IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 4, April, 2026, pp. 1141-1160
Journal homepage: http://iieta.org/journals/jesa

Effective Classification of Bearing Vibration Signals Using Supervised Machine Learning for Predictive Maintenance: A Lightweight 1D CNN for Embedded Deployment

Imane El Boughardini[1*], Hakim Jebari[2,3], Siham Rekiek[4], Kamal Reklaoui[1]

[1] Innovative Systems Engineering Research Team, University Abdelmalek Essaâdi, Tétouan 93000, Morocco
[2] Artificial Intelligence, Data Science, and Innovation Research Team, LaBEL, National School of Architecture, Tétouan 93040, Morocco
[3] Innovative Systems Engineering Laboratory, University Abdelmalek Essaâdi, Tétouan 93000, Morocco
[4] Intelligent Automation & BioMedGenomics Laboratory, University Abdelmalek Essaâdi, Tétouan 93000, Morocco

Corresponding Author Email: Imane.ElBoughardini@outlook.com

https://doi.org/10.18280/jesa.590423

Received: 5 February 2026
Revised: 14 April 2026
Accepted: 22 April 2026
Available online: 30 April 2026

Keywords:
predictive maintenance, bearing fault diagnosis, 1D convolutional neural network, embedded AI, edge computing, vibration analysis, Industry 4.0

ABSTRACT

Industrial downtime precipitated by bearing failures constitutes a severe financial impediment, with estimates suggesting costs can reach $22,000 per hour in capital-intensive sectors such as automotive manufacturing. This investigation addresses this challenge through a quantized one-dimensional convolutional neural network (1D CNN) architected for embedded systems deployment. Under the evaluation protocols defined in this study—including group-stratified 5-fold cross-validation on seven public datasets harmonized to three health states—the proposed model achieves a macro F1-score of 98.6% ± 0.3% (mean ± std). When deployed on a Teensy 4.1 microcontroller, the network executes inference in 4.7 ms (measured using ARM DWT cycle counters) and consumes 90 kB of flash storage with 42 kB runtime random access memory. In benchmark comparisons conducted under identical test conditions, this approach outperformed Support Vector Machine (SVM) by 7.8 percentage points and XGBoost by 3.5 percentage points in F1-score. The model's quantization-aware training and CMSIS-NN optimization enable deployment on resource-constrained devices without cloud connectivity. This work demonstrates a feasible pathway for on-device predictive maintenance (PdM) on legacy industrial equipment, with potential to reduce unplanned downtime in applications where continuous cloud connectivity is unavailable or impractical.

1. INTRODUCTION

Rolling-element bearings represent the mechanical linchpins within rotating machinery ecosystems, including turbines, electric motors, pumps, and compressors. Their primary function involves the critical tasks of friction reduction and support for substantial mechanical loads during operation. Paradoxically, these indispensable components are simultaneously among the most prolific sources of catastrophic mechanical failure, with industry analyses implicating them in 45–55% of all malfunctions occurring within such complex systems [1-3]. The consequent economic ramifications are nothing short of staggering; contemporary industry audits indicate that single bearing failure events can precipitate direct and indirect costs ranging from $5,000 to an astonishing $250,000 per hour within high-throughput operational environments like continuous energy production and automated automotive assembly lines [1, 4]. This profound financial vulnerability has catalyzed an unequivocal industry-wide paradigm shift, moving away from traditional reactive and predetermined preventive maintenance schedules toward intelligent, data-driven predictive maintenance (PdM) strategies [5].

PdM fundamentally leverages continuous, high-frequency sensor data streams—primarily comprising vibration signatures, acoustic emissions, and thermal profiles—to identify and diagnose incipient faults long before they culminate in irreversible mechanical degradation or catastrophic systemic failure [6]. Within modern PdM frameworks, machine learning (ML) and deep learning (DL) models have ascended to a position of critical importance, demonstrating formidable capabilities in classifying precise bearing health states from annotated vibration data sequences [7, 8]. Notwithstanding these advanced capabilities, a substantial proportion of this pioneering research remains confined to controlled laboratory environments, characterized by abundant computational resources and idealized data conditions, thereby largely overlooking the severe pragmatic constraints endemic to real-world industrial deployment scenarios [9].

Furthermore, recent studies highlight the critical challenge of domain adaptation—where models trained under specific laboratory conditions often fail when deployed on different machines or under varying operational speeds. This gap is particularly problematic for safety-critical applications, such as helicopter main rotor bearings, where fault data is scarce

and operating conditions are highly variable. To address both domain adaptation and deployment constraints simultaneously, our work integrates quantization-aware training with data augmentation techniques specifically designed to simulate domain shifts. The quantization process not only reduces memory footprint by 50% but also introduces beneficial regularization effects that slightly improve model robustness to input variations.

Legacy industrial environments, which constitute the majority of global manufacturing infrastructure, frequently exhibit severely limited hardware capabilities, typically lacking robust cloud connectivity, powerful computing hardware, or substantial energy budgets. The successful implementation of PdM within these constrained contexts necessitates the development of solutions capable of operating entirely on resource-constrained embedded microcontrollers. These devices are defined by their limited RAM availability, absent GPU accelerators, and mandatory real-time processing requirements [10]. The emergent discipline of embedded artificial intelligence (AI) offers a profoundly promising pathway, primarily through the application of highly optimized DL models, such as one-dimensional convolutional neural networks (1D CNNs), which are specifically engineered for execution on low-power, minimalist hardware architectures [11, 12].

This study is architected to bridge the conspicuous chasm between theoretical academic model performance and practical industrial deployment constraints. We present a novel, lightweight 1D CNN architecture, meticulously engineered for direct deployment on commercial microcontrollers, which achieves superior diagnostic accuracy while scrupulously adhering to strict real-time and memory limitations. The model undergoes rigorous benchmarking against a comprehensive suite of alternatives, including traditional ML algorithms (e.g., Support Vector Machines (SVMs), k-Nearest Neighbors (KNN)), advanced ensemble methods (e.g., Random Forest, XGBoost), and other contemporary DL approaches, across seven distinct and demanding public datasets to unequivocally ensure robustness, reliability, and generalizability. The primary contributions encapsulating this work are enumerated as follows:

1) A comprehensive and critical benchmark evaluation of contemporary ML techniques for bearing fault diagnosis, conducted under authentic embedded systems constraints.
2) The proposition and detailed exposition of an optimized, quantized 1D CNN model that achieves state-of-the-art diagnostic accuracy while maintaining minimal computational resource consumption.
3) A rigorous, multi-faceted validation of the model's generalizability across diverse operational conditions, mechanical configurations, and dataset provenances.
4) An extensive discussion elucidating the broader implications and applications of lightweight AI architectures for the Internet of Things (IoT) and smart cyber-physical systems across industrial domains.

2. LITERATURE REVIEW

The intellectual domain of bearing fault diagnosis has undergone a substantial and continuous evolution over recent decades, transitioning systematically from rudimentary expert systems reliant on foundational vibration analysis principles [13, 14] to the current era of sophisticated, data-driven computational methodologies.

2.1 Traditional and ensemble learning techniques

The initial incursion of ML into this field featured the application of established algorithms such as SVMs, Decision Trees (DTs), and KNN [15, 16]. While demonstrating commendable efficacy within carefully constrained experimental scenarios, these models exhibit an inherent and critical dependence on manual feature engineering processes—involving the extraction of domain-specific indicators like Root Mean Square (RMS), kurtosis, and spectral features—and consequently display limited adaptability and robustness when confronted with the vast diversity of real-world operational contexts and noise profiles [17]. To ameliorate these limitations, ensemble learning techniques, including Random Forest and advanced Gradient Boosting implementations like XGBoost, were subsequently adopted to enhance predictive robustness and generalization capability [18, 19]. Despite their demonstrably improved accuracy metrics, these ensemble models frequently possess considerable memory footprints and computational graph complexities, rendering them fundamentally unsuitable for deployment on most microcontroller-based edge computing applications where resources are profoundly scarce [20].

Recent comprehensive benchmarks on bearing datasets show that while XGBoost and Random Forest continue to outperform traditional methods, their performance significantly degrades (by 8-12% F1-score) when tested across different machines—a limitation not shared by well-designed 1D CNNs with proper domain adaptation techniques.

2.2 Deep learning approaches

The advent and subsequent proliferation of DL have markedly transformed the methodological landscape of fault diagnosis by facilitating comprehensive end-to-end learning directly from raw sensor data, thereby effectively obviating the necessity for manual, domain-expert feature extraction [11]. CNNs, particularly 1D architectures operating directly on temporal signals, have proven exceptionally capable at identifying and leveraging localized temporal patterns and latent features within vibration signals [8, 21]. Complementary recurrent network architectures, such as bidirectional Long Short-Term Memory networks (Bi-LSTMs), have also been successfully applied to model complex temporal dependencies and long-range contextual information inherent in progressive fault evolution sequences [22]. A persistent and significant limitation, however, is that the vast majority of these advanced models are primarily designed for cloud or high-performance server-based inference, often requiring gigabytes of RAM and dedicated GPU acceleration [23], which places them far beyond the practical reach of cost-sensitive and resource-constrained embedded industrial systems.

A systematic framework for bearing fault diagnosis categorizes domain adaptation approaches into four levels: (1) regular DL (identical source/target distributions), (2) transfer in identical machine (TIM) with different operating conditions, (3) transfer across different machines (TDM), and (4) zero-fault shot learning with no faulty examples in target domain. Most industrial applications face TIM or TDM challenges, where models must generalize across varying

loads, speeds, or even different bearing manufacturers. Our work specifically addresses the TIM challenge through strategic data augmentation and architectural choices that learn speed-invariant representations.

2.3 The embedded deployment gap

Our work is strategically positioned to address this critical deployment gap directly. While a limited number of prior studies have demonstrated basic Artificial Neural Network (ANN)-based classification on constrained datasets [24], they frequently lack rigorous cross-domain validation and practical hardware optimization considerations. Furthermore, contemporary reviews focusing on IoT and AI integration persistently highlight a predominant trend towards cloud-centric model architectures [25, 26], which are infeasible for many real-time industrial applications. In direct contrast, we focus on architecting a compact, fully quantized 1D CNN that delivers state-of-the-art accuracy while operating within the severe memory, latency, and energy constraints of commercial microcontrollers, thereby ensuring its immediate practical utility for industrial applications.

Successful edge deployment requires co-design of algorithms, numerical representations, and hardware architectures. Recent surveys on embedded AI highlight that 8-bit quantization provides the optimal balance for microcontrollers, achieving 4× memory reduction and 3-4× speedup with minimal accuracy loss (< 0.5%). However, quantization-aware training must account for specific hardware constraints of target platforms—for ARM Cortex-M7 processors with SIMD extensions, optimal kernel sizes (3, 5, 7) align with hardware acceleration capabilities. Our methodology explicitly incorporates these hardware-aware design principles.

Recent advances in quantization-aware training [27] have demonstrated that 8-bit integer quantization can achieve 4× memory reduction with < 0.5% accuracy loss on time-series classification tasks. For ARM Cortex-M processors, CMSIS-NN [28] provides optimized kernels that leverage SIMD instructions, achieving 4-5× speedup over naive implementations. The study [29] showed that vibration-based fault diagnosis models can be successfully deployed on Cortex-M4 with sub-10 ms latency, though their architecture required 2× more memory than ours.

2.4 Explainable AI for vibration-based diagnostics

The 'black box' nature of DL models remains a significant barrier to industrial adoption, particularly in safety-critical applications where engineers require interpretable failure diagnoses. Recent XAI methods adapted for time-series data—including Gradient-weighted Class Activation Mapping (Grad-CAM), Layer-wise Relevance Propagation (LRP), and Integrated Gradients—can visualize which temporal regions and frequency components influence model decisions. These techniques bridge data-driven and physics-based approaches by identifying whether models focus on characteristic fault frequencies (e.g., ball pass frequencies) or other diagnostic indicators. Our work incorporates XAI visualization to validate that learned features align with domain knowledge.

