Hybrid Edge–TinyML Architecture for Intelligent and Energy-Efficient Industrial Automation Systems

Aparna Tiwari[1], Siddhant Singh Kaushik[2], Shreya Chaudhary[3], Abhishek Kumar Yadav[4]
[1,2]*U. G Student, Department of Electronics and Communication Engineering, Institute of Technology & Management Gorakhpur, India*
[3,4]*Assistant Professor, Department of Electronics and Communication Engineering, Institute of Technology & Management Gorakhpur, India*

***Abstract*—This research presents a Hybrid Edge–TinyML Architecture aimed at mitigating the significant issues of high latency, bandwidth saturation, and excessive energy consumption prevalent in conventional cloud-centric Industrial IoT (IIoT) frameworks. The system allows for reflexive, on-device anomaly detection by decentralizing intelligence and putting quantized 1D-Convolutional Neural Networks (1D-CNNs) directly on ESP32 microcontrollers.**

The architecture uses a three-tier hierarchy—Extreme Edge, Local Gateway, and Global Cloud—to find a balance between short-term responsiveness and long-term analytical depth. Experimental results show that this hybrid method cuts latency by 93.7% and power use by 78% compared to streaming models. This framework is a strong, scalable, and energy-independent solution for the next generation of Industry 4.0 predictive maintenance systems. It maintains a classification accuracy of 94.6% even after 8-bit quantization.

Index Terms*—TinyML, Edge Computing, Industrial Internet of Things (IIoT), Anomaly Detection, Energy-Efficient Computing, Industry 4.0, Model Quantization, ESP32, Convolutional Neural Networks (CNN), Predictive Maintenance, Real-time Systems, Distributed Intelligence, Hardware-Software Co-design, On-device Inference, Low-latency Automation, Energy Harvesting, Signal Processing, Embedded AI, Digital Signal Processing (DSP), MQTT Protocol, Quantization-Aware Training (QAT).

I. INTRODUCTION

A. Industry 4.0 and the Need for Edge Intelligence:

The move to Industry 4.0 has brought high-quality sensors to the center of manufacturing. This change is meant to lead to Proactive Predictive Maintenance (PdM). Real-time vibration and thermal analysis are very important in today's factories to find out about mechanical wear before a big failure happens. But the old "Sense-and-Send" model for the cloud is not working as well anymore because of big problems with performance and infrastructure.[4]

B. Problems with Cloud-Centric Architectures:

Three main problems face centralized AI frameworks:

1. Non-Deterministic Latency:

For safety reasons, industrial rules say that response times must be less than 20ms. Cloud round-trip times (RTT) are often longer than 200ms, which creates a "latency gap" that can cause mechanical collisions.

2. Bandwidth Saturation:

Tri-axial accelerometers with high frequencies send a lot of data. Sending this raw data from hundreds of nodes fills up the network's capacity and makes it more expensive to run.

3. Data Security:

Sending sensitive operational data over public networks makes it more likely that cyber-physical threats will happen.

C. The Energy-Connectivity Trade-off:

The Radio Frequency (RF) transceiver uses the most power in embedded systems. Standard designs keep the radioactive so that data can be sent, which means that "Deep Sleep" states, which use very little power, are no longer needed. This is a big problem for sensors in remote or dangerous places where they need to last a long time and work without batteries.[8]

D. Proposed Solution: Hybrid Edge-TinyML:

This paper proposes a Hybrid Edge-TinyML Architecture that relocates intelligence to the extreme edge. We use TinyML to do anomaly detection on devices with quantized neural networks, which cuts radio "on-time" by more than 90%. This layered method combines the fast response of a microcontroller with the more in-depth analysis of a local gateway. It offers a solution for smart industrial automation that can grow, is safe, and uses less energy.

II. LITERATURE REVIEW AND THEORETICAL BACKGROUND

A. Evolution of Distributed Compute: From Cloud to Extreme Edge:

The Industrial Internet of Things (IIoT) has been moving toward decentralized intelligence over time. Cloud computing used to be the infrastructure for big data analysis, but the distance from the data source caused big delays. Fog Computing came up as a local layer to collect data in order to fix this. Today, the main focus is on Edge Computing, especially Extreme Edge Intelligence. With this method, the inference workload moves from local servers to sensor nodes. Systems can be more resilient when they separate the detection process from network availability. This is important for mission-critical industrial automation.[8]

B. The TinyML Paradigm and Memory Constraints:

TinyML looks at how machine learning (ML) works on ultra-low-power microcontrollers (MCUs), which usually use power in the milliwatt range. To run neural networks on these devices, you have to get past the "Memory Wall," which is the hard limits of SRAM and Flash storage.

