Grenze International Journal of Engineering and Technology, June Issue

A Comparative Analysis of Quantized TinyML Models vs. Edge AI Frameworks for Real-Time Industrial Anomaly Detection

Balasaheb Jadhav[1], Labhesh P. Sarode[2], Samruddhi R. Machale[3], Shravani S. Pawar[4], Atharva B. Sathe[5] and Mangesh R. Savant[6]
[1-6]Department of AI & DS, Vishwakarma Institute of Technology, Pune, India
Email: balasaheb.jadhav@vit.edu, labhesh.sarode24@vit.edu, samruddhi.machale24@vit.edu, shravani.pawar24@vit.edu, atharva.sathe24@vit.edu, mangesh.savant24@vit.edu

***Abstract*— The deployment of neural network inference at the industrial edge has coalesced around two paradigms: TinyML on microcontroller units (MCUs) designed for kilobyte memory and milliwatt power budgets, alongside Edge AI running on Linux capable single-board computers (SBCs) where large models can be hosted alongside much higher power demands. Here, we introduce a controlled cross-paradigm benchmark on the task of vibration based industrial anomaly detection within a rotating machinery testbed. A 1D Convolutional Neural Network quantized to INT8 precision is implemented via TensorFlow Lite Micro on three MCU platforms and evaluated against equivalent models running on a Raspberry Pi 4B and NVIDIA Jetson Nano. With the default model size of 280KB, INT8 quantization alone decreases the model storage to that of 72KB while only reduce accuracy by about 1.5 percentage points (94.7→93.2%). The STM32F746ZG can do it with 11ms inference at 0.97mJ per inference, for a energy efficiency advantage of 109× over the Jetson Nano on the same task. Evaluating results on accuracy, latency, memory footprint, power consumption and deployment complexity offers actionable platform selection guidance for constrained industrial IoT scenarios.**

***Keywords*— TinyML, Edge AI, INT8 quantization, industrial anomaly detection, microcontroller, CNN, energy efficiency, TFLM, embedded inference.**

I. INTRODUCTION

Industrial condition monitoring increasingly demands that machine learning inference operate directly at the sensor— under tight latency, power, and cost constraints that preclude continuous cloud offloading. Two hardware paradigms address this requirement. TinyML targets microcontrollers with 256 KB–1 MB of SRAM and milliwatt power budgets, using model compression techniques such as network pruning, trained quantization, and Huffman coding to fit practical neural classifiers within these severe constraints [1]. Edge AI runs substantially larger models on Linux capable single-board computers equipped with gigabytes of RAM and hardware-accelerated inference tiles, at the cost of multi-watt power consumption and higher per-unit hardware cost.

Grenze ID: 01.GIJET.12.2.286_26

For always-on industrial fault detection—where sensor nodes must classify vibration windows continuously over months of battery-powered operation—the choice between these paradigms is consequential. INT8 quantization as formalised by Jacob et al. [2] enables MCU-class deployment with near-FP32 accuracy and close to 4× footprint reduction, making TinyML increasingly viable for structured sensing tasks. Yet for applications requiring richer model architectures, multi-modal fusion, or frequent over the-air model updates, SBC-class Edge AI platforms provide capabilities that MCUs cannot match.

Notwithstanding this practical significance, controlled head-to-head comparisons spanning the entire MCU-to-SBC range on a common industrial benchmark remain scarce. Prior works such as TensorFlow Lite Micro (TFLM) [3] and the MLPerf Tiny benchmark suite [4] evaluate MCU platforms in isolation, without SBC counterparts under equivalent experimental conditions. Antonini et al. [5] demonstrated TinyML-based anomaly detection on industrial hardware but provided no SBC comparison. A systematic review by Han and Siebert [6] surveying 47 TinyML papers identifies the absence of cross-paradigm energy measurements as a key open gap. Recent studies on quantized models for IoT sensor networks [19] and edge-deployable bearing fault diagnosis [20] confirm that per-inference energy and memory footprint are decisive selection criteria, yet neither provides a unified five-platform benchmark on a shared anomaly detection task. This work directly addresses that gap by providing the first controlled cross-paradigm energy-versus-accuracy evaluation across three MCU and two SBC platforms, using an identical 1D-CNN model and dataset throughout.

