

Article

A Resource-Aware Method for Evaluating AI Models in Data Streaming Systems for Predictive Maintenance

Luisiana Sabbatini * , Sara Raggiunto , Marco Esposito , Alberto Belli  and Paola Pierleoni 

Department of Information Engineering, Università Politecnica delle Marche, Via Brecce Bianche 12, 60131 Ancona, Italy

* Correspondence: l.sabbatini@staff.univpm.it

Abstract

Computation, data transmission, and all the activities involved in Artificial Intelligence of Things (AIoT) systems raise growing concerns in terms of resource consumption and environmental impact. Industrial asset monitoring is a data-intensive use case requiring a balance between predictive performance, responsiveness, and resource efficiency. In this context, edge and cloud computing represent complementary paradigms, enabling trade-offs between latency, scalability, and resource availability. However, understanding how these trade-offs affect model behavior and system-level resource demands in data streaming systems remains an open challenge, particularly in the context of resource-efficient and sustainable AI system design. In this work, we present a resource-aware method for evaluating AI models in data streaming systems for predictive maintenance, applied to the NASA IMS Bearing Dataset. The method is based on a controlled streaming setup that emulates a resource-constrained configuration and a resource-rich configuration, enabling a systematic analysis of model behavior under heterogeneous conditions. The evaluation considers predictive performance and operational metrics, including accuracy, latency, CPU utilization, and memory usage, to capture interactions between model characteristics and system-level factors. The proposed method enables a structured analysis of trade-offs between performance and resource utilization, offering insights for the design of resource-efficient and sustainable AI-based predictive maintenance systems.

Keywords: predictive maintenance; data streaming systems; resource-aware computing; sustainable AI; model evaluation; heterogeneous resource configurations

1. Introduction

The integration of Artificial Intelligence (AI) and the Internet of Things (IoT) in the industrial sector—commonly referred to as AIoT—represents one of the main frontiers of innovation for sustainability and smart manufacturing [1]. The convergence of these technologies is transforming traditional industry across multiple domains, including supply chain management, industrial asset maintenance, and overall industrial sustainability. However, computation, data transmission, and other digital activities involved in the deployment of AIoT solutions raise increasing concerns in terms of resource consumption and environmental impact due to the significant costs associated with hardware, software, energy consumption, and bandwidth requirements [2].

Among AIoT industrial use cases, advanced management of industrial assets' maintenance deserves particular attention, given that it is usually associated with data-intensive architectures. Thanks to AIoT, the industrial maintenance paradigm is moving from reactive



Academic Editor: Marko Mladineo

Received: 7 May 2026

Revised: 26 June 2026

Accepted: 27 June 2026

Published: 1 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

approaches to highly effective predictive strategies [3]. AI-driven predictive maintenance has undergone a radical transformation in recent years, with 50% of manufacturers expected to use AI, IoT, and analytics to achieve operational resilience by 2025 according to market research reports. In this context, bearing fault diagnosis represents a critical application, as bearings are vital components in rotating machinery, and their failures can cause significant economic losses and operational disruptions [4]. High-frequency sensors (e.g., accelerometers above 20 kHz) generate large volumes of data, requiring efficient processing and transmission strategies. Data, processing techniques (Fast Fourier Transform among others), and models adopted in the context of industrial assets' condition-based maintenance management are associated with critical issues at both model development and deployment stages [5].

In data-intensive industrial applications such as Predictive Maintenance (PdM), both diverse hardware deployment tiers play vital and complementary roles within AIoT architectures [6]. Heterogeneous execution nodes have emerged as two leading architectures to support Industrial Internet of Things (IIoT) applications, addressing latency, bandwidth, and compute challenges [7]. Lower-tier resource-constrained nodes enable low-latency data processing and real-time responsiveness at the device or plant level, supporting autonomy, resilience, and privacy [8]. Furthermore, localized processing ensures continuous operations even when the connection to the central cloud is lost, which is vital for critical applications. Conversely, higher-tier resource-rich infrastructures provide scalable resources for data aggregation, long-term storage, and advanced model training. Benefits of unthrottled infrastructures include scalability to handle large volumes of data from numerous sensors, access to powerful analytics tools, and remote access capabilities. In this context, heterogeneous resource profiles represent complementary paradigms, enabling different trade-offs between latency, scalability, and resource availability. However, rather than focusing on physical deployment infrastructures, it is increasingly important to understand how these trade-offs affect model behavior under structurally different constraints [9,10].

Sustainability associated with optimized energy consumption is another hot topic of today, given the crucial role energy plays against climate change and environment. As computationally intensive modeling solutions arise, more attention is given to the operational footprint these analytical solutions have, for both inferential and generative models [11]. The literature connected to empirical studies on AI models' software intensity and associated energy consumption (generally measuring also hardware metrics like CPU and RAM) is rapidly increasing [12]. In this perspective, process-level resource efficiency becomes a key auxiliary proxy for exploratory AI system design, especially in data-intensive streaming applications. The authors of [13] investigate the application of advanced machine learning models, specifically Long Short-Term Memory (LSTM) networks and Gradient Booster models, for accurate energy consumption estimation within a Kubernetes cluster environment. As highlighted by [14], the convergence of IoT, AI, and computing architectures enables sustainable manufacturing through energy-aware predictive maintenance. However, the exclusive reliance on unconstrained centralized infrastructures introduces computational overhead and latency issues. In [15], the authors outline green IoT and localized execution paradigms, highlighting their crucial role for a sustainable digital transition toward Industry 5.0.

When it comes to model deployment on edge and on cloud, requirements of the two are very different and deserve dedicated strategies for managing the constrained quantity of available resources. Comparative studies between edge and cloud versions of analytical solutions are essential for easing the concrete development of such systems in reality [16]. Deploying AI models on edge devices presents unique challenges related to computational, memory, and energy limitations. Model compression techniques have become essential for

enabling AI on resource-constrained devices: Ref. [17] evaluates compression techniques on ConvNeXt models in image classification tasks using the CIFAR-10 dataset. Structured pruning, unstructured pruning, and dynamic quantization methods are evaluated to reduce model size and computational complexity while maintaining accuracy. Unlike existing works that focus on model performance or deployment optimization separately, this study proposes a unified evaluation method that explicitly captures the interaction between model behavior and system-level resource constraints in streaming environments. The novelty of this work lies in the proposal of a resource-aware evaluation method for analyzing AI models in data streaming systems under controlled resource constraints. It is important to note that the contribution does not stem from the individual technologies employed, all of which are well established in the literature, but rather from their integration within a unified evaluation approach. By jointly considering predictive effectiveness, streaming-system behavior, computational workload, and sustainability-oriented metrics, the proposed method enables the systematic analysis of performance–resource trade-offs that are typically addressed separately in existing studies. To this end, the framework is applied to a predictive maintenance use case based on the NASA IMS Bearing Dataset, under both resource-constrained and resource-rich architectural configurations.