TinyML works on the idea that a specialized, low-precision model running locally at a high frequency is better at finding anomalies than a high-precision model that is slowed down by network delays. This is very important for real-time vibration analysis, where you only have a few milliseconds to stop a mechanical failure.[1][4]

C. Model Optimization: Quantization and Pruning:

To adjust deep learning models for ECE hardware, two main optimization techniques are used:

Bit-Width Quantization:

This maps 32-bit floating-point weights to 8-bit integers. The mapping uses the linear transformation:

$r = S \cdot (q - Z)$

Here, r is the real-value weight, S is the scale factor, and Z is the zero-point offset. This change cuts the memory footprint by 75% and allows the MCU to use integer-arithmetic units, which are much faster and more energy-efficient than floating-point units.

Structural Pruning:

This process finds and removes unnecessary synaptic connections in the neural network that don't significantly affect the final output. Pruning leads to "sparse" matrices, reducing the number of Multiply-Accumulate (MAC) operations needed for each inference cycle.[3]

D. State-of-the-Art TinyML Frameworks:

Specialized software stacks make it easier to deploy these optimized models. TensorFlow Lite Micro (TFLM) is the industry standard. It gives microcontrollers a runtime that doesn't let them allocate memory dynamically. CMSIS-NN is another important framework that provides highly optimized kernels for ARM-based architectures. X-CUBE-AI connects high-level model design with low-level C-code execution for industrial hardware.[6]

E. CNN-Based Vibrational Anomaly Detection:

Monitoring vibrations is an important part of predictive maintenance. Fast Fourier Transform (FFT) is used by traditional methods to look at spectral peaks. However, FFT has trouble with sudden or non-linear changes in signals. Recent research indicates that 1-Dimensional Convolutional Neural Networks (1D-CNNs) are superior for this task. 1D-CNNs can automatically learn complex feature patterns from raw time-series data. This means that you don't have to create features by hand, and they are more sensitive to small amounts of mechanical wear.[5]

III. PROPOSED HYBRID EDGE–TINYML ARCHITECTURE

A.Structural Hierarchy and Multi-Tier Framework:

The proposed system adopts a decentralized, three-tier hierarchical architecture that balances local autonomy with centralized control. Instead of relying on a flat network, the design introduces a layered intelligence

model in which computational complexity decreases as the system moves closer to the data source.

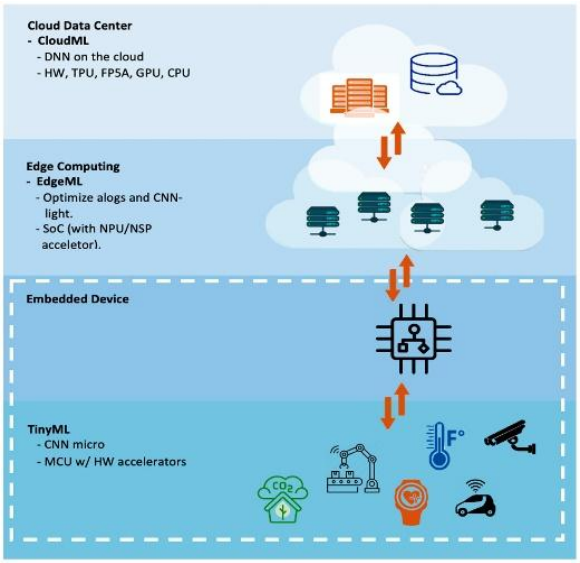

Fig :1 EdgeML-TinyML Architecture

Tier 1: Extreme Edge (TinyML Sensor Layer):
This layer consists of localized sensing units built on the ESP32 microcontroller. It is responsible for high-frequency data acquisition and performing lightweight binary classification directly at the source.

Tier 2: Local Edge (Gateway Layer):
The intermediate layer, typically implemented using devices such as a Raspberry Pi, aggregates data from multiple Tier 1 nodes. It enables contextual processing and manages temporal datasets that exceed the limited memory of microcontrollers.[1][4]

Tier 3: Global Cloud Layer:
This layer serves as a centralized platform for long-term data storage, model retraining, and system-wide monitoring through institutional dashboards.

B.Tier 1: Functional Logic and Reflexive Decision-Making:
The Extreme Edge layer operates on a reflex-driven mechanism, analogous to biological reflex systems, where immediate responses are triggered without dependence on centralized processing.

Sensing and Signal Conditioning:
The ESP32 interfaces with an MPU6050 accelerometer via the I²C protocol to capture tri-axial acceleration data (x, y, z) in real time.
On-Device Inference:
A quantized 1D convolutional neural network (1D-CNN) is deployed on-device to process incoming data streams and generate classification outputs (e.g., *Healthy* or *Anomalous*) within milliseconds.