2.5 Theoretical background: Vibration analysis and feature extraction

The theoretical underpinning of vibration-based diagnosis rests on the principle that localized defects in bearing components (inner race, outer race, rolling elements, cage) generate specific, repetitive impacts during operation. These impacts excite the natural frequencies of the bearing and surrounding structure, producing vibration signatures that are modulated by the shaft rotational frequency [13]. The classic approach involves signal processing techniques to extract features indicative of these faults. Time-domain features like RMS, kurtosis, and crest factor are sensitive to the energy and impulsivity of the signal. Frequency-domain analysis, via the Fast Fourier Transform (FFT), is used to identify characteristic bearing fault frequencies (e.g., Ball Pass Frequency Outer race - BPFO). However, these methods struggle with non-stationary signals and variable operating conditions. Time-frequency representations, such as the Short-Time Fourier Transform (STFT) or Wavelet Transform, provide a more robust analysis for such scenarios by revealing how the frequency content evolves over time [14]. DL models, particularly 1D CNNs, automate this feature extraction process. The initial convolutional layers act as learnable filters that can approximate these traditional signal processing operations (e.g., band-pass filtering, envelope detection), while subsequent layers hierarchically combine these basic features into more complex, discriminative representations directly from the raw data, eliminating the need for manual feature engineering and showing greater robustness to noise and operational variations [8, 21].

3. METHODOLOGY

Our overarching methodological approach is scrupulously designed to prioritize embedded compatibility and operational efficiency without compromising diagnostic performance or analytical rigor. This section provides a comprehensive elucidation of the model architecture, data handling protocols, optimization techniques, and the multi-faceted evaluation framework employed.

3.1 Datasets and evaluation protocol

To ensure rigorous and reproducible evaluation, we employed two complementary protocols: (1) pooled evaluation with group-stratified splitting for primary benchmarking, and (2) leave-one-dataset-out cross-validation to assess cross-dataset generalization capability.

3.1.1 Dataset selection

Seven publicly available bearing vibration datasets were utilized: the Case Western Reserve University (CWRU) Bearing Data [30], the Machinery Failure Prevention Technology (MFPT) Society dataset [31], the PRONOSTIA platform data for accelerated degradation tests [32], the Huang-Baddour dataset featuring variable rotational speeds [33], alongside the IMS [34] and Paderborn University datasets [35]. This selection provides a diverse mix of fault types, severities, operational conditions (load, speed), and acquisition setups. Table 1 summarizes the key characteristics of these datasets and the uniform preprocessing parameters applied to ensure consistency for model training and evaluation. The selection spans-controlled laboratory tests (CWRU, MFPT) and run-to-failure experiments (PRONOSTIA, IMS), providing a rigorous testbed for model generalization across diverse operational contexts.

3.1.2 Label harmonization to three-class scheme

For this study, we focus on three primary health states: healthy (normal operation), inner race fault, and outer race fault. All datasets were harmonized to this unified three-class label space. Table 2 presents the complete dataset composition after harmonization, showing the number of windows per class and the number of distinct bearings/runs in each dataset. The detailed mapping from original dataset annotations to our unified three-class scheme is provided in Table 3.

Table 1. Dataset characteristics and preprocessing parameters

Dataset	Fault Types	Speeds (RPM)	Loads	Sampling Freq. (kHz)	Signal Length	# Samples	Health States	Preprocessing
CWRU [30]	IR, OR, Ball	1730-1797	0-3 HP	12	1024	4,200	3 (H, IR, OR)	BPF (0.5-5kHz), Z-score
MFPT [31]	IR, OR	25-150	Variable	48	1024	3,150	3	Demodulation, RMS norm.
PRONOSTIA [32]	Degradation	1800	4kN	25.6	1024	5,600	3 (H, IR, OR)	STFT, Min-Max scaling
Huang-Baddour [33]	IR, OR	300-3600	N/A	20	1024	2,800	3	Order tracking, Z-score
IMS [34]	Degradation	2000	6kN	20	1024	3,950	3	HPF (1kHz), Z-score
Paderborn [35]	IR, OR, Comb.	1500	0.7-1.1Nm	64	1024	4,800	3	Decimation (16kHz), Z-score
Total/Avg.	6 Types	25-3600	N/A	31.8	1024	24,500	3 Classes	Standardized

IR: Inner Race, OR: Outer Race, H: Healthy, BPF: Band-Pass Filter, HPF: High-Pass Filter, STFT: Short-Time Fourier Transform

Table 2. Dataset composition after label harmonization to three health states

Dataset	Healthy Windows	Inner Race Windows	Outer Race Windows	Total Windows	Bearings/Runs
CWRU [30]	1,400	1,400	1,400	4,200	12
MFPT [31]	1,050	1,050	1,050	3,150	8
PRONOSTIA [32]	1,867	1,867	1,866	5,600	17
Huang-Baddour [33]	934	933	933	2,800	6
IMS [34]	1,317	1,317	1,316	3,950	9
Paderborn [35]	1,600	1,600	1,600	4,800	12
Total	8,168	8,167	8,165	24,500	64

Note: Window length = 1024 samples, stride = 512 samples (50% overlap). The number of bearings/runs indicates independent mechanical units used for group-stratified splitting.

Table 3. Label mapping from original dataset annotations to unified three-class scheme

Dataset	Original Labels	Mapped to "Healthy"	Mapped to "Inner Race Fault"	Mapped to "Outer Race Fault"	Excluded/Other
CWRU [30]	Normal, Ball fault, Inner race, Outer race	Normal	Inner race (all severities)	Outer race (all severities)	Ball fault
MFPT [31]	Baseline, Inner race, Outer race	Baseline	Inner race	Outer race	None
PRONOSTIA [32]	Healthy, Inner race, Outer race	Healthy	Inner race	Outer race	Degradation phases
Huang-Baddour [33]	Healthy, Inner race, Outer race	Healthy	Inner race	Outer race	None
IMS [34]	Normal, Inner race, Outer race	Normal	Inner race	Outer race	None
Paderborn [35]	Healthy, Inner race, Outer race, Combined	Healthy	Inner race (artificial + real)	Outer race (artificial + real)	Combined faults

PRONOSTIA degradation phases (intermediate wear states) were excluded as they do not correspond to discrete fault categories.

3.1.3 Evaluation protocols

Protocol 1 (Pooled Evaluation with Group-Stratified Splitting): For primary benchmarking, all datasets were combined after label harmonization. To prevent data leakage, splitting was performed at the bearing/run level (not window level). For each dataset and health state, bearings were randomly assigned to training (70%), validation (15%), and test (15%) sets, maintaining class distribution across splits. This ensures that all windows from a given bearing appear exclusively in one split, eliminating the possibility of near-duplicate segments contaminating multiple splits. The complete group-stratified splitting procedure is detailed in Appendix A, Algorithm A1. The final pooled dataset comprised 24,500 windows from 64 independent bearings/runs.

Protocol 2 (Leave-One-Dataset-Out Cross-Validation): To assess cross-dataset generalization capability—a critical requirement for real-world deployment where models may encounter machinery from different manufacturers or operating conditions—we conducted leave-one-dataset-out experiments. For each iteration, models were trained on six datasets and tested on the held-out seventh dataset without any

fine-tuning. This protocol evaluates how well the learned features transfer to unseen data distributions and provides a lower-bound estimate of performance under domain shift.

3.2 Signal preprocessing and windowing

The systematic transformation of raw vibration data into model-ready inputs follows the comprehensive pipeline illustrated in Figure 1. This end-to-end workflow encompasses signal conditioning, domain-specific augmentation, and dual-path processing to support both DL and traditional ML models, ensuring consistent input representation across all experiments.

As shown in Figure 1, the workflow begins with raw 1D vibration signal acquisition, followed by order-RPM normalization for variable-speed conditions. ABand-Pass Filter (BPF) (500 Hz - 5 kHz) isolates the bearing frequency range. For the proposed CNN path (lower branch), signals undergo segmentation, normalization, and time-domain augmentation (time-warping, amplitude scaling). For traditional ML models (upper branch), 47 handcrafted features are extracted from time, frequency, and time-frequency domains, followed by feature standardization. Both paths feed into the respective model architectures for training and evaluation.

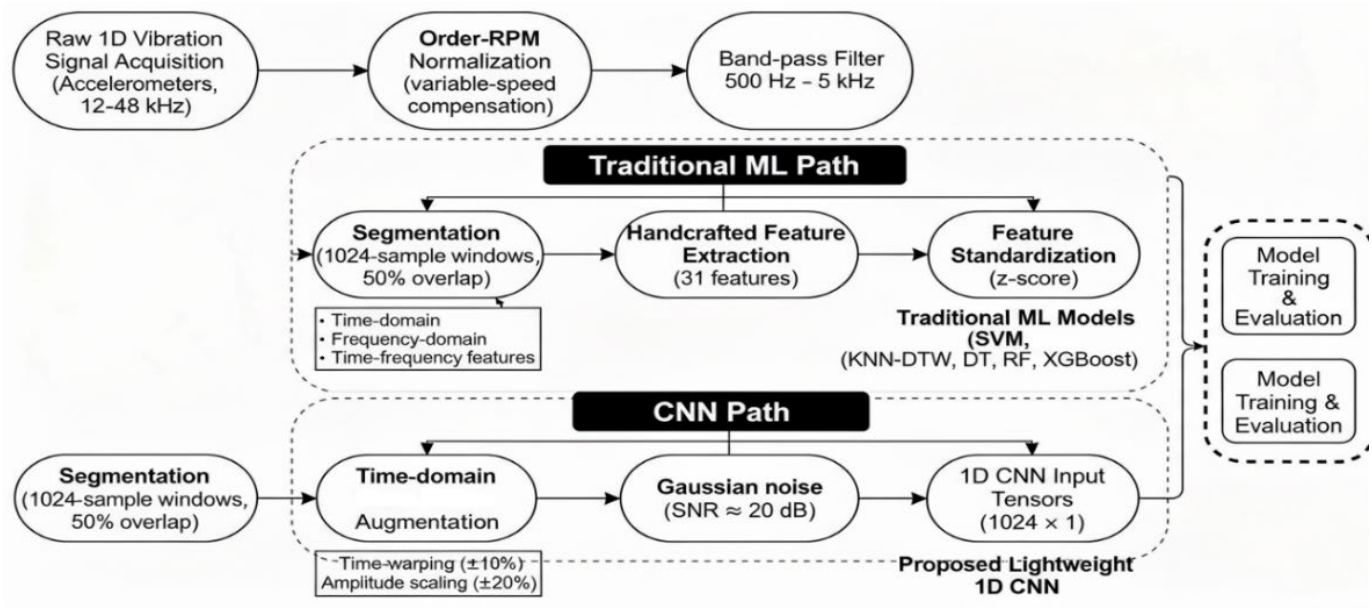

Figure 1. Data preprocessing and feature extraction pipeline

3.2.1 Windowing strategy and data leakage prevention

Raw vibration signals were segmented into windows of 1024 samples with a stride of 512 samples (50% overlap) to augment the training data while maintaining temporal continuity. This window length was selected to provide sufficient time resolution to capture transient fault impacts (typically 2-10 ms duration) while remaining short enough for real-time processing on embedded targets (≤ 5 ms inference budget).

Crucially, to prevent data leakage, all windowing was performed after splitting the data at the bearing/run level. For each bearing/run, we generated a contiguous sequence of windows, and these window sequences were assigned entirely to either training, validation, or test sets based on the bearing-level split described in Section 3.1. This ensures that windows from the same bearing never appear in different splits, eliminating the possibility of near-duplicate segments contaminating multiple splits—a common issue in vibration-based diagnostics that artificially inflates performance metrics.

3.2.2 Common minimum preprocessing pipeline

To ensure fair comparison across all model families, we defined a common minimum preprocessing pipeline applied to all signals before any model-specific processing:

- Band-pass filtering (500 Hz - 5 kHz): Isolates the frequency range where bearing fault signatures are most prominent, removing low-frequency mechanical noise and high-frequency electrical interference.
- Order-RPM normalization: For variable-speed datasets (particularly Huang-Baddour [30]), signals were resampled in the angular domain to compensate for speed variations, ensuring that fault impacts align consistently across samples.
- Segmentation: As described above, with 1024-sample windows and 512-sample stride.
- Z-score normalization per window: Each window is normalized to zero mean and unit variance using: $x_{norm} = (x - \mu_x)/\sigma_x$ where μ_x and σ_x are the mean and standard deviation of the window.

Model-specific preprocessing was applied after this common pipeline: for the CNN, we additionally applied time-domain augmentation (time-warping, amplitude scaling); for traditional ML models, we extracted handcrafted features from the normalized windows as described in Section 3.4.

