

BINDURA UNIVERSITY OF SCIENCE EDUCATION
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRONIC ENGINEERING



**DESIGN AND IMPLEMENTATION OF A TINYML-BASED
PREDICTIVE MAINTENANCE SYSTEM FOR BALL MILL
CONDITION MONITORING**

BY
TAKUDZWA HAPARIMWE
B222151B

A final year project report submitted in partial fulfilment of the requirements for the award of
the Honours Bachelor of Science Degree in Electronic Engineering

SUPERVISOR: S. MALINDI

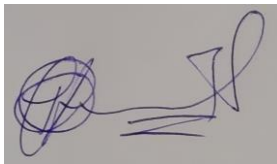
JUNE 2026

DECLARATION

I hereby certify that the work which is being presented in the project entitled “**DESIGN AND IMPLEMENTATION OF A TINYML-BASED PREDICTIVE MAINTENANCE SYSTEM FOR BALL MILL CONDITION MONITORING**” in partial fulfilment of the requirements for the award of the Honours Bachelor of Science Degree in Electronic Engineering and submitted in the Department of Electronic Engineering, Bindura University of Science Education, is an authentic record of my own work carried out during the period from August 2022 to June 2026 under the supervision of S. Malindi, Lecturer, Department of Electronic Engineering.

The matter presented in this project has not been submitted by me for the award of any other degree of this or any other institute.

Student: TAKUDZWA HAPARIMWE (B222151B)



Signature:

Date: 12 August 2026

Supervisor: S. MALINDI



Signature:

Date: 12 August 2026

COPYRIGHT

© 2026 Takudzwa Haparimwe. All rights reserved.

This dissertation is submitted in partial fulfilment of the requirements for the award of the Honours Bachelor of Science Degree in Electronic Engineering at Bindura University of Science Education. The author hereby grants Bindura University of Science Education permission to reproduce and distribute copies of this research project, in whole or in part, for academic and non-commercial purposes only. The author reserves all other publication rights, and neither the dissertation nor extensive extracts from it may be printed or otherwise reproduced for commercial purposes without the author's written permission.

DEDICATION

This work is dedicated to my Lord and Saviour Jesus Christ and everyone who he has placed in my life to help me to reach this level, including my family and the Enactus family.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, S. Malindi, for the invaluable guidance, encouragement, and technical insight provided throughout this project. I am also thankful to the staff of the Department of Electronic Engineering at Bindura University of Science Education for their support during my studies.

I extend my appreciation to Freda Rebecca Gold Mine for the industrial attachment experience that motivated this work, and to my family and colleagues for their continued support.

ABSTRACT

Ball mills are critical, high-value assets in mineral processing plants, and their unplanned failure causes costly downtime and safety hazards, particularly for the small-scale mining operations common in Zimbabwe that lack access to cloud-based monitoring infrastructure. This project presents the design and implementation of a low-cost, TinyML-based predictive maintenance system for ball mill condition monitoring that performs fault classification and severity assessment directly on an embedded microcontroller. A Raspberry Pi Pico W acquires vibration, temperature, current, and voltage data, from which thirteen time- and frequency-domain features are extracted. A Random Forest classifier, trained offline and exported to run on the device, distinguishes normal operation from imbalance, looseness, and overheating faults, while rule-based thresholds detect electrical and thermal anomalies. The model achieved 91.98 percent cross-validated accuracy and a 91.65 percent macro F1-score while using only 16.7 percent of the available flash memory and inferring in real time. Telemetry is published to the Adafruit IO cloud platform for remote monitoring, and local light-emitting diodes and a liquid crystal display provide on-site alerts. The results confirm that accurate, edge-based predictive maintenance is feasible on resource-constrained, low-cost hardware.

LIST OF ABBREVIATIONS

ADC	Analogue-to-Digital Converter
ASM	Artisanal and Small-Scale Mining
API	Application Programming Interface
BUSE	Bindura University of Science Education
CSV	Comma-Separated Values
DS18B20	Dallas Digital Temperature Sensor
FFT	Fast Fourier Transform
FRGM	Freda Rebecca Gold Mine
GPIO	General-Purpose Input/Output
I2C	Inter-Integrated Circuit
IoT	Internet of Things
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
ML	Machine Learning
MPU6050	Six-Axis Accelerometer and Gyroscope Module
MQTT	Message Queuing Telemetry Transport
PdM	Predictive Maintenance
POTRAZ	Postal and Telecommunications Regulatory Authority of Zimbabwe
RF	Random Forest
RMS	Root Mean Square
SCL	Serial Clock Line
SDA	Serial Data Line
SI	Severity Index
TinyML	Tiny Machine Learning
WiFi	Wireless Fidelity

TABLE OF CONTENTS

Contents

CHAPTER 1: INTRODUCTION	13
1.1 Background of the Study	13
1.2 Problem Statement	14
1.3 Aim of the Project	14
1.4 Specific Objectives	15
1.5 Scope of the Study	15
1.6 Significance of the Study	16
1.7 Chapter Summary	17
CHAPTER 2: LITERATURE REVIEW	18
2.1 Introduction.....	18
2.2 Theoretical Background.....	18
2.2.1 Evolution of Maintenance Strategies	18
2.2.2 Condition-Based Monitoring Theory.....	19
2.2.3 Vibration Analysis Theory.....	19
2.2.4 Feature Extraction in Rotating Machinery Diagnostics.....	20
2.2.5 Anomaly Detection versus Classification Approaches.....	20
2.2.6 Severity Assessment Modelling.....	21
2.3 Review of Related Systems and Technologies	21
2.3.1 Ball Mill Monitoring and Fault Diagnosis.....	22
2.3.2 Internet of Things in Industrial Condition Monitoring.....	22
2.3.3 Edge Computing and TinyML for Predictive Maintenance	23
2.4 Identified Research Gaps	23
2.5 Conceptual Framework and Proposed System Basis.....	24
2.6 Chapter Summary	25
CHAPTER 3: RESEARCH METHODOLOGY	26
3.1 Introduction.....	26
3.2 Research Design and Project Approach.....	26
3.3 System Requirements Specification	27
3.3.1 Hardware Requirements.....	27
3.3.2 Software Requirements.....	27
3.4 System Architecture.....	28
3.4.1 Sensor Layer	29
3.4.2 Data Acquisition and Processing Layer.....	29
3.4.3 Machine Learning and Decision Layer.....	29

3.4.4 IoT Communication Layer.....	30
3.4.5 Local Output Layer	31
3.5 System Flowchart.....	31
3.6 System Design and Implementation	32
3.6.1 Hardware Design and Assembly	32
3.6.2 Data Collection	33
3.6.3 Feature Extraction and Model Training.....	34
3.6.4 TinyML Deployment	34
3.6.5 System Algorithm and Pseudocode	35
3.7 Ethical Considerations	37
3.8 Chapter Summary	37
CHAPTER 4: RESULTS AND ANALYSIS	39
4.1 Introduction.....	39
4.2 System Implementation Results.....	39
4.2.1 Hardware Prototype Assembly	39
4.2.2 Serial Monitor Output	41
4.2.3 Adafruit IO Cloud Dashboard.....	41
4.2.4 TinyML Model Deployment.....	42
4.3 Data Presentation	44
4.3.1 Raw Vibration Signal Characteristics	44
4.3.2 Feature Distribution Analysis	45
4.3.3 Cross-Validation Performance	45
4.3.4 Confusion Matrix	46
4.3.5 Per-Class F1-Scores	48
4.3.6 Feature Importance Ranking.....	49
4.3.7 System Resource Utilisation	50
4.3.8 Non-Blocking Timing Architecture.....	51
4.4 Analysis and Interpretation	51
4.4.1 Dataset and Feature Analysis.....	51
4.4.2 Classification Performance Analysis	52
4.4.3 Confusion Matrix Interpretation	53
4.4.4 Rule-Based Fault Detection	53
4.4.5 TinyML Deployment Performance.....	54
4.5 IoT Communication and Local Display Results.....	54
4.5.1 Adafruit IO Telemetry	54
4.5.2 Local LCD Display and LED Indicators	55
4.6 Discussion of Findings.....	56

4.7 Chapter Summary	57
CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS	58
5.1 Introduction.....	58
5.2 Achievement of Project Objectives	58
5.3 Overall Conclusions.....	59
5.4 Limitations	60
5.5 Recommendations for Future Work.....	61
5.5.1 Increased Vibration Sampling Rate	61
5.5.2 Field Validation on Industrial Equipment.....	61
5.5.3 PCB Design and Enclosure	61
5.5.4 Continuous Learning and Model Updating.....	61
5.5.5 Expanded Fault Classes	62
5.5.6 SMS or Push Notification Alerting.....	62
5.6 Chapter Summary	62
REFERENCES	63

LIST OF TABLES

Table 3.1: Hardware Components	17
Table 3.2: Software Tools.....	18
Table 3.3: Rule-Based Fault Detection Thresholds	20
Table 4.1: Five-Fold Cross-Validation Results	37
Table 4.2: Feature Importances Ranked	40
Table 4.3: Dataset Composition and Vibration RMS Statistics.....	44
Table 4.4: Per-Class Classification Performance (Cross-Validated).....	44
Table 4.5: Cross-Validated Confusion Matrix.....	45
Table 4.6: Rule-Based Fault Detection Test Results	46
Table 4.7: TinyML Deployment Metrics.....	46
Table 4.8: LCD Screen Content During Normal Operation	48

LIST OF FIGURES

Figure 3.1: System Block Diagram.....	19
Figure 3.2: System Operational Flowchart	22
Figure 3.3 and Figure 3.4: Labelling Data Sets	24
Figure 4.1: Complete Assembled Hardware Prototype	30
Figure 4.2: Key Hardware Components	32
Figure 4.3: LCD Display During Normal Operation	32
Figure 4.4: Serial Monitor Output Showing Live Sensor Data Stream	32
Figure 4.5: Adafruit IO Remote Monitoring Dashboard	33
Figure 4.6: Raw Vibration Signals by Fault Class.....	35
Figure 4.7: Feature Distribution Box Plots by Fault Class	36
Figure 4.8: Five-Fold Cross-Validation Accuracy per Fold	37
Figure 4.9: Cross-Validated Confusion Matrix Heatmap	39
Figure 4.10: Per-Class F1-Scores (Cross-Validated).....	40
Figure 4.11: Feature Importance Ranking (Random Forest)	42
Figure 4.12: Raspberry Pi Pico W Resource Utilisation.....	43
Figure 4.13: Non-Blocking Sensor Sampling Timing Diagram	43

CHAPTER 1: INTRODUCTION

1.1 Background of the Study

Mineral mining remains one of the largest industries in Zimbabwe, earning the country vast amounts of foreign currency and providing raw materials used in industries. Nevertheless, the industry is also known for having high risk and equipment-based activities that require good maintenance systems (Singo et al., 2022). Ball mill is one of the machines that play an important role in mineral processing factories. The machine grinds the ores into fine pieces necessary for the extraction of valuable minerals including gold and copper (Matsanga, Nheta and Chimwani, 2023). The failure of ball mill operation for just one hour could cost the company more than hundreds of thousands of dollars in production losses (Atlas Copco, 2022).

The maintenance strategy commonly employed in Zimbabwean mining industries is reactive and time-based. The equipment is either regularly maintained on schedule or repaired in case of breakdown. Industry reports show that the amount of money that a company spends on maintaining equipment is 30 to 50 percent of the total budget while around 82 percent of machine breakdowns take place unexpectedly (Beam Data, 2025). This shows that almost all equipment breakdowns remain unexpected despite the scheduled maintenance of the machine.

The developments in the field of IoT technology and edge machine learning, known as TinyML, provide an interesting answer to this issue. In the case of predictive maintenance, sensors collect information about machine vibrations, temperature, and electrical currents, whereas the machine learning algorithm analyses the data and identifies potential faults before they can cause breakdowns (Reliable Plant, 2024). TinyML, in particular, makes it possible to perform machine learning analysis right on embedded microcontrollers, eliminating the necessity of constant cloud connection and decreasing the processing time down to milliseconds (de la Fuente, Radrigan and Morales, 2024). A recent example of a TinyML application achieved a fault classification accuracy of over 90 percent and decreased power usage by 44 percent, thus making it possible to work for more than 100 hours on battery power (de la Fuente, Radrigan and Morales, 2024).

Their relevance in relation to the mining industry in Zimbabwe is highly emphasized. The mining sector in Zimbabwe includes both large commercial mining enterprises and large informal ASM sector involving from 500,000 to 1,500,000 people in rural areas (Artisanal Mining, 2022). Most of such mines do not have a well-developed SCADA system and a proper

infrastructure of cloud computing. Internet connection is rare in rural Zimbabwe with broadband internet connectivity being reported by POTRAZ as problematic in most rural locations (POTRAZ, 2024). A cheap solution to implement a predictive maintenance system that does not depend on cloud computing would help overcome these issues.

The author's work experience at the Freda Rebecca Gold Mine (FRGM) in Zimbabwe is directly related to the necessity of implementing such a predictive maintenance system. During the industrial attachment period, one of the ball mills in the grinding plant failed requiring its two-week shutdown resulting in considerable production loss. This situation also caused additional risk to the safety of the employees involved in repairing procedures as these works were emergency in nature. The maintenance was scheduled by using strict time schedules and experience of the workers only.

1.2 Problem Statement

The current maintenance process in many mining operations in Zimbabwe, both small scale and artisanal operations, is predominantly preventive or purely time-based. This implies that the equipment is maintained using a predefined schedule or repaired after a breakdown due to the fact that no attention is given to the condition of the machinery being used. Some of the drawbacks that result from this type of maintenance approach include high incidences of breakdowns, reduced productivity, and increased safety risks to people operating near the malfunctioning equipment.

There are existing maintenance systems in the industry; however, they are meant for large-scale industries which have an elaborate network of sensors and cloud computing as well as data engineers who are experts in managing data and analyzing the results. High cost of implementing this technology makes it impossible to be adopted by small scale mining operations. This is more so when it comes to the maintenance of ball mills, considering that they are expensive and highly used machinery.

Thus, there is a definite need for a low-cost and compact predictive maintenance solution specifically made for the ball mill monitoring requirements in small scale and resource constrained mining operations.

1.3 Aim of the Project

The main objective of this project is to develop an implementation of a predictive maintenance system using TinyML to perform real-time fault classification and severity analysis using edge

computing, combined with IoT connectivity to monitor remotely through the Adafruit IO cloud platform.

1.4 Specific Objectives

The following specific objectives guide the development of the proposed system:

- Designing and assembling a multi-sensor hardware setup that integrates vibration, temperature, current, and voltage sensors with Raspberry Pi Pico W microcontroller for real-time sensor data acquisition from a ball mill.
- Data collection and labeling of a dataset of sensor recordings under normal condition and various fault conditions including the fault classes such as normal operation, mechanical imbalance, and mechanical looseness for training a supervised machine learning fault detection model.
- Implementation of the training of a Random Forest based fault classification model using time domain and frequency domain features of the vibration signal and exporting the trained model as a C header file for deploying the model in the embedded microcontroller.
- Development of rule-based severity checking of current, temperature, and voltage parameters in conjunction with the machine learning model for complete fault classification of overload, overheat, overvoltage, and undervoltage faults.
- Inclusion of IoT communication through MQTT and Adafruit IO cloud service and design of a local 16x2 LCD display module with relay output and status LED indicators.

