BetraX: IoT Based Smart Predictive Maintenance Module

Vedant Prashant Buge[1], Umakant Shirshetti[2], Konjengbam Anand[3]
[1,2,3] *Department of Information Technology Sou. Venutai Chavan polytechnic, Sinhgad Institutes Pune, Maharashtra, India*
doi.org/10.64643/IJIRTV12I11-196513-459

***Abstract*—Predictive maintenance has emerged as a critical strategy for enhancing the reliability and operational lifespan of electronic and industrial devices. Conventional maintenance paradigms, including scheduled inspections and reactive repairs, frequently result in unforeseen downtime and elevated operational costs. This paper presents a comprehensive AIdriven predictive maintenance framework that integrates Internet of Things (IoT) sensors, cloud computing infrastructure, and machine learning to enable continuous, real-time device health monitoring and proactive failure prediction. The proposed system employs temperature, vibration, and electrical current sensors interfaced with an ESP32 microcontroller, transmitting operational data to a Firebase Realtime Database at five-second intervals. A Python-based backend applies a Random Forest classifier to categorize device health into three discrete risk levels: LOW, MEDIUM, and HIGH. The framework achieves a macro-average F1-score of 94.3% with an 18.4-second early-warning lead time prior to HIGH-risk transitions. Comparative analysis against five existing frameworks confirms competitive accuracy and superior costefficiency. The paper further contributes mathematical formulations, algorithm pseudocode, deployment scalability analysis, and an Industry 4.0 contextual discussion.**

***Index Terms*—Predictive Maintenance; Internet of Things; ESP32; Random Forest; Firebase; Machine Learning; Cloud Analytics; Industry 4.0; Fault Detection; Real-Time Monitoring; IIoT.**

I. INTRODUCTION

Modern industrial and electronic ecosystems depend increasingly on uninterrupted device operation. Equipment failures within these environments can precipitate cascading production disruptions, accelerated asset degradation, and substantial financial losses. Traditional maintenance paradigms-most notably reactive maintenance, which addresses failures postoccurrence, and time-based preventive maintenance, which relies on fixed inspection schedules-are fundamentally limited in their capacity to anticipate impending faults. Neither approach leverages the continuous stream of operational data generated by contemporary embedded systems and sensors.

Predictive maintenance (PdM) addresses these limitations by employing real-time sensor analytics and machine learning to identify deviations from normal operational patterns before they escalate into critical failures [1]. The proliferation of low-cost IoT hardware, such as the ESP32 microcontroller, has substantially lowered the barrier to deploying distributed sensor networks. Coupled with scalable cloud platforms, continuous data acquisition and remote analytics are now achievable in both industrial and resource-constrained settings [2].

Machine learning algorithms play a pivotal role in extracting actionable intelligence from the multivariate time-series data produced by IoT sensor networks. Among the available approaches, ensemble methods such as Random Forest have demonstrated robust generalization on noisy, high-dimensional sensor data, rendering them particularly suitable for field-deployed predictive maintenance applications [3].

This paper presents a comprehensive AI-driven predictive maintenance framework comprising four tightly integrated layers: (i) a hardware sensor array connected to an ESP32 microcontroller; (ii) a Firebase Realtime Database serving as the cloud data backbone; (iii) a Python-based machine learning pipeline employing a Random Forest classifier; and (iv) a Chart.js real-time visualization dashboard. The framework continuously monitors temperature, vibration, and electrical current, and generates risk-level predictions to guide maintenance decision-making.

The remainder of this paper is organized as follows. Section II reviews the pertinent literature. Section III defines the problem statement and enumerates research contributions. Section IV describes the system architecture. Section V presents the data flow and workflow. Section VI describes the algorithm and mathematical formulations. Section VII details the implementation. Section VIII reports experimental results. Section IX presents comparative analysis. Sections X–XIV address limitations, deployment, discussion, conclusion, and future work.

II. LITERATURE REVIEW

Substantial research effort has been directed toward IoTenabled predictive maintenance across diverse industrial domains. The field has evolved from simple thresholdbased alarm systems to sophisticated multi-sensor fusion frameworks integrating deep learning and edge computing. This section provides a structured review of existing approaches, categorized by sensing modality, computational architecture, and deployment context.

A. Vibration-Based Fault Detection

Vibration analysis has historically been the dominant sensing modality for rotating machinery fault detection. Kolok et al. [1] proposed a low-cost ESP32 and MEMS accelerometer system achieving 73% anomaly detection accuracy via RMS/FFT feature extraction and Isolation Forest anomaly scoring on small rotating machinery. While cost-effective, the 73% detection rate reveals a meaningful gap relative to supervised classifier approaches, attributable to the unsupervised nature of Isolation Forest, which requires no labeled training data but sacrifices discriminative power. IoT-integrated vibration analysis has similarly been applied to structural health monitoring of mechanical frames, where ML classifiers trained on frequency-domain features demonstrated reduced false-positive rates compared to single-threshold alarm logic [8].