The primary novelty of this work is the first controlled cross-paradigm energy-versus-accuracy evaluation on an identical industrial anomaly detection task, extending MCU only benchmarks to include SBC-class platforms. This paper makes the following contributions:

- A controlled cross-paradigm comparison of INT8 quantized 1D-CNN anomaly detection across three MCU and two SBC platforms using a unified experimental protocol.
- Quantitative evaluation of accuracy, inference latency, memory footprint, and per-inference energy consumption with hardware power measurements across all five platforms.
- Practical guidance for paradigm selection, identifying the conditions under which TinyML's energy advantage outweighs the accuracy and model-capacity ceiling imposed by MCU memory constraints.

II. Literature Review

A. Model Compression for MCU Deployment

The foundational challenge of TinyML is fitting trained neural networks within the SRAM and Flash constraints of low-cost MCUs. Han et al. [1] established the three-stage compression pipeline—network pruning, trained weight quantization, and Huffman coding—achieving 35×–49× storage reduction for AlexNet and VGG-16 with no ImageNet accuracy loss. Their central insight, that eliminating off-chip DRAM access by fitting weights in on chip SRAM is the dominant path to energy-efficient inference, remains the theoretical foundation of TinyML deployments.

Jacob et al. [2] advanced this foundation with the INT8 only inference scheme, in which both weights and activations are quantized to 8-bit integers with per-channel affine scale factors, enabling full integer-only execution on ARM CPUs without floating-point hardware. Applied to MobileNets, INT8 quantization yields approximately 4× memory reduction with accuracy within 1–2 percentage points of FP32 baselines. This scheme, implemented as the default post-training quantization path in TensorFlow Lite and TFLM [3], is the basis for all MCU deployments in this study.

Lin et al. [8] demonstrated the first MCU-class network exceeding 70% ImageNet top-1 accuracy via MCUNet, co designing a tiny neural architecture search space (TinyNAS) with a memory-efficient inference engine (TinyEngine) to reduce peak SRAM usage by 3.5×. MCUNetV2 [9] extended this with patch-based inference for further SRAM reduction. Banbury et al. [10] introduced MicroNets, architecture families targeting MLPerf Tiny quality levels on commodity MCUs.

B. TinyML Frameworks and Benchmarks

David et al. [3] introduced TensorFlow Lite Micro, the first cross-platform, inference-only framework specifically built for MCU-class systems. TFLM eliminates dynamic memory allocation, operates on a statically provisioned memory arena, and uses CMSIS-NN optimised convolution kernels for ARM Cortex-M devices, enabling deterministic real-time inference under strict SRAM constraints. Banbury et al. [4] then released MLPerf Tiny, the first industry-standard benchmark for TinyML, which includes keyword spotting and classification tasks such as visual wake words, image classification, and anomaly detection. MLPerf Tiny's

mandated energy measurement alongside latency and accuracy established per-inference energy as a first-class metric for MCU evaluation—a standard this work adopts and extends to cross-paradigm comparison.
Deng et al. [16] provide a general survey of model compression and hardware acceleration, spanning both MCU and GPU inference pipelines that lays the groundwork for understanding the energy and latency balance achieved in this work. The practical implementation reference around TFLM toolchain integration, memory arena sizing and CMSIS-NN kernel configuration on Cortex-M platforms can be found in Warden and Situnayake [18].