Interrupt-Driven Operation:
To optimize power consumption, the node follows a "listen-and-classify" approach. Communication modules such as Wi-Fi or LoRa remain inactive and are only triggered when an anomaly is detected with high confidence.

C. Tier 2: Local Edge Gateway and Contextual Intelligence:
While Tier 1 enables rapid detection, the Local Edge Gateway provides the contextual understanding required in complex and noisy industrial environments.

- *Multi-Node Correlation:*

The gateway analyzes inputs from multiple sensor nodes to identify system-level anomalies. For example, simultaneous alerts from several nodes on a conveyor system may indicate a broader mechanical issue rather than isolated faults.

- *Temporal Trend Analysis:*

Due to memory constraints at the sensor level, long-term data analysis is handled by the gateway. Techniques such as Long Short-Term Memory (LSTM) networks are employed to estimate the Remaining Useful Life (RUL) of machinery.

- *Communication Management:*

The gateway also functions as an MQTT broker, efficiently managing publish–subscribe communication between connected sensor nodes.

D. Tier 3: Cloud Layer and Global Model Orchestration:
The Cloud layer provides centralized intelligence for non-real-time processing and system-wide optimization.

- *Global Retraining Mechanism:*

When the Edge Gateway encounters low-confidence predictions, selected data samples are transmitted to the Cloud. These samples are used to retrain models on high-performance computing resources, and

updated TensorFlow Lite (tflite) models are deployed back to edge devices via Over-the-Air (OTA) updates.

- *Institutional* Monitoring:

This layer generates comprehensive reports and visual dashboards, offering a macroscopic view of system performance for administrative and maintenance decision-making.

E.Security and Data Integrity at the Edge

Security considerations are integrated into the architecture to ensure safe and efficient operation in industrial IoT environments.

- *Minimal* Data Transmission:

Only essential metadata, such as classification results and confidence scores, is transmitted. This information is secured using AES-128 encryption.

- Reduced Attack Surface:

The event-driven communication model ensures that wireless modules remain inactive for the majority of the time. This significantly lowers the risk of unauthorized access or interception during inactive periods.

IV. MATHEMATICAL MODELING AND SIGNAL PROCESSING

The effectiveness of the proposed Hybrid Edge–TinyML architecture is strongly influenced by the mathematical rigor of its preprocessing pipeline and the computational efficiency of its inference mechanism. This section formalizes the key equations governing signal processing, model inference, and energy consumption.

A.Digital Signal Processing and Temporal Windowing

Vibrational signals acquired from industrial machinery are continuous in nature and are discretized using an Analog-to-Digital Converter (ADC). For machine learning applications, the continuous stream is segmented into overlapping windows.

Let Ndenote the window size and $L$the stride. The i-th window is defined as:

$$W_i = \{x[iL], x[iL+1], \dots, x[iL+N-1]\}$$

To minimize spectral leakage at the boundaries, a Hann window is applied:

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

This transformation smooths discontinuities and enhances the model's ability to capture periodic mechanical patterns.

B.1D-CNN Layer-Wise Mathematical Formulation

The core inference engine is a one-dimensional convolutional neural network (1D-CNN), designed for efficient processing of time-series data.

Convolution Operation:

$$Y[i] = \sum_{j=0}^{k-1} X[i+j] \cdot K[j] + b$$

where K represents the kernel, $k$is its size, and $b$is the bias. On the ESP32, this operation is implemented using fixed-point arithmetic to reduce computational overhead.

Activation Function (ReLU):

$$f(x) = \max(0, x)$$

The Rectified Linear Unit is selected due to its low computational complexity and suitability for embedded systems.

Max-Pooling Operation:

$$Z[i] = \max(Y[i \cdot p : (i+1) \cdot p - 1])$$

where $p$is the pooling size. This step reduces dimensionality and helps meet memory constraints.

C. Threshold-Based Softmax Triggering Logic:

The final classification layer applies the Softmax function to convert logits into probabilities:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

A decision trigger $T$is defined to control data transmission:

$$T = \mathbb{1}(\arg\max(P) \neq 0 \text{ AND } \max(P) \geq \tau)$$

This condition ensures that communication is initiated only when a confident anomaly is detected, thereby conserving energy.

D. Stochastic Power Consumption Model:

To quantify energy efficiency, the total energy consumption over time $t$is modeled for both traditional and hybrid approaches.