2. Related Work

The intersection of predictive maintenance, distributed data streaming, and sustainable computing has recently gained significant traction under the umbrella of AIoT. To position our contribution within the current state of the art, it is essential to contrast the existing literature across the diverse domains that converge in this study. Traditional predictive maintenance research has heavily focused on maximizing model accuracy using static benchmarks. Within this domain, the Bearing Dataset [18] represents a widely recognized and validated real-world dataset, providing high-frequency operational data that reflect authentic industrial degradation processes. Over the years, a multitude of machine learning and deep learning architectures ranging from classical classifiers like Support Vector Machines and Random Forests [19] to complex convolutional networks [20] have been tested on this real-world benchmark to achieve high prognostic performance. However, the vast majority of these studies operate under an offline, static paradigm. Training and inference are typically conducted on high-performance workstations with unthrottled computational resources. Consequently, while these works achieve exceptional predictive metrics, they generally pay limited attention to the operational constraints of real-time industrial deployment, such as data transmission latency, network jitter, and hardware resource saturation. Crucially, the objective of this work is not to benchmark or introduce highly complex, over-parameterized models to marginally improve baseline predictive accuracy. Instead, the core contribution lies in demonstrating how models with fundamentally different architectures behave when embedded in a realistic, resource-constrained ecosystem. In fact, the literature provides only limited examples of comparative frameworks that simultaneously evaluate predictive maintenance, real-time data streaming, and software-driven sustainability metrics within a controlled, hardware-throttled environment. Rather than evaluating AI models in isolation, our approach contrasts diverse algorithmic families to investigate how structural resource boundaries dynamically affect the trade-off between predictive performance and environmental costs. In practical Industry 4.0 scenarios, data from real-world systems must be ingested and processed continuously. Distributed message brokers like Apache Kafka have become the industry standard for managing such high-throughput data pipelines [21], with research frequently focusing on benchmarking the infrastructure's quality of service, message consumption rates, or broker CPU utilization under varying workloads [22]. Concurrently, the paradigm shift toward localized

execution has motivated the migration of AI models closer to the physical data source to optimize local resource consumption, reduce network bandwidth requirements, and minimize latency [23]. Several frameworks have been proposed to evaluate the trade-offs of deploying predictive models under specific hardware limitations at the edge [24]. While these studies provide valuable insights into architectural scalability, they primarily focus on infrastructure-level characteristics and only partially address the interaction between model behavior and resource constraints. They evaluate hardware efficiency, such as RAM footprint or CPU cycles, but only limited attention is devoted to jointly analyzing these hardware metrics alongside predictive performance under varying resource constraints. Recent research has further emphasized the importance of lightweight and resource-aware AI solutions for constrained computing environments. Lightweight machine-learning systems designed for IoT security applications have demonstrated that competitive predictive performance can be achieved while maintaining low computational overhead, highlighting the practical importance of balancing detection performance and resource efficiency in constrained deployments [25]. Similarly, edge-based predictive maintenance and prognostic health-management frameworks have shown how localized AI inference can support timely decision making and low-latency fault diagnosis in industrial environments [26]. Beyond industrial analytics, resource-aware intelligent decision-support systems have also explored the interplay between predictive quality, resource utilization, and operational objectives, highlighting the broader relevance of process- and resource-aware optimization strategies [27]. Collectively, these studies reinforce the importance of evaluating AI systems not only in terms of predictive performance but also with respect to their operational footprint and deployment feasibility. However, these works primarily focus on the design or optimization of specific intelligent systems, whereas the present study proposes a unified evaluation methodology that systematically analyzes the interaction between predictive performance, streaming-system behavior, resource constraints, and sustainability-related indicators under controlled experimental conditions. This operational gap is strictly connected to the broader necessity of incorporating environmental efficiency and sustainable computing principles directly into the software design phase of AI systems [28]. Within this context, empirical energy tracking tools such as CodeCarbon or Scaphandre have emerged to provide software-level estimations of power consumption and equivalent carbon emissions based on localized hardware utilization and regional grid intensities [29]. To date, however, the application of these empirical consumption tracking protocols has been more extensively explored in large-scale deep learning training workloads, particularly within Natural Language Processing and Computer Vision [30]. Very few studies have translated these software-driven measurements to the continuous inference phase of streaming-based industrial networks. Existing sustainable manufacturing studies often rely on macro-level, theoretical energy models that estimate aggregate infrastructure overheads but may provide limited visibility into workload-specific software execution dynamics. Conversely, while software-level trackers offer precise, empirical insights into local workload dynamics, they inherently omit external infrastructural factors, such as network transmission energy and data center cooling overheads. Acknowledging this trade-off, our methodology utilizes software-defined carbon tracking not as an absolute environmental audit, but as an exploratory, workload-driven metric to analyze how architectural throttling directly impacts the computational intensity of continuous predictive maintenance pipelines. To demonstrate how this proposed approach addresses the limitations of existing research and to clearly outline the specific boundaries of our contributions, Table 1 provides a comparative taxonomy contrasting our evaluation method against the current state of the art across five critical structural dimensions.

Table 1. Comparative taxonomy of the proposed method against the existing literature across critical evaluation dimensions.

Reference	Industrial PdM Dataset	Data Streaming Pipeline	Architectural Emulation	Empirical Workload
Qiu et al. [19]	Yes (IMS)	No (offline)	No	No
Khorram et al. [20]	Yes (IMS)	No (offline)	No	No
Loghin et al. [21]	No	Yes (Kafka)	No	No
Schwartz et al. [28]	No	No	No	Yes (software-defined)
PremSankar et al. [24]	No (generic IoT)	No	Constrained Only	Yes (hardware profiling)
Xu et al. [23]	No (generic IoT)	No	Constrained Only	Yes (hardware profiling)
Proposed Method	Yes (IMS)	Yes (Kafka)	Yes (constrained and rich)	Yes (software-defined)

As evidenced by Table 1, the existing literature addresses the challenges of modern industrial applications through fragmented and isolated lenses. While traditional predictive maintenance methods [19,20] leverage robust, real-world benchmark datasets such as the IMS bearing data to validate their prognostic capabilities, they are generally evaluated in offline settings, and only limited attention is devoted to their integration with live data-streaming pipelines and resource-constrained deployment scenarios. Conversely, infrastructure-oriented streaming benchmarks [21] thoroughly evaluate distributed message brokers like Apache Kafka, yet they generally do not integrate machine learning models or resource-aware evaluation methodologies. Similarly, edge computing frameworks [23,24] provide localized architectural throttling to assess hardware profiling metrics (e.g., RAM footprint, CPU cycles, and FLOPs), but they fail to integrate continuous high-throughput streaming pipelines and do not account for software-defined energy footprinting. Finally, early methodology-driven sustainable AI frameworks [28] establish the foundations for software-defined consumption tracking, but their scope remains restricted to generic workloads and does not explicitly address the continuous inference constraints typically encountered in industrial IoT environments. In contrast to these partial approaches, our proposed method integrates all four dimensions within a unified evaluation methodology. By deploying fundamentally diverse algorithmic families on a validated real-world industrial dataset and routing the data through a containerized Apache Kafka Apache Software Foundation (<https://www.apache.org/>), Wakefield, MA, USA) streaming pipeline under throttled conditions, the proposed framework enables the joint analysis of architectural resource emulation (resource-constrained vs. resource-rich) and empirical workload tracking through CodeCarbon. Consequently, the method provides a structured approach for investigating how computational resource boundaries influence the trade-off between prognostic performance and environmental execution costs. Existing studies do not provide a unified methodology that jointly evaluates predictive maintenance models, streaming-based execution, controlled resource emulation, and software-defined sustainability indicators within a common experimental setting. This integration enables the identification of performance–resource interactions that would remain hidden when these dimensions are analyzed independently.