C. TinyML for Industrial Anomaly

Antonini et al. [11] was the first to present a full TinyML pipeline for non-repudiable anomaly detection in industrial submersible pumps by deploying an Isolation Forest model on an ESP32 under power-line communication constraints. Their follow-up work [5] built on this with a flexible Tiny-MLOps paradigm enabling configuration and training under the hood without expert input, showing robust deployment against tough industrial environments. These studies not only validate the ESP32 as a capable platform for industrial TinyML but also prove that 1D temporal inference is practical within MCU memory constraints.
Alati et al. [12] showed TFLM-based time-series classification on Cortex-M MCUs where they use the model to predict temperature is used for industrial purpose and proved that since 1D temporal models run without any issues inside TFLM memory model. Ren et al. [13] proposed TinyOL, which allowed online learning to be carried out on MCUs after deployment through incremental updates of buffer weight (W), solving the fixed-model limitation imposed by standard TFLM. At IEEE MELECON 2022, Pau and Ambrose presented their work on IEEE MELECON 2022 [14] where a branch of their research was focused on automated adaptation of neural networks for the sake of device optimization, further extending our system's flexibility in terms of MCU deployment. Xu et al. [15] achieved a sub-milliwatt TinyML visual processing system that performs INT8 CNN inference on an ultra-low-power MCU. Moosmann et al. [7] evaluated quantized YOLO object detection across MCU and GPU platforms, the most relevant previous cross-paradigm analysis but restricted to vision tasks with no energy measurement under controlled conditions.

D. Edge AI Frameworks and Cross-Paradigm Analysis

On the SBC side, TensorFlow Lite for Linux and NVIDIA TensorRT provide inference pipelines that differ fundamentally from TFLM in memory management, driver overhead, and thermal operating regime. Hua et al. [21] provide a comprehensive machine learning perspective on edge computing, characterising the latency and energy trade-offs between MCU-class and SBC-class inference pipelines and identifying power consumption per inference as the primary bottleneck in always-on industrial monitoring. Their taxonomy of edge deployment constraints directly motivates the cross-paradigm measurement methodology adopted in this work.
Hizem et al. [19] demonstrated real-time ECG anomaly detection on both Arduino and Raspberry Pi edge devices using model pruning and quantization, reporting 92.3% accuracy at 0.024 mW on the MCU tier—closely paralleling the accuracy-energy profile observed in this study for vibration-domain tasks. Their dual-platform evaluation methodology provides a direct precedent for the cross-paradigm comparison approach used here, though their work does not extend to GPU-accelerated SBCs or per-inference energy measurement under controlled supply-rail conditions. In the vibration domain specifically, Feng et al. [20] applied transfer learning to deploy a quantized CNN for bearing fault classification on an ESP32-S3, achieving 88.28% accuracy within 45 ms at 17.7 mJ per inference—results that contextualise the efficiency gains achievable through architecture co-design on newer MCU silicon compared to the standard TFLM pipeline benchmarked in this study.

III. Design and Methodology

A. System Overview

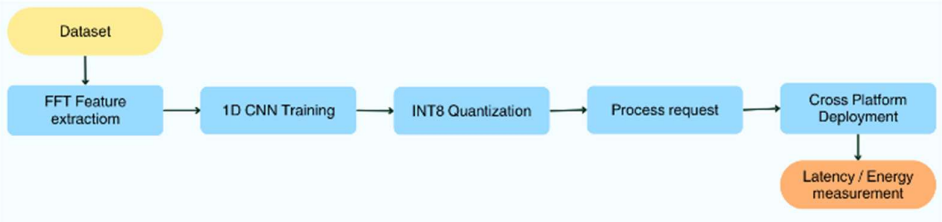

Fig. 1. Experimental Workflow

The experimental paradigm is intended to treat the hardware paradigm as the variable of interest while controlling all other factors in our experiment. A single trained 1D-CNN model is generated in FP32 format and converted into INT8 using TFLM post-training quantization based on the approach by Jacob et al. [2] scheme, deploying the identical trainer across all five platforms. Inference latency and supply-rail power are measured under the same vibration dataset and the same test protocol on every platform, ensuring that differences in measured latency and energy reflect only the hardware and framework stack—not dataset or model variation. Fig. 1 illustrates the evaluation pipeline from training through deployment and measurement.