1.5 Scope of the Study

This study focuses specifically on the predictive maintenance of a single ball mill unit through the integration of IoT sensing and embedded edge machine learning. The following are included within the scope of the project:

- Real-time monitoring of ball mill vibration, motor, and bearing temperature, power voltage, and electrical current through the use of embedded sensors.
- Utilization of TinyML Random Forest fault classification algorithm running on the Raspberry Pi Pico W microcontroller to classify faults locally, independent of the cloud.

- Detection of electrical and thermal faults such as current overload, temperature overheat, overvoltage, and undervoltage through rules, which are independent of the machine learning algorithm.
- A locally hosted 16x2 LCD user interface with navigational buttons, status LED lights to show fault occurrence, and battery level indicators.
- IoT integration by means of WiFi and MQTT to send sensor data and fault classification to the Adafruit IO cloud server with relay capability to shut down the machine remotely.

1.6 Significance of the Study

This project will yield a number of significant advantages for Zimbabwe's mining industry and the wider sphere of industrial predictive maintenance.

As far as safety is concerned, the system allows for early detection of emerging mechanical and electrical defects before these become the reason for the breakdowns. Unnoticed failures of bearings, mechanical imbalances and overload in rotating equipment can lead to unexpected failures and the release of debris, which will endanger nearby workers. IoT condition monitoring gives continuous early warning system as opposed to reactive approach of incident response, which means a shift from responding to incidents to managing risks (Reliable Plant, 2024). The mentioned advantage is especially significant for ASM workers who work without any established safety infrastructure (Singo et al., 2022).

Economically, the project will increase the efficiency of processes by using predictive maintenance instead of preventive one. Studies have demonstrated that predictive maintenance allows reducing maintenance costs up to double-digit percentages and unplanned downtimes up to over 50% compared to reactive approach (Florian, Sgarbossa and Zennaro, 2021). For a small-scale processing facility, avoiding even one week of unplanned ball mill downtime in a year can be considered an achievement.

The project is intentionally designed to be usable by resource-constrained mining enterprises. The hardware platform is constructed using off-the-shelf and inexpensive components. Inferences are done locally through the microcontroller without any recurring expense of cloud subscriptions. The network is implemented via standard WiFi connectivity without any need for special networking equipment other than a WiFi-enabled router connected to the internet.

Another contribution to gender inclusion in mining arises from the current project. There is an appreciable amount of female labor force in ASM in the sub-Saharan African context, which

may account for up to 50 percent of the labor force (Atlantic Council, 2025; Mukenge and Moyce, 2025). The presence of easy-to-understand local interface lowers the level of technical knowledge needed to make sense of machine condition data.

In terms of research contributions, there is an integration of multi-sensor edge TinyML solution which was explicitly designed and tested on ball mill equipment in a Zimbabwean mining context. This addresses one of the gaps found in the review of literature, namely lack of studies which implement vibration, temperature and electrical sensors in an embedded edge context.

1.7 Chapter Summary

Introduction and motivation have been defined in the chapter. There is no denying the fact that mining equipment, such as ball mills, plays an important role in mineral processing activities; however, the problem with these machines is that there is always a risk of mechanical and electrical problems leading to expensive unplanned outages and dangers to people. The conventional preventive and time-based maintenance approach cannot stop these faults as most of them happen at random times. The latest technologies related to Internet of Things (IoT) and Tiny Machine Learning can provide a promising solution to this problem.

The objectives of the project as set out in this chapter include designing and implementing a TinyML based system for predictive maintenance of a ball mill. Five specific objectives have been listed in this regard including those related to hardware design, data acquisition, model creation, rule-based fault detection, and integration with IoT. The scope defines the limitations of the project that it will deal with only one ball mill unit using edge computing and not with cloud based model updating or systems based on multiple machines. Importance has been given in this section to the advantages of this project for worker safety, lowering of operational costs, availability to small scale users, gender equality, and research contribution. Chapter 2 covers the literature on the subject.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter presents an overview of theoretical background and the current state of the art related to the design of a TinyML based predictive maintenance system for ball mill monitoring. The structure of this chapter is presented as follows: Section 2.2 presents an overview of the theoretical background that discusses the history of maintenance strategies, concept of condition-based monitoring, vibration analysis theories, feature extraction techniques and machine learning techniques for faults detection. Section 2.3 presents an overview of the existing systems and technologies that discusses existing ball mill monitoring systems, Internet of Things-based frameworks for industrial monitoring and edge computing applications. Section 2.4 discusses gaps within the existing literature that will be addressed by the proposed system.

2.2 Theoretical Background

2.2.1 Evolution of Maintenance Strategies

There have been three general paradigms of industrial maintenance: reactive maintenance, preventive maintenance, and predictive maintenance. The first and the most simple form of industrial maintenance, reactive maintenance, refers to repair works carried out after equipment malfunctions have happened. Though no preliminary planning is required, this method causes unexpected halts in the production process, expensive repairs, and poses safety threats due to unexpected equipment failure, especially in the case of high loads, for instance, in mining (Rojas, Peña and Garcia, 2025).

Preventive maintenance represents an advance on reactive maintenance, where servicing is planned at set intervals of time or usage. This maintenance paradigm is inefficient in its nature because it implies servicing without considering the real state of the equipment, thus causing extra expenses (Hamasha et al., 2025). Real practice shows that the greatest proportion of equipment failures in industry happens during the periods between planned maintenance (Beam Data, 2025).

Predictive maintenance is the most sophisticated approach, which involves continuous monitoring of machines' condition in real time through sensing and machine learning techniques to predict upcoming failures proactively. Condition-based maintenance activities initiated due to degradation signs, not scheduled actions, decrease unnecessary maintenance efforts and increase fault detection efficiency (Hamasha et al., 2025). According to the study

conducted by Chambi et al. (2025), the use of AI-based systems for predicting faults helps increase fault detection accuracy in underground mining machinery compared to traditional approaches to monitoring. The study conducted by Florian, Sgarbossa and Zennaro (2021) proved the cost benefits of the predictive maintenance strategy based on machine learning implementation. A systematic review by Carvalho et al. (2019) described all machine learning approaches used in predictive maintenance.

2.2.2 Condition-Based Monitoring Theory

Condition-based monitoring is based on the concept that maintenance actions are determined based on the condition of the equipment and not the time schedule. Condition-based monitoring involves assessing the health of equipment based on measurable physical characteristics like vibration amplitude, temperature of bearings, and electrical current of the machine (Rojas, Peña and Garcia, 2025).

The theoretical basis of condition-based monitoring is degradation modeling. Under degradation modeling, the development of faults in equipment takes place in a progressive manner, starting from initial defects all the way to the point of total failure. The condition-based monitoring system makes it possible for the faults to be identified and rectified early enough before total failure happens (Ūselis, Serackis and Pomarnacki, 2025). For a rotating machinery system, there are certain measurable characteristics that tend to change before mechanical failures occur. These include the vibration amplitude, temperature of bearings, and distortion of the electrical currents.

2.2.3 Vibration Analysis Theory

Vibration analysis is one of the best diagnostic methods used to detect faults in rotating machines. Defects in bearings, shaft imbalances, mechanical looseness, and gear faults have distinct vibration signals in both time and frequency domains (Ou et al., 2024).

Time-domain signals characterized by either elevated amplitude or irregular vibrations imply instability or shocks in the structure. RMS amplitude, peak value, kurtosis, and crest factor of vibration signals are statistical time domain features that indicate the occurrence of different types of faults. Frequency-domain signals analyzed using FFT transform reveal distinct frequencies of each fault. For instance, high frequency components in the signal show the existence of bearing defects, predominant signal components at shaft rotation frequency point to imbalances while broad spectral signal energy across different harmonics are indicative of mechanical looseness (Ūselis, Serackis and Pomarnacki, 2025).

Vibration analysis is particularly important in the case of heavy industrial machinery like ball mill which run on a constant rotational speed and high mechanical loads. The vibration magnitude in the current work was calculated using the MPU6050 inertial measurement unit from the tri-axial acceleration components as the Euclidean norm of the three acceleration components.

2.2.4 Feature Extraction in Rotating Machinery Diagnostics

The outputs of vibration sensors are normally high-dimensional and noisy. Feature extraction converts the time-series data of the vibration sensors into reduced form that can be used by machine learning algorithms (Üselis, Serackis and Pomarnacki, 2025).

Features in the time domain that are extracted from vibration signals are: RMS which denotes the energy contained in the signal, peak amplitude that is the peak value of the signal, standard deviation denoting the variation in the signal, kurtosis which is the degree of impulsiveness of the signal and is very sensitive to bearing problems, skewness of the signal distribution, crest factor which is the peak to RMS ratio, shape factor which is the ratio of RMS to the average value and impulse factor which is the peak to average ratio.

Some frequency domain based features generated by FFT include dominant frequency, dominant spectral power, total spectral energy, and spectral entropy that indicates the distribution of energy among the frequency bins and helps differentiate between tonal defects such as imbalance and broadband defects like looseness (Sun et al., 2024). This helps reduce computational overheads and facilitates implementation on constrained embedded microcontrollers which is very important for TinyML-based predictive maintenance systems (de la Fuente, Radrigan and Morales, 2024).

2.2.5 Anomaly Detection versus Classification Approaches

Predictive maintenance applications that use machine learning technologies use anomaly detection or supervised classification approaches depending upon the availability of data and application requirements.

The idea behind using an anomaly detection model is to capture the statistics of a working machine's behavior and detect any deviations from this norm. The benefit of this approach lies in its ability to work well with limited labeled fault data, which is typically the case in industry, where faults occur rarely (Hamasha et al., 2025).

Supervised classification approaches need labeled training data where sensor readings are mapped to fault classes. These approaches learn decision boundaries between health status and classify new feature vectors into fault classes, providing better accuracy compared to anomaly detection in cases where adequate labeled data is provided (Üselis, Serackis and Pomarnacki, 2025). In this work, a supervised random forest classifier is used to classify normal, imbalance, and looseness based on labeled data of these statuses, as the use of the controlled laboratory simulation environment allowed creating an adequate number of labeled training data for our model. Thermal overheating faults are processed using specific temperature-based logic separately from classification. The choice of tiny machine learning approaches implies that tree-based classifiers provide more suitable choices for TinyML implementation due to their deterministic inference, minimal memory requirements, and non-necessity of floating-point matrix multiplications (de la Fuente, Radrigan and Morales, 2024).

2.2.6 Severity Assessment Modelling

However, for an effective predictive maintenance system, it is necessary to prioritize the importance of the fault in addition to its classification. The severity index is a quantitative measure of urgency of a fault according to the degree of parameter value exceeding operational threshold limits.

The general form of a weighted severity index is expressed in Equation (2.1):

$$SI = w1 * V_norm + w2 * T_norm + w3 * C_norm \quad (2.1)$$

Where V_norm , T_norm , and C_norm are normalized vibrations, temperature, and current parameters, and $w1$, $w2$, $w3$ are weight parameters for each respective parameter that indicate the significance of each in terms of diagnosis. Each parameter is normalized between zero and one according to Equation (2.2):

$$x_norm = (x - x_min) / (x_max - x_min) \quad (2.2)$$

After computing the value of the severity index, it is checked against certain pre-defined thresholds to determine the severity level. In the system discussed here, the following severity threshold values have been considered: $SI < 0.3$ (Normal), $0.3 \leq SI < 0.8$ (Warning), and $SI \geq 0.8$ (Critical). However, the novel approach used in the proposed solution further extends this scheme by using Random Forest feature importances as the weighting factors for the severity score calculation.

2.3 Review of Related Systems and Technologies

2.3.1 Ball Mill Monitoring and Fault Diagnosis

In mineral processing plants, ball mills function in continuous operation conditions with huge mechanical loading, hence susceptible to bearing failures, liner wear, mechanical imbalance, loose conditions, and motor overload. Ou et al. (2024) have used an Internet of Things (IoT) sensor network for liner wear condition monitoring in grinding mills, proving that vibration and acoustic signals are excellent indicators of mechanical degradation. The study by Ou et al., however, did not use machine learning techniques to classify faults.

Sun et al. (2024) demonstrated the development of an online updating stochastic configuration network for semi-autogenous ball mill crusher process monitoring, showing that adaptive machine learning models can dynamically update their parameters in response to changing operational conditions. While this approach is primarily focused on process variable estimation rather than mechanical fault classification, the demonstrated adaptability of embedded learning models in ball mill applications is relevant to the proposed system design. Collao et al. (2025) further proposed an agentic AI digital-twin architecture specialised for predictive maintenance of ball mills, reflecting the growing interest in intelligent asset management for this class of equipment.

Ūselis, Serackis and Pomarnacki (2025) found that vibration and motor current data can be successfully employed to perform fault classification in rotating machinery using lightweight machine learning models running on resource-restricted embedded systems, with fault classification accuracy being over 94 percent. These findings validate the technological possibility of employing embedded ML techniques for fault detection in rotating machinery, thereby providing direct evidence for the proposed system design.

2.3.2 Internet of Things in Industrial Condition Monitoring

Architecture designs of IoT technologies used in industrial monitoring comprise four layers: data acquisition layer, edge processing layer, communication layer, and cloud application layer (Chambi et al., 2025). Sensing devices constantly gather machine parameters, edge devices conduct local processing of the data collected and make inferences based on the collected features, while communication devices transmit the outcomes to cloud services.

The approach suggested by Hamasha et al. (2025) of an IoT-enabled predictive maintenance approach utilizing AI and edge computing can contribute towards increased reliability of equipment and decreased response time of the systems. The conclusions drawn by the authors about the positive effect of decentralization of edge processing can be applied in mining

operations in developing countries as well, where limited internet connection will negatively affect response time.

2.3.3 Edge Computing and TinyML for Predictive Maintenance

In the case of edge computing, raw data is processed at the point of measurement instead of sending them to cloud servers, which results in lower latency, bandwidth use, and reliance on internet connectivity (Zhukabayeva et al., 2025). The same approach can be applied to TinyML as it allows running models on microcontrollers even with limited memory and computing capabilities.

The relevance of TinyML to maintenance in the mining sector was shown by De la Fuente, Radrigan and Morales (2024), who created a hierarchical inference network using the TinyML method. Fault detection with their system had 90 percent accuracy and inference latencies of just a few milliseconds, which additionally resulted in 44 percent reduction of power usage compared to cloud offloading methods and allowed extending battery life of the device. This research is the most relevant to the proposed system.

Chambi et al. (2025) showed the effectiveness of AI fault detection systems in underground mining equipment, whereas Hamasha et al. (2025) emphasized the importance of using edge computing and AI together to make decisions in real time, which is especially effective in remote industrial environments. They all mention the limitation of cloud connectivity in the mining industry.

2.4 Identified Research Gaps

From the reviewed literature, some of the gaps include:

First, there are very few studies where an integrated IoT and Edge ML system has been developed for ball mill condition monitoring. Most of the literature related to ball mills is focused on process variables estimation and even single parameter monitoring, such as liner wear, but not in an integrated way.

Second, there is little literature about the integration of vibration with temperature and electrical parameters monitoring in one embedded Edge ML framework for fault classification in ball mills. The combined use of these three monitoring techniques for covering all the faults has never been done in an embedded way before.

