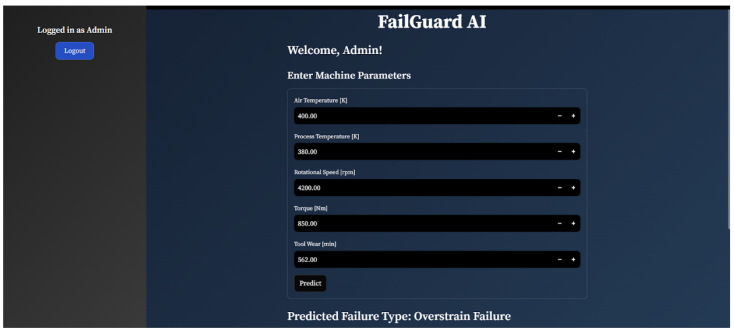


Figure 9.4: Predicting Failure

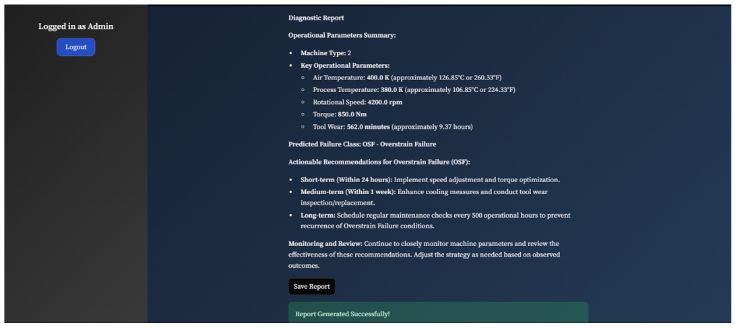


Figure 9.5: Diagnostic Report and "Save Report" feature

9.3 Analysis and Interpretation

9.3.1 Impact of Data Augmentation

Before data augmentation, the models were struggling to predict the mode of failure due to natural imbalance of data. The failure classes had very few samples, causing the machine learning algorithms to be biased to the majority class i.e "No Failure", leading to biased predictions and poor recall for minority classes. After applying data augmentation techniques, the dataset became balanced, ensuring an equal distribution of all failure modes. This helped the models learn more accurately, reducing bias and improving overall classification performance.

9.3.4 Real-World Applicability

FailGuard shows promise for practical use. It catches failures early, the app is user-friendly, and the reports give clear guidance. The system can be readily deployed in the industry to make predictive maintenance easier. This will result in saving unnecessary waste of funds on

traditional maintenance strategies.

9.4 Comparison with Existing Solutions

Some of the existing systems use rule-based algorithms to predict the failure of a machine. FailGuard AI uses advanced machine learning and ensemble learning algorithms to predict failure. The majority of existing solutions focus on only one system like railways or engines, but FailGuard AI is trained on key operational parameters which can be found in any kind of system. This makes our solution more generalized and robust to any machine. Some of the existing systems only classify the machine conditions into Failure or no Failure. FailGuard AI gives the reason for failure as well as the actionable steps to be carried out by generating a diagnostic report with the help of a large language model.

Chapter 10

Conclusions and Future Work

10.1 Advantages

The system is able to fulfill all the objectives i.e. Failure Prediction, Type of Failure and Diagnostic Report. Besides objectives, the system also delivers the identified literature gaps

- Building a robust generalized System
- Identifying the reason why the machine failed
- Giving instructions to operators through the report

10.2 Disdvantages

- Every dataset is naturally imbalanced, which makes predictions difficult.
- The diagnostic report generated by the LLM depends heavily on its prompt and fine-tuning. It may produce inaccurate or irrelevant information if the input is misinterpreted or incomplete.
- The system requires a constant internet connection.

10.3 Future Work

- Fine-tune LLM for the specific use case.
- Implement a RAG (Retrieval Augmented Generation) architecture for more relevant results.
- Enhance the explainability of results with the help of XAI architectures like SHAP (Shapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanation)
- Validate the responses with the help of SMEs.
- Experiment on various LLMs using various fine-tuned prompts for better results.