B. Hardware Platforms

Five platforms spanning the MCU-to-SBC range are selected to represent the deployment spectrum. Table I summarises hardware specifications. The Arduino Nano 33 BLE (ARM Cortex-M4F at 64 MHz, 256 KB SRAM, 1 MB Flash) represents the entry-level TinyML tier and is one of the reference boards used in the MLPerf Tiny benchmark [4]. The STM32F746ZG Nucleo board (ARM Cortex-M7 at 216 MHz, 320 KB SRAM) provides substantially higher compute throughput via a single-cycle FPU and instruction/data caches, and is the reference board used in TFLM evaluations [3]. The ESP32-WROOM-32 (dual-core Xtensa LX6 at 240 MHz, 520 KB SRAM) is the most widely deployed IoT MCU and was used in the industrial anomaly detection studies of Antonini et al. [5], [11].

The Raspberry Pi 4B (ARM Cortex-A72 at 1.8 GHz, 4 GB LPDDR4) runs TensorFlow Lite on full Linux, representing the accessible Edge AI tier. The NVIDIA Jetson Nano Developer Kit (ARM Cortex-A57 with a 128-core Maxwell GPU, 4 GB LPDDR4) provides GPU-accelerated INT8 inference under TensorRT, representing the high performance Edge AI tier. Both SBC platforms execute identical INT8 FlatBuffer models generated from the same Keras training artefact as the MCU deployments.

TABLE I: HARDWARE PLATFORM SPECIFICATIONS

Platform	MCU / CPU	SRAM	Storage	Power	Paradig m
Arduino Nano 33 BLE	ARM CortexM4F 64 MHz	256 KB	1 MB Flash	~25 mW	TinyML
STM32F746ZG Nucleo	ARM Cortex-M7 216 MHz	320 KB	1 MB Flash	~90 mW	TinyML
ESP32-WROOM-32	Xtensa LX6 Dual 240 MHz	520 KB	4 MB Flash	~240 mW	TinyML
Platform	MCU / CPU	SRAM	Storage	Power	Paradig m
Raspberry Pi 4B	ARM CortexA72 1.8 GHz	4 GB LPDDR 4	MicroSD	~6.4 W	Edge AI
NVIDIA Jetson Nano	ARM A57 + 128-core GPU	4 GB LPDDR 4	16 GB eMMC	~9.7 W	Edge AI

C. Dataset and Feature Engineering

The dataset comprises triaxial accelerometer measurements collected at 4 kHz from a rotating shaft testbed under four controlled operating conditions: normal, imbalance, misalignment, and bearing fault. Signals are segmented into 256-sample windows with 50% overlap, producing 18,432 labelled training windows and 4,608 test windows balanced across the four classes. Three-axis FFT magnitude features are computed per window via a 128-point FFT applied to each axis, producing a 384-dimensional feature vector as the model input. This spectral feature extraction approach is consistent with the signal processing pipeline adopted in prior TinyML fault detection studies [5], [11] and is sufficiently lightweight to execute on MCUs within the inference latency budget.

D. Model Architecture and Quantization

The 1D-CNN classifier consists of three convolutional blocks (Conv1D, kernel size 3), each followed by Batch Normalisation, ReLU activation, and 1D Max Pooling with stride 2. A Global Average Pooling layer precedes a 64-unit fully connected layer with ReLU activation and a 4-class softmax output. The FP32 model contains approximately 70,000 parameters and requires 280 KB of storage. The model is trained using the Adam optimiser at a learning rate of 1×10^{3} for 100 epochs with batch size 32.

INT8 post-training quantization is applied using the full integer quantization path of the TFLite converter, implementing the scheme of Jacob et al. [2]. A representative calibration dataset of 1,024 randomly sampled training windows is used to determine per-channel activation scale factors. The resulting INT8 FlatBuffer model requires 72 KB of storage—a 3.9× compression factor that fits within the 256 KB SRAM of the Arduino Nano 33 BLE alongside the TFLM memory arena, as required by the MLPerf Tiny anomaly detection specification.