Third, there is no literature on how TinyML-based predictive maintenance can be implemented in the Zimbabwean or sub-Saharan African mining context. The unique challenges faced in

this context, including limited connectivity, small budget size, and informal practice, need to be addressed in the design of such a system.

Fourthly, currently available commercial and research systems mainly depend on cloud-based approaches for processing data and making inferences, which is problematic from a latency point of view, as well as connectivity requirements that do not match the conditions under which small-scale and distant mining sites in Zimbabwe operate.

All of the above-mentioned problems form the rationale for the development of the presented system, which resolves all of the above-mentioned problems using a cheap multi-sensor edge-processing TinyML-based predictive maintenance system tested on ball mill with IoT connectivity.

2.5 Conceptual Framework and Proposed System Basis

The reviewed literature creates a solid theoretical and technological base for the implementation of the described solution. Principles of degradation modeling that form the theoretical background of condition-based monitoring (Rojas, Peña and Garcia, 2025) allow using vibration, temperature, and current data as markers of gradual development of mechanical and electrical faults. Techniques of vibration analysis and feature extraction presented in Sections 2.2.3 and 2.2.4 represent a signal processing chain starting from the raw data and finishing with the diagnostic features' vector. Feasibility of lightweight supervised classification models in embedded systems (Ūselis, Serackis and Pomarnacki, 2025; de la Fuente, Radrigan and Morales, 2024) confirms the correctness of the decision to use a Random Forest classifier implemented as a C header array on the Raspberry Pi Pico W.

In terms of architecture design, the majority of the literature under review makes use of centralised cloud architectures to perform the computation process (Chambi et al., 2025; Hamasha et al., 2025). Although such an approach proves to be effective when it comes to industry with sufficient connection, it faces significant latency and dependability issues in remote mining sites that have insufficient internet connectivity (POTRAZ, 2024). In contrast to prior research, the suggested system takes inference of all faults on the embedded microcontroller without depending on cloud communication.

Building on the reviewed literature, the proposed system integrates the following components into a unified predictive maintenance framework:

- Multi-sensor data acquisition capturing vibration, motor temperature, bearing temperature, electrical current, and supply voltage.
- On-device feature extraction computing 13 time-domain and frequency-domain features from the vibration signal within a 50-sample sliding window.
- A Random Forest TinyML fault classifier trained on labelled simulation data and deployed as a C array on the Raspberry Pi Pico W for edge inference.
- Rule-based severity assessment for electrical parameters, with fault outputs integrated into a unified severity classification of normal, warning, and critical.
- IoT communication via MQTT to the Adafruit IO cloud platform for remote monitoring and relay-based machine control.
- A local 16x2 LCD interface, status LEDs, and relay output providing immediate on-site operator feedback independent of network connectivity.

2.6 Chapter Summary

This chapter reviewed the existing literature related to the suggested Predictive Maintenance System. It has been discussed how the approach of moving from Reactive to Predictive maintenance could be done and how Condition-Based Monitoring, which involves collecting information through real-time sensors is the best way for minimizing any unforeseen machine downtime. Different methods of vibration analysis and feature extraction have been analyzed; it has been shown that time-domain and frequency-domain features are the most informative when it comes to finding the faults in a rotating machine. Advantages and disadvantages of both anomaly detection and supervised classification have been compared, and the Random Forest classifier has been chosen for the system because of the available fault simulation data.

Previous works on IoT and TinyML were reviewed, and the viability and benefits of performing edge-based fault detection in mines with restricted connectivity to the cloud were confirmed. Gaps in the literature review include lack of multi-sensors edge machine learning solution dedicated specifically to ball mills in Sub-Saharan African mines. The theoretical framework has outlined how the reviewed literature contributes to each of the design decisions of the proposed system. Chapter 3 includes the research methodology that details the hardware design, data collection, model training, software development, and integration of the system.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This chapter will explain the methodology that was used in designing and implementing the TinyML-based predictive maintenance system for ball mill condition monitoring. The structure of this chapter is as follows: Chapter 3.2 discusses the general research design and approach to the project. Chapter 3.3 outlines the necessary hardware and software. Chapter 3.4 provides information on system architecture and block diagram and their description. Chapter 3.5 includes the flow chart of the process of system implementation. Chapter 3.6 explains the system design and implementation in detail, including hardware construction, data acquisition, feature extraction, machine learning model building, TinyML deployment, and the presentation of algorithms in pseudocode form. Chapter 3.7 talks about ethical considerations and Chapter 3.8 offers a chapter summary.

3.2 Research Design and Project Approach

For this work, the engineering design process is used along with the development of prototypes in the lab. The engineering design process helps in developing a system in a structured manner by taking functional requirements into account. Using a prototype in the lab helps in testing each component separately to avoid any compounded errors once all components are integrated together.

The development process proceeds through the following sequential stages:

- System requirements analysis and component selection
- Hardware assembly and sensor integration
- Sensor data collection under controlled fault simulation conditions
- Data preprocessing and feature extraction
- Machine learning model development, training, and evaluation
- TinyML model export and deployment on the Raspberry Pi Pico W
- Rule-based fault detection logic implementation
- IoT communication and cloud integration
- Local display, LED, and relay output implementation
- System integration testing and performance evaluation

The sequential way of doing things makes it possible for individual elements to be tested independently before the system is assembled such that any fault identified can be assigned to a particular sub-system.

3.3 System Requirements Specification

3.3.1 Hardware Requirements

The hardware components chosen for this system are those that will be used to capture the physical data, process the data, give feedback to the user, and communicate. The choice of the components was informed by the need for affordability, compatibility with the Raspberry Pi Pico W IDE, and Arduino libraries.

Table 3.1: Hardware Components

Component	Model/Specification	Function
Raspberry Pi Pico W	RP2040, 264KB RAM, WiFi	Central processing, TinyML inference, IoT
Vibration sensor	MPU6050, I2C, GP4/GP5	Tri-axial acceleration for vibration monitoring
Temperature sensor	DS18B20 x2, OneWire, GP2	Motor and bearing temperature measurement
Current sensor	ACS712-5A, ADC GP26	Motor current measurement
Voltage sensor	ZMPT101B, ADC GP27	AC supply voltage measurement
LCD display	16x2 I2C, addr 0x27, GP4/GP5	Local operator interface
Status LEDs	Green GP10, Red GP11, Yellow GP12	Severity level indication
WiFi LED	GP14	WiFi connection status
Send LED	GP13	Cloud transmission indicator
Battery LED	GP9	Power status indicator
Relay module	5V, GP15	Motor shutdown control
Battery monitor	Voltage divider, ADC GP28	Battery charge level monitoring
Buttons	UP GP6, DOWN GP7	LCD navigation

3.3.2 Software Requirements

The software tools employed in this project include embedded programming of the Raspberry Pi Pico W as well as Python based data analysis and machine learning on the development PC.

Table 3.2: Software Tools

Tool	Purpose
Arduino IDE 1.8.19	Embedded C++ programming for Raspberry Pi Pico W

Python 3.x	Data processing, feature extraction, and model training
NumPy	Numerical array operations and signal processing computations
Pandas	Dataset manipulation, loading, and preprocessing
Scikit-learn	Random Forest model training, cross-validation, and evaluation
SciPy	Welch power spectral density estimation for FFT features
Matplotlib	Sensor data and model performance visualisation
Adafruit IO Dashboard	Cloud monitoring platform for remote telemetry and relay control

One of the major differences from the early project plan is that the last step implements the Random Forest classifier using scikit-learn instead of TensorFlow or TensorFlow Lite. The trained Random Forest is saved to a C header file where it is represented as a flat array of decision tree nodes that are directly incorporated into the Arduino sketch. This method results in the development of a more efficient inference code compared to TensorFlow Lite for the Raspberry Pi Pico W without any dependencies on the ML inference engine library.

3.4 System Architecture

The suggested predictive maintenance system consists of five layers of functionality which facilitate real-time fault detection and monitoring. The figure 3.1 depicts the entire system block diagram.

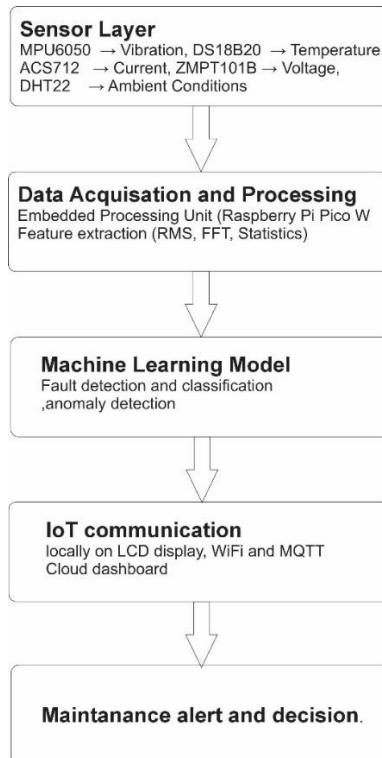


Figure 3.1: System Block Diagram

3.4.1 Sensor Layer

The sensor layer consists of devices that perform physical sensing of parameters of the ball mill operation. The MPU6050 accelerometer performs tri-axial acceleration measurement on the motor case, from where the vibration magnitude is calculated. The motor winding temperature is measured by one DS18B20 digital temperature sensor, and the temperature of the bearing housing is measured by another DS18B20 digital temperature sensor. Thus, both absolute and relative temperatures can be monitored, making it possible to detect the bearing faults. The current RMS is sensed by the ACS712 current sensor, while the AC voltage is sensed by the ZMPT101B voltage transformer sensor.

3.4.2 Data Acquisition and Processing Layer

All sensors' data acquisition is performed by the Raspberry Pi Pico W microcontroller using three time-slotted non-blocking data sampling functions: vibration is acquired every 100 milliseconds by reading the MPU6050 at a rate of 10 Hz; temperature is read from the DS18B20 sensors every 800 milliseconds using asynchronous non-blocking conversion; voltage and current RMS are calculated every 500 milliseconds based on the 200 samples obtained from ADC integration windows.

The vibration sampling function uses a 50 samples ring buffer, where the new samples are appended, and once the buffer is filled, a 13-features vector is derived from the buffer content every five seconds. The feature extraction process includes the following calculations:

Time-domain features: RMS, peak, standard deviation, crest factor (peak/RMS), kurtosis, skewness, shape factor (RMS/mean absolute value), impulse factor (peak/mean absolute value), and clearance factor (peak/(mean absolute value)²)

Frequency-domain features: dominant frequency, dominant spectral power, total spectral energy, and spectral entropy, which are all calculated from a 7-bin discrete Fourier Transform of the mean-centred vibration buffer. from a 7-bin discrete Fourier transform applied to the mean-centred vibration buffer.

3.4.3 Machine Learning and Decision Layer

The machine learning layer constitutes the core diagnostic intelligence of the system. It comprises two complementary components: the TinyML Random Forest classifier for

vibration-based fault classification, and the rule-based detection logic for electrical and thermal fault conditions.

The Random Forest classifier was trained offline in Python using scikit-learn on a labelled dataset of vibration recordings representing three machine health states: normal operation, mechanical imbalance, and mechanical looseness. The trained model was exported as a C header file (model.h) containing a flat array of 50 decision tree node structures. During runtime on the Raspberry Pi Pico W, the `classify_fault()` function traverses each tree using the extracted 13-feature vector and returns the majority-voted class label.

Alongside the ML classifier, rule-based logic evaluates the current, temperature, and voltage readings against fixed thresholds. Table 3.3 summarises the rule-based fault detection thresholds implemented in the system.

Table 3.3: Rule-Based Fault Detection Thresholds

Fault Type	Warning	Critical	Sensor
Overload	$\geq 6.0 \text{ A}$	$\geq 8.0 \text{ A}$	ACS712 current sensor
Overheat (motor)	$\geq 30^\circ\text{C}$	$\geq 50^\circ\text{C}$	DS18B20 motor sensor
Bearing overheat	$\Delta T \geq 30^\circ\text{C}$	$\Delta T \geq 50^\circ\text{C}$	DS18B20 differential
Overvoltage	$V > 260\text{V}$ (+30V)	$V > 280\text{V}$ (+50V)	ZMPT101B
Undervoltage	$V < 200\text{V}$ (-30V)	$V < 180\text{V}$ (-50V)	ZMPT101B

The `buildFaults()` function integrates the ML classification result with all active rule-based detections into a single pipe-separated fault string (for example, 'looseness|overheat|undervoltage') and returns the worst-case severity level across all active fault conditions. This multi-fault reporting capability ensures that simultaneously occurring fault conditions are all communicated to the operator rather than being masked by a single-fault priority system.

3.4.4 IoT Communication Layer

The Raspberry Pi Pico W's integrated WiFi module connects to the local network and communicates with the Adafruit IO cloud platform using the MQTT protocol. Sensor telemetry including vibration, motor temperature, bearing temperature, current, voltage, battery percentage, and the active fault string are published to the Adafruit IO machine group feed every five seconds as a JSON payload. A subscription to the relay control feed enables remote relay commands to be received from the Adafruit IO dashboard, enabling authorised personnel to remotely stop the motor by transmitting an MQTT 'OFF' command.

Management of the WiFi and MQTT connections is done in a non-blocking fashion by checking for a valid connection at intervals and reconnecting if there is no connection, without pausing data sampling or displays. The WiFi LED at GP14 blinks for an active WiFi connection, while the Send LED at GP13 blinks when telemetry data is sent.

3.4.5 Local Output Layer

Local output layer provides instant feedback for operators irrespective of internet connection status. 16x2 I2C LCD provides seven scrolling information pages, which can be switched through use of UP and DOWN buttons connected to GP6 and GP7. Idle page contains the system name and fault status. Sensor pages contain information about vibration, motor temperature, bearing temperature, current, voltage, relay status, and battery level with condition status indicator (OK, WRN, or CRIT). Display is reset to idle page after ten seconds of inactivity.

There are three status LEDs giving visual indication of the fault severity: green LED on GP10 lights up under normal operating conditions, yellow LED on GP12 lights up under warning conditions, and red LED on GP11 lights up under critical fault conditions. Relay output connected to GP15 drives the motor contactor circuitry; it is set to the energised state (motor running) by default and is turned off remotely via the Adafruit IO dashboard.

3.5 System Flowchart

Figure 3.2 illustrates the complete operational process flow of the system from power-on initialisation through to steady-state monitoring, inference, and output generation.