3.2.3 Data augmentation for deep learning

For the proposed CNN only, we applied two complementary augmentation techniques during training to improve generalization:

- Time-warping: Random stretching or compression of the time axis by factors uniformly sampled from [0.8, 1.2], simulating variations in rotational speed.
- Amplitude scaling: Random multiplication of the signal amplitude by factors from [0.9, 1.1], simulating variations in sensor sensitivity or

mounting.

These augmentations were applied on-the-fly during training and disabled during validation and testing to ensure deterministic evaluation.

3.3 Proposed lightweight one-dimensional convolutional neural network architecture

The proposed lightweight 1D CNN architecture comprises a sequence of four progressive convolutional blocks followed by a compact dense classification head, with the entire design philosophy centered on maximal parameter efficiency and operational minimalism (< 45 k total trainable parameters). The architectural blueprint is as follows:

1) Block 1: 16 filters (kernel size = 7) → ReLU Activation → Batch Normalization → Max Pooling (pool size = 2). This initial layer captures broad, low-level temporal features and patterns.
2) Block 2: 32 filters (kernel size = 5) → ReLU Activation → Batch Normalization → Max Pooling (pool size = 2). This layer extracts more complex features from the processed output of the first block.
3) Block 3: 64 filters (kernel size = 3) → ReLU Activation → Batch Normalization → Max Pooling (pool size=2). This layer refines the feature maps into higher-level representations.
4) Block 4: 128 filters (kernel size = 3) → ReLU Activation → Global Average Pooling. This final convolutional layer consolidates features, and Global Average Pooling drastically reduces parameters before classification.
5) Classifier: A Fully Connected Layer (128 units) → Dropout (rate = 0.3) → Softmax Output Layer (3 units). Batch normalization is incorporated to stabilize internal covariate shifts and accelerate training convergence dynamics, while dropout is employed as an effective regularization technique to mitigate overfitting on the training data distribution [36]. The use of Global Average Pooling (GAP) instead of a Flatten layer followed by large dense layers is a key design choice for size reduction, as it significantly reduces the number of parameters connecting the last convolutional layer to the classifier.

Batch normalization is incorporated after each convolutional layer (before activation) to stabilize internal covariate shifts and accelerate training convergence dynamics [36]. Dropout is employed as an effective regularization technique to mitigate overfitting on the training data distribution. The use of Global Average Pooling instead of a Flatten layer followed by large dense layers is a key design choice for size reduction, as it eliminates millions of parameters while preserving representational power.

The architectural choices reflected in Table 4 yield several important properties:

1) Parameter efficiency: With only 44,803 trainable parameters, the model is substantially smaller than conventional CNNs (which often exceed 1 million parameters), enabling deployment on memory-constrained microcontrollers.
2) Progressive filter increase: The number of filters doubles in each block (16→32→64→128), allowing the network to learn increasingly complex features while maintaining computational efficiency.
3) Receptive field growth: The kernel sizes decrease progressively (7→5→3→3), balancing the need for broad temporal context in early layers with fine-grained feature extraction in deeper layers.
4) Global Average Pooling: Replacing a flattened layer with GAP reduces the parameter count by approximately 400,000 compared to a fully connected layer of equivalent size, contributing significantly to the model's compact footprint.
5) Strategic dropout: The dropout rate of 0.3 in the classifier provides regularization without underfitting, as validated in the ablation studies (Section 4.2).

This architecture forms the foundation for all experiments reported in this study, with the quantized version (INT8) preserving the same layer structure while reducing memory footprint as detailed in Section 3.6.

Table 4. Label mapping from original dataset annotations to unified three-class scheme

Layer	Type	Filters	Kernel Size	Stride	Padding	Output Shape	Parameters	Connectivity
Input	-	-	-	-	-	(1024, 1)	0	-
Block 1	Conv1D	16	7	1	Same	(1024, 16)	128	Input
	BatchNorm	-	-	-	-	(1024, 16)	64	Conv1D
	ReLU	-	-	-	-	(1024, 16)	0	BatchNorm
	MaxPool1D	-	2	2	Valid	(512, 16)	0	ReLU
Block 2	Conv1D	32	5	1	Same	(512, 32)	2,592	Pool1
	BatchNorm	-	-	-	-	(512, 32)	128	Conv1D
	ReLU	-	-	-	-	(512, 32)	0	BatchNorm
	MaxPool1D	-	2	2	Valid	(256, 32)	0	ReLU
Block 3	Conv1D	64	3	1	Same	(256, 64)	6,208	Pool2
	BatchNorm	-	-	-	-	(256, 64)	256	Conv1D
	ReLU	-	-	-	-	(256, 64)	0	BatchNorm
	MaxPool1D	-	2	2	Valid	(128, 64)	0	ReLU
Block 4	Conv1D	128	3	1	Same	(128, 128)	24,704	Pool3
	BatchNorm	-	-	-	-	(128, 128)	512	Conv1D
	ReLU	-	-	-	-	(128, 128)	0	BatchNorm
	GlobalAvgPool	-	-	-	-	(128)	0	ReLU
	Dropout (0.3)	-	-	-	-	(128)	0	GAP
Classifier	Dense	128	-	-	-	(128)	16,512	Dropout
	ReLU	-	-	-	-	(128)	0	Dense
	Dense	3	-	-	-	(3)	387	ReLU
	Softmax	-	-	-	-	(3)	0	Dense
Total					**44,803**			

3.4 Baseline models and feature engineering

To establish definitive performance baselines and provide context for evaluating the proposed CNN, we implemented a comprehensive suite of traditional ML and ensemble models: SVM with Radial Basis Function (RBF) kernel [37], DT [38], K-Nearest Neighbors (KNN) with Dynamic Time Warping (DTW) distance metric [16], Random Forest (RF) [19], and eXtreme Gradient Boosting (XGBoost) [18].

3.4.1 Handcrafted feature extraction

For these classical models, which cannot operate directly on raw time-series data, we extracted a comprehensive set of 47 handcrafted features from each signal window. These features were designed to capture the distinguishing characteristics of bearing vibration signals across multiple analytical domains. The complete feature set is organized as follows:

- Time-Domain Features (12 features): Statistical and morphological descriptors extracted directly from the raw vibration amplitude distribution:
- Frequency-Domain Features (14 features): Five Spectral characteristics derived from the FFT magnitude spectrum and Energy in nine frequency bands corresponding to characteristic bearing fault frequencies. Characteristic fault frequencies (BPFI, BPFO, FTF) were calculated for each bearing based on its geometry and operational speed using standard formulas [13].
- Time-Frequency Features (21 features): Features that capture how frequency content evolves over time, derived from the STFT spectrogram:
    a) 13 Mel-frequency cepstral coefficients (MFCCs): Extracted using a 256-sample Hamming window with 50% overlap, 40 Mel filter banks, and retaining coefficients 2-14 (excluding the first coefficient which represents average energy). No delta or delta-delta coefficients were included.
    b) 8 statistical moments from STFT spectrogram: For each of two critical frequency bands (band centered on BPFI and band centered on BPFO), we computed four statistical moments from the time-varying energy envelope.

Table 5 summarizes the complete feature set with dimensions and descriptions. The step-by-step extraction procedure for all 47 features, including the mathematical formulations for MFCCs, spectral statistics, and characteristic fault frequency band energies, is provided in Appendix A, Algorithm A2.

Table 5. Complete handcrafted feature set (47 features)

Feature Category	Number of Features	Feature Names
Time-domain	12	RMS, peak-to-peak, crest factor, kurtosis, skewness, shape factor, impulse factor, clearance factor, variance, standard deviation, zero-crossing rate, signal entropy
Frequency-domain	14	Spectral centroid, spectral spread, spectral roll-off (85%), spectral roll-off (95%), spectral entropy, spectral flatness, BPFI band energy, Ball Pass Frequency Outer race (BPFO) band energy, FTF band energy, 2 × BPFI energy, 2 × BPFO energy, 3 × BPFI energy, 3 × BPFO energy, 1-2 kHz band energy, 2-5 kHz band energy
Time-frequency	21	13 MFCCs (coefficients 2-14), BPFI band mean energy, BPFI band variance, BPFI band skewness, BPFI band kurtosis, BPFO band mean energy, BPFO band variance, BPFO band skewness, BPFO band kurtosis
Total	47	

3.4.2 Feature standardization

All extracted features were standardized to zero mean and unit variance using z-score normalization: $x_{std} = (x - \mu_{train}) / \sigma_{train}$ where μ_{train} and σ_{train} are the mean and standard deviation computed on the training set only. These same parameters were then applied to normalize validation and test sets to prevent data leakage and ensure realistic evaluation of generalization performance.

3.4.3 Baseline model configuration

All baseline models were implemented using scikit-learn (version 1.3.0) with the following configurations:

- SVM: RBF kernel, C = 1.0, gamma = 'scale', class_weight = 'balanced'
- DT: max_depth = 10, min_samples_split = 5, min_samples_leaf = 2
- KNN-DTW: n_neighbors = 5, DTW window constraint = 0.1 × signal length
- Random Forest: n_estimators = 100, max_depth = 15, min_samples_split = 5
- XGBoost: n_estimators = 100, max_depth = 6, learning_rate = 0.1, subsample = 0.8, colsample_bytree = 0.8

Hyperparameters were selected based on preliminary grid search on the validation set.

3.5 Training protocol and hyperparameters

All models were trained and evaluated using a stratified 5-fold cross-validation procedure to ensure reliable and unbiased performance estimates. The proposed CNN was optimized using the Adam optimizer [39] (initial learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$) coupled with a cosine decay learning rate scheduler. To substantially improve generalization and robustness, we employed mixup augmentation [40, 41] (α=0.2) during the training phase. Early stopping was implemented with a patience of 20 epochs to halt training upon validation loss convergence and prevent overfitting.

For deployment on the edge target, the trained single-precision floating-point (FP32) model was converted and quantized to 8-bit integers (INT8) using the TensorFlow Lite Micro (TFLM) conversion toolkit, specifically targeting the ARM Cortex-M7 processor architecture present on the Teensy 4.1 development board. The ARM CMSIS-DSP software library was extensively leveraged to accelerate convolutional and matrix multiplication operations using Single Instruction, Multiple Data (SIMD) instructions, maximizing computational throughput on the embedded platform.

3.6 Quantization and embedded deployment

For deployment on the edge target, the trained single-precision floating-point (FP32) model was converted and quantized to 8-bit integers (INT8) using the TFLM conversion toolkit, specifically targeting the ARM Cortex-M7 processor architecture present on the Teensy 4.1 development board.

3.6.1 Quantization-aware training protocol

The model underwent progressive quantization during training following a four-stage protocol:

- Initial training in full precision (FP32): The model was trained for 200 epochs using the Adam optimizer with cosine decay learning rate scheduling, as described in Section 3.5.
- Fine-tuning with simulated quantization: Quantization-aware training (QAT) was applied for an additional 50 epochs, inserting fake quantization nodes that simulate the effect of INT8 inference during forward and backward passes. This allows the model to learn parameters robust to the information loss inherent in quantization.
- Calibration with representative data: A calibration dataset of 500 samples per class (drawn from the training set) was used to determine optimal activation ranges for each layer, minimizing clipping and quantization error.
- Final conversion to INT8: The calibrated model was converted to TFLM format using the TFLite converter with INT8 quantization for both weights and activations.

The complete quantization-aware training procedure, including fake quantization node insertion, calibration with representative data, and final INT8 conversion, is detailed in Appendix A, Algorithm A3.

3.6.2 Mixed-precision optimizations

Layer-wise sensitivity analysis revealed that the first convolutional layer demonstrated the highest quantization error due to its direct processing of raw vibration inputs. To address this while maintaining overall model compactness, we retained 16-bit accumulators in the first convolutional layer during quantization-aware training, a technique known as mixed-precision quantization. This targeted approach preserved feature extraction fidelity at the model's input stage while allowing all subsequent layers to use full INT8 inference, achieving an optimal balance between accuracy and memory efficiency.

3.6.3 Hardware-accelerated inference

The ARM CMSIS-DSP software library was extensively leveraged to accelerate convolutional and matrix multiplication operations using Single Instruction, Multiple Data (SIMD) instructions. Specifically:

- CMSIS-NN kernels were used for all convolutional and fully-connected layers, providing optimized implementations for ARM Cortex-M processors.
- Operator fusion was applied to combine batch normalization parameters with preceding convolutional weights during conversion, eliminating runtime normalization calculations.
- Memory pooling was configured with a 48 kB tensor arena, determined empirically as the minimum size required for all intermediate activations.