3. Materials and Methods

This study adopted a resource-aware evaluation method designed to analyze the behavior of AI models in data streaming systems under controlled resource constraints. The method was based on three main components: (i) a streaming data pipeline, (ii) configurable resource profiles emulating distinct computational execution environments, and (iii) a set of predictive and operational metrics for assessing performance–resource trade-offs. The evaluation pipeline was designed as an architectural emulation framework utilizing containerized isolation on a single physical host equipped with an Apple M1 Max processor (Apple Inc., Cupertino, CA, USA) featuring a 10-core CPU and 32 GB of RAM. Rather than deploying bare-metal distributed hardware, resource boundaries were strictly enforced via Docker (<https://www.docker.com/>, Docker Inc., Palo Alto, CA, USA) container engine quotas to emulate distinct operational hardware tiers. To capture heterogeneous deployment scenarios, two specific profiles were established:

- **Resource-Constrained Configuration:** Throttled strictly to 1 CPU core and 2 GB of RAM, emulating the baseline hardware boundaries of low-power industrial IoT gateways or embedded single-board computers (such as a Raspberry Pi 4 configuration).
- **Resource-Rich Configuration:** Allocated 10 CPU cores and 8 GB of RAM, emulating a high-performance local industrial server or an unconstrained centralized compute node execution environment.

While this setup accurately replicates CPU and memory availability bottlenecks, it acts as a system-level emulation and inherently does not model physical network distance, multi-tenancy noise, heterogeneous instruction set architectures (ISA), or localized hardware thermal and power throttling behaviors. To accurately simulate the behavioral dynamics of distributed data transmission, a stochastic communication network delay was explicitly introduced into the data ingestion pipeline of the resource-rich configuration. Rather than employing a static baseline, the network overhead was governed by random uniform distributions representing distinct operational stress profiles. Specifically, a low-latency profile varied between 0.02 and 0.05 s to emulate optimized local environments, a medium profile varied between 0.08 and 0.12 s for stable networks, and a high profile varied between 0.15 and 0.25 s to replicate congested Wide Area Network (WAN) or cellular backhaul latencies in distributed industrial monitoring frameworks. Consequently, the reported End-to-End Latency dynamically encompasses communication delays, data ingestion, processing operations, and model execution, providing a system-level characterization of the streaming pipeline. The streaming data pipeline was built upon the NASA IMS Bearing Dataset, provided by the NASA Prognostics Data Repository under the Center for Intelligent Maintenance Systems (IMS) [31]. The NASA IMS Bearing Dataset has become a standard benchmark in the research community for bearing failure diagnostics [4]. The dataset comprises three folders collected during test-to-failure experiments. Each folder contains individual snapshot files of 1-second vibration signals (20,480 points for each axis) recorded at specific intervals. Standardization on this dataset has enabled direct comparison studies between different machine learning and deep learning algorithms. Numerous studies have used this dataset to validate the effectiveness of CNN, LSTM, and hybrid architecture approaches for classifying different types of bearing failures [4,32].

The dataset consists of vibration signals collected from four double-row bearings mounted on a rotating shaft, recorded under run-to-failure conditions. Measurements were acquired at 20 kHz sampling frequency with a constant load of 2000 lbs and a rotational speed of 2000 rpm. Signal degradation gradually increases until bearing failure occurs.

The process followed in this study is summarized in Figure 1.

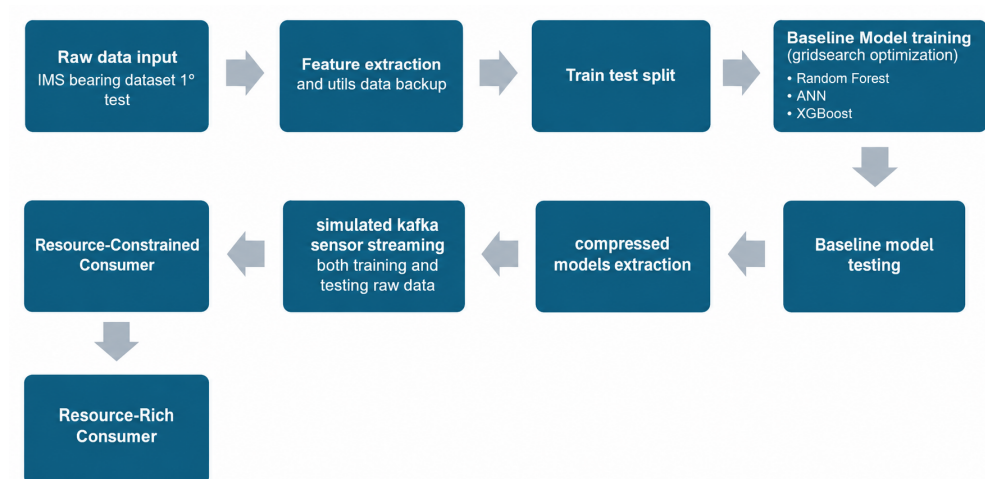


Figure 1. Research process followed.

In this study, the raw accelerometer signals of the first folder of the IMS dataset (two axis for each monitored bearing) were preprocessed to extract statistical and frequency-domain features. Specifically, time-domain features: mean, standard deviation, Root Mean Square (RMS), peak-to-peak, skewness, kurtosis, crest factor, shape factor, impulse factor; frequency-domain features: frequency centroid, frequency bandwidth, spectral entropy, peak frequency, PSD computed using Welch’s method; and time–frequency-domain features: mean value, max value and standard deviation of the Short-Time Fourier Transform (STFT). Each feature was computed over fixed-length windows (1 s, equal to 20,480 samples acquired) of raw vibration data, over each of the two axes. The detailed formulas of computed features are summarized in Table 2.

Table 2. Extracted features and associated equations.

Feature	Equation
Time-domain features	
acquired value x_i ; number of acquired samples N ; numeric stabilization term $\varepsilon \ll 1$	
Mean	$\mu = \frac{1}{N} \sum_{i=1}^N x_i$
Standard Deviation	$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu)^2}$
Root Mean Square (RMS)	$\text{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2}$
Kurtosis	$\text{Kurt}(x) = \frac{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^4}{\sigma^4}$
Peak-to-Peak	$\text{P2P} = \max(x_i) - \min(x_i)$
Crest Factor	$\text{CF} = \frac{\max(x_i)}{\text{RMS}}$
Skewness	$\text{Skew}(x) = \frac{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^3}{\sigma^3}$
Shape Factor	$\text{ShapeFactor} = \frac{\sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2}}{\frac{1}{N} \sum_{i=1}^N x_i + \varepsilon}$
Impulse Factor	$\text{ImpulseFactor} = \frac{\max_n x_i }{\frac{1}{N} \sum_{i=1}^N x_i + \varepsilon}$

Table 2. Cont.