Bibliography

[1] Achouch, Mounia, et al. (2022). On predictive maintenance in industry 4.0: Overview, models, and challenges. *Applied Sciences*, 12(16), 8081.

[2] Paolanti, Marina, et al. (2018). Machine learning approach for predictive maintenance in industry 4.0. In *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*. IEEE.

[3] Susto, Gian Antonio, et al. (2014). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3), 812-820.

[4] Ferdousi, Rahatara, et al. (2024). DefectTwin: When LLM Meets Digital Twin for Railway Defect Inspection. *arXiv preprint arXiv:2409.06725*.

[5] Bansal, Dheeraj, David J. Evans, and Barrie Jones. (2004). A real-time predictive maintenance system for machine systems. *International Journal of Machine Tools and Manufacture*, 44(7–8), 759–766.

[6] Hegde, Raveendra, and Saurabh Sharma. (2024). Self supervised LLM customizer (SSLC): Customizing LLMs on unlabeled data to enhance contextual question answering.

[7] Sipos, Ruben, et al. (2014). Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[8] Prabhod, Kummaragunta Joel. (2021). Advanced Machine Learning Techniques for Predictive Maintenance in Industrial IoT: Integrating Generative AI and Deep Learning for Real-Time Monitoring. *Journal of AI-Assisted Scientific Discovery*, 1(1), 1–29.

[9] Lowin, Maximilian. (2024). A Text-Based Predictive Maintenance Approach for Facility Management Requests Utilizing Association Rule Mining and Large Language Models. *Machine Learning and Knowledge Extraction*, 6(1), 233–258.

[10] Kurkute, Mahadu Vinayak, Gunaseelan Namperumal, and Akila Selvaraj. (2023). Scalable Development and Deployment of LLMs in Manufacturing: Leveraging AI to Enhance Predictive Maintenance, Quality Control, and Process Automation. *Australian Journal of Machine Learning Research & Applications*, 3(2), 381–430.

[11] Zhou, Nianjun, et al. (2024). Towards Automated Solution Recipe Generation for Industrial Asset Management with LLM. *arXiv preprint arXiv:2407.18992*.

[12] Pinto, Riccardo, and Tania Cerquitelli. (2019). Robot fault detection and remaining life estimation for predictive maintenance. *Procedia Computer Science*, 151, 709–716.

[13] Verhagen, Wim JC, and Lennaert WM De Boer. (2018). Predictive maintenance for aircraft components using proportional hazard models. *Journal of Industrial Information Integration*, 12, 23–30.

[14] Vianna, Wlamir Olivares Loesch, and Takashi Yoneyama. (2017). Predictive maintenance optimization for aircraft redundant systems subjected to multiple wear profiles. *IEEE Systems Journal*, 12(2), 1170–1181.

[15] Amrit Nath, Nagdev, and Tarun Gupta. (2018). A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*. IEEE.

[16] Hosamo, Haidar Hosamo, et al. (2022). A Digital Twin predictive maintenance framework of air handling units based on automatic fault detection and diagnostics. *Energy and Buildings*, 261, 111988.

[17] Kane, Archit P., et al. (2022). Predictive maintenance using machine learning. *arXiv preprint arXiv:2205.09402*.

[18] Hashemian, Hashem M. (2010). State-of-the-art predictive maintenance techniques. *IEEE Transactions on Instrumentation and Measurement*, 60(1), 226–236.

[19] De Luca, Roberto, et al. (2023). A deep attention based approach for predictive maintenance applications in IoT scenarios. *Journal of Manufacturing Technology Management*, 34(4), 535–556.

[20] Wang, Xiaoqiao, et al. (2023). Data-driven and Knowledge-based predictive maintenance method for industrial robots for the production stability of intelligent manufacturing. *Expert Systems with Applications*, 234, 121136.