The study reported in [9] combined ADXL335 three-axis vibration sensing with DS18B20 temperature acquisition on a NodeMCU ESP32, displaying processed data on both local LCD panels and cloud-hosted IoT dashboards. While this system demonstrates hardware integration competency, the absence of a machine learning inference layer limits its predictive capability to static threshold exceedance detection rather than probabilistic health classification.

B. Temperature and Current Sensing Approaches

Sabri and Aziz [2] developed an ESP32-based solution integrating DS18B20 temperature sensors and ADXL335 accelerometers for real-time three-phase motor monitoring, issuing threshold-based alerts via Blynk and LCD when temperature exceeded 55°C or vibration surpassed 0.5G. Similarly, [4] presented an integrated IoT condition monitoring system for AC induction motors, employing vibration and temperature sensing to detect faults and enable PdM. A consistent limitation across threshold-based approaches is their inability to detect gradual degradation that progresses through a MEDIUM-risk zone before reaching critical levels-precisely the scenario that machine learning-based classification addresses.

C. Machine Learning-Augmented Predictive Maintenance

Machine learning-centric approaches have significantly advanced state-of-the-art classification performance. A study presented at ICAST-ES [5] demonstrated integration of Random Forest and LSTM models with IoT sensor data, achieving prediction accuracies exceeding 90% in simulated environments. The study in [10] specifically applied accelerometer-based vibration data to ML-driven PdM for electrical motors and CNC machines, confirming that Random Forest consistently outperformed Support Vector Machine (SVM) baselines when feature engineering was applied to the raw timeseries input.

D. Cloud and Edge Computing Architectures

Cloud-based PdM platforms have enabled centralized data aggregation and analytics at scale. A comprehensive framework proposed in [7] leverages IIoT sensors measuring temperature, vibration, pressure, and current in conjunction with AI and edge computing to optimize maintenance scheduling and reduce decision latency. This work explicitly recognizes the trade-off between cloud-centralized inference and edge-deployed inference, advocating a tiered architecture in which lightweight anomaly detection runs at the edge while more computationally intensive prognostic models execute in the cloud.

Table 1: Comparative Overview of Representative PdM Systems

Ref.	Sensors	Platform	ML Method	Acc.	Cost
[1]	Vib.+Acoustic	ESP32	Isolation Forest	73%	Low
[2]	Temp.+Vib.	ESP32	Threshold	N/A	Low
[5]	Vib.,Temp.	Custom IoT	RF + LSTM	>90%	Med.
[7]	T+V+P+I	IIoT+Edge	AI/Edge	High	High
[10]	Vibration	Custom IoT	RF vs SVM	~88%	Med.
Proposed	Temp+Vib+I	ESP32+Firebase	Rand. Forest	94.3%	Low

III. PROBLEM STATEMENT AND RESEARCH CONTRIBUTIONS

A. Problem Statement

Despite significant progress in predictive maintenance research, a persistent gap exists between laboratoryvalidated high-accuracy systems and cost-effective deployable solutions suitable for small-to-medium enterprises (SMEs) and resource-constrained industrial environments. High-performing systems identified in the literature frequently rely on proprietary industrial sensor suites, licensed cloud analytics platforms, or computationally intensive deep learning models that demand GPU-accelerated inference infrastructure. These requirements impose substantial capital expenditure and operational complexity that renders many proposed frameworks impractical for widespread adoption.

B. Comparative Positioning

The principal contributions of this work are enumerated Table 1 summarizes key attributes of representative as follows:

existing systems against the proposed framework. A 1. A cost-effective, open-source predictive consistent observation across the reviewed literature is maintenance framework integrating temperature, that high-accuracy solutions often impose significant vibration, and electrical current sensing with a hardware complexity, proprietary cloud dependencies, or cloud-based analytics pipeline, deployable using large labeled training datasets. The present system widely available commercial off-the-shelf distinguishes itself by delivering competitive components costing less than USD 30 per classification accuracy using entirely open-source monitored node. components, a three-modality sensor fusion strategy, and a confidence-gated prediction mechanism not present in methodology that augments raw three-sensor the majority of reviewed systems.

2. A nine-dimensional feature engineering readings with rolling statistical descriptors and first-Threshold-based IoT monitoring systems, at the opposite end of the cost spectrum, lack the probabilistic reasoning necessary to distinguish gradual equipment degradation from transient sensor noise. The binary nature of threshold exceedance detection provides no earlywarning capability and generates high false-positive rates in environments with variable operational loads. Consequently, a clear technical need exists for a framework that (i) operates exclusively on commodity hardware and open-source software; (ii) performs multiclass probabilistic health classification rather than binary alarm triggering; (iii) provides a quantifiable earlywarning window prior to critical failure; and (iv) supports real-time remote monitoring through an intuitive operator dashboard. **B.** Research Contributions

order temporal derivatives, demonstrably improving Random Forest classification accuracy by 6.2 percentage points relative to raw-feature baselines.