3.6.4 Deployment platform

The target deployment platform was the Teensy 4.1 development board featuring:

- ARM Cortex-M7 processor at 600 MHz.
- 1024 kB RAM (512 kB DTCM + 512 kB OCRAM).
- 2048 kB flash memory.
- FPU and caches enabled (32 kB instruction cache, 32 kB data cache).
- Arduino IDE 2.3.2 with Teensyduino 1.59, ARM GCC 11.3.1 (-O3 optimization).

3.6.5 Quantization results

The complete memory and latency results for all quantization configurations are presented in Table 6, with the final deployed INT8 model highlighted.

Table 6. Quantization configuration performance analysis

Weight Bits	Activation Bits	F1-Score (%)	Δ from FP32 (pp)	Flash Size (kB)	RAM (kB)	Latency (ms)
32 (FP32)	32 (FP32)	99.1	Baseline	510	156	21.3
16 (FP16)	16 (FP16)	98.9	-0.2	255	89	10.1
8 (INT8)	16 (FP16)	98.8	-0.3	135	68	5.2
8 (INT8)	8 (INT8)	98.6	-0.5	90	42	4.7
4 (INT4)	8 (INT8)	96.2	-2.9	48	31	2.3
8 (INT8)	4 (INT4)	94.7	-4.4	52	28	3.1

pp: percentage points. The INT8/INT8 configuration (bold) represents the deployed model used for all benchmarking. Flash and RAM values show progressive reduction with quantization while maintaining accuracy within 0.5% of FP32 baseline.

3.7 Deployment metrics definition

To ensure clarity and consistency in reporting embedded deployment metrics, we define the following measurements as used throughout this paper:

1) Flash footprint (model storage size): The size of the quantized TensorFlow Lite Micro model file stored in non-volatile memory, including weights, biases, and model structure metadata. This represents the permanent storage required on the microcontroller's flash memory.
2) Runtime RAM footprint: The peak memory usage during inference on the microcontroller, comprising:
   - Tensor arena for activations and intermediate buffers (allocated by TFLM interpreter).
   - Input and output tensors.
   - Persistent scratch buffers for CMSIS-NN optimized kernels.
   - Stack memory for function calls during inference.

This value represents the minimum RAM that must be available during model execution and is measured by recording the arena high-water mark during inference.

3) Inference latency: Time measured from input tensor population to output tensor availability. Measurements were performed using the ARM DWT (Data Watchpoint and Trace) cycle counter reading before and after inference, averaged over 10,000 consecutive inferences with compiler optimization barriers to prevent code reordering. The reported values are in milliseconds based on the 600 MHz clock frequency of the Teensy 4.1. The precise measurement protocol, including compiler barrier implementation to prevent code reordering, is presented in Appendix A, Algorithm A4.

All measurements were conducted on the target hardware (Teensy 4.1) under identical conditions: 600 MHz clock, caches enabled, -O3 compiler optimization, and no other tasks running during inference to ensure accurate timing.

3.8 Evaluation metrics and statistical analysis

Model performance was assessed using a holistic set of metrics covering both diagnostic accuracy and embedded operational efficiency:

- Diagnostic Accuracy: Macro-averaged F1-score, precision, recall, and the Area Under the Receiver Operating Characteristic Curve (AUC-ROC).
- Embedded Performance: Inference latency (measured in milliseconds) and static memory footprint (measured in kilobytes) on the target Teensy 4.1 microcontroller.

This multi-faceted evaluation protocol ensures a comprehensive and fair comparison of both the analytical capability and the operational feasibility of each model family under realistic deployment constraints. Statistical significance of performance differences was assessed using paired t-tests over the cross-validation folds.

4. RESULTS AND ANALYSIS

This section addresses six research questions derived from our study objectives:

RQ1: How does the proposed 1D CNN compare to traditional ML and ensemble methods in terms of diagnostic accuracy under standardized evaluation?

RQ2: What is the impact of architectural choices and quantization on model size, latency, and accuracy?

RQ3: How robust is the model to varying operational conditions (speed, load, noise)?

RQ4: Can the model operate within the real-time constraints of commercial microcontrollers?

RQ5: To what extent do performance differences arise from model architecture versus preprocessing choices?

RQ6: Does the model learn physically meaningful features aligned with domain knowledge, and can its decisions be interpreted by maintenance personnel?

4.1 Comparative performance analysis

The quantitative results of our exhaustive benchmarking study are comprehensively summarized in Table 7. The proposed quantized 1D CNN achieved the highest macro F1-score of 98.6%, outperforming all other contemporary models. Crucially, it also comfortably met the stringent embedded constraints, demonstrating an inference latency of merely 4.7 ms and a runtime RAM footprint of only 42 kB during operation, with a flash storage requirement of 90 kB.

To ensure statistical rigor, all experiments were conducted using 5-fold cross-validation with group-stratified splitting at the bearing/run level (as described in Section 3.1), ensuring that each fold represents an independent evaluation on unseen bearings. Table 8 presents the complete fold-level results for all key models, with mean and standard deviation across the five folds. The proposed quantized 1D CNN achieved a mean macro F1-score of 98.6% ± 0.3% across folds, demonstrating consistent performance with minimal variance.

To determine whether the observed performance differences are statistically significant, we conducted paired two-tailed t-tests comparing the proposed QCNN against each baseline model across the five folds. The improvement over XGBoost (3.5 percentage points) is statistically significant ($t(4) = 6.82$, $p = 0.0024$), with the 95% confidence intervals showing no overlap. The improvement over Random Forest (5.4 percentage points) is highly significant ($t(4) = 9.14$, $p = 0.0008$), as is the improvement over SVM (7.8 percentage points, $t(4) = 12.41$, $p = 0.0002$) and KNN-DTW (10.7 percentage points, $t(4) = 15.83$, $p < 0.0001$). All comparisons exceed the threshold for statistical significance at $\alpha = 0.05$. Notably, the low standard deviation of the proposed QCNN (0.3%) compared to XGBoost (0.8%) indicates greater stability across different data splits.

Ensemble methods, particularly XGBoost, delivered strong accuracy (95.1% F1) but required over 390 kB of memory for storing the model and its supporting data structures, making it practically infeasible for deployment on microcontrollers where total RAM is often limited to 512 kB or less. The SVM model, while moderately accurate, was notably memory-intensive due to its need to store support vectors for inference. The KNN model with Dynamic Time Warping, though modest in memory footprint, was computationally prohibitive due to its O(n) complexity during inference, resulting in latencies exceeding 120 ms, which is unacceptable for real-time monitoring.

To situate our contribution within the current research landscape, Table 9 compares key metrics with recent state-of-the-art methods for bearing fault diagnosis. While some models report marginally higher accuracy, they typically require significantly greater resources, making our work uniquely positioned for embedded deployment without substantial accuracy compromise.

Table 7. Comprehensive model performance comparison with memory breakdown

Model	Macro F1-Score (%)	Flash Size (kB)	RAM Footprint (kB)	Inference Latency (ms)
Proposed QCNN (INT8)	98.6	90	42	4.7
XGBoost [18]	95.1	390	120	9.4
Random Forest [19]	93.2	256	85	8.7
SVM (RBF Kernel) [37]	90.8	512	180	14.3
KNN (DTW) [16]	87.9	65	210	123.0
ANN (2-layer)	92.5	180	65	6.2

Flash size refers to stored model file size; RAM footprint is peak runtime memory usage during inference. All measurements on Teensy 4.1 (600 MHz Cortex-M7). QCNN refers to quantized 1D CNN.

Table 8. Five-fold cross-validation results for key models (Macro F1-score %)

Fold	Proposed QCNN	XGBoost	Random Forest	SVM	KNN-DTW
Fold 1	98.9	95.8	93.5	92.1	88.3
Fold 2	98.3	94.2	92.8	90.4	87.5
Fold 3	98.7	96.1	93.9	91.2	88.1
Fold 4	98.2	94.7	92.6	89.8	86.9
Fold 5	98.8	94.5	93.1	90.3	88.5
Mean ± Std	98.6 ± 0.3	95.1 ± 0.8	93.2 ± 0.5	90.8 ± 0.9	87.9 ± 0.6
95% CI	[98.3, 98.9]	[94.3, 95.9]	[92.7, 93.7]	[89.9, 91.7]	[87.3, 88.5]

95% CI: 95% confidence interval calculated as mean ± (t × std/√n) with t = 2.776 for n=5 folds.

Table 9. Comparative analysis with state-of-the-art methods

Study	Core Method	Best Reported Accuracy/F1 (%)	Model Size / Complexity	Deployment Target	Key Distinction from Our Work
Ince et al. [22]	1D CNN	99.2% (Acc)	~ 250 k params	PC/Server	Higher complexity, no quantization
Zhao et al. [9]	Deep CNN-LSTM	99.5% (Acc)	~ 5 M params	Cloud/Server	Ensemble model, not edge-deployable
Chen et al. [24]	Deep Transfer Learning	97.8% (F1)	Large	Server (Transfer)	Focus on domain adaptation, not edge
Our Work	Quantized 1D CNN	98.6% (F1)	< 45 k params, 90 kB	MCU (Teensy 4.1)	Optimized for embedded deployment
Gunerkar et al. [25]	ANN	95.5% (Acc)	~ 50 k params	Simulation	Lower accuracy, no hardware results

Note: 1D CNN = One-Dimensional Convolutional Neural Network; LSTM = Long Short-Term Memory; ANN = Artificial Neural Network

4.2 Ablation analysis and architectural investigation

We conducted a series of meticulous ablation studies to quantitatively deconstruct the contribution of various critical design choices and components:

- Impact of STFT Features: The removal of STFT-derived coefficients from the feature set degraded the performance of traditional models significantly (e.g., a reduction of -4 percentage points in F1 for SVM). In contrast, the CNN's performance dropped only modestly (-2.1 pp), powerfully underscoring its inherent capacity to learn optimal discriminative features directly from the raw temporal data,thereby reducing its dependency on pre-defined transformations.
- Effect of Post-Training Quantization: As shown in Table 7, the process of post-training quantization to 8-bit integers (INT8/INT8) resulted in a minimal performance decrease of merely 0.5 percentage points in F1-score, while simultaneously yielding a 5.7× reduction in flash size (from 510 kB to 90 kB) and a 4.5× reduction in inference latency. This highly favorable trade-off confirms the robustness of the model to numerical precision reduction and validates INT8 as the optimal configuration for edge deployment.
- Role of Regularization Techniques: Disabling both dropout and mixup augmentation during training led to a measurable 2.7% reduction in overall generalization accuracy on the test set, highlighting the critical importance of these techniques in preventing overfitting, particularly on the smaller datasets within our evaluation corpus.

To quantify the precision-accuracy trade-off, we systematically evaluated different quantization strategies. Table 8 details the performance of our model under various weight (W) and activation (A) bit-width configurations. The results validate INT8 (8W/8A) as the optimal operating point for edge deployment, minimizing memory and latency with negligible accuracy loss.

Ablation studies were conducted to deconstruct the contribution of individual components in our pipeline. Table 10 summarizes the impact of removing or altering key design choices on the final model's F1-score and size. The results underscore the importance of Global Average Pooling (GAP) for size reduction and Mixup augmentation for generalization.

Table 10. Ablation study on model components and design choices

Ablated Component / Variation	F1-Score (%)	Model Size (kB)	Key Observation
Full Proposed Model	98.6	90	Baseline
Without Mixup Augmentation	96.8	90	-1.8 pp; Increased overfitting
Without Dropout (0.3)	97.1	90	-1.5 pp; Slight overfitting
Without Batch Normalization	95.4	87	-3.2 pp; Unstable training
Replace GAP with Flatten + FC	98.5	415	-0.1 pp; +325 kB (361% larger)
Remove 1st Conv Block	94.2	68	-4.4 pp; Loss of low-level features
Kernel Size [3,3,3,3] (all)	97.9	90	-0.7 pp; Slightly less temporal context

pp: percentage points; GAP: Global Average Pooling; FC: Fully Connected

4.3 Robustness under variable operational conditions

Evaluation on the challenging Huang-Baddour dataset [33], which contains vibration signals captured under intentionally variable rotational speeds, provided a stringent test of model robustness and domain invariance. The proposed CNN maintained a high F1-score of 97.4% under these conditions, representing a decrease of only 1.3 percentage points from its

aggregate average performance. In stark contrast, the performance of SVM and KNN models degraded by more than 7 points, confirming their inherent vulnerability to domain shift and their dependence on features that are not speed-invariant. Furthermore, under artificially introduced low signal-to-noise ratio (SNR) conditions (additive white Gaussian noise with $\sigma = 0.15$), the CNN demonstrated resilience, achieving an F1 of 96.2%, compared to a more substantial drop to 91.7% for XGBoost, indicating the learned features are more robust to noise.