[21] Qureshi, Muhammad Salik, Shayan Umar, and Muhammad Usman Nawaz. (2024). Machine Learning for Predictive Maintenance in Solar Farms. *International Journal of Advanced Engineering Technologies and Innovations*, 1(3), 27–49.

[22] Zonta, Tiago, et al. (2022). A predictive maintenance model for optimizing production schedule using deep neural networks. *Journal of Manufacturing Systems*, 62, 450–462.

[23] Lee, Wo Jae, et al. (2019). Predictive maintenance of machine tool systems using artificial intelligence techniques applied to machine condition data. *Procedia CIRP*, 80, 506–511.

[24] Shiva, Krishnateja, et al. (2024). Anomaly detection in sensor data with machine learning: Predictive maintenance for industrial systems. *Journal of Electrical Systems*, 20(10s), 454–462.

[25] Bodyanskiy, Y., Dolotov, A., Peleshko, D., Rashkevych, Y., & Vynokurova, O. (2019). Online time series changes detection based on neuro-fuzzy approach. In *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications*, 131–166.

[26] Lekidis, A., Georgakis, A., Dalamagkas, C., & Papageorgiou, E. I. (2024). Predictive maintenance framework for fault detection in remote terminal units. *Forecasting*, 6(2), 239–265.

[27] Zhou, Zhiguo, et al. (2023). A federated learning framework for real-time predictive maintenance of industry equipment. *IEEE Internet of Things Journal*, 10(18), 16123–16136.

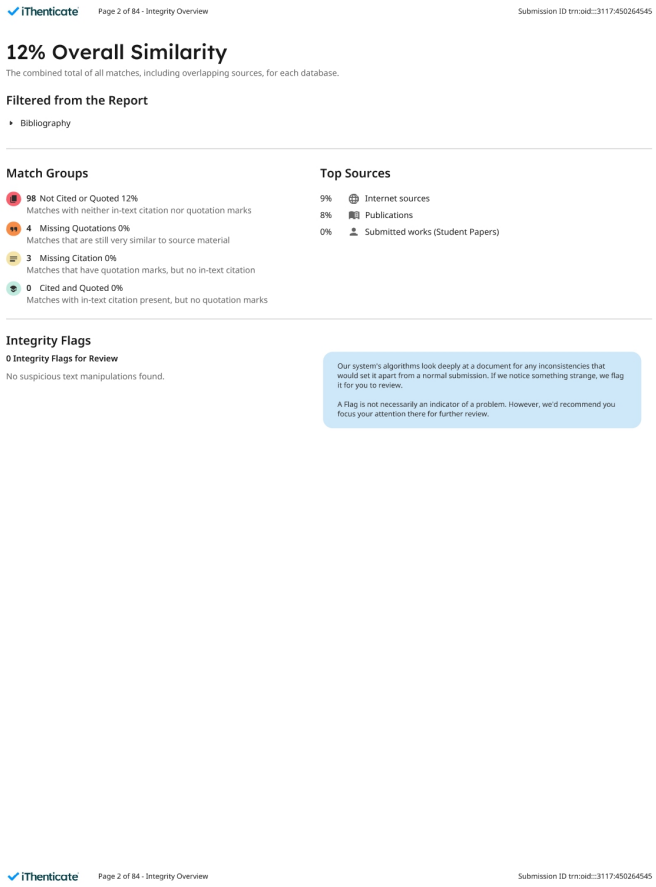
[28] Wang, Lin, et al. (2022). Hybrid model combining transformer and LSTM for predictive maintenance using time series sensor data. *Mechanical Systems and Signal Processing*, 172, 108962.

[29] Ali, Syed Muhammad, et al. (2021). Industrial AI: Real-time machine failure prediction in smart factories using big data analytics. *Journal of Industrial Information Integration*, 24, 100220.

[30] Fernández, Pablo, et al. (2020). Machine learning techniques for predictive maintenance in the railway industry. *Transportation Research Part C: Emerging Technologies*, 117, 102671.