Feature	Equation
Frequency-domain features computed using Welch's method i -th frequency f_i ; frequency bin number N_f .	
Power Spectral Density	$P_{xx}(f_i)$ = PSD according to Welch's formula
Normalized PSD	$\tilde{P}_{xx}(f_i) = \frac{P_{xx}(f_i)}{\sum_{i=1}^{N_f} P_{xx}(f_i)}$
Frequency centroid	Centroid = $\sum_i f_i \tilde{P}_{xx}(f_i)$
Frequency bandwidth	Bandwidth = $\sqrt{\sum_i (f_i - \text{Centroid})^2 \tilde{P}_{xx}(f_i)}$
Spectral entropy	Entropy = $-\sum_i \tilde{P}_{xx}(f_i) \log(\tilde{P}_{xx}(f_i) + \varepsilon)$
Peak frequency	PeakFreq = $f_{\arg \max_i P_{xx}(f_i)}$
Time–frequency-domain features computed on the Short-Time Fourier Transform (STFT) i -th frequency f_i ; frequency bin number N_f ; j -th time t_j ; number of time slots N_t .	
Short-Time Fourier Transform	$Z_{xx}(f_i, t_j)$
STFT magnitude	$ Z_{xx} (f_i, t_j)$
Mean value	MeanMag = $\frac{1}{N_f N_t} \sum_{i,j} Z_{xx}(f_i, t_j) $
Max value	MaxMag = $\max_{i,j} Z_{xx}(f_i, t_j) $
Standard deviation	StdMag = $\sqrt{\frac{1}{N_f N_t} \sum_{i,j} (Z_{xx}(f_i, t_j) - \text{MeanMag})^2}$

The resulting dataset was labeled according to health states derived from degradation trends found by past studies on the same dataset [19,20]. The adopted labels were not generated within the scope of the present study. Instead, they were derived from degradation stages identified in previous investigations of the NASA IMS Bearing Dataset [19,20], where signal degradation patterns were analyzed through traditional condition-monitoring indicators and failure progression analyses. Consequently, the health-state categories should be interpreted as literature-derived degradation labels adopted to ensure consistency with previous research using the same benchmark dataset.

- Early (initial run-in of the bearings).
- Normal.
- Suspect (the health seems to be deteriorating).
- Imminent failure (for bearings 1 and 2, which did not fail but were prone to damage).
- Inner race failure (bearing 3).
- Rolling element failure (bearing 4).
- Stage 2 failure (bearing 4).

This enabled multiclass modeling to be applied to the dataset.

3.1. Model Training and Deployment

Two ML models (Random Forest, and XGBoost) were trained using the extracted features after normalization. The training set was used to compute normalization parameters, which were then applied to the test set. The 1D Convolutional Neural Network (CNN) was trained using the raw bi-axial signals. Although recurrent architectures such as LSTM and GRU are commonly adopted for vibration time-series analysis and may provide competitive predictive performance, they were not included in the present evaluation to keep the experimental scope focused on the proposed methodology. A CNN was selected as the deep learning representative because it can directly process raw bi-axial vibration signals while offering a comparatively simpler inference structure for constrained streaming scenarios. This choice was considered particularly appropriate for the objectives of the present study, where the focus was placed on understanding performance–resource

trade-offs under constrained deployment conditions rather than on maximizing predictive performance through increasingly complex temporal architectures.

All models were trained offline on the preprocessed NASA IMS dataset, using a 80–20% training–test split. Specifically, the DL model was trained using as the raw signal as input, while ML models used the preprocessed data after features extraction and normalization as described in the previous subsection.

Hyper-parameter optimization for each of the three models was performed via grid search with 5-fold cross-validation, using accuracy, F1-score, and AUC as selection criteria. For reproducibility, the training/test split was performed using a stratified 80/20 partition with a fixed random seed (42). The same random seed was adopted for the Random Forest and XGBoost classifiers. Hyper-parameter optimization for the Random Forest considered `n_estimators` values of 100 and 200, `max_depth` values of none, 10, and 20, and `min_samples_split` values of 2 and 5. For XGBoost, the search space included `n_estimators` values of 100 and 200, `max_depth` values of 3 and 6, and `learning_rate` values of 0.05 and 0.1. The CNN was trained for up to 30 epochs with a batch size of 32, using the Adam optimizer, categorical cross-entropy loss, early stopping on validation loss with a patience of 5 epochs, and learning-rate reduction on plateau with a patience of 3 epochs. The hyper-parameters selected during the optimization phase were kept fixed across both the resource-constrained and resource-rich configurations. This choice was made to isolate the impact of resource availability on model behavior and operational performance without introducing additional variability associated with configuration-specific model tuning.

The trained models were evaluated under different resource configurations. Specifically, models were executed under a resource-rich configuration, where higher computational resources were available, and under a resource-constrained configuration, where resource constraints were enforced.

The Random Forest and XGBoost models were deployed in their original form in both configurations. The CNN model was converted to TensorFlow Lite, and the same TensorFlow Lite implementation was used in both resource configurations. This choice ensured consistency of model architecture and numerical representation while enabling deployment-oriented evaluation under different resource allocations.

3.2. Simulation Hardware and Software

The simulation experiments were carried out on an Apple MacBook Pro 16" 2021 (Apple Inc., Cupertino, CA, USA), equipped with an M1 Max chip and 32 GB of RAM. Experiments were carried out on a local machine using Docker containers. Resource configurations were defined parametrically to emulate heterogeneous execution conditions. All experiments were conducted using Python 3.10 within containerized environments. Dependencies were installed without strict version pinning to ensure compatibility with ARM-based architectures. Although all experiments were conducted on a single physical machine, resource constraints were applied at the container level to reproduce resource-constrained and resource-rich configurations. This controlled setup enabled consistent and reproducible comparison of model behavior under varying resource conditions. However, while the proposed approach allowed the systematic evaluation of resource-performance trade-offs, it did not fully reproduce the complexity of physically distributed industrial environments. In particular, factors such as heterogeneous hardware platforms, network variability, resource contention among distributed nodes, and infrastructure-level orchestration effects were outside the scope of the present evaluation. Therefore, resource-constrained and resource-rich configurations should be interpreted as controlled emulations of different resource conditions rather than exact representations of real deployment infrastructures. Furthermore, the implemented data streaming setup was fundamentally synthetic and

idealized. It assumed deterministic message delivery and did not model real-world distributed streaming challenges such as data rate matching against the physical 20 kHz sampling frequency, network unreliability, packet jitter, out-of-order data ingestion, or late arrivals. Consequently, critical real-time engineering constraints, including system-level backpressure handling, consumer deadline misses, and buffering overflow dynamics under network stress, were outside the scope of this evaluation. Specifically, containers were configured with different hardware limits (e.g., CPU cores and memory allocation) to represent constrained and resource-rich execution conditions. Process-level energy tracking was performed via CodeCarbon to ensure isolated measurement of computational workloads. Full project dependencies are available in the requirements file provided in the associated GitHub (GitHub Inc., San Francisco, CA, USA) repository (see Data Availability Statement for the link to the public repository). The main python packages used for performing the experiments are:

- codecarbon 3.0.4.
- confluent-kafka 2.11.1.
- fastparquet 2024.11.0.
- keras 3.11.3.
- pandas 2.3.2.
- psutil 7.0.0.
- scikit-learn 1.7.2.