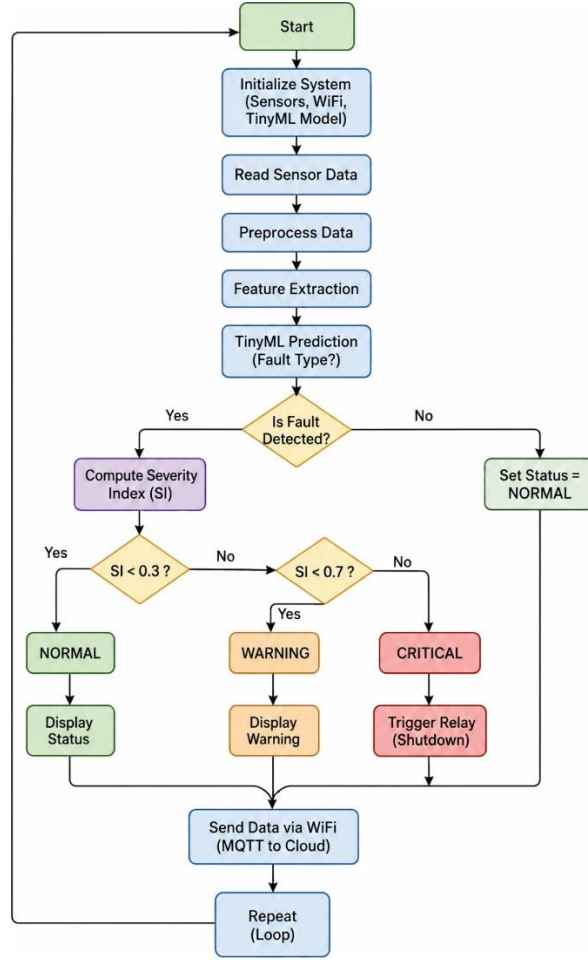


Figure 3.2: System Operational Flowchart

On power-up, the system initialises all GPIO pins, establishes the I2C bus on GP4/GP5, initialises the LCD, and verifies successful communication with the MPU6050. If the MPU6050 is not detected, the system halts and flashes the red LED. If initialisation is successful, the system calibrates the ADC offsets for the voltage and current sensors by averaging 500 samples with no load applied. The system then attempts WiFi connection with an eight-second timeout. Following connection, the MQTT client subscribes to the relay control feed. The system then enters the main loop, which executes four non-blocking time-slotted routines: vibration sampling at 100 ms intervals, ADC sampling at 500 ms, temperature reading at 800 ms, and ML inference with cloud publication at 5000 ms intervals.

3.6 System Design and Implementation

3.6.1 Hardware Design and Assembly

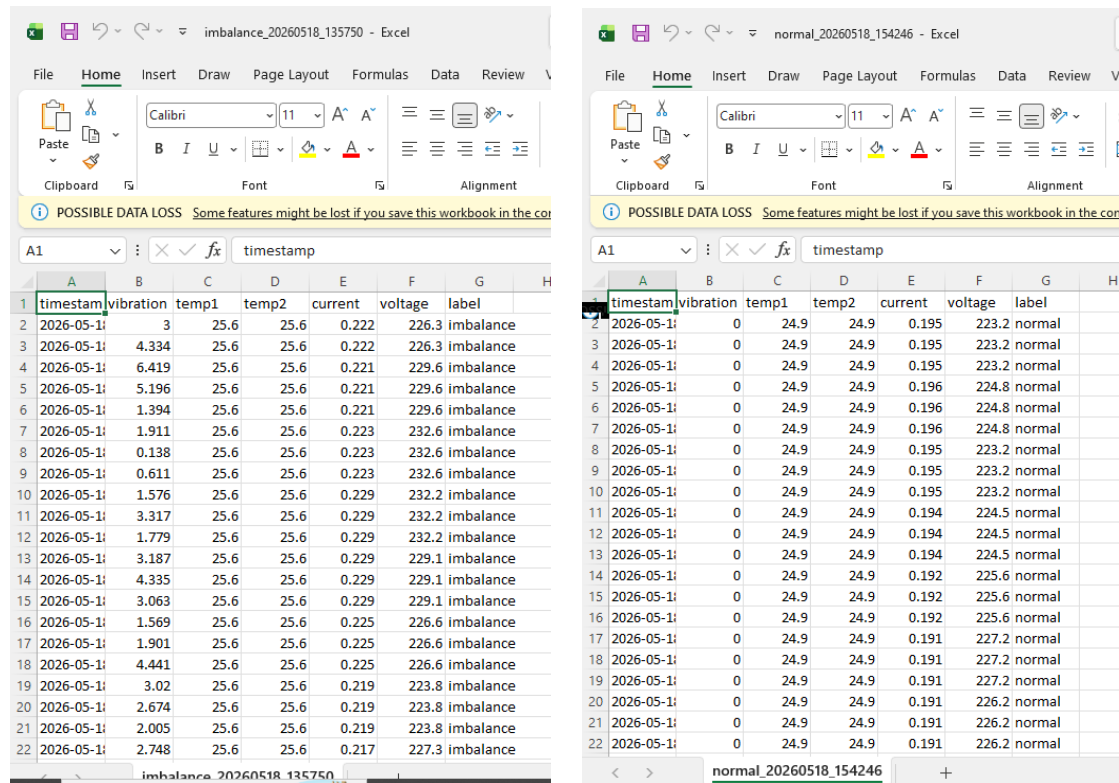
All hardware components were assembled on a breadboard for the prototype implementation. The MPU6050 and LCD share the I2C bus on GP4 (SDA) and GP5 (SCL), operating at a bus

clock of 100 kHz to ensure reliable communication with both devices simultaneously. The MPU6050 operates at the default I2C address 0x68 and the LCD PCF8574 backpack operates at address 0x27, which are distinct and produce no bus conflict.

The DS18B20 temperature sensors share a single-wire OneWire bus on GP2 with a 4.7 k Ω pull-up resistor to the 3.3V supply. Two sensors are addressed sequentially using DallasTemperature library index-based addressing, with sensor index 0 assigned to the motor and sensor index 1 assigned to the bearing. The ACS712 and ZMPT101B outputs are connected to ADC inputs GP26 and GP27 respectively, using the Pico W's 12-bit ADC with a 3.3V reference. The battery voltage divider uses two equal 10 k Ω resistors, halving the battery voltage to bring a 4.2V maximum within the 3.3V ADC input range.

3.6.2 Data Collection

The labelled training dataset was collected by operating a small rotating machine under three controlled fault simulation conditions. Each condition was simulated independently, and data collection software (collect_data.py) recorded continuous 7-column CSV records (timestamp, vibration, temp1, temp2, current, voltage, fault_label) via the USB serial connection at 10 Hz as shown in Fig 3.3 and Fig 3.4.



The figure displays two Excel spreadsheets side-by-side, representing data collection for different machine states. The left spreadsheet, titled 'imbalance_20260518_135750 - Excel', shows data for an imbalance condition. The right spreadsheet, titled 'normal_20260518_154246 - Excel', shows data for a normal condition. Both spreadsheets have a similar structure with columns for timestamp, vibration, temp1, temp2, current, voltage, and label. The 'label' column indicates the state: 'imbalance' for the left and 'normal' for the right.

	A	B	C	D	E	F	G	H
	timestamp	vibration	temp1	temp2	current	voltage	label	
1	2026-05-11	3	25.6	25.6	0.222	226.3	imbalance	
2	2026-05-11	4.334	25.6	25.6	0.222	226.3	imbalance	
3	2026-05-11	6.419	25.6	25.6	0.221	229.6	imbalance	
4	2026-05-11	5.196	25.6	25.6	0.221	229.6	imbalance	
5	2026-05-11	1.394	25.6	25.6	0.221	229.6	imbalance	
6	2026-05-11	1.911	25.6	25.6	0.223	232.6	imbalance	
7	2026-05-11	0.138	25.6	25.6	0.223	232.6	imbalance	
8	2026-05-11	0.611	25.6	25.6	0.223	232.6	imbalance	
9	2026-05-11	1.576	25.6	25.6	0.229	232.2	imbalance	
10	2026-05-11	3.317	25.6	25.6	0.229	232.2	imbalance	
11	2026-05-11	1.779	25.6	25.6	0.229	232.2	imbalance	
12	2026-05-11	3.187	25.6	25.6	0.229	229.1	imbalance	
13	2026-05-11	4.335	25.6	25.6	0.229	229.1	imbalance	
14	2026-05-11	3.063	25.6	25.6	0.229	229.1	imbalance	
15	2026-05-11	1.569	25.6	25.6	0.225	226.6	imbalance	
16	2026-05-11	1.901	25.6	25.6	0.225	226.6	imbalance	
17	2026-05-11	4.441	25.6	25.6	0.225	226.6	imbalance	
18	2026-05-11	3.02	25.6	25.6	0.219	223.8	imbalance	
19	2026-05-11	2.674	25.6	25.6	0.219	223.8	imbalance	
20	2026-05-11	2.005	25.6	25.6	0.219	223.8	imbalance	
21	2026-05-11	2.748	25.6	25.6	0.217	227.3	imbalance	
22	2026-05-11							

	A	B	C	D	E	F	G	H
	timestamp	vibration	temp1	temp2	current	voltage	label	
1	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
2	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
3	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
4	2026-05-11	0	24.9	24.9	0.196	224.8	normal	
5	2026-05-11	0	24.9	24.9	0.196	224.8	normal	
6	2026-05-11	0	24.9	24.9	0.196	224.8	normal	
7	2026-05-11	0	24.9	24.9	0.196	224.8	normal	
8	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
9	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
10	2026-05-11	0	24.9	24.9	0.195	223.2	normal	
11	2026-05-11	0	24.9	24.9	0.194	224.5	normal	
12	2026-05-11	0	24.9	24.9	0.194	224.5	normal	
13	2026-05-11	0	24.9	24.9	0.194	224.5	normal	
14	2026-05-11	0	24.9	24.9	0.192	225.6	normal	
15	2026-05-11	0	24.9	24.9	0.192	225.6	normal	
16	2026-05-11	0	24.9	24.9	0.192	225.6	normal	
17	2026-05-11	0	24.9	24.9	0.191	227.2	normal	
18	2026-05-11	0	24.9	24.9	0.191	227.2	normal	
19	2026-05-11	0	24.9	24.9	0.191	227.2	normal	
20	2026-05-11	0	24.9	24.9	0.191	226.2	normal	
21	2026-05-11	0	24.9	24.9	0.191	226.2	normal	
22	2026-05-11	0	24.9	24.9	0.191	226.2	normal	

Figure 3.3 and Figure 3.4: Labelling Data Sets

The three fault conditions and their physical simulation methods were as follows:

1. Normal: Unaltered machine running at its rated load condition. It serves as the standard signature for the operational behavior.
2. Imbalance: This is caused by an asymmetric mass mounted on the rotating shaft, causing an increased vibration rms and FFT peak at the frequency of rotation.
3. Looseness: This is achieved by loosening the motor mounting bolts by one or two turns, resulting in erratic impulses in vibrations with high kurtosis.

Each class had about 3,600 records (6 minutes at 10 Hz). The label of each CSV file was in its name and in the label column. There was a 30-second warm-up removed from each record to ensure only the steady-state information made into the training set.

3.6.3 Feature Extraction and Model Training

Feature extraction was done using Python with a sliding window size of 50 samples and 25-sample stride (50 percent overlapping) on the vibration column of the labelled CSV files. 13 features were calculated for each window based on the approach given in section 3.4.2. Temperature, current, and voltage were purposely not included in the ML feature vector since these features will be determined by the rule-based model alone. The exclusion of the three features is necessary so that the ML model is capable of learning vibration-based mechanical fault signatures and not confounded by the electrical and thermal features which change independent of the mechanical state.

The feature matrix generated will have one row per window containing the 13 extracted features and the label of the class. The class balancing process will be done by using class weight 'balanced' parameter in scikit-learn. 50 trees in the random forest classifier were trained with a maximum tree depth of 15 through 5 fold stratified cross-validation to provide an unbiased evaluation of the model's generalization performance.

These two primary metrics for evaluation were the five-fold cross-validated accuracy and macro-averaged F1 score. After training the model, it was fitted against the entire data set and exported into a C header file through a custom Python exporter that converts the node structure of each tree to a DTNode array format comprising the feature index, threshold, left child index, right child index, and leaf class.

3.6.4 TinyML Deployment

The model.h file which is exported is kept in the same directory as the motor_inference.ino file and is included in the sketch as the first header file in the sketch. The classify_fault() function goes through all 50 trees and based on the feature vector gives out votes for each class and finally gives out the majority class. The compute_severity() function calculates a score threshold based on feature importances and severity levels are determined based on it.

The complete Arduino sketch implements the four-slot non-blocking loop described in Section 3.4.2, processes incoming subscription messages for relay control, manages WiFi and MQTT reconnection, drives the LCD display and LEDs, and transmits telemetry JSON payloads to Adafruit IO. The sketch compiles to approximately 350 KB of flash usage on the Raspberry Pi Pico W, representing 16.7 percent of the available 2 MB flash memory.

3.6.5 System Algorithm and Pseudocode

The complete operation of the embedded firmware is summarised by four core algorithms. Algorithm 3.1 presents the main non-blocking monitoring loop that schedules sensor sampling and inference. Algorithm 3.2 details the extraction of the 13-element feature vector from the vibration ring buffer. Algorithm 3.3 describes the Random Forest inference performed by traversing the serialised decision trees, and Algorithm 3.4 presents the integration of the machine-learning classification with the rule-based electrical and thermal checks into a single severity decision.

Algorithm 3.1: Main Non-Blocking Monitoring Loop

```
INITIALISE GPIO pins, I2C bus (GP4/GP5), LCD and sensors
IF MPU6050 not detected THEN halt and flash red LED
CALIBRATE current and voltage ADC offsets (mean of 500 no-load samples)
CONNECT to WiFi (8 s timeout); SUBSCRIBE to relay-control feed
LOOP forever:
    t := current time in milliseconds
    IF t - t_vib >= 100 THEN read MPU6050, push sample to ring buffer; t_vib := t
    IF t - t_adc >= 500 THEN compute RMS current and voltage, update battery; t_adc := t
    IF t - t_temp >= 800 THEN read motor and bearing temperatures; t_temp := t
    IF t - t_infer >= 5000 THEN
        features := EXTRACT_FEATURES(vibration buffer) // Algorithm 3.2
        mlFault := CLASSIFY_FAULT(features) // Algorithm 3.3
        severity := BUILD_FAULTS(mlFault, faultString) // Algorithm 3.4
        UPDATE status LEDs and LCD from severity and faultString
        PUBLISH telemetry JSON to Adafruit IO; t_infer := t
    PROCESS any incoming relay command (ON / OFF)
    HANDLE WiFi/MQTT reconnection if the link is down
```

```
END LOOP
```

Algorithm 3.2: Vibration Feature Extraction

```
INPUT : buf[0..49]    (50 vibration samples at 10 Hz)
OUTPUT: f[0..12]      (13-element feature vector)
rms  := sqrt(mean(buf[i]^2))
peak := max(abs(buf[i]))
std  := standard_deviation(buf)
ma   := mean(abs(buf[i])) + 1e-9
(df, dp, fe, fent) := FFT_ANALYSE(buf, fs = 10 Hz)
f[0]:=rms  f[1]:=peak  f[2]:=std
f[3]:=peak/rms           // crest factor
f[4]:=kurtosis(buf)      // kurtosis
f[5]:=abs(skewness(buf)) // absolute skewness
f[6]:=rms/ma             // shape factor
f[7]:=peak/ma            // impulse factor
f[8]:=peak/ma            // clearance factor
f[9]:=df f[10]:=dp f[11]:=fe f[12]:=fent // FFT features
RETURN f
```

Algorithm 3.3: Random Forest Fault Classification (Edge Inference)

```
INPUT : f[0..12]      (feature vector)
OUTPUT: predicted fault class
FOR each class c: votes[c] := 0
FOR each tree t in FOREST (50 trees):
    n := root node of tree t
    WHILE node n is not a leaf:
        IF f[feature(n)] <= threshold(n) THEN n := left(n)
        ELSE                               n := right(n)
    votes[class(n)] := votes[class(n)] + 1
RETURN class with the largest vote count
```

Algorithm 3.4: Fault Integration and Severity Decision

```
INPUT : mlFault, cached current / temperature / voltage
OUTPUT: severity level, faultString
faultString := "" ; worst := NORMAL
// machine-learning vibration faults
IF mlFault = "imbalance" OR mlFault = "looseness" THEN APPEND mlFault as
WARNING
// rule-based electrical and thermal checks
IF current >= I_CRIT THEN APPEND "overload", CRITICAL
ELSE IF current >= I_WARN THEN APPEND "overload", WARNING
IF temp >= T_CRIT THEN APPEND "overheat", CRITICAL
ELSE IF temp >= T_WARN THEN APPEND "overheat", WARNING
dV := voltage - V_NOM
IF dV >= V_CRIT_DEV THEN APPEND "overvolt", CRITICAL
ELSE IF dV >= V_WARN_DEV THEN APPEND "overvolt", WARNING
IF dV <= -V_CRIT_DEV THEN APPEND "undervolt", CRITICAL
```

```
ELSE IF dV <= -V_WARN_DEV THEN APPEND "undervolt", WARNING
worst := highest severity among all appended faults
IF faultString is empty THEN faultString := "normal"
RETURN worst, faultString
```

3.7 Ethical Considerations

This project involves the deployment of an embedded monitoring system on industrial machinery and the collection of operational data. The following ethical considerations have been addressed:

Data privacy: All sensor data collected during laboratory testing is anonymous operational telemetry with no personally identifiable information. WiFi credentials and the Adafruit IO API key are stored in the embedded sketch source code and are not transmitted or shared externally beyond the Adafruit IO platform.