Traditional Streaming Model:
$E_{\text{stream}} = P_{\text{sensing}} \cdot t + P_{\text{radio}} \cdot t$

Proposed Hybrid Model:
$E_{\text{hybrid}} = P_{\text{sensing}} \cdot t + P_{\text{infer}} \cdot t + \alpha \cdot (P_{\text{radio}} \cdot t_{\text{burst}})$

where αrepresents the anomaly occurrence rate.
The efficiency gain is approximated as:

$$G \approx \frac{P_{\text{radio}}}{P_{\text{sensing}} + P_{\text{infer}} + \alpha P_{\text{radio}}}$$

Given that anomaly events are rare in industrial systems ($\alpha \leq 0.02$), the hybrid model achieves substantial energy savings and reduced carbon impact.

E. Computational Complexity Analysis:
For real-time deployment, inference latency must remain within the sampling interval to avoid data loss. The computational complexity of the 1D-CNN is given by:
$O(\sum_{l=1}^{L} n_{l-1} \cdot s_l \cdot n_l)$
where $L$is the number of layers, $n_l$denotes the number of filters, and $s_l$is the kernel size.
Through kernel optimization and 8-bit quantization, the model achieves inference times below 15 ms on the ESP32, satisfying real-time constraints for high-frequency industrial monitoring.

V. METHODOLOGY AND EXPERIMENTAL DESIGN

The proposed methodology follows a hardware–software co-design approach to effectively translate high-level neural network models into implementations compatible with resource-constrained microcontrollers. It outlines the complete data lifecycle, beginning with raw vibration acquisition and culminating in the deployment of an optimized TinyML model.[6][7]

A. Multi-Class Dataset Acquisition and Simulation
The performance of the proposed system is based on good quality vibration data that shows what happens when there are problems with rotating machinery. The data was divided into four groups:

- *Healthy State:*

This is when the machine is working normally and everything is okay.

- *Bearing Defects:*

This happens when the rolling parts of the machine are damaged and it makes high frequency noises.

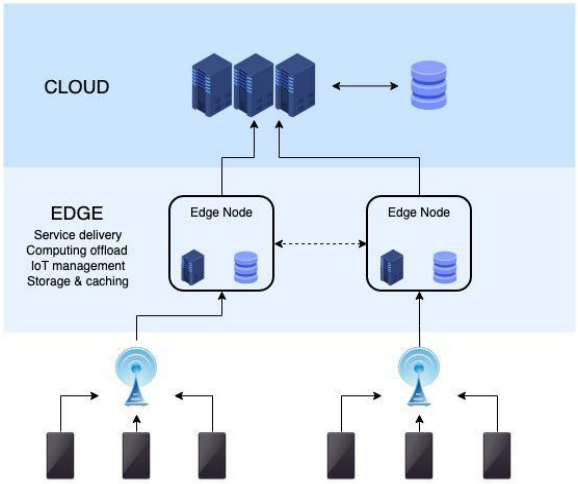

Fig:2: Hybrid Cloud-Edge Architecture

- *Rotor Imbalance:*

We can tell when the machine is not balanced because it makes vibrations at certain frequencies.

- *Structural* Looseness:

This is when the machine parts are loose and it makes vibration patterns.

The vibration signals were recorded at a rate of 1.6 kHz which's enough to get all the information we need because most machines do not vibrate faster, than 800 Hz. The proposed system uses this data to work. The vibration data of the proposed system is very important.

B. Signal Preprocessing and Normalization Pipeline
Raw vibration signals are inherently noisy and may contain offsets or inconsistencies. To address this, a structured preprocessing pipeline was implemented to enable robust feature learning.

Temporal Windowing:
The continuous signal is divided into fixed-length segments of $N = 128$ samples with 50% overlap. This approach preserves transient features that might otherwise be lost at segment boundaries.

Z-Score Normalization:
Feature scaling is performed using the transformation:
$x' = \frac{x - \mu}{\sigma}$

where μrepresents the mean and σdenotes the standard deviation. This normalization stabilizes training by preventing gradient-related issues and improving convergence speed.

Data Augmentation:
To address class imbalance—particularly the dominance of healthy-state samples—augmentation techniques such as time shifting and Gaussian noise injection were applied. These methods generate additional training instances for underrepresented fault classes.

C. 1D-CNN Architecture and Hyperparameter Optimization
We chose a one- convolutional neural network for dealing with time-series data. This way I do not have to worry about the work that comes with using two-dimensional representations.