3.3. Simulated System Architecture

A streaming-based system architecture was developed to support the proposed resource-aware evaluation method under different execution configurations. The overall evaluation workflow is summarized in Figure 2.

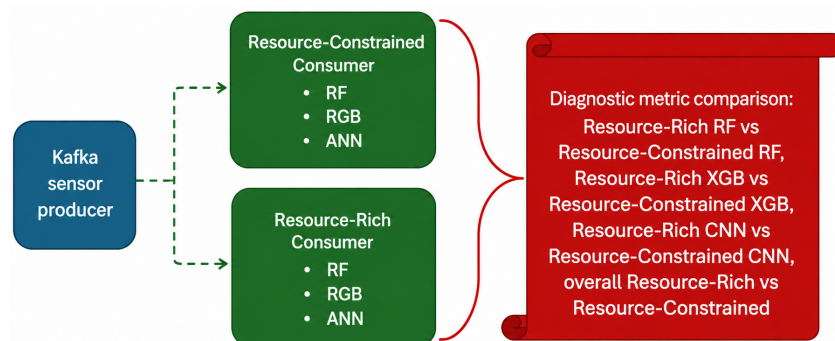


Figure 2. Overview of the streaming-based evaluation setup under different resource configurations.

The architecture comprised the following main components:

- **Kafka Producer (Sensor Stream Simulator)**—a Python-based producer that streamed vibration signals from the dataset to an Apache Kafka topic, emulating real-time data acquisition.
- **Resource-Constrained Consumer**—a constrained inference module operating under limited computational resources, representing Resource-constrained execution conditions.
- **Resource-Rich Consumer**—a higher-resource inference module operating under relaxed constraints, representing resource-rich execution conditions.

This setup enabled the evaluation of model behavior under different resource profiles in a controlled streaming environment.

It should be noted that the adopted streaming pipeline is based on the replay of pre-recorded vibration data and therefore represents a controlled and idealized approximation

of a real-time industrial streaming environment. The objective was to provide reproducible evaluation conditions rather than to fully reproduce the complexity of operational data streams.

The overall pipeline was implemented using Python, Apache Kafka, and standard machine learning and deep learning libraries (scikit-learn, TensorFlow/Keras), ensuring full reproducibility.

Apache Kafka is widely used for event streaming in industrial IoT systems, providing scalable and fault-tolerant data processing capabilities [33]. In this context, Kafka supports real-time data ingestion and processing, enabling the implementation of streaming-based AI inference pipelines.

3.4. Evaluation Metrics

The sustainability of industrial AI has become a critical priority, with increasing attention to the energy consumption of ML models [13]. This aspect becomes even more relevant when considering the different computational resources available across deployment configurations, which can significantly influence both the operational behavior and the overall efficiency of AI-based systems.

Edge computing is often associated with improved sustainability, as it can reduce the need for data transmission to centralized data centers, thereby decreasing overall energy consumption. Furthermore, the adoption of energy optimization strategies at edge computing sites can contribute to lowering the environmental impact of AIoT applications. However, the extent of these benefits depends on the specific deployment conditions and system design choices.

In this context, the comparison between deployment configurations considered both predictive performance and system-level efficiency, which represent the two main aspects analyzed in this study. To support the sustainability perspective, resource utilization and computational load were examined with respect to their impact on model deployment. Accordingly, both predictive and operational metrics were evaluated to provide a comprehensive assessment of model behavior under different resource-based configurations.

In this study, the predictive performance of the models was primarily evaluated in terms of accuracy, in order to provide an interpretable assessment of model effectiveness. Since the objective of the proposed approach was to analyze performance–resource trade-offs at the system level, accuracy was used as a concise indicator of predictive effectiveness that can be directly interpreted alongside latency, memory utilization, and CPU usage. To complement this assessment and account for the class-frequency distribution of the dataset, the Weighted F1-score is also reported. This metric provides a class-sensitive measure of predictive effectiveness and enables a more comprehensive evaluation of model behavior across the different health-state categories. In this context, the primary goal of our resource-aware method was to investigate how fundamentally different algorithmic families (ensemble trees versus deep neural networks) dynamically behaved when embedded in a realistic, hardware-throttled streaming ecosystem characterized by different resource boundaries (resource-constrained vs. resource-rich). Accordingly, Random Forests, XGBoost, and Convolutional Neural Networks were intentionally selected as representative standards for distinct computational paradigms to map their structural trade-offs between processing latency, hardware footprint, software-defined energy consumption, and prognostic stability under strict isolation constraints. Consequently, accuracy is reported as a percentage-point gain with respect to a random baseline corresponding to uniform class assignment across the seven classes of the dataset. This representation highlights the effective discriminative capability of each model beyond trivial classification performance.

Operational behavior was evaluated strictly in terms of system-level End-to-End Latency, CPU utilization, and memory usage. Because the architecture relies on the asynchronous polling, batching, and non-blocking I/O dynamics inherent to Apache Kafka, attempting to isolate the bare computational inference execution time at a microsecond level would introduce artificial profiling overhead, thereby distorting the empirical CPU and memory measurements. Consequently, End-to-End Latency was captured as an emergent, holistic metric of the entire streaming pipeline, encompassing data ingestion, Kafka message polling, deserialization, feature processing, and model inference. For the resource-constrained configuration, since execution occurs entirely within a localized container with zero network hops, this value directly represents the local processing throughput. Conversely, it is important to note that latency values associated with the resource-rich configuration include the stochastic communication-delay profiles (low, medium, and high uniform distributions) introduced to emulate distributed wide-area network environments. Therefore, the End-to-End Latency should be interpreted as a scenario-dependent indicator of integrated system behavior rather than as a pure measure of standalone mathematical model inference time, an essential distinction in streaming-based industrial applications where communication overhead and infrastructure-level orchestration significantly govern the overall response time. Although inference-only latency may provide additional insight into model-level computational behavior, the present study prioritized End-to-End Latency in order to capture the operational characteristics of the complete streaming pipeline. Future work will investigate the separate contribution of inference execution, communication delays, and pipeline overhead to overall system latency.

CPU utilization is reported as percentage usage, computed as the ratio between the observed CPU usage. This normalization enables comparability between resource-constrained and resource-rich configurations. Memory usage is reported as the percentage of allocated container memory utilized during execution, allowing consistent comparison across configurations with different memory limits.

To support the sustainability perspective, process-level energy and emission estimates were obtained using CodeCarbon v3.0.4 (Mila, Montréal, QC, Canada). However, since all experiments were conducted on the same physical machine, these estimates primarily reflect local computational workload rather than infrastructure-level differences. In particular, CodeCarbon does not account for factors such as data center overhead, cooling systems, or network transmission beyond the host machine. For this reason, energy and emission values were interpreted as relative indicators of computational load and were used as secondary metrics to complement the analysis, rather than as absolute measures of sustainability differences between resource-constrained and resource-rich infrastructures. In this context, resource utilization metrics can be interpreted as proxies for environmental impact in controlled experimental settings, enabling comparative sustainability analysis even in the absence of infrastructure-level measurements.