3. A confidence-gated prediction mechanism that suppresses low-certainty model outputs, reducing false-positive HIGH-risk alerts to 2.3% across the full evaluation dataset.
4. Experimental validation demonstrating an 18.4second early-warning lead time prior to HIGH-

risk transitions, enabling actionable operator intervention within the sensor update cycle.
5. A structured comparative analysis against five representative predictive maintenance frameworks from the recent literature, contextualizing the proposed system's performance within the current state of the art.

IV. SYSTEM ARCHITECTURE

The proposed predictive maintenance framework is organized into four hierarchical and functionally independent layers. This modular design promotes component replaceability, supports parallel development of individual subsystems, and ensures clean separation of concerns across the full data lifecycle from physical sensing to operator-facing visualization.

A. Layer 1: Data Acquisition

The data acquisition layer constitutes the physical foundation of the framework. Three sensor modalities operate concurrently to capture a multidimensional representation of device operating state. Temperature monitoring detects thermal anomalies associated with excessive friction, cooling circuit degradation, or electrical resistance increases caused by winding insulation breakdown. Vibration analysis identifies mechanical imbalances, bearing race defects, gear tooth fatigue, and resonance amplification phenomena. Electrical current measurement reveals load variations caused by mechanical drag, coil short circuits, and power supply instability.

All sensor modules are interfaced directly with the GPIO and analog input pins of the ESP32 microcontroller. The ESP32 polls each sensor at the start of every five-second epoch, applies a lightweight hardware-level averaging filter (four consecutive ADC readings averaged per sample) to mitigate quantization noise, and formats the three readings into a timestamped JSON structure. NTP synchronization via Wi-Fi ensures temporal consistency across all records stored in the cloud database.

B. Layer 2: Data Transmission and Cloud Storage

The ESP32 leverages its integrated 802.11 b/g/n Wi-Fi transceiver to transmit structured JSON sensor payloads to the Firebase Realtime Database. Each payload contains the three sensor values, a Unix epoch timestamp, a device identifier string, and a sequence number for packet-loss detection. Transmission occurs over a TLS-encrypted HTTPS connection authenticated via a per-device JSON service account credential stored in ESP32 flash memory. Connection resilience is maintained through an automatic Wi-Fi reconnection routine and an exponential back-off retransmission strategy with a maximum of five retry attempts per epoch before the sample is discarded.

Firebase Realtime Database was selected as the cloud backbone for several practical reasons. Its WebSocketbased real-time synchronization protocol enables both the Python backend and the JavaScript dashboard to receive new data records within approximately 200 milliseconds of a write event without polling. Its security rules engine provides field-level access control, ensuring that dashboard clients can read but not write sensor data, while the ESP32 device tokens carry write-but-not-read authorization.

C. Layer 3: Data Processing and Machine Learning

The Python backend operates as a persistent event-driven listener registered on the Firebase data stream. Upon detection of a new sensor record write event, the backend retrieves the most recent N sensor observations (default N = 12, representing the trailing 60 seconds) and executes the preprocessing and inference pipeline. The pipeline proceeds through four sequential stages: outlier suppression, normalization, feature extraction, and Random Forest inference.

The Random Forest classifier returns a class label (LOW,

MEDIUM, or HIGH) and a three-element classprobability vector. The argmax probability constitutes the prediction confidence score. Predictions exceeding the configurable confidence threshold (default: 0.60) are written back to a dedicated Firebase predictions node for audit and retraining purposes. Predictions below the threshold are logged locally but not surfaced to the dashboard, preventing low-certainty outputs from triggering premature maintenance alerts.

D. Layer 4: Real-Time Visualization

The visualization layer comprises a single-page web application implemented in HTML5, CSS3, and vanilla JavaScript, leveraging Chart.js 4.x for interactive timeseries rendering. The dashboard

subscribes to both the Firebase sensor data node and the predictions node, registering onValue listeners that trigger chart data updates and risk indicator refreshes upon each new record. Three stacked line charts display the trailing 60second history of temperature, vibration magnitude, and current readings. A prominently displayed risk panel presents the current model prediction alongside a colourcoded severity badge-green for LOW, amber for MEDIUM, and red for HIGH-and the associated confidence percentage.

V. DATA FLOW AND SYSTEM WORKFLOW

The end-to-end data flow of the proposed framework follows a unidirectional pipeline from physical sensor acquisition through cloud storage, machine learning inference, and finally dashboard visualization.

A. Sensor Polling and Payload Construction (ESP32)

At the commencement of each five-second epoch, the ESP32 firmware executes a sensor polling subroutine. The temperature sensor is read via a one-wire protocol query returning a 12-bit digital value converted to degrees Celsius. The vibration sensor output is sampled via a 12-bit ADC channel; four successive samples are averaged to reduce quantization noise, and the result is converted to gravitational acceleration units (g) using the sensor's known sensitivity coefficient. The current sensor produces an analog voltage proportional to the sensed current; the ESP32 ADC reading is converted to RMS current in Amperes using the ACS712 sensitivity formula.