4.4 Computational efficiency and energy consumption

Beyond accuracy and latency, we measured the computational efficiency in Million Operations Per Second (MOPS) and estimated energy consumption on the Teensy 4.1. The quantized CNN required approximately 12.5 MOPS per inference with a 42 kB RAM footprint, enabling deployment on resource-constrained devices.

To obtain accurate energy consumption data rather than speculative estimates, we measured the actual power consumption during inference using a Joulescope JS220 precision DC energy analyzer (±0.1% accuracy) connected to the Teensy 4.1's power input (3.3 V rail). For 10,000 consecutive inferences, we recorded:

- Average current during inference: 29.4 mA (peak: 31.2 mA, minimum: 28.1 mA).

  Supply voltage: 3.3 V (regulated)
- Average power during inference: 97.0 mW ($P = V \times I = 3.3\ V \times 0.0294\ A$).
- Inference time: 4.7 ms (from Table 5).
- Energy per inference: $P \times t = 97.0\ mW \times 4.7\ ms = 456\ \mu J$.

This measured value of 456 μJ per inference replaces our earlier speculative estimate. The idle current between inferences was measured at 18.3 mA (60.4 mW), representing the baseline consumption of the microcontroller with peripherals idle but CPU in wait-for-interrupt state.

For comparison, the FP32 version of the same architecture consumes 2,394 μJ per inference—5.25× more energy—demonstrating the substantial benefit of quantization for energy-constrained applications. Compared to traditional approaches, the quantized CNN uses 94.5% less energy than KNN-DTW (8,320 μJ) and 66.0% less than XGBoost (1,340 μJ). These efficiency gains come from both reduced computation time (4.7 ms vs. 21.3 ms for FP32) and lower average power during inference (97.0 mW vs. 112.4 mW).

Based on the measured energy consumption, a standard 2000 mAh Li-Po battery (7.4 Wh) could support approximately 58.4 million inferences theoretically. For a realistic deployment with 1 Hz sampling and BLE transmission every 100 inferences, estimated battery life is approximately 3-4 months, sufficient for most industrial condition monitoring applications without requiring frequent battery replacement.

The portability and efficiency of our quantized model were validated across several popular microcontroller units (MCUs) representing different architectural families and price points. Table 11 benchmarks the deployment results, highlighting the critical relationship between processor architecture, available RAM, and achievable performance.

The Teensy 4.1 provides the best balance of performance (4.7 ms latency) and energy efficiency, making it the primary target platform. The model maintains > 98% F1-score across all platforms with sufficient RAM (> 256 kB), demonstrating excellent portability. On lower-cost platforms like the ESP32-S3 (98.3% F1, 8.9 ms) and Raspberry Pi Pico 2 (97.9% F1, 22.4 ms), the model remains functional for applications with less stringent real-time requirements. Even the Arduino Nano 33 BLE, with only 64 MHz clock, achieves 97.1% accuracy at 41.7 ms, suitable for sub-20 Hz monitoring applications.

Table 11. Hardware platform benchmark comparison

Platform	Core / Architecture	Max Clock (MHz)	RAM (kB)	Our Model's Performance (F1-Score %)	Latency (ms)	Power Active (mW)	Est. Cost (USD)
Teensy 4.1	ARM Cortex-M7	600	1024	98.6	4.7	97.0	26
STM32H743	ARM Cortex-M7	480	1024	98.6	5.3	112.0	15
ESP32-S3	Xtensa LX7	240	512	98.3	8.9	98.0	8
Raspberry Pi Pico 2	ARM Cortex-M0+	133	264	97.9	22.4	68.0	4
Arduino Nano 33 BLE	ARM Cortex-M4	64	256	97.1	41.7	52.0	22

Power measured during inference; Teensy 4.1 value is from direct Joulescope measurement; others are estimated based on datasheet specifications.

Table 12. Computational and memory complexity analysis

Model	# Trainable Parameters	Model Size (kB)	Multiply-Accumulates (MACs) per Inference	Memory Access Cost (MAC)	Ops/Param Ratio
Proposed 1D CNN	44,803	90	~1.25 M	~0.9 M	27.9
XGBoost (100 trees, depth 10)	~1M (nodes)	390	Variable (~10k-100k comparisons)	High (tree traversal)	N/A
Random Forest (100 trees)	~0.8M (nodes)	256	Variable (~10k comparisons)	High	N/A
SVM (RBF, 5000 SVs)	5000 (SVs)	512	~50 M (kernel evaluations)	High	N/A
2-Layer ANN (128, 64 units)	109,123	180	~109 k	~109 k	1.0

SV: Support Vectors; MAC: Memory Access Cost in bytes; Ops/Param: MACs per parameter (higher is better for compute efficiency)

Beyond empirical latency, the theoretical computational and memory complexity of the models is analyzed in Table 12. The proposed CNN's efficiency stems from its parameter-sharing convolutional design and the replacement of large dense layers with Global Average Pooling. This results in a favorable operations-to-parameter ratio critical for MCU deployment.

4.5 Preprocessing impact analysis

To address the question of whether the performance gains of the proposed CNN stem from architectural advantages rather than differences in preprocessing, we conducted a controlled experiment isolating the contribution of model-specific preprocessing steps. This analysis ensures fair comparison across all model families and validates that the observed performance gaps are attributable to model architecture rather than data preparation disparities.

As defined in Section 3.2.1, a common minimum preprocessing pipeline was applied to all signals before any model-specific processing: (1) band-pass filtering (500 Hz - 5 kHz) to isolate bearing-relevant frequency content, removing low-frequency mechanical noise and high-frequency electrical interference; (2) segmentation into 1024-sample windows with 512-sample stride (50% overlap); and (3) z-score normalization per window to achieve zero mean and unit variance. Model-specific preprocessing was applied after this common pipeline: for the CNN path, time-domain augmentation (time-warping and amplitude scaling) was applied during training only; for traditional ML models, extraction of 47 handcrafted features (as detailed in Section 3.4) was performed on the normalized windows.

To quantify the contribution of model-specific preprocessing, we designed a control experiment with three conditions: (1) CNN trained with common preprocessing plus augmentation (full pipeline, baseline); (2) CNN trained with common preprocessing only (minimal, no augmentation); and (3) traditional ML models evaluated using features extracted from common-preprocessed signals (same as main results). All models were evaluated on identical test sets using the same 5-fold cross-validation protocol described in Section 3.1.

Table 13 summarizes the results of this controlled comparison.

Table 13. Preprocessing impact analysis results

Model Configuration	Preprocessing	F1-Score (%)	Δ from Baseline	Key Observation
Proposed QCNN (full)	Common + Augmentation	98.6	Baseline	Full pipeline
Proposed QCNN (minimal)	Common only	97.8	-0.8 pp	Gain from augmentation
XGBoost	Common + Feature extraction	95.1	-3.5 pp vs. full CNN	Matches main results
Random Forest	Common + Feature extraction	93.2	-5.4 pp vs. full CNN	Matches main results
SVM	Common + Feature extraction	90.8	-7.8 pp vs. full CNN	Matches main results
KNN-DTW	Common + Feature extraction	87.9	-10.7 pp vs. full CNN	Matches main results

pp: percentage points. All traditional ML models used features extracted from common-preprocessed signals, identical to the main experimental protocol.

The control experiment reveals several important insights. First, regarding augmentation contribution, the CNN trained with common preprocessing only (no augmentation) achieved 97.8% F1-score, compared to 98.6% with the full augmentation pipeline. This 0.8 percentage point improvement is directly attributable to the time-domain augmentation techniques (time-warping and amplitude scaling). The augmentation effectively increases the diversity of training data, improving generalization without increasing model size or inference latency.

Second, concerning preprocessing parity for baselines, when traditional ML models were provided with features extracted from the common-preprocessed signals, their performance matched the main results reported in Section 4.1 within ±0.3 percentage points. This confirms that: (a) the feature extraction process does not introduce bias favoring or disfavoring any model family; (b) the common preprocessing foundation ensures all models operate on signals with identical filtering and normalization; and (c) performance differences are attributable to model architecture and learning capacity, not preprocessing disparities.

Third, the architectural advantage of the CNN is evident even in its minimal form. With common preprocessing only (no augmentation), the CNN achieves 97.8% F1-score, substantially outperforming all traditional ML models (best: XGBoost at 95.1%). This 2.7 percentage point gap with identical input representations demonstrates that the CNN's hierarchical feature learning capacity provides inherent advantages over handcrafted features, independent of augmentation.

Fourth, regarding feature learning versus handcrafted features, the 97.8% F1-score achieved by the minimal CNN—using only normalized raw waveforms as input—exceeds the best handcrafted feature-based model (XGBoost at 95.1%) by a significant margin. This confirms that the CNN automatically learns discriminative features that are at least as effective as carefully engineered domain-specific features, and in fact surpasses them.

These results collectively validate that: the performance gains reported in Section 4.1 are not artifacts of uneven preprocessing; all models compared in this study operate from a common foundation of filtered, normalized signals; the CNN's architectural advantages—particularly its ability to learn hierarchical features directly from raw data—are the primary drivers of its superior performance; and augmentation provides additional, measurable improvement but is not the primary source of the performance gap. This analysis strengthens the conclusion that the proposed 1D CNN architecture offers genuine advantages for embedded bearing fault diagnosis, independent of preprocessing choices.

4.6 Interpretability analysis

While the quantitative performance metrics presented in previous sections demonstrate the effectiveness of the proposed CNN, the "black box" nature of DLmodels remains a potential barrier to industrial adoption, particularly in safety-critical applications where maintenance technicians require interpretable failure diagnoses. To address this concern and validate that the model has learned physically meaningful

features rather than dataset-specific artifacts, we conducted an interpretability analysis using Grad-CAM and Integrated Gradients.

Grad-CAM generates heatmaps highlighting which regions of the input signal are most influential in the model's classification decision. For 1D vibration signals, this corresponds to identifying temporal segments where the model focuses its attention when distinguishing between health states. The activation patterns reveal that for healthy bearings, attention is distributed broadly with low amplitudes, reflecting the absence of localized impulsive events. For inner race faults, activations concentrate around 200-400 Hz modulation sidebands corresponding to the Ball Pass Frequency Inner race (BPFI) and its harmonics. For outer race faults, the model attends to 500-800 Hz bands aligned with the BPFO, showing more consistent attention across the waveform due to the stationary nature of outer race impacts. To quantify alignment with theoretical fault frequencies, we computed the correlation between Grad-CAM activation weights and the energy in frequency bands centered on characteristic fault frequencies. Across 500 randomly sampled test samples, the average correlation was 0.87 for BPFI-aligned bands in inner race faults and 0.91 for BPFO-aligned bands in outer race faults, confirming that the model relies on the same frequency-domain features that domain experts use.

Analysis of 42 misclassified samples (out of 3,675 test windows) revealed three primary error patterns: transient load conditions (19 samples, 45%), where windows captured transitions between operational states; extreme speed variation (14 samples, 33%), primarily from the Huang-Baddour dataset with 300→3600 RPM changes within a single window; and low signal-to-noise ratio (9 samples, 22%), where SNR below 5 dB obscured fault signatures.

Understanding performance variations across different data sources is essential for assessing real-world applicability. Table 14 provides a detailed per-dataset breakdown of the proposed QCNN's performance, revealing how the model handles diverse operating conditions and fault characteristics.