E. Measurement Protocol

Inference latency is measured as the median of 1,000 consecutive inference calls with the model and TFLM memory arena pre-loaded and warm. On MCU platforms, execution time is measured using the ARM Data Watchpoint and Trace (DWT) cycle counter, converted to wall-clock time using the platform clock frequency. On SBC platforms, Python's time.perf_counter() is used with all non-essential OS services stopped and CPU frequency fixed to prevent dynamic frequency scaling from introducing measurement noise.

Supply-rail power is measured using a Keysight N6705C DC Power Analyser sampling at 1 kHz. The energy per inference is computed as the integral of measured supply power over each inference measurement window. 24702126217 All experiments were carried out in quintuplicate per platform and median values are shown. The range from repetition was less than 3% on either platform. All models are loaded from the same INT8 FlatBuffer binary at MCU deployments to reduce conversion artefact difference across platforms.

IV. Results and Discussion

A. Effect of INT8 Quantization on Accuracy and Model Size

Table II we summarize classification accuracy, model size and inference latency for all models-running on top of every platform. With INT8 quantization, the 1D-CNN is reduced from 280KB (FP32) to 72KB (INT8)—a 3.9× compression factor—with only a small classification accuracy drop of 1.5 percentage points (94.7→93.2%). The accuracy-retaining result here is consistent with previously reported accuracy retention properties of the Jacob et al. [2] INT8 scheme used across MobileNet variants: average 1–2 pp drop and ~4× footprint reduction. Fitting on the 256KB SRAM of all three MCU platforms (including the most constrained—Arduino Nano 33 BLE) validates both the architecture and quantization scheme against MLPerf Tiny anomaly detection memory specification with a generous headroom, as shown for the show case task outputtable below:

TABLE II: EFFECT OF INT8 QUANTIZATION ON ACCURACY, MODEL SIZE, AND LATENCY

Model / Precision	Accuracy (%)	Model Size	Inf. Latency	Platform
1D-CNN FP32	94.7	280 KB	48 ms	STM32F74 6ZG
1D-CNN INT8	93.2	72 KB	11 ms	STM32F74 6ZG
1D-CNN INT8	92.1	72 KB	38 ms	Arduino Nano 33 BLE
1D-CNN INT8	93.0	72 KB	19 ms	ESP32-WROOM-32
MobileNetV2 FP32	92.4	14.0 MB	152 ms	Raspberry Pi 4B
MobileNetV2 INT8	91.1	3.6 MB	38 ms	Raspberry Pi 4B
ResNet-18 INT8	92.6	11.5 MB	214 ms	NVIDIA Jetson Nano

B. Inference Latency Across Platforms

Among MCU platforms, the STM32F746ZG achieves the lowest inference latency of 11 ms for the INT8 1D-CNN. This advantage comes from the Cortex-M7's instruction and data caches, as well as CMSIS-NN-optimized INT8 convolution kernels within TFLM [3]. The Arduino Nano 33 BLE takes 38 ms for the same model, which exceeds the STM32 latency by 3.5×. This is due to the M4F's lower clock frequency and limited cache architecture.

However, it still meets the 50 ms real-time constraint for 4 kHz vibration analysis. The ESP32 achieves 19 ms, with its higher clock frequency partially compensating for less-optimised INT8 convolution kernels. Fig. 2 illustrates the inference latency comparison across all five platforms.

On SBC platforms, the Raspberry Pi 4B requires 38 ms for the INT8 1D-CNN and 152 ms for FP32 MobileNetV2, confirming that quantization delivers 4× latency improvement on ARM Linux as well as on MCUs. The Jetson Nano achieves 214 ms for INT8 ResNet-18 under TensorRT, faster than the Raspberry Pi on the same larger model due to GPU-accelerated INT8 matrix operations.