B. Secure Transmission to Firebase

The JSON payload is transmitted via an authenticated HTTPS POST request to the Firebase Realtime Database REST endpoint. The ESP32 firebase-arduino library manages TLS handshake, token refresh, and retry logic transparently. Upon successful acknowledgment from Firebase, the ESP32 increments the sequence counter and enters a low-power delay state for the remainder of the five-second cycle. On transmission failure after five retries, the payload is discarded and a fault indicator LED is toggled to alert local operators of connectivity loss.

C. Backend Event Processing

The Python backend maintains a persistent WebSocket connection to Firebase via the firebase-admin SDK. The onValue callback registered on the sensor data node fires within approximately 200–400 ms of each new record write. The callback appends the incoming record to an in-memory sliding window deque of length $N = 12$ and invokes the preprocessing pipeline. Upon completion of inference, the prediction record is written to the /predictions/{device_id}/latest Firebase node, overwriting the previous prediction. An append-only history is simultaneously maintained at /predictions/{device_id}/history for audit and retraining data collection.

D. Dashboard Rendering Cycle

The dashboard JavaScript runtime maintains two concurrent Firebase listeners: one on the sensor data node (for live chart updates) and one on the predictions latest node (for risk indicator updates). Chart.js updates are batched within a requestAnimationFrame callback to prevent render-thread blocking. The total perceived latency from sensor acquisition to dashboard update, inclusive of all transmission, processing, and rendering stages, averages 1.8 seconds under standard Wi-Fi conditions.

VI. ALGORITHM DESCRIPTION AND MATHEMATICAL FORMULATION

A. Preprocessing Pipeline

Let the raw sensor observation at time step t be defined as the vector:

$x_t = [T_t, V_t, I_t] \in \mathbb{R}^3$ where T_t denotes temperature (°C), V_t denotes vibration magnitude (g), and I_t denotes RMS current (A). For a sliding window of $N = 12$ consecutive observations, the window matrix is:

$X\_W = \{ x_{t-n+1}, \ldots, x_t \} \in \mathbb{R}^{(N\times 3)}$

Outlier suppression is applied independently to each sensor channel. For channel j, values exceeding three standard deviations from the window mean are replaced by the channel mean:

$x_t[j] = \mu\_W[j] \quad if\ |x_t[j] - \mu\_W[j]| > 3\sigma\_W[j]$

Min-max normalization is subsequently applied using parameters fitted on the historical training dataset:

$\hat{x}_t[j] = (x_t[j] - min\_j) / (max\_j - min\_j) \in [0,1]$

B. Feature Extraction

From the normalized window $\hat{X}\_W$, nine features are extracted to form the final feature vector $\varphi_t \in \mathbb{R}^9$. For each sensor channel $j \in \{T, V, I\}$, three features are

computed. The rolling mean captures the baseline level within the window:

$\mu\_j = (1/N) \sum_{i=1}^{N} \hat{x}\_i[j]$

The rolling standard deviation captures intra-window variability and instability:

$\sigma\_j = \sqrt{((1/(N-1)) \sum_{i=1}^{N} (\hat{x}\_i[j] - \mu\_j)^2)}$

The first-order temporal difference captures the rate of change between the most recent and preceding observation:

$\delta\_j = \hat{x}_t[j] - \hat{x}_{t-1}[j]$

The complete nine-dimensional feature vector is:

$\varphi_t = [\mu\_T, \sigma\_T, \delta\_T, \mu\_V, \sigma\_V, \delta\_V, \mu\_I, \sigma\_I, \delta\_I] \in \mathbb{R}^9$

C. Random Forest Classification

The Random Forest classifier [3] is an ensemble of B decision trees {h_1, h_2, ..., h_B}, where each tree h_b is trained on a bootstrap sample of the training data using a random feature subset of size $m = \lfloor \log_2(9) \rfloor + 1 = 4$ at each split node. The ensemble prediction is determined by majority vote:

$\hat{y} = \text{argmax}\_c \sum\_b^{B} \mathbb{T}(h\_b(\varphi_t) = c)$

The class probability estimate for class c, used as the confidence score, is:

$\hat{P}(y = c \mid \varphi_t) = (1/B) \sum\_b^{B} \mathbb{T}(h\_b(\varphi_t) = c)$

A prediction is committed to Firebase only when the confidence exceeds the threshold θ:

$\max\_c \hat{P}(y = c \mid \varphi_t) \geq \theta \quad (\theta = 0.60)$

Each decision tree is grown using the Gini impurity criterion. For a node containing samples from classes with proportions p_1, ..., p_K:

$Gini = 1 - \sum\_k^{K} p\_k^2$

D. Algorithm Pseudocode

Algorithm 1 summarizes the complete inference workflow executed by the Python backend upon receipt of each new sensor record.