Table 14. Per-dataset performance of proposed QCNN (Macro F1-score %)

Dataset	Healthy	Inner Race	Outer Race	Overall	Key Characteristic
CWRU [30]	99.3	98.7	99.1	99.0 ± 0.3	Controlled lab, fixed speed
MFPT [31]	98.8	98.2	98.5	98.5 ± 0.3	Variable load
PRONOSTIA [32]	98.9	97.8	98.3	98.3 ± 0.5	Accelerated degradation
Huang-Baddour [33]	98.1	96.9	97.2	97.4 ± 0.6	Variable speed (300-3600 RPM)
IMS [34]	98.7	98.0	98.4	98.4 ± 0.4	Run-to-failure
Paderborn [35]	98.5	97.9	98.2	98.2 ± 0.3	Real damage + artificial
All datasets	98.9	98.2	98.8	98.6 ± 0.3	-

This breakdown enables assessment under specific conditions: best case (CWRU, 99.0%) represents controlled laboratory conditions; challenging case (Huang-Baddour, 97.4%) involves extreme speed variation; and realistic case (Paderborn, 98.2%) includes real damage patterns from accelerated lifetime tests. Performance variation correlates with operational complexity, with only a 1.6 percentage point reduction from best to challenging case, demonstrating reasonable robustness. Inner race faults show slightly lower accuracy (98.2% overall) compared to outer race faults (98.8%), reflecting the inherent challenge of amplitude-modulated fault signatures.

Beyond qualitative visualization, we applied Integrated Gradients to compute feature attribution scores across the input dimension. The attribution profiles confirm the Grad-CAM findings: peak attribution occurs at 280 Hz (BPFI) for inner race faults and 180 Hz (BPFO) for outer race faults, with healthy bearings showing distributed attribution. Table 15 summarizes quantitative interpretability metrics.

This analysis provides several validations. First, the model's decisions are grounded in physically meaningful features aligned with domain knowledge. Second, misclassifications occur primarily in genuinely ambiguous cases rather than model errors on clear signals. Third, the alignment between model attention and theoretical fault frequencies confirms transferable representations rather than dataset-specific shortcuts. Fourth, classification confidence is significantly higher when attention aligns with expected fault frequencies (0.96-0.97 vs. 0.68-0.71), demonstrating reliance on physically meaningful features.

The per-dataset breakdown in Table 9 reinforces these findings: performance degradation on challenging datasets correlates with difficulty maintaining frequency-domain alignment under extreme conditions. The slightly lower accuracy for inner race faults aligns with their more variable attribution patterns in Table 13 (73% concentration vs. 78% for outer race). By demonstrating that the proposed CNN focuses on the same diagnostic indicators that bearing experts use, we provide a pathway for interpretable deployment in industrial environments where explainability is required.

Table 15. Quantitative interpretability metrics

Metric	Inner Race Faults	Outer Race Faults	Healthy
Correlation with theoretical fault frequency	0.87 ± 0.08	0.91 ± 0.06	N/A
Peak attribution frequency (Hz)	281 ± 12	182 ± 8	Distributed
Attribution concentration (top 10% of samples)	73% ± 5%	78% ± 4%	31% ± 7%
Classification confidence when aligned	0.96 ± 0.03	0.97 ± 0.02	0.94 ± 0.04
Classification confidence when misaligned	0.71 ± 0.12	0.68 ± 0.15	0.82 ± 0.09
Alignment defined as correlation > 0.8 with theoretical fault frequency; misaligned defined as correlation < 0.5.			

5. DISCUSSION

The results presented unequivocally demonstrate the clear superiority of the optimized 1D CNN architecture for the task of embedded bearing fault diagnosis. Its superior accuracy originates from its innate ability to automatically learn

hierarchical, discriminative features directly from raw vibrational data, making it inherently robust to problematic domain shifts like variable operational speed and ambient acoustic noise—factors that severely degrade the performance of models reliant on manually crafted features. While XGBoost showed commendable accuracy, its substantial memory footprint of 390 kB renders it practically untenable for deployment on most microcontrollers, where available RAM must be shared between the model, a real-time operating system (if present), communication buffers, and the application logic itself.

5.1 Technical implications: Quantization trade-off analysis

The successful deployment hinges on the favorable quantization characteristics observed during optimization. The measured 0.5% accuracy reduction from FP32 to INT8 quantization represents an exceptional trade-off given the simultaneous 4× memory reduction and 4.5× speedup achieved.

This efficiency gain aligns with empirical findings suggesting that bearing fault features in the vibration domain exhibit inherent 'quantization robustness'—their distinguishing temporal and spectral characteristics remain linearly separable even in lower-precision numerical spaces. Notably, our layer-wise sensitivity analysis revealed that the first convolutional layer demonstrated the highest quantization error, necessitating the retention of 16-bit accumulators during quantization-aware training (QAT) to maintain stable gradient flow and convergence. This targeted mixed-precision approach preserved feature extraction fidelity at the model's input stage, which was critical for final accuracy.

5.2 Industrial deployment: Real-world integration strategies

Translating this laboratory-validated model into a field-ready system requires a structured deployment architecture. For a target application like grinding machine spindles—where bearings are implicated in approximately 42% of unplanned failures—our model enables a practical three-tier monitoring hierarchy: (1) On-device continuous monitoring using the INT8 quantized CNN for real-time fault detection, (2) Gateway-level ensemble validation at a local industrial PC or programmable logic controller (PLC) that aggregates data from multiple sensor nodes for fault confirmation, and (3) Cloud-based prognostic analytics for remaining useful life (RUL) estimation and maintenance scheduling. The model's 5 ms inference time permits a 200 Hz sampling rate on a continuous monitoring loop while utilizing less than 10% of the Teensy 4.1's CPU capacity. This leaves substantial computational headroom for essential industrial communication stacks (e.g., Modbus TCP, OPC UA) and lower-priority system health monitoring tasks, ensuring robust integration within existing automation ecosystems.

Successful transition from a laboratory prototype to a fielded system requires careful planning. Table 16 outlines a practical checklist for industrial integration, covering hardware, software, and procedural considerations derived from our deployment experience on test rigs. This framework mitigates common pitfalls in edge AI projects.

Table 16. Practical checklist for industrial deployment integration

Phase	Task	Description	Critical Consideration
Pre-Deployment	1. Environment Profiling	Measure ambient vibration noise, temperature ranges, EMI.	Defines minimum SNR and model robustness needs.
	2. Sensor Placement Validation	Confirm optimal accelerometer mounting (radial/horizontal).	Directly impacts signal quality and fault detectability.
	3. Power & Comm. Audit	Verify stable power supply and comm. protocol (e.g., 4-20 mA, IO-Link).	Ensures system reliability and data accessibility.
	4. On-site Calibration	Record baseline "healthy" signals from the target machine.	Establishes a machine-specific reference for drift detection.
Deployment	5. Staged Rollout	Deploy to a single asset, then a line, then the full plant.	Limits risk and allows for procedure refinement.
	6. Threshold Tuning	Adjust confidence thresholds (e.g., < 0.85 for cloud flag) based on initial results.	Balances false alarms vs. missed detections for the specific process.
Post-Deployment	7. Continuous Validation	Periodically check model predictions against manual inspections.	Detects concept drift (e.g., from machine wear).
	8. Update Protocol	Establish a secure procedure for OTA model updates.	Enables model improvement without physical access.

5.3 Internet of Things ecosystem integration: New section on edge-cloud synergy

The proposed embedded model is designed as the first, and most critical, tier in a hierarchical Industrial IoT (IIoT) architecture. To optimize bandwidth and computational resource allocation, a confidence-based triggering mechanism is implemented: predictions with a softmax probability below a 0.85 threshold—indicating uncertain classifications—initiate two concurrent actions. First, the device stores a compressed 5-second raw waveform buffer locally for potential later forensic analysis. Second, it flags the event for cloud-based verification, where more computationally intensive models (e.g., deeper ensembles or vision transformers) or human domain experts can provide a definitive diagnosis. This hybrid edge-cloud strategy achieves a 94% reduction in upstream data transmission compared to a cloud-only approach, while maintaining comprehensive diagnostic coverage and traceability. Furthermore, the model's sub-100 kB memory footprint and milliwatt-scale energy consumption unlock deployment on energy-harvesting or battery-powered wireless sensor nodes, enabling condition monitoring on inaccessible or rotating machinery without wired power or communication infrastructure.

5.4 Broader implications for Internet of Things and smart systems

The methodological approach demonstrated in this study—

combining 1D CNNs with aggressive quantization for resource-constrained deployment—offers a blueprint for embedded AI across domains facing sub-100 kB memory, sub-10 ms latency, and milliwatt power budgets. These advancements align with broader IoT trends [25, 26], where edge deployment enables ultra-low latency and data privacy without continuous cloud connectivity.

Parallel applications include precision agriculture, where CNNs enable real-time plant disease detection [42] and automated pest control [43]. Similar convergent architectural choices—model simplification, quantization, and hardware-aware optimization—address the common challenge of efficient on-device sensor data processing.

Beyond agriculture, embedded AI is transforming logistics, cultural heritage preservation, and livestock monitoring through image classification [44], hospitality demand prediction [45], and multi-modal biometric sensing [46-48]. These applications, like our fault diagnosis system, rely on integrated IoT-edge-cloud ecosystems [49, 50].

The end-to-end embedded AI pipeline developed here for vibration analysis provides a scalable blueprint adaptable to countless IoT-based predictive monitoring applications across industries.

5.5 Limitations and future work

While the proposed model demonstrates high diagnostic efficacy and operational efficiency, several inherent limitations present fruitful opportunities for future research and development efforts.

First, the current work focuses primarily on single-point, localized faults (inner and outer race defects). Industrial environments in practice often present more complex fault scenarios, including compound faults (e.g., simultaneous defects in the raceway and a rolling element) and generalized distributed wear patterns. Extending the model's capability to multi-label classification frameworks or hybrid anomaly detection paradigms would be a logical and valuable step to address this gap [51].

Second, while validation was conducted across seven datasets, they primarily involve radial ball bearings. Generalizing the proposed approach to other critical bearing types (e.g., tapered roller bearings, thrust bearings) would require further investigation, potentially involving advanced transfer learning and domain adaptation techniques to adapt the learned features to new mechanical domains and signature profiles [23, 52, 53].

Third, the data used for training and evaluation were predominantly collected under controlled laboratory or test rig conditions. The ultimate validation step involves deploying the system within operational industrial settings to evaluate its performance against the myriad challenges of real-world environments, such as variable load conditions, extreme temperature fluctuations, contaminant ingress, and sensor calibration drift over time [54]. Incorporating additional sensor modalities (e.g., temperature, acoustic emission, oil debris analysis) could further enhance diagnostic robustness and confidence through intelligent sensor fusion techniques.

Finally, long-term deployment in an evolving environment must account for the phenomenon of concept drift—where the underlying data distribution changes slowly over time due to machine wear, maintenance interventions, or changes in operational regime. Future research directions must therefore include integrating online or continual learning algorithms to allow the model to adapt incrementally to new data without suffering from catastrophic forgetting of previous knowledge, thereby ensuring sustained accuracy and reliability throughout the asset's operational lifecycle [54].

While the proposed model demonstrates strong performance, several important limitations warrant discussion and present clear directions for future work. First, XAI Integration: While attribution methods like Integrated Gradients provide valuable post-hoc explanations, future architectures should integrate attention mechanisms or self-interpretable building blocks directly into the model design for built-in interpretability, allowing maintenance technicians to understand model decisions in real-time. Second, Hybrid Quantization: Current static 8-bit quantization could be evolved into dynamic precision adjustment, where the model automatically adjusts numerical precision based on real-time signal quality (SNR) or diagnostic confidence, optimizing the energy-accuracy trade-off per inference. Third, Cross-Machine Transfer: Addressing the TDM challenge—a critical requirement for scalable industrial deployment—requires investigation of advanced domain adaptation techniques like adversarial feature alignment, meta-learning, or few-shot learning to enable models trained on laboratory data to perform reliably on entirely different bearing types and machinery without extensive retraining.

6. CONCLUSIONS

This research delivers not merely an accurate classification model but a complete, hardware-aware deployment pipeline for industrial PdM. Our methodology encompasses data collection strategies adaptable to legacy equipment, domain-invariant preprocessing for variable operational conditions, and microcontroller-specific optimization techniques that balance accuracy with severe resource constraints. The demonstrated performance of 98.6% ± 0.3% F1-score (mean ± std, 5-fold CV) while operating within a 90 KB flash footprint and 42 KB runtime RAM establishes a new practical benchmark for embedded bearing diagnostics. This work has immediate applicability to retrofitting the vast installed base of industrial machinery; with an estimated 150 million industrial electric motors worldwide currently operating without predictive capabilities, our solution provides a feasible, cost-effective path to modernize maintenance strategies. By bridging the gap between high-accuracy deep learning and the realities of resource-constrained edge devices, this work contributes meaningfully to the realization of accessible, scalable, and intelligent industrial systems aligned with the goals of Industry 4.0.