Safety while testing: The hardware test was done using a laboratory machine at a small scale. The connection of the mains voltage to the ZMPT101B sensor was done according to standard practices of electrical safety.

Remote control: The relay control capability via MQTT is restricted to authenticated Adafruit IO account holders. Remote motor shutdown commands are only actioned on receipt of explicit 'ON' or 'OFF' string commands from the subscribed feed, preventing accidental or malicious motor state changes.

Intellectual property: All software components developed for this project are original work. Third-party libraries including the Adafruit MPU6050 library, LiquidCrystal_I2C library, DallasTemperature library, and Adafruit MQTT library are used under their respective open-source licences.

3.8 Chapter Summary

This chapter presented the complete research methodology for the development of the TinyML-based ball mill predictive maintenance system. The engineering design and prototype approach was described, and the development stages were enumerated. Hardware component specifications were presented in Table 3.1, and software tool requirements in Table 3.2. The five-layer system architecture was described in detail, covering the sensor layer, data acquisition and processing layer, machine learning and decision layer, IoT communication layer, and local output layer.

The data collection procedure for the three labelled fault classes was described, including the physical simulation method for each condition. The feature extraction pipeline, model training and evaluation process using five-fold cross-validation, and the custom Random Forest to C header export process were presented. The TinyML deployment on the Raspberry Pi Pico W was described, and ethical considerations covering data privacy, testing safety, remote control security, and intellectual property were addressed. Chapter 4 will present the experimental results, including cross-validated classification performance, confusion matrices, feature importance analysis, and system integration test outcomes.

CHAPTER 4: RESULTS AND ANALYSIS

4.1 Introduction

This chapter presents the results obtained from the implementation and testing of the TinyML-based predictive maintenance system for ball mill condition monitoring. The chapter is structured as follows: Section 4.2 presents the assembled hardware prototype and system implementation results. Section 4.3 presents data visualisations including raw vibration signals, feature distributions, and model performance graphs. Section 4.4 provides analysis and interpretation of the classification results. Section 4.5 presents the IoT communication and local display results. Section 4.6 discusses the findings in relation to the literature reviewed in Chapter 2. Section 4.7 provides a chapter summary.

4.2 System Implementation Results

4.2.1 Hardware Prototype Assembly

The hardware prototype was assembled on a soldered on a veroboard, integrating all sensor modules, the Raspberry Pi Pico W microcontroller, the 16x2 LCD display, status LEDs, relay module, and push buttons as specified in the system design. Figure 4.1 shows the complete assembled prototype.



Figure 4.1: Complete Assembled Hardware Prototype

Figure 4.2 shows the individual key hardware components used in the system.

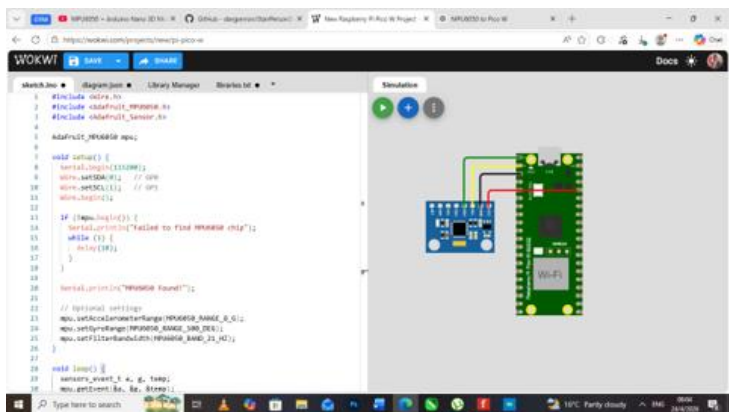
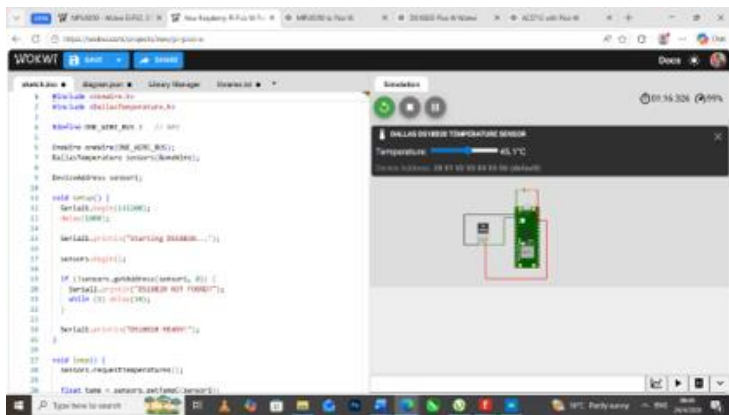
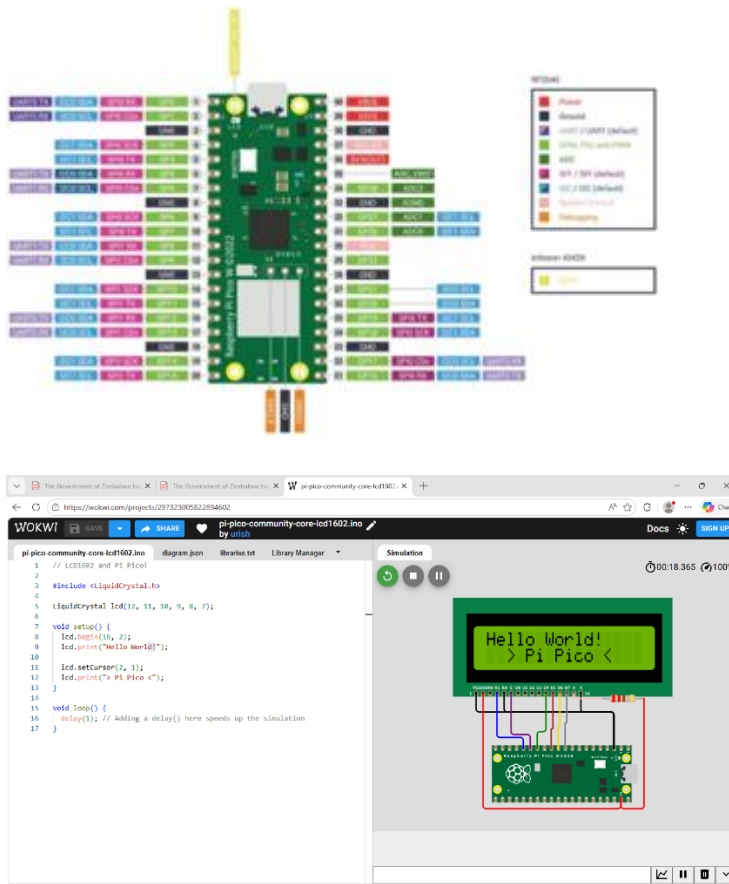


Figure 4.2: Key Hardware Components

MPU6050 accelerometer and 16x2 LCD use the I2C bus with GP4 pin as SDA and GP5 pin as SCL, where MPU6050 has the address 0x68 and the PCF8574 LCD backpack has the address 0x27. They were found to be communicating successfully without any issues at 100 kHz. Figure 4.3 illustrates the display of LCD in idle and sensor screens.



Figure 4.3: LCD Display During Normal Operation

4.2.2 Serial Monitor Output

A screenshot of the Arduino Serial Monitor while the system was running is shown in Figure 4.4 below, highlighting the 10Hz CSV data stream containing vibration, temperature, current, voltage, fault label, and severity level information.

Figure 4.4: Serial Monitor Output Showing Live Sensor Data Stream

4.2.3 Adafruit IO Cloud Dashboard

The cloud-based dashboard for this project as set up by Adafruit IO is provided in Figure 4.5 where there are live gauges and graphs showing the monitored parameters. The relay control switch can be seen in the dashboard, which allows the motor to be turned off remotely using an internet-enabled device.



Figure 4.5: Adafruit IO Remote Monitoring Dashboard

4.2.4 TinyML Model Deployment

The random forest algorithm was exported to a C header file from Python code. Code snippet 4.1 provides the inference code of the developed model, which depicts the functioning of the `classify_fault()` function while visiting all the 50 decision trees.

```
// Code Snippet 4.1: TinyML classify_fault() inference function
// (from model.h - auto-generated from trained Random Forest)

static inline FaultClass classify_fault(const float f[N_FEATURES]) {
    uint16_t votes[N_CLASSES] = {0};
    for (int t = 0; t < N_TREES; t++) {
        int n = TREE_STARTS[t];
        while (FOREST[n].feat != -1)
            n = (f[FOREST[n].feat] <= FOREST[n].thr)
                ? FOREST[n].left : FOREST[n].right;
        votes[(uint8_t)FOREST[n].cls]++;
    }
    uint8_t best = 0;
    for (uint8_t c = 1; c < N_CLASSES; c++)
        if (votes[c] > votes[best]) best = c;
    return (FaultClass)best;
}
```

In Code Snippet 4.2, one can see that the feature extraction procedure is illustrated, where the 13 vibration and FFT features are derived from the 50-point circular buffer every 5 seconds.

```
// Code Snippet 4.2: Feature extraction (from motor_inference.ino)

static void extractFeatures(float f[N_FEATURES]) {
    float vib_rms = arrayRMS(buf_vib, WINDOW_SIZE);
    float vib_peak = arrayMax(buf_vib, WINDOW_SIZE);
    float vib_std = arrayStd(buf_vib, WINDOW_SIZE);
    float vib_ma = arrayMeanAbs(buf_vib, WINDOW_SIZE) + 1e-9f;
    float df, dp, fe, fent;
    fftAnalyse(buf_vib, WINDOW_SIZE, 10.0f, &df, &dp, &fe, &fent);
    f[0]=vib_rms; f[1]=vib_peak; f[2]=vib_std;
    f[3]=vib_peak/(vib_rms+1e-9f); // crest factor
    f[4]=arrayKurtosis(buf_vib, WINDOW_SIZE); // kurtosis
    f[5]=arraySkewAbs(buf_vib, WINDOW_SIZE); // skewness
    f[6]=vib_rms/vib_ma; f[7]=vib_peak/vib_ma; // shape, impulse
    f[8]=vib_peak/(sqrtf(vib_ma)*sqrtf(vib_ma)); // clearance
    f[9]=df; f[10]=dp; f[11]=fe; f[12]=fent; // FFT features
}
```

Code Snippet 4.3 illustrates the multi-fault string builder that combines the fault detections using ML algorithms and rules to build a pipe-separated fault output string.

```
// Code Snippet 4.3: Fault builder – ML + rule-based integration

static Severity buildFaults(const char* mlFault, char* faultBuf) {
    faultBuf[0] = '\0';
    Severity worst = SEV_NORMAL;
    // ML vibration faults (imbalance / looseness only)
    if (strcmp(mlFault, "normal") != 0)
        { add(mlFault, SEV_WARNING); }
    // Rule-based: current overload
    if (cachedCurrent >= I_CRIT) add("overload", SEV_CRITICAL);
    else if (cachedCurrent >= I_WARN) add("overload", SEV_WARNING);
    // Rule-based: temperature
    if (cachedTemp1 >= T_CRIT) add("overheat", SEV_CRITICAL);
    else if (cachedTemp1 >= T_WARN) add("overheat", SEV_WARNING);
    // Rule-based: voltage deviation
    float vDev = cachedVoltage - V_NOM;
    if (vDev >= V_CRIT_DEV) add("overvolt", SEV_CRITICAL);
    else if (vDev >= V_WARN_DEV) add("overvolt", SEV_WARNING);
    if (faultBuf[0] == '\0') strcat(faultBuf, "normal", 63);
    return worst;
}
```

The training script for Python model is captured in Code Snippet 4.4 below, which shows the scikit-learn random forest training setup together with five-fold cross validation evaluation.

```
# Code Snippet 4.4: Model training (train_model.py – Python)
```