Algorithm 1: Real-Time Predictive Maintenance Inference

```
Input:  Sensor record r = {T, V, I, timestamp}
Output: Risk label ŷ, confidence P^
1.  Append r to sliding window deque W (capacity N = 12)
2.  If |W| < N: return   ▷ defer until window is full
3.  For each channel j ∈ {T, V, I}:
    a. Apply IQR outlier winsorization to W[:,j]
    b. Apply min-max normalization using training stats
4.  Extract φ_t = [μ_T, σ_T, δ_T, μ_V, σ_V, δ_V, μ_I, σ_I, δ_I] 5.  Run Random Forest: ŷ, P^ ←
RF.predict(φ_t)
6.  If max(P^) < θ: log locally, return ▷ suppress
7.  Write {ŷ, P^, timestamp, φ_t} to Firebase /predictions/{id}
```

VII. IMPLEMENTATION

A. Hardware Configuration

The hardware subsystem is constructed around the Espressif ESP32 development board, chosen for its dualcore Xtensa LX6 processor (240 MHz), 520 KB SRAM, integrated 802.11 b/g/n Wi-Fi, and rich peripheral interface. Table 2 summarizes the sensor components and interfacing details.

Table 2: Hardware Component Specifications

Component	Module	Interface	Range	Parameter
Microcontroller	ESP32 DevKit v1	GPIO/ADC	-	Control & Tx
Temperature	DS18B20	1-Wire	-55 to 125 °C	Temp. (°C)
Vibration	ADXL335	Analog ADC	±3 g	Accel. (g)
Current	ACS712 (20A)	Analog ADC	0–20 A RMS	Current (A)

B. Software Stack

The firmware executed on the ESP32 is developed within the Arduino IDE (v2.3) using C++17. The firebasearduino-client-library v4.4 facilitates authenticated JSON writes. The Python 3.10 backend employs the firebase-admin SDK v6.2 for database interaction, scikitlearn v1.3 for model training and inference, pandas v2.1 for data manipulation, and NumPy v1.26 for numerical feature computation. The front-end dashboard is a singlepage application utilizing Chart.js v4.4 and the Firebase JavaScript SDK v9.x modular API.

C. Model Training Details

The Random Forest classifier is configured with B = 200 decision trees, a maximum depth of 15, and a minimum samples-per-leaf of 5. Gini impurity is the splitting criterion. The model achieves a 10-fold stratified crossvalidated accuracy of 94.3% on the full training dataset of 3,320 labeled observations. Feature

importance analysis identifies rolling standard deviation of vibration ($\sigma\_V = 0.231$) and rate-of-change of current ($\delta\_I = 0.194$) as the two most discriminative features, collectively contributing approximately 42.5% of predictive weight.

VIII. RESULTS AND EXPERIMENTAL EVALUATION

Experimental evaluation of the framework was conducted across three distinct operating scenarios on a test bench comprising a DC motor with attached load, instrumented with the three sensor modules described in Section VII-A. The total evaluation dataset contains 3,320 labeled observations: 1,560 LOW-risk, 990 MEDIUM-risk, and 770 HIGH-risk records. An 80/20 stratified train-test split was employed; the test set comprises 664 observations.

A. Classification Performance

Table 3 presents the per-class and macro-average precision, recall, F1-score, and support metrics obtained by the trained Random Forest model on the held-out test set.

Table 3: Classification Performance Metrics on Hold-Out Test Set

Risk Level	Precision	Recall	F1-Score	Support
LOW	0.96	0.97	0.97	312
MEDIUM	0.91	0.89	0.90	198
HIGH	0.94	0.95	0.95	154
Macro Avg.	0.94	0.94	0.94	664

B. Confusion Matrix Analysis

The confusion matrix for the test set is presented in Table

4. The matrix reveals that the predominant misclassification pattern is confusion between the MEDIUM and LOW classes (14 instances), which is expected given the inherent ambiguity at the boundary between gradual degradation and normal operation. HIGH-risk misclassification is minimal (8 instances), confirming that critical fault conditions are reliably distinguished from lower-severity states.

Table 4: Confusion Matrix on Hold-Out Test Set (Rows: Actual, Columns: Predicted)

Actual \ Predicted	LOW	MEDIUM	HIGH
LOW (312)	303	8	1
MEDIUM (198)	14	176	8
HIGH (154)	2	6	146

C. ROC Analysis and AUC

One-versus-rest ROC analysis was conducted for each class. The Area Under the Curve (AUC) values are presented in Table 5. AUC scores uniformly exceed 0.97 across all three classes, indicating excellent discriminative capability across varying decision thresholds. The HIGH-risk class achieves the highest AUC of 0.989, reflecting the strong separability of failure-indicative sensor signatures from the combined LOW and MEDIUM distributions.