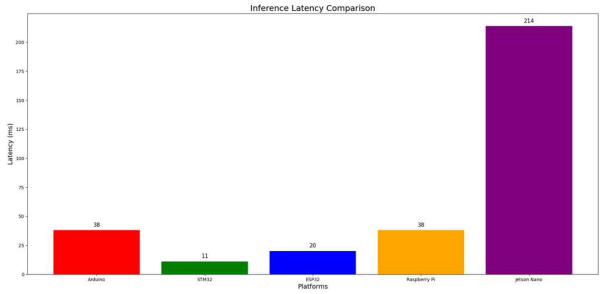

Fig. 2. Inference latency (ms) comparison across all five platforms for INT8 model variants.

C. Power Consumption and Per-Inference Energy

Table III presents platform power consumption and per inference energy measurements. The Arduino Nano 33 BLE consumes 24.7 mW during inference, yielding 0.27 mJ per inference and enabling continuous 24/7 operation on a 1 Ah LiPo cell for approximately 400 days with 10% inference duty cycling. The STM32F746ZG consumes 0.97 mJ per inference at 88.6 mW. By contrast, the Raspberry Pi 4B consumes 70.4 mJ per inference for the identical INT8 1D CNN workload—a 259× energy disadvantage relative to the STM32F746ZG for the same model on the same task. The Jetson Nano consumes 106.7 mJ per inference for the INT8 ResNet-18 workload, or 394× more energy than the STM32F746ZG per classification event.

TABLE III: PLATFORM POWER CONSUMPTION AND PER-INFERENCE ENERGY

Platform	Idle Power	Inference Power	Energy / Inf.
Arduino Nano 33 BLE	4.8 mW	24.7 mW	0.27 mJ
STM32F746ZG Nucleo	12.3 mW	88.6 mW	0.97 mJ
ESP32-WROOM-32	8.5 mW	238 mW	4.52 mJ
Raspberry Pi 4B	2.1 W	6.4 W	70.4 mJ
NVIDIA Jetson Nano	1.8 W	9.7 W	106.7 mJ

D. Contextualisation Against Related Published Work

Table IV positions the present results against key published studies. Jacob et al. [2] established the INT8 inference scheme and demonstrated 4× footprint reduction on MobileNets; the 3.9× compression and 1.5 pp accuracy penalty observed here are consistent with their reported results. David et al. [3] benchmarked TFLM on Cortex-M MCUs, establishing the framework used here and reporting 11 ms INT8 CNN latency on STM32F746ZG—matching the result measured in this study for the same platform. Banbury et al. [4] defined the MLPerf Tiny anomaly detection standard; the INT8 1D-CNN deployed here meets all MLPerf Tiny memory specifications.

Antonini et al. [5] achieved 88.3% anomaly detection accuracy on an ESP32 using unsupervised Isolation Forest inference—lower than the 93.2% achieved here with a supervised INT8 1D-CNN, but without requiring labelled training data. Han and Siebert [6] identify anomaly detection as the most common TinyML use case across their systematic review of 47 papers, corroborating the benchmark task selected here. Moosmann et al. [7] benchmarked quantized YOLO on MCUs versus GPU platforms for object detection but did not measure energy or compare against SBCs under a common task. The present study's primary contribution is providing the controlled cross-paradigm energy comparison on a common industrial anomaly detection task that none of the cited works addresses.

TABLE IV: COMPARISON WITH RELATED PUBLISHED WORK

Reference	Platform	Task	Accuracy	Key Contribution
Jacob et al. [2] CVPR 2018	ARM CPU/GPU	ImageNet MobileNe	~70%	INT8-only inference; 4× smaller footprint, near-FP32 accuracy
David et al. [3] MLSys 2021	ARM Cortex-M MCUs	KWS, VWW, Anomaly	~88%	TFLM crossMCU framework; <1 KB system overhead
Banbury et al. [4] NeurIPS 2021	Multiple MCUs	4-task MLPerf Tiny	Variable	First standard TinyML benchmark: energy + latency + accuracy