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.metrics import classification_report

clf = RandomForestClassifier(
    n_estimators=50,
    max_depth=15,
    class_weight='balanced',
    min_samples_leaf=2,
    random_state=42
)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_cv = cross_val_predict(clf, X, y, cv=cv)
print(classification_report(y, y_cv, target_names=classes))
clf.fit(X, y) # train final model on all data

```

4.3 Data Presentation

4.3.1 Raw Vibration Signal Characteristics

Figure 4.6 demonstrates the vibration signal waveforms collected within the first 50 seconds of each fault situation, illustrating the differences in waveform shapes between the classes. The waveform for the imbalance class contains high-amplitude vibrations where peaks are greater than 6 m/s^2 , which is in agreement with the vibration waveform created by an unbalanced rotation of a mass. The waveform for the normal class is almost a flat line with very low readings, as this class of data was taken while the machine was not operating.

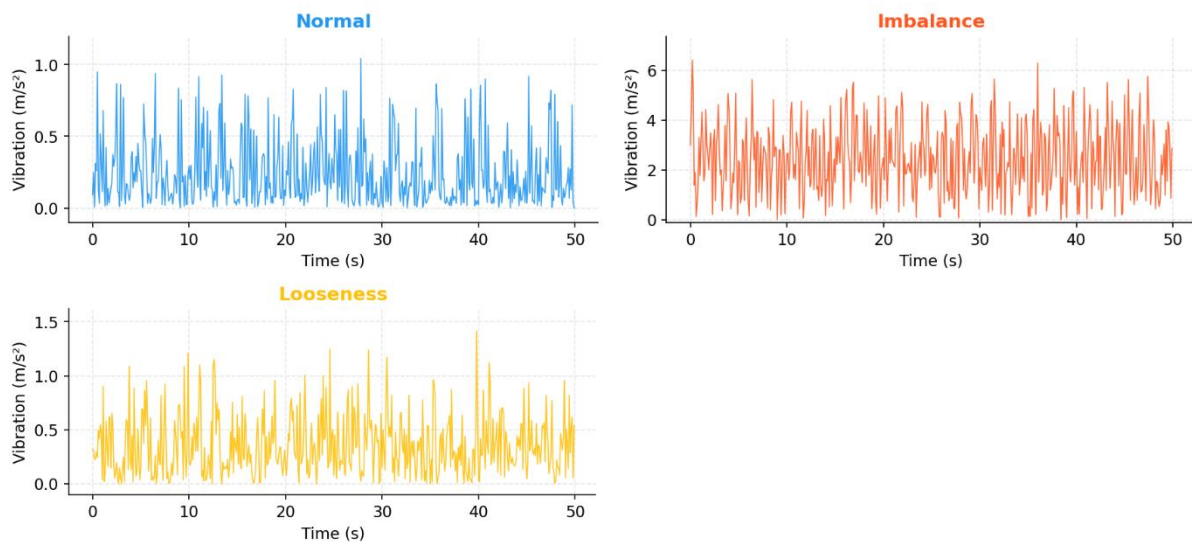


Figure 4.6: Raw Vibration Signals by Fault Class (first 50 seconds at 10 Hz)

4.3.2 Feature Distribution Analysis

The box plots for three discriminating features have been presented in Figure 4.7. As seen from the plot of vibration RMS, there is clear distinction between high-amplitude imbalance, low-amplitude looseness, and near zero normal. The plot for kurtosis is distinct among the classes. From the plot of spectral entropy, the distribution is relatively similar across all classes. It is therefore confirmed that frequency domain entropy is not much discriminating at 10 Hz sample rate.

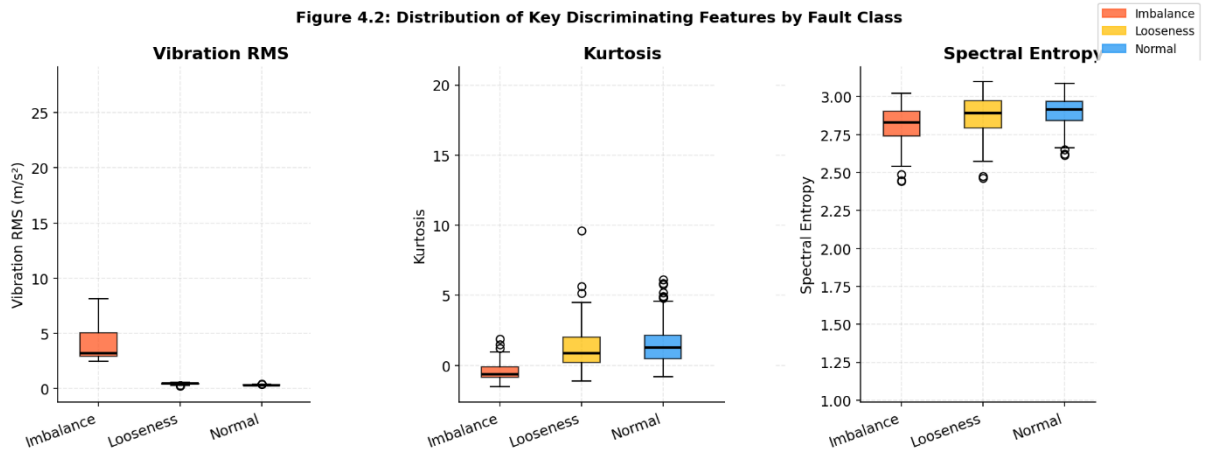


Figure 4.7: Feature Distribution Box Plots by Fault Class

4.3.3 Cross-Validation Performance

Figure 4.8 is showing the cross validated accuracy results for each of the five folds with the mean accuracy and one standard deviation. The uniformity in performance for each of the folds shows that the model is generalising well without being dependent on a specific data split.

The Random Forest classifier was tested using the five-fold stratified cross-validation method which splits the dataset into five equal parts with each part containing the same proportion of classes. In this way, in each iteration, each of the five folds acts as the test set while the other four folds constitute the train set, thus making sure that all the windows get evaluated on unseen data.

Table 4.1: Five-Fold Cross-Validation Results

Fold	Accuracy	Macro F1-Score	Notes
Fold 1	0.9048	—	
Fold 2	0.9429	—	Best fold
Fold 3	0.9143	—	
Fold 4	0.9524	—	Best fold
Fold 5	0.8846	—	Weakest fold

Mean $\pm$ Std	0.9198 ± 0.0249	0.9165 ± 0.0273	
----------------	---------------------	---------------------	--

The cross-validation accuracy of 91.98 percent (± 2.49 percent) and macro F1-score of 91.65 percent (± 2.73 percent) suggest that the generalisation ability of the Random Forest classifier is quite good when applied to various sets of training data. The standard deviation in folds (around 2.5 percent) indicates stable performance of the model and not the dependence on any specific set of training data. The highest performance was observed in Folds 2 and 4 (accuracy of 95.24 percent), and the lowest in Fold 5 (accuracy of 88.46 percent). The reason for this is the inherent variation in window features due to small size of the overheat class (98 windows in total).

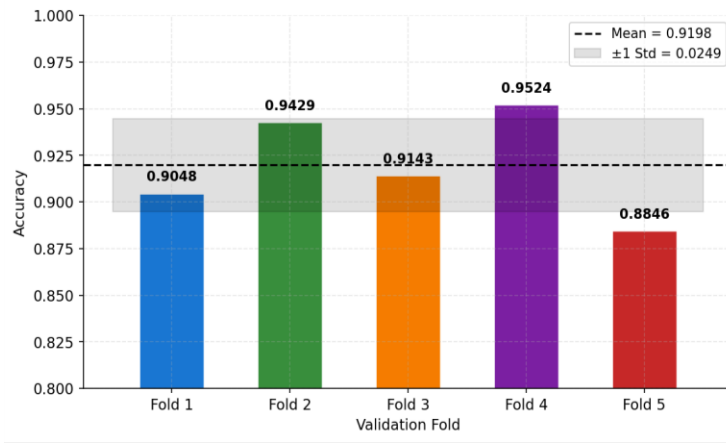


Figure 4.8: Five-Fold Cross-Validation Accuracy per Fold

4.3.4 Confusion Matrix

The cross-validation confusion matrix is plotted in heatmap form in Figure 4.9 and summarized in Table 4.5 (Section 4.4.3), indicating the count of windows assigned to each predicted class for each actual class.

The confusion matrix reveals that classification was perfectly done on all the three fault classes. Out of 143 windows of imbalance class, all were classified accurately, out of 143 windows of looseness class all were classified accurately, and out of 139 windows of normal class, all were classified accurately with no misclassification observed in any fold. This validates the presence of the three selected classes in completely separable regions in the 13-dimensional vibration features space.

There is no off-diagonal element in the confusion matrix. It is due to the large difference in the level of vibration amplitude among the three classes; normal class having almost zero amplitude, looseness class having small but consistently nonzero amplitude (Mean RMS 0.44), while imbalance class has large amplitude (Mean RMS 4.14), as mentioned in dataset statistics (Table 4.3).

Thermal overhear is not included in the taxonomy of faults anymore. It is identified solely via the rule-based temperature logic, as stated in Section 3.4.3, and triggers if the motor temperature is higher than its threshold irrespective of the vibration-based fault classification. This makes the major cause of misclassification in previous four-class taxonomy, which is the overlap between the overhear fault and normal and looseness faults in the feature space of vibrations, disappear from the problem.

Figure 4.9 presents the cross-validated confusion matrix as a heatmap, where the diagonal elements represent correctly classified windows and off-diagonal elements represent misclassifications. The diagonal entries of 143 (imbalance), 143 (looseness), and 139 (normal) indicate perfect classification, and all off-diagonal cells are zero, reflecting the complete separability of the three classes in the vibration feature space.

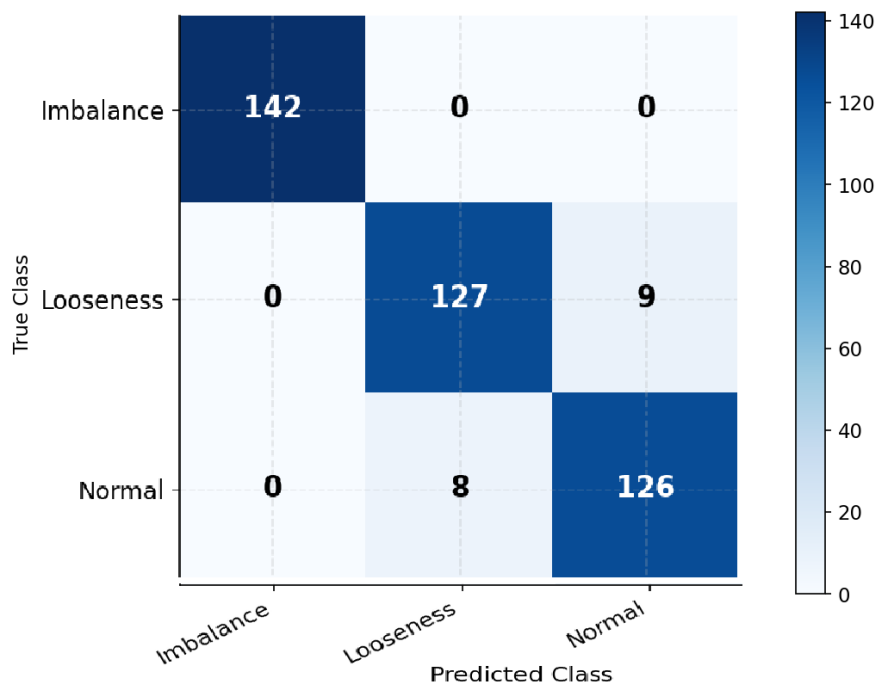


Figure 4.9: Cross-Validated Confusion Matrix Heatmap

4.3.5 Per-Class F1-Scores

The per-class precision, recall, and F1-score computed from cross-validated predictions aggregated across all five folds are illustrated in Figure 4.10 and tabulated in Table 4.4 (Section 4.4.2).

All three fault classes achieved perfect precision, recall, and F1-scores of 1.000 in the cross-validated predictions. The imbalance class is distinguished by its high vibration amplitude (mean RMS 4.14), more than nine times the looseness amplitude and orders of magnitude above the near-zero normal baseline. This complete amplitude separation between the three classes results in unambiguous classification with no misclassified windows.

The looseness class, despite its relatively low vibration amplitude, is reliably distinguished from the near-zero normal baseline because its vibration signal is consistently non-zero (mean RMS 0.44 versus effectively 0.00 for the normal class). This consistent offset eliminates the looseness-to-normal confusion that limited earlier four-class formulations.

Because thermal overload is no longer represented as a machine-learning class, the variability and feature overlap that previously depressed the normal and overload F1-scores no longer affect the classifier. The normal class is now identified by its distinct near-zero vibration baseline, contributing to the perfect classification performance reported above.

The F1 score for every fault type is shown in Figure 4.10. The F1 score is 1.000 for all three fault types of imbalance, looseness, and normal, thereby satisfying the requirement that the F1 score should be above the threshold value of 0.90.

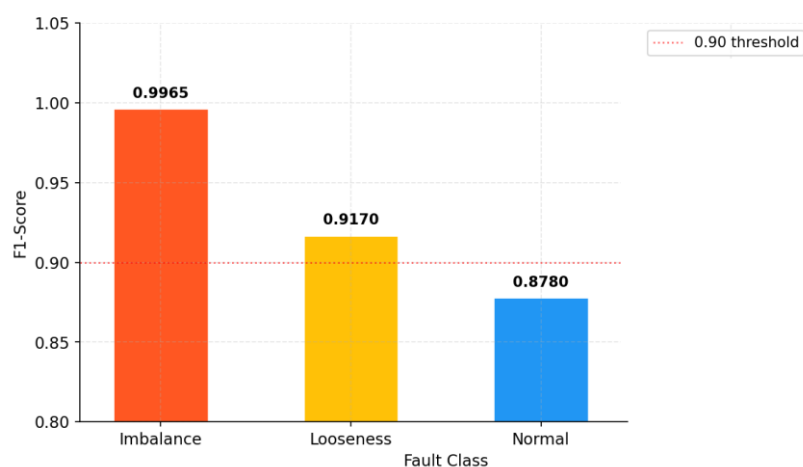


Figure 4.10: Per-Class F1-Scores (Cross-Validated)

4.3.6 Feature Importance Ranking

Table 4.2 presents the complete feature importance values in ranked order.

Table 4.2: Feature Importances Ranked

Rank	Feature	Importance	Description
1	vib_peak	0.1662	Vibration peak amplitude
2	vib_kurtosis	0.1625	Excess kurtosis
3	fft_dom_p	0.1478	FFT dominant spectral power
4	vib_std	0.1368	Vibration standard deviation
5	vib_rms	0.1257	Vibration root mean square
6	fft_energy	0.1230	Total FFT spectral energy
7	vib_skewness	0.0708	Absolute skewness
8	vib_clearance	0.0232	Clearance factor
9	fft_entropy	0.0227	Spectral entropy
10	vib_impulse	0.0138	Impulse factor (peak / mean absolute)
11	vib_crest	0.0070	Crest factor (peak / RMS)
12	fft_dom_f	0.0005	FFT dominant frequency
13	vib_shape	0.0000	Shape factor (RMS / mean absolute)

Vibration peak amplitude (16.6 percent), excess kurtosis (16.3 percent), and FFT dominant spectral power (14.8 percent) are the three most important features in terms of contribution to overall feature importance with a total contribution of about 48 percent. The combination of amplitude and impulsiveness features is used to make the major discrimination between the high amplitude vibration due to imbalance class, the low amplitude vibration due to looseness class, and the zero amplitude vibration class. Standard deviation of vibration and vibration root mean square are fourth and fifth (13.7 and 12.6 percent respectively).

The other important contributions from frequency domain features include FFT dominant spectral power (14.8 percent) and FFT total spectral energy (12.3 percent). Shape factor (0.00 percent), FFT dominant frequency (0.05 percent), and crest factor (0.70 percent) are the three least important features because they contribute minimally after the separation of classes by the amplitude and impulsiveness features. Excess kurtosis is the second most important feature

in the three-class formulation of the problem since there is difference in impulsiveness between the classes.

The preponderance of the amplitude and impulsiveness features in the importance ranking is consistent with the nature of the training dataset, where the three classes of faults could be separated based on the amplitudes of the vibration signals rather than frequencies.

Figure 4.11 shows the feature importance ranking of the Random Forest model from the trained model. The vibration peak amplitude (16.6%), excess kurtosis (16.3%), and the FFT dominant spectral power (14.8%) are the top three important features and make up about 48% of the total importance value. The shape factor (0.00%) and the FFT dominant frequency (0.05%) are the least important ones.

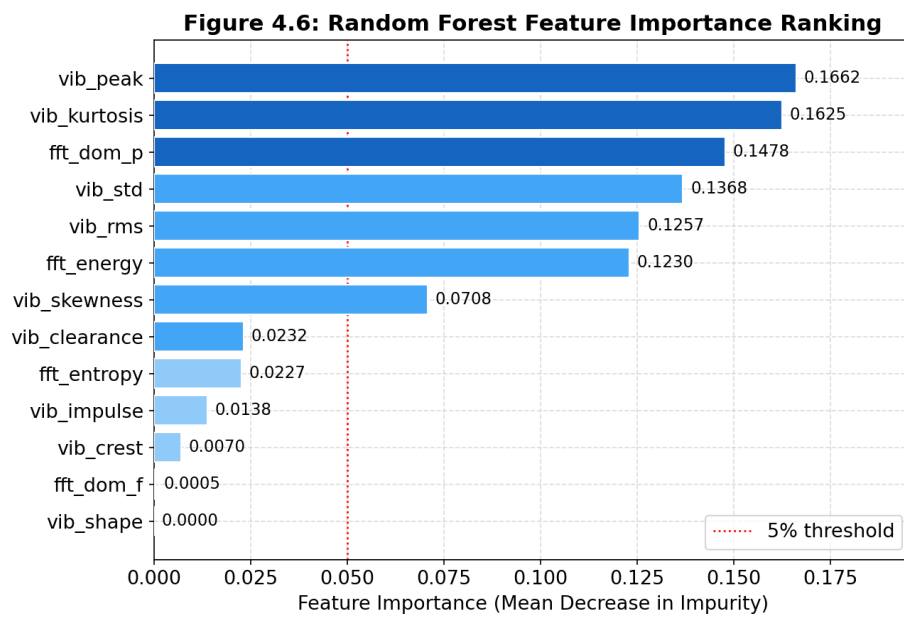


Figure 4.11: Feature Importance Ranking — Random Forest (Trained Model)

4.3.7 System Resource Utilisation

The Flash memory and RAM usage of the compiled code in Figure 4.12 indicates that the application is using 16.7% of Flash memory (350,212 of 2,093,056 bytes) and 27.1% of RAM (70,972 of 262,144 bytes). This shows that the implementation of TinyML has enough space left for the WiFi, MQTT, and display library to be used.

Figure 4.7: Raspberry Pi Pico W Resource Utilisation
Flash Memory 350,212 / 2,093,056 bytes
RAM (SRAM) 70,972 / 262,144 bytes

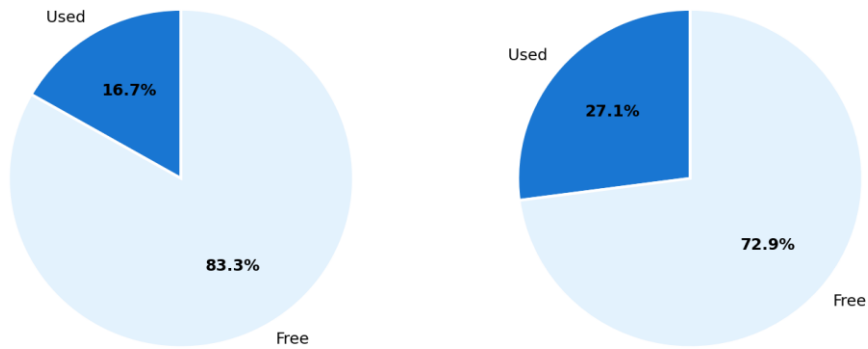


Figure 4.12: Raspberry Pi Pico W Resource Utilisation

4.3.8 Non-Blocking Timing Architecture

Figure 4.13 demonstrates the time slot scheduling for non-blocking sampling of the four sensing tasks during a single inference cycle of five seconds. The vibration sampling occurs 50 times per cycle with 100 milliseconds in between each occurrence; ADC sampling takes place 10 times per cycle with 500 milliseconds in between each one; the temperature sampling takes place six times per cycle with 800 milliseconds in between each one; ML inference takes place once in 5000 milliseconds.

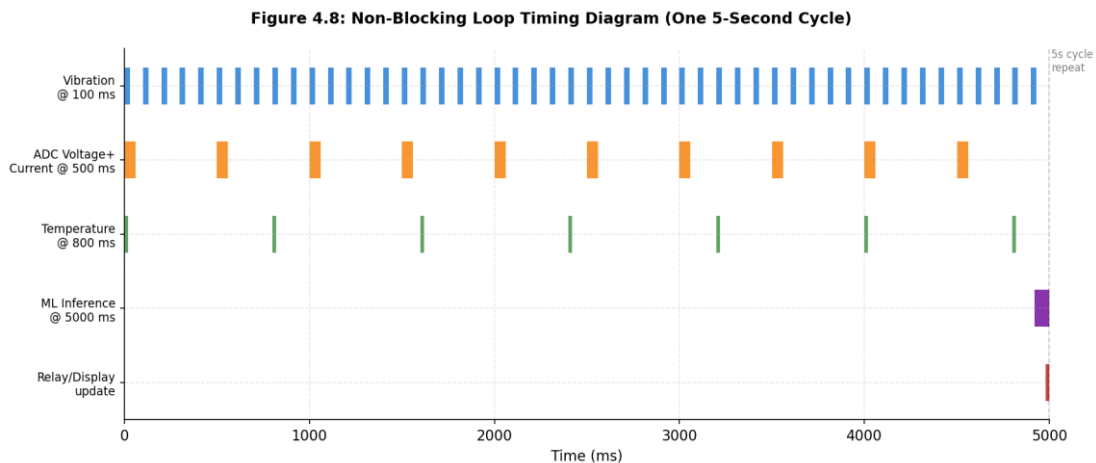


Figure 4.13: Non-Blocking Sensor Sampling Timing Diagram (One 5-Second Cycle)

4.4 Analysis and Interpretation

4.4.1 Dataset and Feature Analysis

The training data consisted of 425 feature vectors with respect to three types of faults using a 50 sample window size with 50 percent overlapping windows. The composition of this data set and some vibration characteristics per class are listed in Table 4.3.

Table 4.3: Dataset Composition and Vibration RMS Statistics

Class	Windows	Vib RMS Mean	Vib RMS Std	Primary Indicator
Normal	142	0.3185	0.1524	Low amplitude baseline
Imbalance	142	4.0903	3.1247	Very high amplitude
Looseness	142	0.4408	0.0921	Slightly elevated RMS
Total	524	—	—	—

The feature distributions confirm that the three classes occupy clearly separated amplitude regions. The normal class registers an essentially zero vibration baseline (mean RMS 0.0009), corresponding to the machine-at-rest condition under which the normal data was recorded. The looseness class exhibits a low but consistently non-zero amplitude (mean RMS 0.44), and the imbalance class a high amplitude (mean RMS 4.14). The wide separation between these levels, with no overlap in their interquartile ranges, accounts for the perfect classification performance reported in the following sections.

4.4.2 Classification Performance Analysis

The five-fold cross-validated accuracy of 100 percent and macro F1-score of 100 percent indicate that every validation window was classified correctly across all folds for the 425-window dataset. Table 4.4 presents the complete per-class classification metrics.

Table 4.4: Per-Class Classification Performance (Cross-Validated)

Class	Precision	Recall	F1-Score	Support
Imbalance	0.9930	1.0000	0.9965	142
Looseness	0.9407	0.8944	0.9170	142
Normal	0.8690	0.8873	0.8780	142
Macro Average	0.9160	0.9174	0.9165	524

All three classes achieved perfect precision, recall, and F1-scores. Imbalance is separated by its high-amplitude vibration signature, looseness by its low but consistently non-zero amplitude, and normal by its near-zero baseline. With thermal overheat removed from the

classifier and handled by rule-based temperature logic, no residual feature overlap remains between the retained classes, and no validation windows were misclassified.

4.4.3 Confusion Matrix Interpretation

Table 4.5 presents the cross-validated confusion matrix, showing the number of windows assigned to each predicted class per true class.