Table 5: One-vs-Rest ROC AUC Scores

Risk Class	AUC (OvR)
LOW	0.974
MEDIUM	0.971
HIGH	0.989
Macro Average	0.978

D. System Latency and Early-Warning Performance

End-to-end latency averaged 1.8 seconds under standard Wi-Fi conditions, well within the five-second sensor update cycle. Firebase write/read round-trip contributed approximately 0.6 seconds; Python inference time per sample was consistently below 5 milliseconds. During stress testing, all 12 injected fault conditions were successfully detected, yielding a 100% fault detection rate for HIGH-risk transitions. The system issued a MEDIUM-risk prediction an average of 18.4 seconds prior to HIGH-risk onset, providing a meaningful earlywarning window. The false positive rate for HIGH-risk predictions was 2.3% across the full evaluation dataset.

IX. COMPARATIVE ANALYSIS

To contextualize the proposed framework's performance within the current state of the art, a structured comparative analysis was conducted against five representative predictive maintenance systems. The comparison encompasses four dimensions: classification accuracy, hardware cost, deployment complexity, and early-warning capability.

A. Accuracy Comparison

The proposed system's macro-average F1-score of 0.943 exceeds the 73% anomaly detection accuracy reported by Kolok et al. [1] by a margin of 20.3 percentage points. This improvement is attributable to two factors: the use of a supervised multi-class Random Forest classifier trained on labeled fault data versus the unsupervised Isolation Forest approach of [1], and the three-modality sensor fusion strategy that provides richer discriminative features. The proposed system is comparable to the >90% accuracy reported by [5], with the advantage of substantially lower hardware and infrastructure cost and a simpler, more interpretable model architecture.

B. Cost and Deployment Analysis

The per-node hardware cost of the proposed framework is estimated at under USD 30, comprising the ESP32 development board (~USD 5), DS18B20 temperature sensor (~USD 2), ADXL335 accelerometer module (~USD 4), ACS712 current sensor (~USD 3), and ancillary passive components and PCB (~USD 16). This compares favorably against the IIoT edge-computing architecture of [7], which requires dedicated edge gateway hardware and typically incurs per-node costs an order of magnitude higher. The exclusive use of opensource software eliminates licensing costs entirely.

C. Summary Comparison

Table 6 presents a quantitative summary of the proposed system compared to the five reviewed frameworks across all evaluated dimensions.

Table 6: Quantitative Comparative Analysis: ProposedSystem vs. Existing Frameworks

System	F1/Acc.	Sensors	Cost (USD)	Early-Warn.	Complexity
[1] 2025	73%	Vib.+Acous.	~15	No	Low
[2] 2025	N/A	Temp.+Vib.	~20	No	Low
[5] 2025	>90%	Multi	>100	Partial	Medium
[7] 2025	High	T+V+P+I	>200	Yes	High
[10] 2023	~88%	Vibration	~80	No	Medium
Proposed	94.3%	Temp+Vib+I	<30	18.4 s	Low

X. LIMITATIONS OF THE PROPOSED SYSTEM

A transparent delineation of the current system's limitations is essential for accurate assessment of its applicability and for guiding future development. The following limitations have been identified through both experimental evaluation and critical design review.

A. Dataset Scope and Generalizability

The training and evaluation dataset was generated from a single DC motor type operating under controlled laboratory conditions with three deliberately induced fault modes. The Random Forest model's generalization to different equipment classes (e.g., three-phase induction motors, pneumatic actuators, hydraulic pumps), alternative fault morphologies (e.g., stator winding faults, gear tooth fatigue fractures), or realworld environmental variability has not been validated. Cross-equipment generalization will require substantially larger and more heterogeneous training corpora, potentially necessitating domain adaptation or transfer learning approaches.

B. Sensor Modality Coverage

The three sensing modalities employed-temperature, vibration, and current-do not capture all failure mechanisms relevant to complex industrial machinery. Acoustic emission sensing can detect early-stage subsurface crack initiation preceding macro-scale vibration signatures. Humidity monitoring is critical for environments where moisture ingress causes corrosion or electrical insulation degradation. Pressure sensing is essential for hydraulic and pneumatic systems.

C. Cloud Dependency and Connectivity

The current architecture requires continuous Wi-Fi connectivity for operation. Intermittent or absent network access results in prediction deferral and potential missed fault events. The Firebase Realtime Database dependency introduces a single point of failure; service outages or quota exhaustion on the Firebase free tier would halt data ingestion entirely. For highcriticality industrial applications, a local edge inference fallback and an offline data buffering strategy with deferred cloud synchronization would be mandatory reliability requirements.

D. Firebase Free-Tier Scalability

The Firebase Spark (free) plan imposes limits of 1 GB stored data, 100 simultaneous connections, and 10 GB monthly download. These constraints are manageable for small deployments but would be exceeded by a fleet of more than approximately 20 continuously monitored devices. Migration to a paid Firebase Blaze plan or an alternative time-series database architecture would be required for larger deployments.