All models were evaluated on identical data streams generated through the Kafka-based pipeline. The resource-constrained and resource-rich configurations differed only in terms of allocated computational resources and simulated communication conditions. This controlled setup enabled a consistent comparison of model behavior under different deployment profiles while isolating system-level effects. To ensure statistical rigor and evaluate the reproducibility of the streaming pipeline, all experimental runs were repeated 5 times under independent data streaming iterations. Operational metrics, including End-to-End Latency, CPU utilization, and memory allocation, are therefore reported as mean $\pm$ standard deviation across the five executions. Predictive metrics, namely accuracy gain and Weighted F1-score, are deterministic for a fixed training/test partition and model configuration and are therefore reported as single values. To determine the statistical signif-

icance of the operational variations between the resource-rich and resource-constrained profiles, two-tailed Student's *t*-tests were performed with a significance threshold of $\alpha = 0.05$. The statistical analysis confirmed that the computational and transmission discrepancies between the two environmental profiles were highly significant ($p < 0.01$), validating that the architectural constraints realistically differentiated the benchmarked paradigms.

4. Results

This section reports the experimental results obtained by applying the proposed resource-aware evaluation method to different machine learning and deep learning models under resource-constrained and resource-rich configurations. Predictive performance and operational metrics are jointly analyzed to assess model behavior under varying resource conditions.

Table 3 summarizes the results for Random Forest (RF), XGBoost (XGB), and Convolutional Neural Network (CNN) models.

Table 3. Experimental results for RF, XGB, and CNN models under resource-constrained and resource-rich configurations, jointly reporting predictive effectiveness and operational performance metrics. Operational metrics are reported as mean $\pm$ standard deviation over 5 independent executions.

Model	Accuracy Gain (pp)	Weighted F1-Score	End-to-End Latency (ms $\pm$ SD)	CPU Utilization (% $\pm$ SD)	Mem Utilization (% $\pm$ SD)
Resource-Constrained Configuration					
RF	+25.7	0.389	26.48 $\pm$ 1.15	0.17 $\pm$ 0.04	2.90 $\pm$ 0.11
XGB	+23.8	0.327	19.75 $\pm$ 0.92	0.23 $\pm$ 0.05	2.38 $\pm$ 0.08
CNN	+51.6	0.653	12.35 $\pm$ 0.65	0.08 $\pm$ 0.02	0.95 $\pm$ 0.04
Resource-Rich Configuration					
RF	+25.7	0.389	161.75 $\pm$ 28.45	0.45 $\pm$ 0.08	2.96 $\pm$ 0.12
XGB	+23.8	0.327	162.53 $\pm$ 29.10	0.46 $\pm$ 0.08	2.41 $\pm$ 0.09
CNN	+51.6	0.653	152.99 $\pm$ 31.25	0.14 $\pm$ 0.03	6.36 $\pm$ 0.22

From a predictive-performance perspective, the results highlight that model complexity is the dominant factor influencing classification effectiveness. Classical machine learning models achieve moderate performance, with RF and XGB reaching accuracy gains of +25.7 and +23.8 percentage points over the random baseline, respectively. In contrast, the CNN model achieves a substantially higher gain (+51.6), confirming its superior ability to capture complex patterns in vibration signals. Notably, these performance levels remain consistent across resource configurations, indicating that predictive capability is primarily model-driven rather than configuration-dependent. The Weighted F1-score confirms the same trend observed through accuracy gain, with the CNN consistently achieving higher predictive effectiveness than the machine learning models across both resource configurations.

Regarding latency, the resource-constrained configuration consistently exhibits lower end-to-end response times across all models. However, latency values associated with the resource-rich configuration include predefined communication-delay assumptions introduced to emulate remote inference conditions. As such, latency should be interpreted as a system-level metric rather than a pure measure of model inference time. Therefore, the latency differences observed between the resource-constrained and resource-rich configurations should not be attributed solely to computational resource availability but rather to the combined effects of resource allocation, communication latency, and pipeline-level processing. Under this interpretation, the results suggest that communication overhead

and pipeline-related factors play a critical role in shaping response time in streaming-based deployments.

Resource utilization patterns further highlight the impact of resource allocation on system behavior. CPU utilization patterns indicate that the resource-rich configuration exhibits higher CPU usage for ML models compared to the resource-constrained setup, meaning that the values represent the aggregate usage across the 10 allocated cores compared to the single-core constraint. This suggests that increased resource availability may lead to more intensive system-level activity, including communication, buffering, and orchestration overhead, rather than a simple redistribution of computational load. The relatively low CPU utilization values observed across all configurations are expected because utilization is measured as an average over the complete streaming execution. Since inference operations are lightweight and data are processed intermittently, Kafka consumers spend a significant portion of the execution time waiting for incoming messages. Consequently, the reported values should be interpreted as indicators of average system-level processor utilization rather than peak computational demand during model inference.

A similar trend is observed for memory usage. Memory utilization, expressed as a percentage of allocated container memory, remains relatively low across all configurations but shows slightly higher values in the resource-rich setup. This suggests that increased resource availability may introduce additional buffering or system-level overhead rather than strictly model-driven memory consumption.

Process-level energy and emission estimates obtained through CodeCarbon do not show significant differences between configurations. This outcome is expected, as all experiments were conducted on the same physical machine. Therefore, these metrics primarily reflect local computational workload and should be interpreted as relative indicators rather than as evidence of infrastructure-level sustainability differences.

Overall, the results demonstrate that while predictive performance is largely determined by model characteristics, operational behavior is strongly influenced by resource allocation and system-level factors. These findings validate the usefulness of the proposed method in capturing the interaction between model complexity and deployment conditions in streaming environments.

5. Discussion

The results highlight the importance of evaluating predictive maintenance models from a system-level perspective, where predictive performance and operational behavior are jointly considered under resource constraints.

From a predictive standpoint, the superior performance of the CNN model confirms the effectiveness of deep learning approaches in capturing complex temporal and spectral patterns in vibration signals. However, the consistency of accuracy across resource-constrained and resource-rich configurations indicates that predictive performance is largely independent of resource availability, reinforcing the idea that model design remains the primary driver of classification capability.

In contrast, operational metrics reveal a strong dependence on resource configuration. The lower latency observed in the resource-constrained setup is primarily attributable to the absence of communication overhead, rather than to differences in model inference efficiency. This suggests that, in streaming scenarios, system-level factors such as data transfer and pipeline design can dominate overall response time. Consequently, latency should be interpreted as an emergent property of the entire system rather than a model-specific characteristic.

CPU and memory utilization patterns provide further insight into the interaction between model execution and resource allocation. The higher CPU utilization observed

in the resource-rich configuration suggests that additional system-level overhead, such as data transfer, buffering, and orchestration, contributes significantly to computational activity. This indicates that increased resource availability does not necessarily lead to more efficient execution but may instead introduce additional processing demands at the system level. Interestingly, higher resource availability does not translate into lower CPU utilization per core, highlighting the presence of non-negligible system-level overhead in resource-rich configurations.