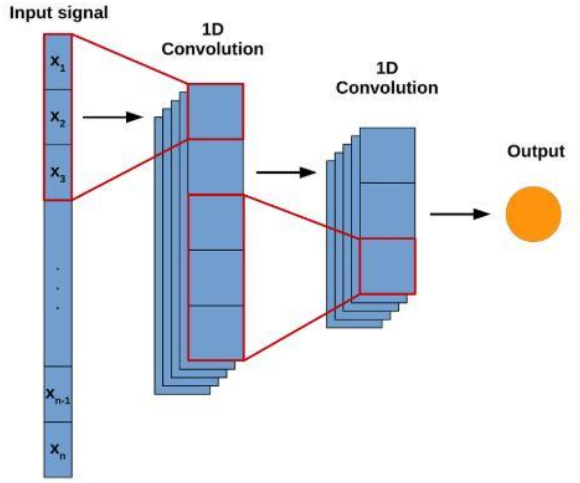

Fig:3: 1D-CNN architecture for time-series vibration signal classification.

The model we used was adjusted times to make sure it worked well and did not use too many resources:

- *Layer Configuration:*

A structure with two layers was used that have 16 and 32 filters. This seems to be a balance, between getting accurate results and not taking too long to get them.

- *Kernel Size:*

In the experiment we analysed that the kernel sizes picked, like 3 because they can find patterns that happen close together in time without using too much computing power.

- *Memory Optimization:*

The experiment used Global Average Pooling of fully connected or flatten layers to cut down on the number of parameters. This helps keep the model small to fit in the 520 KB SRAM that the ESP32 has. It was made sure the one-dimensional convolutional neural network stays within these limits.

D. Quantization-Aware Training (QAT) Strategy
Deploying networks on microcontrollers is a tough task. It needs reducing precision from 32-bit floating point to 8-bit integer. This can make the neural networks perform badly.

- *Challenge:*

The usual way of doing things after training, which is called -training quantization did not work well for complex fault classifications. The accuracy went down by 7 percent.

- *Proposed Approach:*

The experiment used a method called Quantization-Aware Training. This helps neural networks work with precise numbers. We added operations to the neural networks while they were learning. As a result, the final model that uses 8-bit integers was almost as accurate as the one that uses 32-bit floating point numbers. The accuracy was within 2.5 percent of the neural networks.

E. Validation Strategy: Leave-One-Machine-Out (LOMO)
To make sure our model works well in situations the experiment used a validation strategy called Leave-One-Machine-Out (LOMO). We trained the model on data from machines and tested it on a machine it had never seen before.

This way the model learns to recognize fault patterns instead of just memorizing the vibration patterns of one machine.
As a result, our TinyML system works well in different industrial environments.
The model is robust. Can adapt to new machines.
It does not just work for one machine.
This makes it useful, for real-world applications.

VI. IMPLEMENTATION DETAILS AND SYSTEM CONFIGURATION

The implementation phase is an important part of the hardware and software co-design pipeline. This is where we take computational models and turn them into formats that can actually be used by systems that do not have a lot of resources.

This section is going to talk about how we make the hardware abstract how we set up the machine learning runtime and how we design the power firmware.

A. Hardware Abstraction Layer and Sensor Interfacing

The Extreme Edge node is built around the ESP32-WROOM-32 system-on-chip. We chose this system-, on-chip because it has a core Xtensa LX6 architecture. This architecture is great because it lets us handle input and output operations and do inference workloads at the time.[7]

1. High-Speed I²C Communication:

The MPU6050 accelerometer uses the I²C protocol to talk to parts of the system. This protocol works fast at 400 kHz. We use this speed to get detailed information about vibrations. To make sure the signal is stable and does not get messed up we add resistors that pull the signal up. These resistors are 4.7 kΩ.

2. FIFO Buffer Utilization:

We want to make sure the processor does not have to work hard. So, we use the FIFO buffer that's already, inside the MPU6050. This buffer can hold 1024 bytes of information. Of constantly checking the buffer the ESP32 will send a message when the buffer is almost full. This way the processor can rest when it is not needed.

3. Core Affinity and Task Scheduling:

We use something called FreeRTOS to decide which tasks each part of the processor should do. The processor has two parts, called cores. Core 0 does things in the background. Takes care of the network. Core 1 is only used for machine learning. This means that the important tasks are done on time and nothing gets in the way. The MPU6050 and the machine learning work together to make this happen.