E. Model Interpretability

While Random Forest provides feature importance scores, the model itself is not inherently interpretable at the individual prediction level. Maintenance engineers typically require not only a risk classification but also a human-readable explanation of which sensor channel and which feature contribution triggered a prediction. Integration of model explanation techniques such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) would substantially improve the framework's operational utility in safety-critical environments.

XI. DEPLOYMENT AND SCALABILITY CONSIDERATIONS

Transitioning the proposed framework from laboratory prototype to industrial deployment requires careful consideration of architectural scalability, network topology, data governance, and operational resilience.

A. Multi-Device Fleet Architecture

In a fleet deployment scenario, each monitored device is assigned a unique device UUID used as the Firebase database key prefix and as the ESP32 firmware configuration parameter. The Python backend is refactored from a single-device listener into a multitenant microservice that registers independent event listeners for each enrolled device. Horizontal scalability of the backend is achieved through stateless containerization (Docker) and orchestration via Kubernetes, enabling dynamic scaling of inference workers in response to fleet size and event rate.

B. Edge Computing Integration

For deployments with stringent latency requirements or unreliable network connectivity, a tiered edge-cloud architecture is recommended. A local edge gateway (e.g., Raspberry Pi 4 or NVIDIA Jetson Nano) receives sensor data from ESP32 nodes over a local Wi-Fi or Ethernet LAN, executes a lightweight quantized version of the

Random Forest model using ONNX Runtime for sub100 ms inference, and forwards predictions to the cloud asynchronously. This architecture eliminates cloud round-trip latency from the critical prediction path and enables continued operation during WAN outages.

C. Database Scalability

For large-scale deployments exceeding 100 monitored devices, the Firebase Realtime Database architecture should be replaced by or augmented with a purpose-built time-series database. InfluxDB OSS or TimescaleDB (PostgreSQL extension) provide optimized write throughput, time-range query performance, and native data retention policies essential for long-running industrial monitoring deployments. A Kafka message broker can be interposed between the ESP32 transmission layer and the database to provide ingestion buffering, back-pressure management, and multiconsumer fan-out.

D. Security and Data Governance

Industrial deployments must satisfy operational technology (OT) security requirements including device authentication, encrypted data-in-transit and data-at-rest, access control with role-based authorization, and audit logging. ESP32 devices should employ certificate-based mutual TLS (mTLS) authentication rather than preshared token credentials. A data governance policy must define retention periods, anonymization requirements, and compliance with applicable industrial cybersecurity standards such as IEC 62443.

E. Maintenance Workflow Integration

For maximum operational value, the predictive maintenance framework should integrate with existing computerized maintenance management systems (CMMS) such as IBM Maximo or SAP Plant Maintenance. Integration can be realized through a webhook adapter that converts HIGH-risk prediction events into CMMS work order creation API calls, automatically populating the work order with the device identifier, predicted fault type, confidence score, and relevant sensor telemetry context. This integration closes the loop between automated fault prediction and human maintenance execution, providing measurable ROI in terms of reduced mean time to repair (MTTR) and decreased unplanned downtime hours.

XII. DISCUSSION

A. Framework Performance and Design Choices

The experimental results validate the core hypothesis of this work: that an integrated stack of commodity IoT hardware, cloud data infrastructure, and ensemble machine learning can deliver reliable real-time predictive maintenance at low deployment cost. The 94.3% macroaverage F1-score obtained by the Random Forest classifier is competitive with state-of-the-art results and meaningfully surpasses threshold-based approaches that provide no early-warning capability. The multi-sensor fusion strategy is a key contributor to classification robustness; feature importance analysis confirms that no single sensor dominates the model, indicating that each modality contributes complementary discriminative information.

The confidence-gated prediction mechanism represents a practically significant design enhancement absent from the majority of reviewed systems. By suppressing predictions below the 0.60 confidence threshold, the framework reduces operationally disruptive falsepositive HIGH-risk alerts while preserving the sensitivity needed to detect genuine fault progression.

B. Industry 4.0 and Industrial IoT Context

The proposed framework is explicitly aligned with the Industry 4.0 paradigm, which defines the fourth industrial revolution as the integration of cyber-physical systems, IoT connectivity, cloud computing, and data analytics into industrial production environments. The nine defining characteristics of Industry 4.0interoperability, virtualization, decentralization, realtime capability, service orientation, modularity, and others-are collectively addressed by the framework's architecture. The ESP32 sensor nodes constitute the cyber-physical interface layer; Firebase provides the interoperable cloud data substrate; the Python ML backend delivers real-time analytical capability; and the dashboard provides the human-machine interface through which decentralized production floor data is made actionable.

Within the Industrial Internet of Things (IIoT) landscape, the proposed framework occupies the category of condition-based monitoring (CBM) systems enhanced with machine learning-based prognostics. Industry analysts estimate that predictive maintenance enabled by IIoT sensor data can reduce equipment downtime by 30– 50%, extend asset lifecycles by 20–40%, and decrease maintenance costs by 10–25% compared to time-based preventive maintenance schedules.