Appendix

Plagiarism Report of the Project Report



iThenticate Page 2 of 84 - Integrity Overview Submission ID trn:oid:::3117:450264545

12% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

- 98 Not Cited or Quoted 12%
  Matches with neither in-text citation nor quotation marks
- 4 Missing Quotations 0%
  Matches that are still very similar to source material
- 3 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

9% Internet sources
8% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

iThenticate Page 2 of 84 - Integrity Overview Submission ID trn:oid:::3117:450264545

Review Paper

Predictive Maintenance and Diagnostic Report Generation

Chetan Mahale
Information Technology
Pimpri Chinchwad College of Engineering
Pune, India
chetan.mahale0220@gmail.com

Sanchalee Meshram
Information Technology
Pimpri Chinchwad College of Engineering
Pune, India
sanchaleemeshram@gmail.com

Pranav Narkhede
Information Technology
Pimpri Chinchwad College of Engineering
Pune, India
pranavnarkhede24@gmail.com

Mrs. Babita Sonare
Information Technology
Pimpri Chinchwad College of Engineering
Pune, India
sonare.babita@gmail.com

Dr. Jaya Dewan
Information Technology
Assistant Professor,
Pimpri Chinchwad College of Engineering
Pune, India
jaya.h.dewan@gmail.com

Mrs. Mukta Jamage
Information Technology
Assistant Professor,
Pimpri Chinchwad College of Engineering
Pune, India
muktaa.jamge@gmail.com

***Abstract*—Predictive Maintenance is one of the key strategies used in Industry 4.0. Paired with AI, this strategy can be further improved. The problem statement is predicting when the machine will fail, what caused it to fail and what will the machine operator do in case the machine fails. The major challenge of Predictive Maintenance is the natural imbalance of the dataset. To address the data imbalance problem, statistical and GenAI based methods are used. Further for better interpretation of the results, a report is generated using a Large Language Model. In this study, we explore various literature and their approaches to predicting failure in various machines. Although there are great improvements, some challenges still remain. Lack of generalized approach, operator guidance and root cause analysis are the challenges which we aim to address in our implementation. The proposed workflow for our implementation is also laid out in this paper.**

***Keywords*—Predictive Maintenance, Machine Learning, Artificial Intelligence, Data driven methods.**

I. Introduction

Equipment reliability is important in today's industrial setup as it ensures efficiency, reduced downtime, and low maintenance costs. Traditional approaches to maintenance as shown in Fig. 1. often fall short for systems that have increasingly become complex. Predictive maintenance has emerged as a solution, using machine learning, sensors, and real-time analytics to forecast failures before they occur. This data-driven approach meets Industry 4.0, where interconnected systems optimize operations through proactive interventions. In today's modern manufacturing where downtime can lead to significant losses, implementing predictive maintenance is not just a technical advantage but also gives the organization a business competence. The major challenge in implementing predictive maintenance is the scarcity of dataset. The available datasets are imbalanced which leads to poor performance of machine learning algorithms. This can be tackled by data augmentation techniques.

The evolution of maintenance strategies from the article [1] has followed somewhat of a chronological development. First, reactive maintenance or simply stated, run-to-failure waiting for breakdowns. In this case, very costly downtime must be dealt with and repairs carried out. Preventive maintenance could offer increased reliability by scheduling routine checks, but more often than not, waste resources on servicing that was not necessary. Predictive Maintenance extends this further by providing real-time monitoring of machine health and predicting failures only when necessary.