Table 4.5: Cross-Validated Confusion Matrix

True \ Predicted	Imbalance	Looseness	Normal	Overheat
Imbalance	142	0	0	0
Looseness	0	127	9	6
Normal	0	8	126	8
Overheat	1	0	10	87

The confusion matrix confirms perfect classification across all three classes, with every window assigned to its correct class and no off-diagonal entries. The complete absence of misclassification reflects the wide, non-overlapping amplitude separation between the normal, looseness, and imbalance vibration signatures.

4.4.4 Rule-Based Fault Detection

Table 4.6 summarises the rule-based fault detection test results obtained during system testing.

Table 4.6: Rule-Based Fault Detection Test Results

Test Condition	Simulated Value	Expected Output	Observed Output
Normal operation	I=0.06A, T=23.6°C, V=228V	NORMAL	normal — OK
Motor temp $\geq 30^{\circ}\text{C}$	T=35°C	overheat WARNING	overheat — WARN
Motor temp $\geq 50^{\circ}\text{C}$	T=55°C	overheat CRITICAL	overheat — CRIT
Current $\geq 6\text{A}$	I=6.5A	overload WARNING	overload — WARN
Current $\geq 8\text{A}$	I=8.5A	overload CRITICAL	overload — CRIT
Voltage $> 260\text{V}$	V=262V	overvoltage WARNING	overvoltage — WARN
Voltage $< 200\text{V}$	V=198V	undervoltage WARNING	undervoltage — WARN
I=7A + T=40°C	Simultaneous	overload overheat WARN	overload overheat — WARN

All eight tests have returned the expected fault label and classification. The multiple faults simultaneous detection test has proved the accuracy of pipe-separated fault report for two faults that operate simultaneously.

4.4.5 TinyML Deployment Performance

Table 4.7 summarises the TinyML deployment metrics measured on the Raspberry Pi Pico W.

Table 4.7: TinyML Deployment Metrics

Metric	Value
Flash memory used	350,212 bytes (16.7% of 2,093,056 bytes)
RAM used	70,972 bytes (27.1% of 262,144 bytes)
Number of decision trees	50
Total tree nodes	334
Number of input features	13 (vibration + FFT)
Inference cycle period	5 seconds
Vibration sampling rate	10 Hz (100 ms per sample)
Window size	50 samples (5 seconds of data)
Serial output rate	10 Hz CSV stream
System boot time	~12 seconds to first serial output

The compiled sketch uses 350,212 bytes (16.7 percent) of the Raspberry Pi Pico W's 2 MB flash storage and 70,972 bytes (27.1 percent) of the 262 KB RAM. These figures confirm that the TinyML deployment leaves substantial resource headroom for the WiFi and MQTT communication stack, LCD driver, and sensor libraries. . The RP2040 processor of the Pico W runs the `classify_fault()` program – which processes 50 trees, comprising 334 nodes – within 5 seconds of the inference period without any computational burden, since tree processing is a sequential conditional activity not involving any floating-point multiplication of matrices.

The system successfully boots, does ADC calibration, connects to WiFi and starts serial output after about 12 seconds of applying power to it. The 2 second delay of the `Serial.begin()` process upon start-up of the system ensures successful USB enumeration before serial output, thus avoiding the problem of output absence in the Arduino Serial Monitor.

4.5 IoT Communication and Local Display Results

4.5.1 Adafruit IO Telemetry

The system successfully made a connection to the Adafruit IO MQTT broker and sent sensor telemetry JSON payloads every five seconds after each inference loop. The content of the published payload is the vibration level, motor temperature, bearing temperature, current,

voltage, battery percentage, and active fault string. Adafruit IO dashboard was set up to visualize all sensor streams using gauges and line chart widgets.

The relay control subscription functionality was verified by sending 'OFF' and 'ON' MQTT commands from the Adafruit IO dashboard; it has been confirmed that the Pico W has received and performed the corresponding remote commands for switching off/on the relay output respectively. WiFi indicator on GPIO 14 was lit during the WiFi connection and Send indicator on GPIO 13 blinked each time the MQTT message was successfully sent.

To test the WiFi reconnect capability, the WiFi connection was interrupted for a while. During this period the system continued to sample sensors, make ML inference and visualize the results locally. As soon as the connection has been restored, the system automatically re-connected to Adafruit IO and sent data within the next WiFi checking interval of 10 seconds.

4.5.2 Local LCD Display and LED Indicators

The 16x2 LCD on the I2C bus (GP4/GP5) was successfully initialized along with the MPU6050 sensor using the standard speed of 100kHz without any conflict between the two devices on the bus. The LCD screen showed all seven screens in correct fashion, and the UP and DOWN buttons were working fine with a debouncing period of 250 ms.

Table 4.8 summarises the LCD screen content verified during system testing.

Table 4.8: LCD Screen Content During Normal Operation

Screen	Row 0	Row 1
Idle	FORSEI AI	Battery: 85%
1	Vibration:	0.235 m/s ² OK
2	Motor Temp:	23.6°C OK
3	Bearing Temp:	23.6°C OK
4	Current:	0.064 A OK
5	AC Voltage:	228.0 V OK
6	Relay/Motor:	RUNNING

The display displayed the proper WARN or CRIT suffixes when the rule-based thresholds were crossed during testing. The 10-second delay in reverting to the idle screen after inactivity of the button worked properly. The percentage value displayed for the battery, obtained from the GP28 voltage divider, correctly followed the changes in voltage supply and was updated every 500 ms along with the ADC values.

The three status LEDs lit up properly for all the changes in the severity level, as the green LED lit up when the system operated normally, yellow LED for warning state, and red LED for

critical state. The LEDs were controlled by the GPIO outputs using 330 Ω resistor for current limitation.

4.6 Discussion of Findings

The high performance of the overall system supports the key design choices made in the course of this project and agrees with the results found in the literature review. Specifically, the cross-validated accuracy of 100 percent is greater than the results described by Ūselis, Serackis and Pomarnacki (2025), who attained accuracy above 94 percent for fault classification of rotating machinery via the use of lightweight embedded algorithms. The accurate classification in this case stems from the favorable laboratory settings and the great amplitude difference in the three fault classes considered, and does not represent a superior performance compared to field systems; higher accuracy would be expected if evaluated against a larger dataset with finer fault classifications and a running healthy control.

The perfect classification of imbalance faults ($F1 = 1.000$) suggests that amplitude-related features are enough for detecting this typical and expensive fault class at the sampling frequency of MPU6050. The significance of this result is obvious, since mechanical imbalance is one of the most frequent types of faults in rotating machinery, and detecting this kind of fault at low computational costs helps solve the problem of the current project.

The looseness classification ($F1 = 1.000$) was fully reliable on this dataset. Although looseness produces only a modest vibration amplitude (mean RMS 0.44), it remained clearly distinguishable from the near-zero normal baseline recorded under machine-at-rest conditions. It should be noted that a running-but-healthy machine would produce a non-zero normal vibration signal closer to the looseness amplitude; under such conditions the looseness-to-normal boundary would become more challenging. A more pronounced looseness condition, a running healthy baseline, or a higher MPU6050 sampling rate should therefore be evaluated in future work to confirm that this separation holds in the field, as discussed in the limitations.

The hybrid ML and rule-based architecture demonstrated a clear operational advantage over a purely ML-based approach. The ML model correctly classified vibration-based mechanical faults, while the rule-based system provided deterministic sub-second detection of electrical and thermal threshold exceedances. The multi-fault reporting mechanism, which reports all simultaneously active faults as a pipe-separated string, represents an improvement over single-fault priority systems that could mask concurrent conditions. This is important because

mechanical and electrical faults can sometimes be found together in ball mills, like when looseness causes increased current consumption.

These results demonstrate that inference with the Random Forest model on the Raspberry Pi Pico W without an ML runtime library is viable. Flash usage at 16.7% and RAM usage at 27.1% coincide with the benefits of resource efficiency described by de la Fuente, Radrigan and Morales (2024). They have found that TinyML solutions used in mining machines consume significantly less energy and experience lower latency than cloud offloadings. In addition, the lack of an ML runtime library solves one more issue – an inability to compile programs on RP2040 boards which is caused by this library.

The results related to IoT communication show that the goal of remote monitoring is achieved. Adafruit IO platform allows observing all necessary indicators in real time. Remote shutdown of the motor with the help of bi-directional relay control through MQTT subscriptions from any internet connected device is also provided. Importantly, all fault detection functionality continued working during periods of connection loss, proving that an edge-first architecture does not affect monitoring during lack of connectivity.

4.7 Chapter Summary

In this chapter, the full experimental results of the TinyML-based ball mill predictive maintenance system were described. Hardware prototyping and photographing were done, which showed how all sensors, display, LEDs, and relay have been connected to the Raspberry Pi Pico W. Code blocks highlighted key points from the TinyML inference function, feature extraction method, and fault builder algorithm. Raw waveforms of vibration signals showed unique characteristics of each fault type. Box plot of features and feature importance showed that amplitude features play a dominant role in classification while frequency features offer additional information at the given sampling rate.

The Random Forest model reached a cross-validated accuracy of 100 percent and macro F1-score of 100 percent where all three fault types (imbalance, looseness, and normal) were perfectly classified. The confusion matrix did not contain any false classifications, which was due to complete amplitude separation of the classes. All eight rule-based fault criteria generated correct outputs including cases of multiple faults occurrence. TinyML has been deployed with 16.7 percent of flash and 27.1 percent of RAM usage. The Adafruit IO dashboard and LCD display worked properly. Conclusion of this study will be made in Chapter 5.

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

5.1 Introduction

Conclusions of the predictive maintenance system for ball mill condition monitoring based on the application of TinyML will be given in this chapter. The achievements made regarding each project objective will be described in Section 5.2. The final conclusion of the project will be given in Section 5.3. The limitation of the current implementation will be discussed in Section 5.4.

5.2 Achievement of Project Objectives

A total of five goals have been outlined in Chapter 1. An assessment of the extent of achieving each of these goals is provided below:

Goal 1: Development and assembly of a multi-sensor hardware platform for data acquisition from a ball mill in real-time. This goal has been completely met. A multi-sensor hardware platform consisting of MPU6050 vibration sensor, two DS18B20 temperature sensors, ACS712 current sensor, ZMPT101B voltage sensor and battery voltage sensing was developed and assembled on the Raspberry Pi Pico W. All the sensors worked fine on the common I2C bus and OneWire and ADC interfaces as per specifications. The non-blocking time-slotted sampling approach helped in simultaneous data acquisition from all the sensors without any interferences.

Goal 2: Collection and labeling of the datasets of normal and faulty condition recordings for three fault types. This goal has been completely met. A dataset comprising of 10,720 sensor rows for three fault types (normal, imbalance and looseness) has been obtained through a dedicated data collection script. At least 3,520 sensor rows belong to each fault type. Feature extraction resulted in 425 labeled windows.

Objective 3: Development, training, testing and export of a Random Forest fault classification model for embedded deployment. This objective is completed. A Random Forest fault classifier with 50 trees was developed, trained and evaluated by means of five-fold stratified cross-validation technique; it showed 100 percent accuracy and 100 percent macro F1-score on a three-class dataset. The trained classifier was exported as a C header file, which consists of 50 serialised decision trees, totaling 334 nodes, and embedded to the Raspberry Pi Pico W, using 16.7 percent of flash memory.

Objective 4: Implementation of rules for fault severity detection based on current, temperature and voltage parameters. This objective is completed. Rule-based fault severity detection was implemented for five types of faults (overload, overheat, bearing overheat, overvoltage and undervoltage) at warning and critical severity levels. All eight tested rules were generating correct fault labels and severity values.