The framework's design also anticipates the evolution toward digital twin architectures, in which a virtual model of each physical device is continuously updated with real-time sensor telemetry and used for simulationbased failure prognosis. The Firebase data backend and structured prediction logging infrastructure provide the data foundation upon which a device digital twin could be incrementally built, suggesting a natural evolutionary pathway within the broader Industry 4.0 technology stack.

C. Comparison with Deep Learning Alternatives

While deep learning approaches, particularly LSTM and Transformer-based sequence models, have demonstrated high accuracy on temporal fault detection tasks, their adoption in resource-constrained embedded and cloudlimited environments entails significant practical tradeoffs. LSTM models require substantially more training data to achieve generalization, are computationally more expensive to train, and are less interpretable than Random Forest ensembles. For the target deployment context of this work-small-to-medium scale, limited training data, interpretability requirements-the Random Forest classifier represents the optimal balance of accuracy, training data efficiency, inference speed, and model interpretability. Hybrid architectures combining LSTM temporal encoding with Random Forest classification heads, as explored in [5], merit investigation as dataset sizes grow.

XIII. CONCLUSION

This paper has presented a comprehensive AI-driven predictive maintenance framework that cohesively integrates IoT sensing, cloud data management, machine learning classification, confidence-gated prediction, and real-time visualization into a unified, cost-effective solution. The framework monitors device health through temperature, vibration, and electrical current sensors connected to an ESP32 microcontroller, streaming data to Firebase Realtime Database at five-second intervals. A Python backend applies a nine-feature Random Forest classifier, achieving a macro-average F1-score of 0.943 and an AUC of 0.978 across three risk classes. A Chart.js dashboard provides operators with continuous situational awareness and early-warning capability averaging 18.4 seconds prior to HIGH-risk transitions. The paper has further contributed a formal problem statement, a mathematical treatment of the feature extraction and classification pipeline, an expanded literature review, a new algorithm description with pseudocode, quantitative comparative analysis against five existing systems, a structured limitations analysis, and an engineering roadmap for industrial-scale deployment within the Industry 4.0 and IIoT context.

The system demonstrates that high-fidelity, multi-class predictive maintenance is achievable with open-source tooling and under USD 30 of hardware per monitored node, substantially reducing the barrier to adoption for SMEs and resource-constrained industrial environments. The measured early-warning lead time provides a meaningful operational window for preventive intervention, with direct implications for reducing unplanned downtime and extending equipment service life.

XIV. FUTURE WORK

Several directions are identified for extending and strengthening the proposed framework:

- Integration of acoustic emission and humidity sensing modalities to extend fault coverage to moisture-induced degradation and early-stage subsurface crack initiation, accompanied by a systematic feature importance re-evaluation across the expanded sensor set.
- Investigation of Transformer-based sequence models and Temporal Convolutional Networks (TCN) for improved capture of long-range temporal dependencies in sensor time-series, with a particular focus on hybrid architectures combining deep sequence encoding with Random Forest classification heads.
- Development of an edge inference pipeline using TensorFlow Lite Micro or ONNX Runtime on ESP32-S3 or Raspberry Pi hardware to eliminate cloud round-trip latency from the critical prediction path and enable offline fault detection during network outages.
- Integration of SHAP explainability analysis into the prediction pipeline to provide maintenance engineers with sensor-level attribution for each risk classification, improving trust, auditability, and regulatory compliance in safety-critical deployments.
- Large-scale field validation across multiple equipment types, fault modes, and industrial environments to quantify framework generalizability and generate sufficiently large and diverse training datasets for fine-tuned domainspecific model variants.
- Development of a CMMS integration adapter enabling automatic work order creation from HIGH-risk predictions, enabling quantitative ROI measurement in terms of MTTR and downtime reduction

REFERENCES

[1] P. Kolok, M. Hodoň, P. Ševčík, L. Hotz, and N. Remy, "Low-Cost IoT-Based Predictive Maintenance Using Vibration," *Sensors*, vol. 25, no. 21, p. 6610, Oct. 2025.

[2] A. S. A. Sabri and R. Aziz, "Motor Vibration and Temperature Detector by Using IoT," *Evol. in Electrical and Electronic Engineering*, vol. 6, no. 2, pp. 347–355, 2025.

[3] "Design and Implementation of an IoT-Based Electric Motor Vibration and Temperature Disruption Handling System," 2024. [Online]. Available: https://www.academia.edu/114230203

[4] "IoT-based health monitoring and fault detection of industrial AC induction motor for efficient predictive maintenance," *Measurement and* Control, 2024. doi: 10.1177/00202940241231473.

[5] "Implementation of Predictive Maintenance using IoT and Machine Learning," in *Proc. ICAST-ES*, 2025.