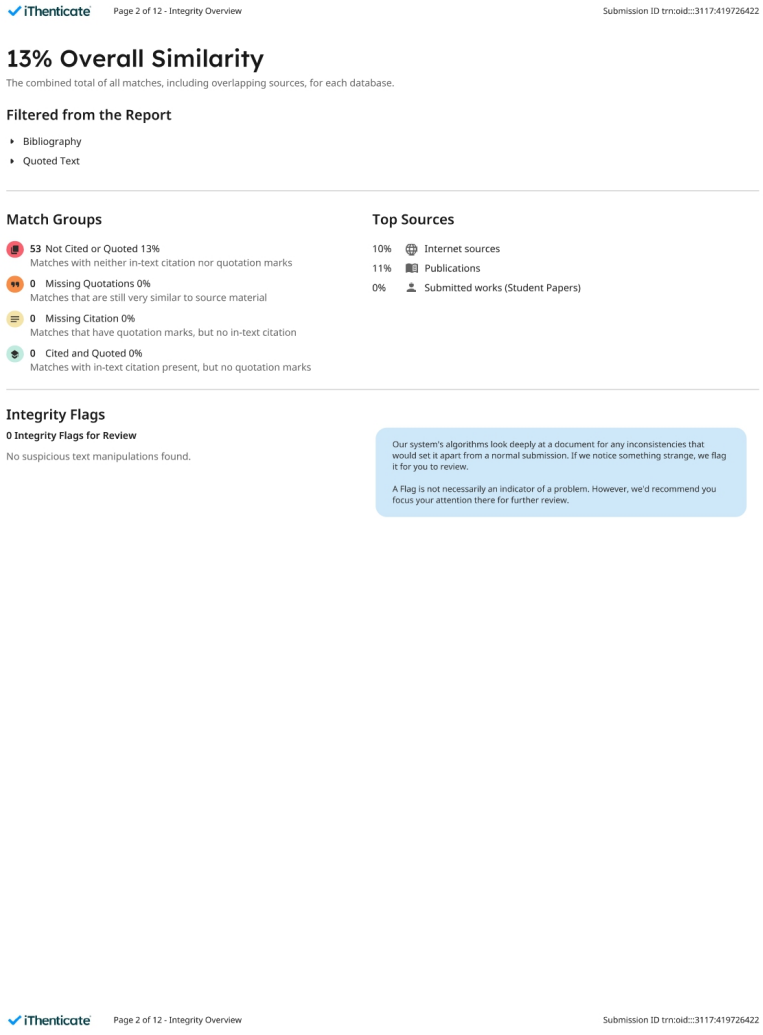
Fig. 1. Evolution of Maintenance Strategies

II. Literature Survey

The authors of [2] implemented predictive maintenance for cutting machines on the shop floor. The dataset was collected from the sensors attached to the machine. Random Forest for multi-label classification was used to predict the failure. They achieved an overall accuracy of 95%. The study in [3] proposed two multi-class predictive maintenance training algorithms that use Monte Carlo simulations and can be adapted to any classification algorithm. The highest accuracy of 98.15% was achieved on 38 simulations The authors used SVM and KNN in the article. The experiment carried out in [4] involves predictive maintenance for railway defect inspection using a multimodal model. The authors used LLM pipeline to analyze visual defects in railways. They achieved precision of 92% on image and 76% on video. The article [5] introduces a real time system for predictive maintenance based on motion current which can detect abnormal electric conditions. Precision of 97.59% was achieved on MLP with softmax activation function and scaled conjugate gradient optimization technique. Another approach [7] used logs for failure detection. The authors achieved a precision of 90% and recall of 86%. The authors of article [8] gave a hypothesis of using Generative AI in the IoT industry to detect real-time anomalies. Proposed metrics include precision, recall, and f1 score.

The study by authors in [9] used temporal association mining, predictive maintenance, temporal filter, machine learning, and semantic similarity on textual data for facility management requests. The English RoBERTa model has the highest acceptance rates for the minimum similarity thresholds of 0.65 and 0.7. In [10] predictive maintenance has been used in the manufacturing industry for process

Review Paper Plagiarism Report



iThenticate Page 2 of 12 - Integrity Overview Submission ID trn:oid:::3117:419726422

13% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

53 Not Cited or Quoted 13%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

10% Internet sources
11% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

iThenticate Page 2 of 12 - Integrity Overview Submission ID trn:oid:::3117:419726422