Energy and emission estimates, obtained through process-level monitoring, show negligible differences between configurations. This is consistent with the experimental setup, where all measurements were performed on a single physical machine. As a result, these metrics should be interpreted as proxies for computational load rather than as direct indicators of sustainability differences between deployment paradigms. Consequently, they do not fully capture the overall environmental footprint of resource-constrained and resource-rich deployments, which is also influenced by infrastructure-level factors beyond the scope of the present study. This observation underscores the importance of complementing process-level measurements with infrastructure-level analyses in future work. Accordingly, the sustainability-related findings reported in this study should be interpreted primarily as indicators of relative resource efficiency and computational workload rather than as direct assessments of the environmental sustainability of alternative deployment configurations.

Overall, the findings demonstrate that increasing resource availability does not necessarily lead to improved system efficiency. Instead, system-level overhead and communication dynamics can offset potential benefits, particularly in distributed or resource-rich scenarios. This reinforces the relevance of the proposed resource-aware evaluation method, which enables a structured analysis of performance–resource trade-offs beyond traditional accuracy-focused assessments.

From a deployment perspective, it is important to contextualize the architectural differences among the evaluated model families. The CNN model was deployed using the TensorFlow Lite runtime in both the resource-constrained and resource-rich configurations. No post-training quantization or precision reduction techniques were applied, and the same model architecture and trained weights were maintained across all experiments. This choice was made intentionally to isolate the impact of resource allocation on operational behavior without introducing additional variability due to model compression or numerical precision changes. Conversely, Random Forest and XGBoost were deployed in their standard implementations and operated on handcrafted statistical and spectral features, whereas the CNN directly processed raw bi-axial vibration signals. Consequently, the comparison should not be interpreted as a strict architectural benchmark but rather as an illustrative evaluation of representative machine learning and deep learning paradigms within the proposed resource-aware evaluation methodology.

A limitation of the present study is that the proposed evaluation framework was validated using only the NASA IMS Bearing Dataset. Although this dataset is a widely recognized benchmark in predictive maintenance research, the use of a single dataset may limit the generalizability of the reported findings. An additional limitation concerns the labeling strategy adopted for the NASA IMS Bearing Dataset. The health-state categories used in this work were derived from degradation stages reported in previous studies rather than from expert annotations independently validated within the present investigation. Consequently, labels such as “Imminent Failure” and “Stage 2 Failure” should be interpreted as literature-derived degradation categories reflecting retrospective analyses of the dataset. While this choice ensures consistency with previous benchmark studies, it also reflects the broader limitations associated with post hoc labeling schemes commonly encountered in

legacy predictive-maintenance datasets. Another limitation of the present study is that the streaming environment is based on a controlled replay of historical data. As a result, several operational characteristics of real industrial streaming systems, including data-rate mismatches, backpressure effects, out-of-order messages, late arrivals, network unreliability, buffering overflows, and deadline misses, are not explicitly represented. Another limitation of the present study is that deployment-specific hyper-parameter optimization was not considered. The same optimized hyper-parameter configurations were maintained across all resource profiles to ensure a controlled comparison of model behavior under different resource conditions. While this design choice supports the methodological objective of isolating resource-related effects, future studies could investigate whether distinct hyper-parameter configurations emerge as optimal under different resource constraints.

From a methodological perspective, the proposed approach provides a generalizable framework for evaluating AI models in streaming environments. By explicitly modeling resource constraints and combining predictive and operational metrics, the method enables a more comprehensive understanding of model behavior under realistic deployment conditions. Nevertheless, future validation on additional industrial datasets characterized by different operating conditions, fault types, and data distributions would further strengthen the assessment of the framework's applicability and the generalizability of the reported findings.

Future work should extend this analysis to physically distributed environments, incorporate direct energy measurements, and explicitly separate model inference latency from communication delay and pipeline overhead. This additional analysis would enable a more detailed understanding of the contribution of individual system components to the overall End-to-End Latency observed in streaming-based predictive maintenance systems. Future work should also include recurrent architectures, such as LSTM and GRU, to assess how their temporal modeling capabilities affect predictive performance, latency, memory usage, and resource efficiency in streaming predictive maintenance scenarios. Such extensions would further strengthen the applicability of the method to real-world industrial scenarios.

6. Conclusions

This work presented a resource-aware method for evaluating AI models in data streaming systems for predictive maintenance, with the aim of analyzing the interaction between predictive performance and system-level resource constraints.

The proposed approach integrates a streaming-based pipeline, configurable resource profiles, and a joint set of predictive and operational metrics, enabling a structured assessment of model behavior under resource-constrained and resource-rich configurations. Unlike traditional evaluation approaches focused primarily on accuracy, the proposed method provides a more comprehensive perspective by explicitly capturing performance–resource trade-offs.

The experimental results highlight that predictive performance is largely driven by model complexity, while operational behavior is significantly influenced by resource allocation and system-level factors. In particular, latency emerges as a system-level property affected by communication overhead and pipeline design, rather than as a purely model-dependent metric. Similarly, resource utilization patterns demonstrate that increased resource availability does not necessarily translate into improved efficiency, as workload distribution and system overhead play a critical role.

These findings emphasize the importance of adopting resource-aware evaluation strategies when designing AI-based predictive maintenance systems, especially in streaming environments where computational constraints and real-time requirements must be jointly considered.

From a methodological perspective, the main contribution of this work lies in the definition of a generalizable evaluation methodology that can be applied to different models and deployment scenarios. By enabling controlled analysis under varying resource conditions, the proposed method supports more informed design choices for sustainable and efficient AI systems. The reported findings are valid within the context of the adopted controlled resource emulation and should not be interpreted as direct evidence of behavior in real distributed resource-constrained and resource-rich infrastructures.

Future work will focus on extending the evaluation to physically distributed environments, incorporating infrastructure-level energy measurements, and further analyzing the contribution of individual system components (e.g., inference, communication, and buffering) to overall performance. Furthermore, future research could investigate the impact of deployment-oriented optimization techniques, including quantization, pruning, and model compression strategies, on both deep learning and traditional machine learning models operating under resource-constrained conditions. Future research could also investigate deployment-aware hyper-parameter optimization strategies to determine whether different resource profiles require distinct model configurations to maximize the trade-off between predictive effectiveness and operational efficiency. These extensions will enhance the applicability of the method to real-world industrial settings and strengthen its role in supporting resource-efficient and sustainable AI deployment strategies. Future work will also include the validation of the proposed framework on additional predictive maintenance datasets to further assess the generalizability of the reported findings.

Author Contributions: Conceptualization, L.S. and A.B.; methodology, S.R.; software, L.S. and M.E.; validation, L.S., A.B., M.E., S.R. and P.P.; formal analysis, L.S.; investigation, L.S. and S.R.; resources, M.E.; data curation, L.S. and M.E.; writing—original draft preparation, L.S. and A.B.; writing—review and editing, A.B., S.R., M.E. and P.P.; visualization, S.R.; supervision, A.B. and P.P.; project administration, P.P.; funding acquisition, P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study and the code used for performing the experiments are available on GitHub at https://github.com/LuisianaSabbatini/edgecloud_docker_project (accessed on 29 June 2026). This work was derived from the following resources (raw data) available in the public domain: <https://data.nasa.gov/dataset/ims-bearings> (accessed on 29 June 2026).