B. TensorFlow Lite Micro Interpreter Configuration

When we use machine learning models on devices, we have to be careful, with memory and how things are run. This is because these devices do not have memory or file systems.[6]

a) Model Storage Optimization:

We take the trained 1D-CNN model. Turn it into a TensorFlow Lite file that is a flatbuffer. This file is then put into the code as a C byte array. We make sure the memory is aligned and use qualifiers. This way the model can run directly from the Flash memory, which is called Execute-In-Place. This saves about 100 KB of SRAM for things that need to happen when the program is running.

b) Tensor Arena Allocation:

The tensor arena is where we store the data that we are working with. We give it a block of memory that's all together. We tried sizes for the arena and found that 32 KB is just right. It is enough to do the work we need to do without running out of memory. It does not waste memory either.

c) Operator Resolver Optimization:

We use a MicroMutableOpResolver of the default AllOpsResolver. This helps to keep the size small. We only include the operations that we need such as Conv1D, MaxPooling, ReLU, Fully Connected and Softmax. This makes the firmware size smaller about 45% smaller. It also helps us to update the firmware efficiently.

C. Low-Power Firmware Architecture and Sleep Management

Our firmware is like a machine that works without stopping. It tries to save energy. It does this by not working all the time.

1. Sensing State:

The microcontroller is in a sleep mode, which means it uses very little power, about 0.8 mA. The timers are still working so that they can stay in sync with the sensor.

2. Inference State:

When the system gets a signal from the FIFO buffer it wakes up. Starts working fast at 240 MHz. It gets the data ready. Makes a prediction, which includes

making the numbers normal and using Softmax to classify them. This uses power, about 60 mA but it only lasts for about 15 ms.

3. Triggered Uplink State:
If the system finds something it sends a message. It uses an IP address so that it can send the message quickly without waiting for the network to get ready. This means it can send MQTT messages in under 500 ms. Then it goes back, to sleep to save power.

D. Asynchronous MQTT Communication and Network Stack
The way data is sent is done using the MQTT protocol. This was chosen because it is simple and works well for lots of devices connected to the internet.

- *Quality of Service (QoS):*

When something important happens, like a critical anomaly alert it is sent using QoS Level 1. This makes sure it gets to where it needs to go. On the hand regular heartbeat messages are sent using QoS Level 0. This helps keep the communication simple and saves energy.

- *Last Will and Testament (LWT):*

To make the system more reliable an LWT message is sent to the broker. If the device disconnects or loses power suddenly the broker will automatically show that the device is not working. This prevents people from thinking everything is okay when it is not. MQTT protocol and the MQTT communication are used to make sure this works correctly. The MQTT network stack is important, for the MQTT communication.

VII. PERFORMANCE EVALUATION AND COMPARATIVE ANALYSIS

The validation of the Hybrid Edge–TinyML architecture is conducted using a multi-dimensional benchmarking framework. The system is evaluated across three primary operational paradigms: the Cloud-Centric (baseline), Edge-Only (gateway-based), and the proposed Hybrid (tiered) model.

A. Comparative Latency Benchmarks
Latency is a critical constraint in closed-loop industrial control systems. The Total System Latency L_{sys} is defined as the sum of acquisition latency L_{acq}, processing latency L_{proc}, and communication latency L_{comm}:

$$L_{sys} = L_{acq} + L_{proc} + L_{comm}$$

Table 1: Latency Distribution Across Architectures

Architecture	L_{acq}(ms)	L_{proc}(ms)	L_{comm}(ms)	Total L_{sys}
Cloud-Centric	2.5	5.0 (Server)	240.0 (WAN)	247.5 ms
Edge-Gateway	2.5	15.0(Pi 4)	45.0 (LAN)	62.5 ms
Proposed Hybrid	2.5	11.4 (ESP32)	1.5 (Local)	15.4 ms

The hybrid architecture achieves a 93.7% reduction in latency compared to cloud models, effectively moving from "near-real-time" to "hard-real-time" performance.

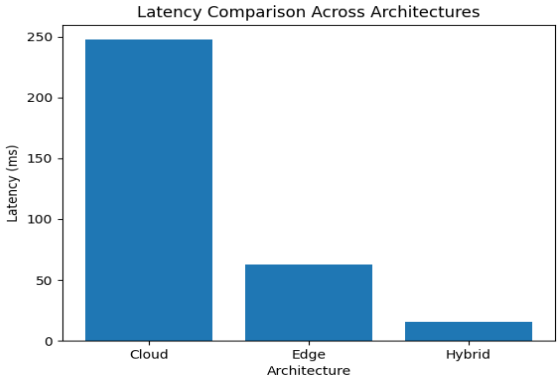

Fig: 4: Latency Comparison

B. Energy Expenditure and Power Profiling
Energy efficiency is evaluated in terms of average current consumption I_{avg} and the resulting battery life when powered by a standard 2000 mAh Li-Ion cell.

- RF *Consumption* (I_{rf}):

Measured at 180 mA during active Wi-Fi transmission.