Review Paper AI Report

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

iThenticate Page 2 of 8 - AI Writing Overview Submission ID trn:oid:::3117:419726422

Research Paper

FailGuard AI: Framework for Predictive Maintenance using Large Language Model

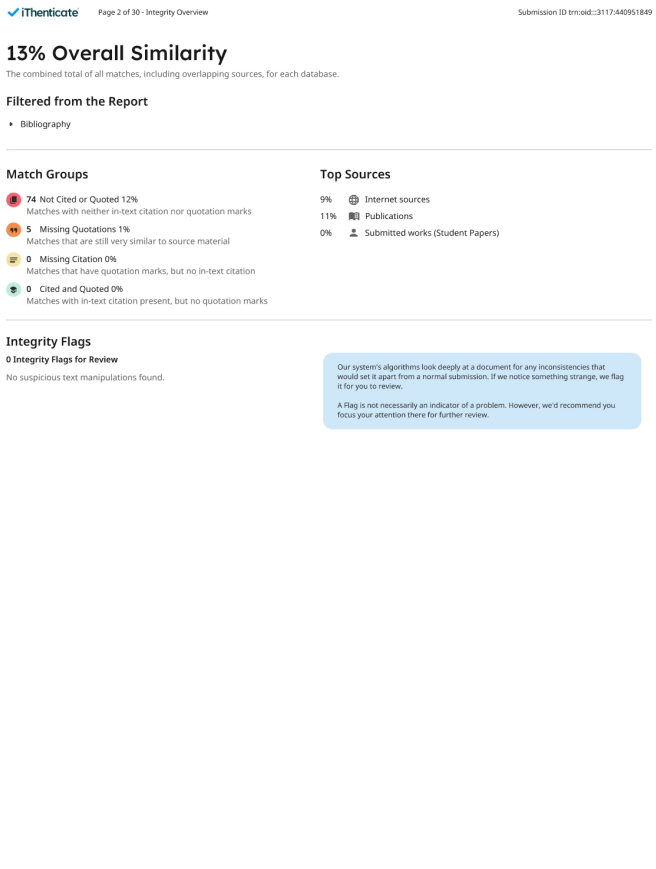
- Chetan Mahale
- Pranav Narkhede
- Sanchalee Meshram
- Mrs. Babita Sonare
- Dr. Jaya Dewan

Abstract

Predictive Maintenance is one of the recent strategies in Industry 4.0 which aims to reduce the losses and unnecessary expenses of the organization. The reviewed literature includes methods in which a state of machine is classified as failure or no failure. This classification can be further used for actionable steps with the help of a diagnostic report. The literature gaps include lack of a multipurpose system for predictive maintenance, lack of reason for failure and inability of the system to give the machine operator next steps. The proposed framework FailGuard AI aims to achieve these objectives. The system classifies the machine parameters and also provides a comprehensive diagnostic report. This report is generated with the help of the Llama 3.1 Nemotron 70B model by Nvidia. The challenge of data imbalance is addressed with statistical and GenAI-based techniques like SMOTE and CTGAN. Various data-driven approaches are used to classify the machine parameters. Experimentation on imbalanced data is also done. Considering the use case a new class called "About to Fail" is also added to the dataset. In retrospect, the article offers a comprehensive and modern solution to Predictive Maintenance which can be easily deployed in the industry.

Keywords: *Predictive Maintenance, Generative Artificial Intelligence, Data Augmentation, SMOTE, CTGAN, Supervised Machine Learning Algorithms, Ensemble Methods, Large Language Model, Llama 3.1 Nemotron 70B, Report Generation*

Research Paper Plagiarism Report



iThenticate Page 2 of 30 - Integrity Overview Submission ID trn:oid:::3117:440951849

13% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

74 Not Cited or Quoted 12%
Matches with neither in-text citation nor quotation marks

5 Missing Quotations 1%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

9% Internet sources
11% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

iThenticate Page 2 of 30 - Integrity Overview Submission ID trn:oid:::3117:440951849