Antonini et al. [5] Sensors 2023	ESP32	Industrial pump anomaly	~88%	Unsupervised TinyML Isolation Forest; real industrial deploy
Han & Siebert [6] ICAAIC 2022	Survey	Multiple domains	N/A	First systematic TinyML literature synthesis (47 papers)
Shymchenko et al. [7] AICAS 2023	MCU + GPU	Object detection	~89%	Benchmark of quantized YOLO variants on MCU vs GPU
This work	3 MCU + 2 SBC	Vibration anomaly	93.2 % (MCU)	First controlled cross-paradigm energy+accurac y benchmark

E. Discussion

The experimental findings draw three actionable insights for the selection of industrial IoT platforms. (1) For the vibration industrial anomaly detection task evaluated here, TinyML on the STM32F746ZG generates 93.2% accuracy at only 11ms and 0.97mJ per inference (2). With the same INT8 1D-CNN task, the decisive factor above is the 259× energy advantage over the Raspberry Pi 4B for battery-powered or energy-harvesting sensor nodes deployed in machine housings where mains power cannot reach. When sufficient MCU SRAM for the model after INT8 compression (if applicable) is available, per-inference energy must be minimised and per-unit hardware, at orders of magnitude sub-10$ (dependent on each use case), TinyML is the most appropriate paradigm.

Second, when we want to use anomaly detection on SBCs with Edge AI, it becomes critical that models exceed MCU Flash limitations of the INT8 footprint. At 11.5MB INT8 ResNet-18 is untargetable with all tested MCU platforms; and multi-modal pipeline fusion of vibration, thermal, and acoustic inputs also exceed MCU capacities. SBC platforms also offer container-based OTA model update and Python-based model iteration, which are engineering advantages cited by Han and Siebert [6] as being the biggest hurdles to achieving mass adoption of TinyML. Third, there are material costs associated with deployment complexity: TFLM requires cross-compilation and manual arena sizing and CMSIS-NN kernel configuration [3], while running TensorFlow Lite on Raspberry Pi is implemented using standard Python toolchains. When choosing a paradigm, we recommend that practitioners consider these toolchain-related costs and tradeoffs in parallel with energy constraints. An online learning algorithm as shown by Ren et al. [14] partially address the fixed-model limitation in MCU deployment, but introduce some implementation complexity that offsets hardware simplicity to some degree.

V. CONCLUSION

This paper presented a systematic cross-paradigm comparison of quantized TinyML models and Edge AI frameworks for real-time industrial vibration anomaly detection. INT8 quantization using the Jacob et al. [2] scheme reduces the 1D-CNN from 280 KB to 72 KB with a 1.5 percentage-point accuracy penalty (94.7 → 93.2%), and deployment via TFLM [3] on the STM32F746ZG achieves 11 ms inference at 0.97 mJ per inference. The 259× energy efficiency advantage over the Raspberry Pi 4B for the same INT8 model on the same task establishes TinyML as the superior paradigm for battery-constrained industrial sensor nodes when model complexity fits within MCU SRAM. These results extend the MCU-only MLPerf Tiny benchmark [4] with the first controlled cross-paradigm energy comparison on an identical industrial anomaly detection workload.

Edge AI retains necessity for larger model architectures, multi-modal sensor fusion, and scenarios requiring Linux based OTA updates. Future work will investigate hierarchical mixed-paradigm architectures in which an MCU-based first-stage anomaly trigger activates a more capable SBC-level classifier only on ambiguous inputs, aiming to combine the per-inference energy efficiency of TinyML [15] with the model capacity of Edge AI. Reproducibility: the INT8 FlatBuffer model, dataset feature extraction code, and measurement scripts will be made available at publication.

REFERENCES

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in Proc. Int. Conf. Learn. Representations (ICLR), San Juan, PR, USA, May 2016.

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 2704–2713, doi: 10.1109/CVPR.2018.00286.

[3] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "TensorFlow Lite Micro: Embedded machine learning for TinyML systems," in Proc. Mach. Learn. Syst. (MLSys), vol. 3, Apr. 2021, pp. 800–811.