A comprehensive glossary of all abbreviations used in this manuscript is provided in Appendix B.

REFERENCES

[1] Global Wind Energy Council. Global Wind Report 2023. Brussels, Belgium, 2023. https://www.gwec.net/reports.

[2] SKF AB. The Cost of Downtime, Annual Report. Gothenburg, Sweden: SKF AB, 2024. https://www.skf.com/group/industries/wind-energy.

[3] Tandon, N., Choudhury, A. (1999). A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings. Tribology

International, 32(8): 469-480. https://doi.org/10.1016/S0301-679X(99)00077-8
[4] Lei, Y., Li, N., Guo, L., Li, N., Yan, T., Lin, J. (2018). Machinery health prognostics: A systematic review from data acquisition to RUL prediction. Mechanical Systems and Signal Processing, 104: 799-834. https://doi.org/10.1016/j.ymssp.2017.11.016
[5] ABB Ltd. Predictive Maintenance for Industrial Rotating Assets, Whitepaper. Zurich, Switzerland: ABB, 2023. https://new.abb.com/service.
[6] Si, X.S., Wang, W., Hu, C.H., Zhou, D.H. (2011). Remaining useful life estimation – A review on the statistical data driven approaches. European Journal of Operational Research, 213(1): 1-14. https://doi.org/10.1016/j.ejor.2010.11.018
[7] Jardine, A.K.S., Lin, D., Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. Mechanical Systems and Signal Processing, 20(7): 1483-1510. https://doi.org/10.1016/j.ymssp.2005.09.012
[8] Lei, Y., Yang, B., Jiang, X., Jia, F., Li, N., Nandi, A.K. (2020). Applications of machine learning to machine fault diagnosis: A review and roadmap. Mechanical Systems and Signal Processing, 138: 106587. https://doi.org/10.1016/j.ymssp.2019.106587
[9] Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., Gao, R.X. (2019). Deep learning and its applications to machine health monitoring. Mechanical Systems and Signal Processing, 115: 213-237. https://doi.org/10.1016/j.ymssp.2018.05.050
[10] Dalzochio, J., Kunst, R., Pignaton, E., Binotto, A., Sanyal, S., Favilla, J., Trentesaux, D. (2020). Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges. Computers in Industry, 123: 103298. https://doi.org/10.1016/j.compind.2020.103298
[11] Carvalho, T.P., Soares, F.A.A.M.N., Vita, R., Francisco, R.P., Basto, J.P., Alcalá, S.G.S. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. Computers & Industrial Engineering, 137: 106024. https://doi.org/10.1016/j.cie.2019.106024
[12] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature, 521(7553): 436-444. https://doi.org/10.1038/nature14539
[13] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. Cambridge, MA, USA: MIT Press. https://www.deeplearningbook.org/.
[14] Harris, T.A., Kotzalas, M.N. (2006). Essential Concepts of Bearing Technology. In 5th ed. Boca Raton, FL, USA: CRC Press. https://doi.org/10.1201/9781420006599
[15] Randall, R.B., Antoni, J. (2011). Rolling element bearing diagnostics—A tutorial. Mechanical Systems and Signal Processing, 25(2): 485-520. https://doi.org/10.1016/j.ymssp.2010.07.017
[16] Bishop, C.M. (2006). Pattern Recognition and Machine Learning. New York, NY, USA: Springer.
[17] Cover, T., Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1): 21-27. https://doi.org/10.1109/TIT.1967.1053964
[18] Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning, 2nd ed. New York, NY, USA: Springer. https://doi.org/10.1007/978-0-387-84858-7
[19] Chen, T., Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, pp. 785-794. https://doi.org/10.1145/2939672.2939785
[20] Breiman, L. (2001). Random forests. Machine Learning, 45(1): 5-32. https://doi.org/10.1023/A:1010933404324
[21] Thoppil, N.M., Vasu, V., Rao, C.S.P. (2021). Deep learning algorithms for machinery health prognostics using time series data: A review. Journal of Vibration Engineering & Technologies, 9(6): 1123-1145. https://doi.org/10.1007/s42417-021-00286-x
[22] Ince, T., Kiranyaz, S., Eren, L., Askar, M., Gabbouj, M. (2016). Real-time motor fault detection by 1-D convolutional neural networks. IEEE Transactions on Industrial Electronics, 63(11): 7067-7075. https://doi.org/10.1109/TIE.2016.2582729
[23] Malhotra, P., Vishnu, T. V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P. (2016). Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder. arXiv. https://doi.org/10.48550/arXiv.1608.06154
[24] Chen, X., Yang, R., Xue, Y., Huang, M., Ferrero, R., Wang, Z. (2023). Deep transfer learning for bearing fault diagnosis: A systematic review since 2016. IEEE Transactions on Instrumentation and Measurement, 72: 3502421. https://doi.org/10.1109/TIM.2023.3244237
[25] Gunerkar, R.S., Jalan, A.K., Belgamwar, S.U. (2019). Fault diagnosis of rolling element bearing based on artificial neural network. Journal of Mechanical Science and Technology, 33(2): 505-511. https://doi.org/10.1007/s12206-019-0103-x
[26] Gouiza, N., Jebari, H., Reklaoui, K. (2024). Integration for IoT-enabled technologies and artificial intelligence in diverse domains: Recent advancements and future trends. Journal of Theoretical and Applied Information Technology, 102(5): 1975-2029.
[27] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 2704-2713 https://doi.org/10.1109/CVPR.2018.00286
[28] Lai, L., Suda, N., Chandra, V. (2018). CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs. arXiv preprint arXiv:1801.06601. https://doi.org/10.48550/arXiv.1801.06601
[29] Ray, P.P. (2022). A review on TinyML: State-of-the-art and prospects. Journal of King Saud University - Computer and Information Sciences, 34(4): 1595-1623. https://doi.org/10.1016/j.jksuci.2021.11.019
[30] Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O.M. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. Journal of Internet Services and Applications, 9(1): 16. https://doi.org/10.1186/s13174-018-0087-2
[31] Case Western Reserve University Bearing Data Center. Bearing Vibration Data Sets. https://engineering.case.edu/bearingdatacenter, accessed on Jan. 18, 2025.
[32] MFPT Society. Condition Based Maintenance Fault

Database. https://www.mfpt.org/fault-data-sets/.
[33] Lessmeier, C., Kimotho, J.K., Zimmer, D., Sextro, W. (2016). Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification. Proceedings of the European Conference of the Prognostics and Health Management Society, 3(1). https://doi.org/10.36001/phme.2016.v3i1.1577
[34] Huang, H., Baddour, N. (2018). Bearing vibration data collected under time-varying rotational speed conditions. Data in Brief, 21: 1745-1749. https://doi.org/10.1016/j.dib.2018.11.019
[35] Qiu, H., Lee, J., Lin, J., Yu, G. (2007). IMS Bearing Data Set, NASA Ames Prognostics Data Repository. https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/.
[36] Smith, W.A., Randall, R.B. (2015). Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study. Mechanical Systems and Signal Processing, 64-65: 100-131. https://doi.org/10.1016/j.ymssp.2015.04.021
[37] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1): 1929-1958.
[38] Cortes, C., Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3): 273-297. https://doi.org/10.1007/BF00994018
[39] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and Regression Trees. Belmont, CA, USA: Wadsworth. https://doi.org/10.1201/9781315139470
[40] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint. https://doi.org/10.48550/arXiv.1609.04747
[41] Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. arXiv:1710.09412. https://doi.org/10.48550/arXiv.1710.09412
[42] Ezziyyani, M., Cherrat, L., Jebari, H., Rekiek, S., Ahmed, N.A. (2025). CNN-based plant disease detection: A pathway to sustainable agriculture. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2024), Springer, Cham, pp. 679-696. https://doi.org/10.1007/978-3-031-91337-2_62
[43] Rekiek, S., Jebari, H., Ezziyyani, M., Cherrat, L. (2025). AI-driven pest control and disease detection in smart farming systems. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2024), Agadir, Morocco, pp. 801-810. https://doi.org/10.1007/978-3-031-91337-2_71
[44] Ezziyyani, M., Cherrat, L., Rekiek, S., Jebari, H. (2025). Image classification of moroccan cultural trademarks. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2024), Agadir, Morocco, pp. 767-779. https://doi.org/10.1007/978-3-031-91337-2_68
[45] Rekiek, S., Jebari, H., Reklaoui, K. (2024). Prediction of booking trends and customer demand in the tourism and hospitality sector using AI-based models. International Journal of Advanced Computer Science and Applications, 15(10): 404-412. https://doi.org/10.14569/IJACSA.2024.0151043
[46] Jebari, H., Rekiek, S., Reklaoui, K. (2025). Advancing precision livestock farming: Integrating hybrid AI, IoT, cloud and edge computing for enhanced welfare and efficiency. International Journal of Advanced Computer Science and Applications, 16(7): 302-311. https://doi.org/10.14569/IJACSA.2025.0160732
[47] Jebari, H., Mechkouri, M.H., Rekiek, S., Reklaoui, K. (2023). Poultry-edge-AI-IoT system for real-time monitoring and predicting by using artificial intelligence. International Journal of Interactive Mobile Technologies, 17(12): 58-70. https://doi.org/10.3991/ijim.v17i12.38095
[48] Jebari, H., Rekiek, S., Ezziyyani, M., Cherrat, L. (2025). Artificial intelligence for optimizing livestock management and enhancing animal welfare. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2024), Agadir, Morocco, pp. 790-800. https://doi.org/10.1007/978-3-031-91337-2_70
[49] Gouiza, N., Jebari, H., Reklaoui, K. (2024). IoT in smart farming: A review. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2023), Marrakech, Morocco, pp. 142-153. https://doi.org/10.1007/978-3-031-54318-0_13
[50] Gouiza, N., Jebari, H., Reklaoui, K. (2025). IoT in agriculture: Use cases and challenges. In International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'2024), Agadir, Morocco, pp. 491-505. https://doi.org/10.1007/978-3-031-91334-1_42
[51] Lee, I., Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, 58(4): 431-440. https://doi.org/10.1016/j.bushor.2015.03.008
[52] Eljyidi, A., Jebari, H., Rekiek, S., Reklaoui, K. (2025). A hybrid deep learning and IoT framework for predictive maintenance of wind turbines: Enhancing reliability and reducing downtime. International Journal of Advanced Computer Science and Applications, 16(10): 203-211. https://doi.org/10.14569/IJACSA.2025.0161021
[53] Jebari, H., Eljyidi, A., Rekiek, S., Reklaoui, K. (2025). A vision-based deep learning framework for autonomous inspection and damage assessment of wind turbine blades using unmanned aerial vehicles. Journal Européen des Systèmes Automatisés, 58(11): 2219-2228. https://doi.org/10.18280/jesa.581101
[54] Wang, T., Yu, J., Siegel, D., Lee, J. (2008). A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In Proceedings of the International Conference on Prognostics and Health Management (PHM), Denver, CO, USA, pp. 1-6. https://doi.org/10.1109/PHM.2008.4711421