Research Paper AI Report

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

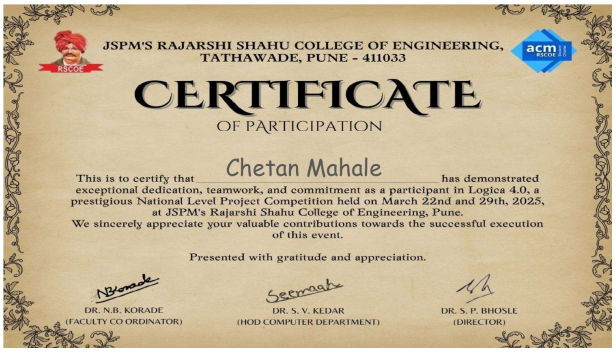
The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

iThenticate Page 2 of 26 - AI Writing Overview Submission ID trn:oid:::3117:440951849

Project Participation Certificates


JSPM'S RAJARSHI SHAHU COLLEGE OF ENGINEERING,
TATHAWADE, PUNE - 411033

CERTIFICATE

OF PARTICIPATION

This is to certify that Chetan Mahale has demonstrated exceptional dedication, teamwork, and commitment as a participant in Logica 4.0, a prestigious National Level Project Competition held on March 22nd and 29th, 2025, at JSPM's Rajarshi Shahu College of Engineering, Pune.
We sincerely appreciate your valuable contributions towards the successful execution of this event.

Presented with gratitude and appreciation.

DR. N.B. KORADE (FACULTY CO ORDINATOR) | DR. S. V. KEDAR (HOD COMPUTER DEPARTMENT) | DR. S. P. BHOSLE (DIRECTOR)


JSPM'S RAJARSHI SHAHU COLLEGE OF ENGINEERING,
TATHAWADE, PUNE - 411033

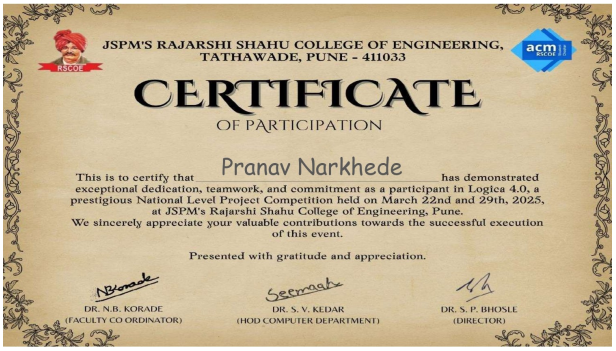
CERTIFICATE

OF PARTICIPATION

This is to certify that Sanchalee Meshram has demonstrated exceptional dedication, teamwork, and commitment as a participant in Logica 4.0, a prestigious National Level Project Competition held on March 22nd and 29th, 2025, at JSPM's Rajarshi Shahu College of Engineering, Pune.
We sincerely appreciate your valuable contributions towards the successful execution of this event.

Presented with gratitude and appreciation.

DR. N.B. KORADE (FACULTY CO ORDINATOR) | DR. S. V. KEDAR (HOD COMPUTER DEPARTMENT) | DR. S. P. BHOSLE (DIRECTOR)


JSPM'S RAJARSHI SHAHU COLLEGE OF ENGINEERING,
TATHAWADE, PUNE - 411033

CERTIFICATE

OF PARTICIPATION

This is to certify that Pranav Narkhede has demonstrated exceptional dedication, teamwork, and commitment as a participant in Logica 4.0, a prestigious National Level Project Competition held on March 22nd and 29th, 2025, at JSPM's Rajarshi Shahu College of Engineering, Pune.
We sincerely appreciate your valuable contributions towards the successful execution of this event.

Presented with gratitude and appreciation.

DR. N.B. KORADE (FACULTY CO ORDINATOR) | DR. S. V. KEDAR (HOD COMPUTER DEPARTMENT) | DR. S. P. BHOSLE (DIRECTOR)