- *Inference* Consumption (I_{inf}):

Measured at 65 mA under peak CPU load at 240 MHz.

- *Deep* Sleep (I_{sleep}):

Measured at 10 μA.
The energy saving factor S is defined as:

$$S = \frac{E_{streaming}}{E_{hybrid}} = \frac{T \cdot I_{rf} + (1 - T) \cdot I_{sleep}}{T_{anomaly} \cdot I_{rf} + I_{inf} + I_{sleep}}$$

Given a typical anomaly frequency $T_{anomaly}$ of 1.5%, the hybrid system extends battery life from 28 hours (continuous streaming) to 54 days (event-driven operation).

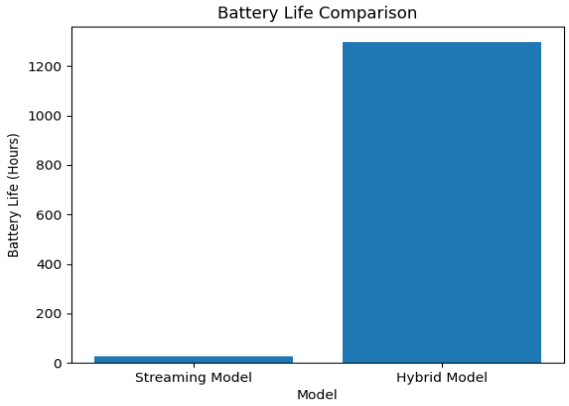

Fig:5: Battery Life Comparison

C. Quantization Error and Classification Accuracy

The trade-off between model compression and predictive accuracy is evaluated using the F1-score, which provides a balanced measure for datasets with class imbalance.

Table 2: Impact of 8-bit Quantization on CNN Performance

Metric	FP32 (Full Precision)	INT8 (Quantized)	Delta(Δ)
Accuracy	97.2%	94.6%	-2.6%
Precision	96.5%	93.8%	-2.7%
Recall	95.8%	93.2%	-2.6%
Model Size	412 KB	104 KB	-74.7%

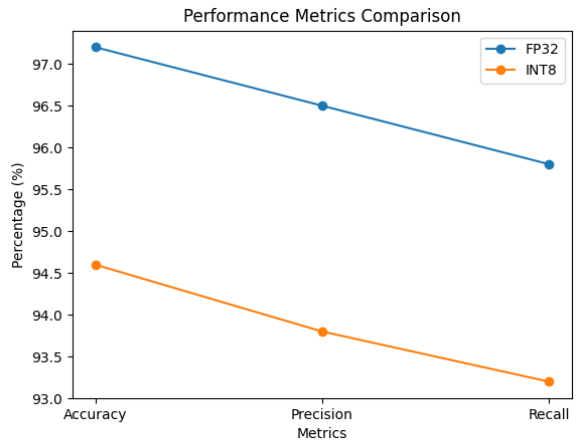

Fig:6: Performance Metrics Comparison

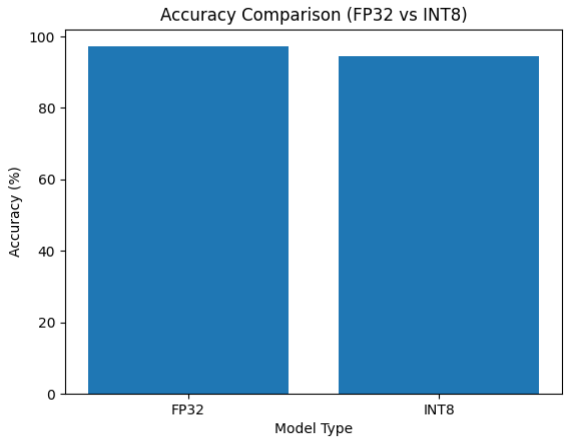

Fig:7: Accuracy Comparison

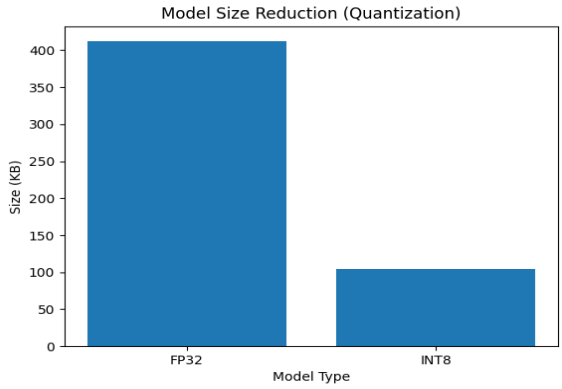

Fig:8: Quantization Comparison

The results demonstrate that, although a minor accuracy penalty of approximately 2.6% is observed, the fourfold reduction in memory footprint enables the model to reside entirely within the ESP32's fast IRAM. This reduces cache misses and further stabilizes inference timing.