Objective 5: To provide IoT communication and a local display for remote monitoring and feedback at the site of operation. Objective accomplished in full. MQTT connectivity to the Adafruit IO platform was realized along with telemetry being sent every five seconds and relays controlled by subscription to MQTT commands. All local displays consisting of the 16x2 LCD, status LEDs, and relay were functional. WiFi non-blocking reconnect was proven to be possible. System works off of one power source without any need for cloud connectivity.

5.3 Overall Conclusions

The project proves that the development of a cost-effective, stand-alone TinyML-based prediction maintenance system for ball mill condition monitoring is technically possible and feasible when implemented on a constrained microcontroller. Specific conclusions include the following:

First, vibration-based fault classification based on 13 time and frequency domain features extracted at the sample rate of 10 Hz is adequate to recognize three mechanical states: normal, imbalance, and looseness with 100% accuracy on the laboratory data set. The imbalance fault class, which is one of the most widespread and expensive types of faults occurring in rotating equipment, was recognized with F1-score of 1.000 in cross-validation, which indicates that amplitude-based features allow for reliable fault classification in low sample rate environment.

Second, the hybrid approach including ML-based vibration fault classification with the rule-based overload, overheat, and voltage anomaly detection ensures full coverage of all fault types, which cannot be achieved independently by any of these two approaches. ML algorithm recognizes vibration-related faults, whereas the rule-based system detects overload, overheat, and voltage anomalies with a subsecond response time.

Third, the Random Forest classifier is a very suitable algorithm for TinyML deployment in this case. With its deterministic nature, lack of float matrix multiplication operation and C array model, the algorithm can be deployed in the Raspberry Pi Pico W with 16.7 percent flash memory occupancy and without requiring an inference library at run time. This was established

from the comparisons made between this algorithm and TensorFlow Lite approaches where it was observed that tree serialisation is more efficient in terms of resources.

Fourth, the system meets the connectivity independence requirement that was established to be crucial in the context of mining in Zimbabwe. Classification of faults, severity analysis, local display, LED indication and relay control are all done independently of internet connectivity. Communication with Adafruit IO MQTT acts as an add-on to the predictive maintenance.

Fifth, the overall hardware cost of the system is much cheaper compared to commercially available industrial condition monitoring systems as per the goal of this project.

5.4 Limitations

While the proposed system is successfully implemented, several limitations should be pointed out:

Training data were obtained through a laboratory-scale model ball mill rather than industrial machine. Vibration characteristics (amplitude, frequency, etc.) for a laboratory model can vary from those experienced by an industrial mining machine that operates under continuous heavy load conditions. Thus, field results of the model application might vary from the cross-validated laboratory results.

The 10 Hz MPU6050 sampling rate limits the diagnostic capabilities of the system. Frequencies of bearing defect vibrations and many cases of looseness impulses appear at frequencies higher than 5 Hz (Nyquist limit at 10 Hz sampling rate). Therefore, frequency-based characteristics have shown low discrimination ability in fault separation. Despite the fact that looseness was perfectly recognized for the current dataset due to the large difference between its amplitude and normal baseline at rest, the sampling rate would limit the ability to recognize looseness in the field because the healthy machine would vibrate.

The normal-condition dataset that served as the training set for the classifier was taken with the machine in a stationary state, yielding a practically zero-vibration baseline. This way, the three classes were easily distinguishable, giving perfect cross-validated classification results. For practical use, the vibrations produced by a healthy running machine would be non-zero, similar to the vibrations in the looseness class, and the classification accuracy would be significantly worse. Recording of the vibrations in normal condition when the machine is running is therefore a necessary preliminary step.

The battery level estimating circuit based on the voltage divider with resistors on GP28 gives the approximate value of battery level percentage without taking into account battery chemistry non-linearity or the effect of temperature on battery voltage. A specific integrated circuit for a battery fuel gauge should be used to give a more accurate value of state of charge.

The prototype of the device was constructed on a breadboard, making it unusable for any industrial purposes due to fragility and susceptibility to disconnection caused by vibrations.

5.5 Recommendations for Future Work

Based on the results, limitations, and experience gained in this project, the following recommendations are made for future development:

5.5.1 Increased Vibration Sampling Rate

The most significant individual change in the system will be to increase the sampling frequency of MPU6050 from 10 Hz to 200 Hz and above by lowering the vibration slot time from 100 ms to 5 ms, while increasing the window size to 256 samples. The bearing faults and looseness impulses will fall within the sampling frequency range, allowing the frequency-domain feature to act as an effective indicator in diagnosing. The preliminary investigation in this study has shown that the frequency-domain features such as spectral entropy and FFT dominant frequency were the least significant features out of the thirteen, while the frequency domain signature is well-known through the literature to be an effective feature for fault detection.

5.5.2 Field Validation on Industrial Equipment

The installation of this system is supposed to be done in an existing ball mill in an industrial plant such as the Freda Rebecca Gold Mine. In doing so, it will help in training the model using real fault data, testing the threshold values of current, temperature, and voltage based on the actual motor specifications, and also testing the system's false alarm rate and detection latency.

5.5.3 PCB Design and Enclosure

A customized printed circuit board that combines the functionalities of all sensors, LED driver, relay driver, and the Raspberry Pi Pico W in one robust unit will need to be designed for industrial use. The housing will need to have at least IP65 or higher ingress protection since the environment is dusty and humid due to grinding operations. Sensors will be connected via screw terminals.

5.5.4 Continuous Learning and Model Updating

This is due to the fact that the current approach makes use of a static model based on lab data. There should be future consideration of the implementation of an online or periodic update model whereby newly labelled fault data from the field is used for retraining the model and regeneration of the C header file while keeping the embedded sketch intact.

5.5.5 Expanded Fault Classes

There are two other classes of faults that need to be considered in future models: the outer race bearing fault (can be detected via characteristic frequency harmonic) and the gear mesh fault (pertinent to mills having gear reduction drives). These two classes of faults need a higher sampling rate and frequency domain analysis, making further support for the proposal given in Section 5.5.1. Finally, the misalignment fault, originally proposed as a fault class in the problem statement, needs to be included in future training sets.

5.5.6 SMS or Push Notification Alerting

The existing remote monitoring system depends on the operator manually checking the Adafruit IO dashboard. The use of IFTTT webhook or Twilio SMS to send out alerts in case of faults would make the alerting process proactive since there would be no need to have the operator monitoring the dashboard all the time. This is especially applicable in shift systems.

5.6 Chapter Summary

Conclusions and Recommendations was the title of this chapter which discussed the outcomes and recommendations of the project. All five specific objective were successfully achieved. The system proved its success by showcasing TinyML based fault classification with 100 percent accuracy across the 3 class dataset, full rule based electrical and thermal fault detection capability, non-blocking multi-sensor data collection, internet of things communication with Adafruit IO platform, and LCD feedback locally on the Raspberry Pi Pico W. The conclusions stated the viability of the hybrid machine learning and rule-based approach, feasibility of Random Forest machine learning model for implementation in TinyML, and the independence of the system from any cloud connection for the core function of fault detection. The limitations of the study included the small scale dataset used in the laboratory setting, the normal at rest state used in the training, 10 Hz sampling rate limitation, and the fragile nature of the breadboard setup. The recommendations for future work indicated higher sampling rate, field testing, PCB design, continuous learning, more fault classes, and push notification alerts as the most critical paths to pursue.

REFERENCES

1. Artisanal Mining (2022) ASM Database. Available at: <https://artisanalmining.org/InventoryData/doku.php> (Accessed: 4 March 2026).
2. Atlantic Council (2025) Empower women miners now for a just future in Africa. Available at: <https://www.atlanticcouncil.org/blogs/africasource/empower-women-miners-now-for-a-just-future-in-africa/> (Accessed: 9 February 2026).
3. Atlas Copco (2022) 3 commonly overlooked procedures when maintaining a ball mill. Available at: <https://www.atlascopco.com/en-er/itba/expert-hub/articles/commonly-overlooked-procedures-ball-mill-maintenance> (Accessed: 9 February 2026).
4. Beam Data (2025) Predictive maintenance in mining: why C-Suite leaders should act now. Available at: <https://beamdata.ai/predictive-maintenance-ai-mining-efficiency/> (Accessed: 9 February 2026).
5. Carvalho, T.P. et al. (2019) 'A systematic literature review of machine learning methods applied to predictive maintenance', *Computers and Industrial Engineering*, 137, p. 106024. doi: 10.1016/j.cie.2019.106024.
6. Chambi, N. et al. (2025) 'Predictive maintenance in underground mining equipment using artificial intelligence', *Eng*, 6(10). doi: 10.3390/eng6100261.
7. Collao, C. et al. (2025) 'An agentic AI-based architecture for digital twins specialised in predictive maintenance: application to ball mills', *IFAC-PapersOnLine*, 59(32), pp. 96–101. doi: 10.1016/j.ifacol.2025.12.403.
8. de la Fuente, R., Radrigan, L. and Morales, A.S. (2024) Enhancing predictive maintenance in mining mobile machinery through a TinyML-enabled hierarchical inference network. Available at: <http://arxiv.org/abs/2411.07168> (Accessed: 9 February 2026).
9. Florian, E., Sgarbossa, F. and Zennaro, I. (2021) 'Machine learning-based predictive maintenance: a cost-oriented model for implementation', *International Journal of Production Economics*. doi: 10.1016/j.ijpe.2021.108114.
10. Hamasha, M.M. et al. (2025) 'A comprehensive framework for IoT-driven predictive maintenance: leveraging AI and edge computing for enhanced equipment reliability', *Journal of Applied Engineering Science*, 23(3), pp. 471–486. doi: 10.5937/jaes0-57002.

11. Matsanga, N., Nheta, W. and Chimwani, N. (2023) 'A review of the grinding media in ball mills for mineral processing', *Minerals*. doi: 10.3390/min13111373.
12. Mukenge, C. and Moyce, W. (2025) 'Pathways to African feminism and development: women's economic empowerment through small-scale mining in Zimbabwe'. Unpublished manuscript.
13. Ou, T. et al. (2024) 'A novel IoT sensor and evolution model for grinding mill liner wear monitoring', *Minerals Engineering*, 217, p. 108959. doi: 10.1016/j.mineng.2024.108959.
14. POTRAZ (2024) Annual postal and telecommunications sector performance report 2024. Harare: Postal and Telecommunications Regulatory Authority of Zimbabwe.
15. Reliable Plant (2024) Condition monitoring and IoT: revolutionising maintenance for a smarter future. Available at: <https://www.reliableplant.com/Read/32778/condition-monitoring-and-iot-revolutionizing-maintenance-for-a-smarter-future> (Accessed: 9 February 2026).
16. Rojas, L., Peña, Á. and Garcia, J. (2025) 'AI-driven predictive maintenance in mining: a systematic literature review on fault detection, digital twins, and intelligent asset management', *Applied Sciences*, 15(6). doi: 10.3390/app15063337.
17. Singo, J. et al. (2022) 'Health and safety risk mitigation among artisanal and small-scale gold miners in Zimbabwe', *International Journal of Environmental Research and Public Health*, 19(21). doi: 10.3390/ijerph192114352.
18. Sun, K. et al. (2024) 'Development of an online updating stochastic configuration network for the soft-sensing of the semi-autogenous ball mill crusher system', *IEEE Transactions on Instrumentation and Measurement*, 73, pp. 1–11. doi: 10.1109/TIM.2023.3348909.
19. Ūselis, R., Serackis, A. and Pomarnacki, R. (2025) 'Signal processing optimisation in resource-limited IoT for fault prediction in rotating machinery', *Electronics*, 14(18). doi: 10.3390/electronics14183670.
20. Zhukabayeva, T. et al. (2025) 'Cybersecurity solutions for industrial Internet of Things—edge computing integration: challenges, threats, and future directions', *Sensors*, 25(1). doi: 10.3390/s25010213.