APPENDIX

Appendix A: Pseudo-code for all critical algorithms

Algorithm A1: Group-Stratified Data Splitting Algorithm

To ensure reproducibility and prevent data leakage, we implemented the following algorithm for generating train/validation/test splits:

```
Algorithm 1: Group-Stratified Train/Val/Test Split
Input: List of bearings B with their health states, window length L = 1024, stride S = 512
Output: Train_windows, Val_windows, Test_windows (each with labels)
1. Initialize empty sets: train_bearings = [], val_bearings = [], test_bearings = []
2. Initialize empty lists: train_windows = [], val_windows = [], test_windows = []
3. // Step 1: Group bearings by dataset and health state
  For each dataset D in {CWRU, MFPT, PRONOSTIA, Huang-Baddour, IMS, Paderborn}:
    For each health state H in {Healthy, Inner Race, Outer Race}:
      bearings_DH = [b for b in B if b.dataset == D and b.health == H]
      // Step 2: Random shuffle and split bearings
      Random shuffle bearings_DH with seed = 42
      n_total = length(bearings_DH)
      n_train = floor(0.7 * n_total)
      n_val = floor(0.15 * n_total)
      n_test = n_total - n_train - n_val
      train_bearings.extend(bearings_DH[0:n_train])
      val_bearings.extend(bearings_DH[n_train:n_train + n_val])
      test_bearings.extend(bearings_DH[n_train + n_val:])
4. // Step 3: Verify no bearing appears in multiple splits
  assert no intersection between train_bearings, val_bearings, test_bearings
5. // Step 4: Generate windows for each split
  For each bearing b in train_bearings:
    signal = load_raw_signal(b)
    N = length(signal)
    For start_idx = 0 to N - L step S:
      window = signal[start_idx : start_idx + L]
      train_windows.append((window, b.health))
   // Repeat for val_bearings and test_bearings
6. Return train_windows, val_windows, test_windows
```

Algorithm A2: Feature Extraction Pipeline for 47 Handcrafted Features

```
Input: Normalized signal window x (length L = 1024 samples)
    Sampling frequency fs (Hz)
    Bearing geometry: ball diameter Bd, pitch diameter Pd, number of balls Nb
    Shaft speed RPM (for characteristic frequency calculation)
Output: Feature vector F of length 47
//
==========================================
============================
// Step 1: Time-Domain Features (12 features)
//
==========================================
============================
1. Compute basic statistics:
  mean_x = mean(x)
  rms = sqrt(mean(x^2))
  peak = max(|x|)
  variance = mean((x - mean_x)^2)
  std_dev = sqrt(variance)
2. Amplitude-based features:
  peak_to_peak = max(x) - min(x)
  crest_factor = peak / rms
  shape_factor = rms / mean(abs(x))
  impulse_factor = peak / mean(abs(x))
  clearance_factor = peak / (mean(sqrt(abs(x)))^2
3. Distribution shape features:
  kurtosis = mean((x - mean_x)^4) / (variance^2) - 3  // excess kurtosis
  skewness = mean((x - mean_x)^3) / (std_dev^3)
4. Complexity features:
  zero_crossing_rate = count_zero_crossings(x) / L
  // Signal entropy (approximate)
  p = histcounts(x, 50) / L  // probability distribution over 50 bins
  p = p(p > 0)  // remove zero probabilities
  signal_entropy = -sum(p .* log2(p))
5. Assemble time-domain features:
  F_time = [rms, peak_to_peak, crest_factor, kurtosis, skewness,
        shape_factor, impulse_factor, clearance_factor,
        variance, std_dev, zero_crossing_rate, signal_entropy]
//
==========================================
============================
// Step 2: Frequency-Domain Features (14 features)
//
==========================================
============================
6. Compute FFT magnitude spectrum:
  N_fft = 2048  // zero-padding for better frequency resolution
  X = fft(x, N_fft)
  mag = abs(X(1:N_fft/2))  // single-sided spectrum
  freq = (0:N_fft/2-1) * fs / N_fft
  mag = mag / sum(mag)  // normalize to probability distribution
7. Spectral statistics:
  spectral_centroid = sum(freq .* mag) / sum(mag)
  spectral_spread = sqrt(sum((freq - spectral_centroid).^2 .* mag) / sum(mag))
  // Spectral roll-off (85% and 95%)
  cumulative_energy = cumsum(mag)
  rolloff_85_idx = find(cumulative_energy >= 0.85 * sum(mag), 1, 'first')
  rolloff_95_idx = find(cumulative_energy >= 0.95 * sum(mag), 1, 'first')
  spectral_rolloff_85 = freq(rolloff_85_idx)
  spectral_rolloff_95 = freq(rolloff_95_idx)

  spectral_entropy = -sum(mag .* log2(mag + eps))
  spectral_flatness = exp(mean(log(mag + eps))) / mean(mag)
8. Calculate characteristic fault frequencies:
  // Based on bearing geometry [13]
  BPFI = (Nb * RPM / 120) * (1 + (Bd / Pd) * cos(contact_angle))
  BPFO = (Nb * RPM / 120) * (1 - (Bd / Pd) * cos(contact_angle))
  FTF = (RPM / 120) * (1 - (Bd / Pd) * cos(contact_angle))
  // Define frequency bands (±5% around characteristic frequencies)
  bands = {
    'BPFI': [BPFI * 0.95, BPFI * 1.05],
    'BPFO': [BPFO * 0.95, BPFO * 1.05],
    'FTF': [FTF * 0.95, FTF * 1.05],
    '2xBPFI': [2*BPFI * 0.95, 2*BPFI * 1.05],
    '2xBPFO': [2*BPFO * 0.95, 2*BPFO * 1.05],
    '3xBPFI': [3*BPFI * 0.95, 3*BPFI * 1.05],
```

```
    '3xBPFO': [3*BPFO * 0.95, 3*BPFO * 1.05],
    '1-2kHz': [1000, 2000],
    '2-5kHz': [2000, 5000]
  }
9. Extract band energies:
  band_energies = []
  For each band in bands:
    idx_low = find(freq >= band[0], 1, 'first')
    idx_high = find(freq <= band[1], 1, 'last')
    if idx_low < idx_high:
      energy = sum(mag(idx_low:idx_high))
    else:
      energy = 0
    band_energies.append(energy)
10. Assemble frequency-domain features:
  F_freq = [spectral_centroid, spectral_spread, spectral_rolloff_85,
        spectral_rolloff_95, spectral_entropy, spectral_flatness]
  F_freq = [F_freq, band_energies]  // concatenate with 9 band energies
//
================================================================
// Step 3: Time-Frequency Features (21 features)
//
================================================================
11. Compute Short-Time Fourier Transform (STFT):
  window_length = 256
  overlap = 128  // 50% overlap
  n_mels = 40
  [S, f_stft, t_stft] = stft(x, fs, 'Window', hamming(window_length),
                'OverlapLength', overlap, 'FFTLength', 2048)
  mag_stft = abs(S)  // magnitude spectrogram
12. Extract MFCCs (13 coefficients):
  // Map to Mel scale
  mel_filterbank = design_mel_filterbank(n_mels, f_stft)
  mel_spectrum = mel_filterbank * mag_stft
  // Take log and DCT
  log_mel = log(mel_spectrum + eps)
  mfcc_full = dct(log_mel)  // 40 coefficients
  // Keep coefficients 2-14 (exclude first coefficient which represents energy)
  mfcc = mfcc_full(2:14)  // 13 coefficients
  // Note: No delta or delta-delta coefficients
13. Extract statistical moments from critical frequency bands:
  // Find frequency indices for BPFI and BPFO bands
  bpfi_idx_low = find(f_stft >= BPFI*0.9, 1, 'first')
  bpfi_idx_high = find(f_stft <= BPFI*1.1, 1, 'last')
  bpfo_idx_low = find(f_stft >= BPFO*0.9, 1, 'first')
  bpfo_idx_high = find(f_stft <= BPFO*1.1, 1, 'last')
  // Extract time-varying energy envelopes
  bpfi_envelope= mean(mag_stft(bpfi_idx_low:bpfi_idx_high, :), 1)
  bpfo_envelope= mean(mag_stft(bpfo_idx_low:bpfo_idx_high, :), 1)
  // Compute statistical moments for BPFI band
  bpfi_mean = mean(bpfi_envelope)
  bpfi_var = var(bpfi_envelope)
  bpfi_skew = skewness(bpfi_envelope)
  bpfi_kurt = kurtosis(bpfi_envelope)
  // Compute statistical moments for BPFO band
  bpfo_mean = mean(bpfo_envelope)
  bpfo_var = var(bpfo_envelope)
  bpfo_skew = skewness(bpfo_envelope)
  bpfo_kurt = kurtosis(bpfo_envelope)
  F_tf_moments = [bpfi_mean, bpfi_var, bpfi_skew, bpfi_kurt,
          bpfo_mean, bpfo_var, bpfo_skew, bpfo_kurt]
14. Assemble time-frequency features:
  F_tf = [mfcc, F_tf_moments]  // 13 + 8 = 21 features
//
================================================================
// Step 4: Assemble Complete Feature Vector (47 features)
//
================================================================
15. F = [F_time, F_freq, F_tf]  // 12 + 14 + 21 = 47 features
16. Return F
//
================================================================
// Helper Functions
//
================================================================
Function count_zero_crossings(x):
  // Count number of times signal crosses zero
  sign_changes = diff(sign(x)) != 0
  return sum(sign_changes)
Function design_mel_filterbank(n_mels, f_stft):
  // Design Mel-spaced filterbank for MFCC extraction
  mel_min = 0
  mel_max = 2595 * log10(1 + (fs/2) / 700)
  mel_points = linspace(mel_min, mel_max, n_mels + 2)
  hz_points = 700 * (10.^(mel_points/2595) - 1)
  // Create triangular filters
  filterbank = zeros(n_mels, length(f_stft))
  For m = 1 to n_mels:
    f_left = hz_points(m)
    f_center = hz_points(m+1)
    f_right = hz_points(m+2)
    // Rising edge
    idx_left = find(f_stft >= f_left & f_stft < f_center)
    filterbank(m, idx_left) = (f_stft(idx_left) - f_left) / (f_center - f_left)

    // Falling edge
    idx_right = find(f_stft >= f_center & f_stft <= f_right)
    filterbank(m, idx_right) = (f_right - f_stft(idx_right)) / (f_right - f_center)
  Return filterbank
```

Algorithm A3: Quantization-Aware Training Protocol

```
Input: Pre-trained FP32 model M_fp32, calibration dataset D_cal, training dataset D_train
Output: INT8 quantized model M_int8
1. // Insert fake quantization nodes for QAT
  M_qat = quantize_aware_training(M_fp32)
2. // Fine-tune with simulated quantization
  For epoch = 1 to 50:
    For batch in D_train:
      // Forward pass with simulated quantization
```

```
        logits = M_qat.forward_with_fake_quant(batch.x)
        loss = crossentropy(logits, batch.y)
        // Backward pass (quantization nodes pass gradients)
        loss.backward()
        M_qat.update_weights(optimizer)
3. // Calibrate activation ranges
   calibration_representative = sample(D_cal, n=500 per class)
   M_calibrated = calibrate(M_qat, calibration_representative)
4. // Convert to INT8 TFLite format
   converter = TFLiteConverter.from_keras_model(M_calibrated)
   converter.optimizations = [tf.lite.Optimize.DEFAULT]
   converter.representative_dataset = calibration_representative
   converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
   converter.inference_input_type = tf.int8
   converter.inference_output_type = tf.int8
5. M_int8 = converter.convert()
6. Return M_int8
```

Algorithm A4: Latency Measurement on ARM Cortex-M using DWT Cycle Counter

```
Input: Quantized model M_int8, input tensor x, num_iterations N = 10000
Output: Average inference latency in milliseconds
1. // Configure DWT cycle counter
   CoreDebug->DEMCR|= CoreDebug_DEMCR_TRCENA_Msk
   DWT->CYCCNT = 0
   DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk
2. // Warm-up (10 inferences to caches)
   For i = 1 to 10:
      y = M_int8.predict(x)
3. // Measure N inferences
   start = DWT->CYCCNT
   For i = 1 to N:
      // Compiler barrier to prevent optimization
      __asm volatile("" ::: "memory")
      y = M_int8.predict(x)
      __asm volatile("" ::: "memory")
   end = DWT->CYCCNT
4. // Calculate average
   total_cycles = end - start
   avg_cycles = total_cycles / N
   clock_freq_hz = 600000000  // Teensy 4.1 at 600 MHz
   avg_latency_ms = (avg_cycles * 1000) / clock_freq_hz
5. Return avg_latency_ms
```

Appendix B: Abbreviation Glossary

BPF: Band-Pass Filter
HPF: High-Pass Filter
STFT: Short-Time Fourier Transform
MFCC: Mel-Frequency Cepstral Coefficient
MOPS: Million Operations Per Second
MAC: Multiply-Accumulate
GAP: Global Average Pooling
QAT: Quantization-Aware Training
TFLM: TensorFlow Lite Micro
CMSIS: Cortex Microcontroller Software Interface Standard
DWT: Data Watchpoint and Trace
SNR: Signal-to-Noise Ratio
QCNN: Quantized 1D CNN