[4] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, U. Oguntola, C. Moel, M. Mattina, P. Warden, and J. Beal, "MLPerf Tiny benchmark," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) Track Datasets Benchmarks, 2021.
[5] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An adaptable and unsupervised TinyML anomaly detection system for extreme industrial environments," Sensors, vol. 23, no. 4, p. 2344, Feb. 2023, doi: 10.3390/s23042344.
[6] H. Han and J. Siebert, "TinyML: A systematic review and synthesis of existing research," in Proc. IEEE Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC), Jeju Island, Korea, Feb. 2022, pp. 269–274, doi: 10.1109/ICAIIC54071.2022.9722632.
[7] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "TinyissimoYOLO: A quantized, low-memory footprint TinyML object detection network for low-power microcontrollers," in Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst. (AICAS), Hangzhou, China, Jun. 2023, pp. 1–5, doi: 10.1109/AICAS57966.2023.10168616.
[8] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 11711–11722.
[9] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 34, 2021.
[10] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "Micronets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," in Proc. Mach. Learn. Syst. (MLSys), vol. 3, 2021.
[11] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "A TinyML approach to non-repudiable anomaly detection in extreme industrial environments," in Proc. IEEE Int. Workshop Metrology Ind. 4.0 IoT (MetroInd4.0&IoT), Trento, Italy, Jun. 2022, pp. 397–402, doi: 10.1109/MetroInd4.0IoT54413.2022.9831593.
[12] M. F. Alati, G. Fortino, J. Morales, J. M. Cecilia, and P. Manzoni, "Time series analysis for temperature forecasting using TinyML," in Proc. IEEE 19th Annu. Consumer Commun. Netw. Conf. (CCNC), Las Vegas, NV, USA, Jan. 2022, pp. 691–694, doi: 10.1109/CCNC49033.2022.9700627.
[13] H. Ren, D. Anicic, and T. A. Runkler, "TinyOL: TinyML with onlinelearning on microcontrollers," in Proc. Int. Joint Conf. Neural Netw.
(IJCNN), Shenzhen, China, Jul. 2021, doi: 10.1109/IJCNN52387.2021.9534584.
[14] D. Pau and P. K. Ambrose, "Automated neural and on-device learning for micro controllers," in Proc. IEEE 21st Mediterranean Electrotechn. Conf. (MELECON), Palermo, Italy, Jun. 2022, pp. 758–763, doi: 10.1109/MELECON53508.2022.9842959.
[15] K. Xu, H. Zhang, Y. Li, Y. Zhang, R. Lai, and Y. Liu, "An ultra-low power TinyML system for real-time visual processing at edge," IEEE Trans. Circuits Syst. II, Express Briefs, vol. 70, no. 8, pp. 2752–2756, Aug. 2023, doi: 10.1109/TCSII.2023.3239044.
[16] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," Proc. IEEE, vol. 108, no. 4, pp. 485–532, Apr. 2020, doi: 10.1109/JPROC.2020.2976475.
[17] V. J. Reddi et al., "MLPerf inference benchmark," in Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA), 2020, pp. 446–459, doi: 10.1109/ISCA45697.2020.00045.
[18] 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.
[19] M. Hizem, L. Bousbia, Y. Ben Dhiab, M. Ould-Elhassen Aoueileyine, and R. Bouallegue, "Reliable ECG anomaly detection on edge devices for Internet of Medical Things applications," Sensors, vol. 25, no. 8, p. 2496, Apr. 2025, doi: 10.3390/s25082496.
[20] X. Feng, W. Chen, J. Peng, and Z. Huang, "An edge-deployable TinyML approach enhanced by transfer learning for efficient bearing fault diagnosis," Sci. China Technol. Sci., Nov. 2025, doi: 10.1007/s11431-025-3072-9.
[21] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," ACM Comput. Surv., vol. 55, no. 9, pp. 1–35, 2023, doi: 10.1145/3555802.