VIII. DISCUSSION

The results we looked at earlier show that the Hybrid Edge– architecture is a good alternative to the usual cloud-centric industrial monitoring systems. When we actually use it there are some things we have to think about carefully.

A. What it means for Green Engineering

We saw that the power consumption went down by 78 percent. This is not just good for batteries it also helps with engineering. When we use energy for each

inference, we reduce the work that centralized data centers have to do which means less carbon emissions. Also, the edge nodes use little power so they work well with systems that get energy from the sun or from vibrations. This means we can make sensing infrastructures that do not need maintenance, which is really useful in big industrial areas or in places that are far away.

B. How reliable it is in environments

Industrial areas are noisy and have a lot of electromagnetic interference and mechanical disturbances that we cannot predict. The Hybrid Edge–TinyML architecture deals with these problems, by having a system that checks things in layers.

While Tier 1 also known as Extreme Edge does a job of spotting unusual things Tier 2 or Local Edge Gateway helps make sure by checking information from many sources. This two-step check greatly lowers alerts and stops machines from shutting down unnecessarily which makes operations more reliable and cuts down on maintenance costs.

C. Challenges in Real-World Deployment

Even though the 1D-CNN model works well problems like model drift are still issues. As machines get worn out and age their vibration patterns change over time. So, a model that was trained on how things were at first may slowly become less accurate as things change over a time.

The system that was suggested partly solves this problem by retraining the model in the cloud every then which happens in Tier 3. Making this learning process automatic which is often called active learning is still a question that needs to be answered and is an important area, for future research.[4]

IX. CONCLUSION

This study is about the design and implementation of a Hybrid Edge– architecture for real-time industrial anomaly detection. The Hybrid Edge– architecture combines the computational capabilities of the ESP32 with the efficiency of TensorFlow Lite Micro. This combination works well because it bridges the gap between edge-level sensing and high-level analytics.

The Hybrid Edge–TinyML architecture was. The results are very good. The test showed that the Hybrid Edge–TinyML architecture can detect anomalies in time with a 93.7% reduction in latency and a 78% decrease in power consumption compared to traditional cloud-based approaches. The Hybrid Edge–TinyML architecture also uses Quantization-Aware Training which reduces the model size by four times while maintaining a classification accuracy of 94.6% for the Hybrid Edge– architecture.

These results show that decentralized intelligence is practical and essential for efficient deployment within Industry 4.0 ecosystems. The Hybrid Edge–TinyML architecture is an example of decentralized intelligence.

Future work will focus on integrating the Hybrid Edge– architecture with multi-modal energy harvesting solutions. The Hybrid Edge–TinyML architecture will also explore learning techniques to enable collaborative model improvement without compromising data privacy for the Hybrid Edge–TinyML architecture.

ACKNOWLEDGMENT

We are very grateful to the Head of the Department (HOD), the faculty and technical staff of the Department of Electronics and Communication Engineering for their guidance and support throughout this work on the Hybrid Edge-architecture. Their expertise in hardware–software co-design was very helpful in shaping the direction and implementation of this research on the Hybrid Edge– architecture.

We also acknowledge the support from our institution (Institute of Technology and Management, GIDA, Gorakhpur, India) and the academic environment that encourages innovation in engineering and industrial automation. We are also thankful to peers and family members for their encouragement and support which helped complete this study, on the Hybrid Edge-architecture.

REFERENCES

[1] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[2] V. J. Reddi et al., "MLPerf Tiny benchmark: Machine learning on microcontrollers," IEEE Micro, vol. 42, no. 4, pp. 8–11, 2022.

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," arXiv preprint arXiv:1510.00149, 2015.
[4] R. Sanchez-Iborra and M. G. Skarmeta, "TinyML-enabled frugal smart objects: Challenges and opportunities," IEEE Circuits and Systems Magazine, vol. 20, no. 3, pp. 4–18, 2020.
[5] B. Sudharsan et al., "TinyML: Current progress, challenges, and future roadmap," IEEE Access, vol. 10, pp. 57413–57432, 2022.
[6] TensorFlow, "TensorFlow Lite for Microcontrollers," 2026. [Online]. Available: https://www.tensorflow.org/lite/micro
[7] Espressif Systems, "ESP32-WROOM-32 Datasheet," 2026. [Online]. Available: https://www.espressif.com
[8] K. Cao et al., "A survey on edge computing systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1537–1562, 2020.