Acknowledgments: This study has been partially supported within the project entitled “CoSoTePa—Conveyor Solid technology Path”, funded within POR CAMPANIA FESR 2021/2027, CUP B65H23004430007. During the preparation of this manuscript, the authors used ChatGPT (OpenAI, San Francisco, CA, USA; GPT-5.5) to assist with language editing and to improve the clarity of the English expression. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Onu, P.; Pradhan, A.; Madonsela, N.S.; Ajayi, S.A.; Osueke, C.O.; Samuel, D.R.; Adebayo, S. Integration of AI and IoT in Smart Manufacturing: Exploring Technological, Ethical, and Legal Frontiers. *Procedia Comput. Sci.* **2025**, *253*, 654–660. [CrossRef]
2. Mansour, M.; Lafta, A.M.; Salman, A.M.; Salman, H.S. Exploring the Impact of AI and IoT on Production Efficiency, Quality Precision, and Environmental Sustainability in Manufacturing. *Vokasi Unesa Bull. Eng. Technol. Appl. Sci.* **2025**, *2*, 342–355. [CrossRef]

3. Awaisi, K.S.; Ye, Q.; Sampalli, S. A Survey of Industrial AIoT: Opportunities, Challenges, and Directions. *IEEE Access* **2024**, *12*, 96946–96996. [\[CrossRef\]](#)
4. Zhang, S.; Zhang, S.; Wang, B.; Habetler, T.G. Machine Learning and Deep Learning Algorithms for Bearing Fault Diagnostics: A Comprehensive Review. *IEEE Access* **2020**, *8*, 29857–29881. [\[CrossRef\]](#)
5. Park, S.; Ahn, G.J.; Im, D.H. Auto Labeling Methods Developed Through Semi-Weakly Supervised Learning in Prognostics and Health Management Applications for Rolling Ball Bearing. *IEEE Sens. J.* **2022**, *22*, 16223–16233. [\[CrossRef\]](#)
6. Dey, S.; Sharma, P. Predictive Maintenance for Smart Manufacturing: An AI and IoT-Based Approach. *Libr. Prog.-Libr. Sci. Inf. Technol. Comput.* **2024**, *44*, 11406–11423.
7. Arifin, M.; Ramadhan, A.; Wijaya, H. A Comparative Study of Edge and Cloud Computing Architectures for Industrial IoT Applications. *Int. J. Inf. Eng. Sci.* **2024**, *1*, 11–15. [\[CrossRef\]](#)
8. Zhu, S.; Ota, K.; Dong, M. Green AI for IIoT: Energy Efficient Intelligent Edge Computing for Industrial Internet of Things. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 79–88. [\[CrossRef\]](#)
9. Matin, A.; Islam, M.R.; Wang, X.; Huo, H.; Xu, G. AIoT for sustainable manufacturing: Overview, challenges, and opportunities. *Internet Things* **2023**, *24*, 100901. [\[CrossRef\]](#)
10. Sathupadi, K.; Achar, S.; Bhaskaran, S.V.; Faruqui, N.; Abdullah-Al-Wadud, M.; Uddin, J. Edge-Cloud Synergy for AI-Enhanced Sensor Network Data: A Real-Time Predictive Maintenance Framework. *Sensors* **2024**, *24*, 7918. [\[CrossRef\]](#) [\[PubMed\]](#)
11. Li, P.; Islam, M.J.; Ren, S. A Case Study of Environmental Footprints for Generative AI Inference: Cloud versus Edge. *ACM SIGMETRICS Perform. Eval. Rev.* **2025**, *53*, 21–26. [\[CrossRef\]](#)
12. Arpaia, P.; Capobianco, L.; Caputo, F.; Cioffi, A.; Esposito, A.; Isgrò, F.; Moccaldi, N.; Pau, D.; Siorpaes, D.; Toscano, E. Accurate Energy Measurements for Tiny Machine Learning Workloads. In Proceedings of the 2024 IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRINE), St Albans, UK, 21–23 October 2024; pp. 831–836. [\[CrossRef\]](#)
13. Kassai, K.; Dagiklas, T.; Bashir, S.; Iqbal, M. GreenBytes: Intelligent Energy Estimation for Edge-Cloud. *arXiv* **2024**, arXiv:2403.04665.
14. Rehan, H. Energy Efficiency in Smart Factories: Leveraging IoT, AI, and Cloud Computing for Sustainable Manufacturing. *J. Comput. Intell. Robot.* **2021**, *1*, 18–55.
15. Fraga-Lamas, P.; Lopes, S.I.; Fernández-Caramés, T.M. Green IoT and Edge AI as Key Technological Enablers for a Sustainable Digital Transition towards a Smart Circular Economy: An Industry 5.0 Use Case. *Sensors* **2021**, *21*, 5745. [\[CrossRef\]](#) [\[PubMed\]](#)
16. Rakholia, R.; Suárez-Cetrulo, A.L.; Singh, M.; Carbajo, R.S. Integrating AI and IoT for Predictive Maintenance in Industry 4.0 Manufacturing Environments: A Practical Approach. *Information* **2025**, *16*, 737. [\[CrossRef\]](#)
17. Francy, S.; Singh, R. Edge AI: Evaluation of Model Compression Techniques for Convolutional Neural Networks. *arXiv* **2024**, arXiv:2409.02134.
18. NASA Prognostics Data Repository. IMS Bearing Dataset. 2006. Available online: <https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/> (accessed on 11 June 2026).
19. Qiu, H.; Lee, J.; Lin, J.; Yu, G. Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics. *J. Sound Vib.* **2006**, *289*, 1066–1090. [\[CrossRef\]](#)
20. Khorram, A.; Khalooei, M.; Rezghi, M. End-to-end CNN+ LSTM deep learning approach for bearing fault diagnosis. *Appl. Intell.* **2021**, *51*, 736–751. [\[CrossRef\]](#)
21. Loghin, D.; Ramachandran, L.; Zhang, J.; Teo, Y.M. An Evaluation of Skywriting and Apache Kafka for Distributed Data Streaming. In Proceedings of the 2020 IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, 2 November 2020; pp. 121–131. [\[CrossRef\]](#)
22. Frampton, M.; Barker, A. Benchmarking Distributed Message Brokers: Kafka, Pulsar and RabbitMQ. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 3102–3111. [\[CrossRef\]](#)
23. Xu, J.; Wang, Y.; Yuan, B.; Jiang, H.; Liu, S. Diet CNN: A resource-aware framework for edge-based intelligence. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *40*, 1124–1135. [\[CrossRef\]](#)
24. Premasankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [\[CrossRef\]](#)
25. Alomari, Z.; Li, Z.; Makanju, A. Lightweight machine learning-based IDS for IoT environments. In *Proceedings of the 2024 8th Cyber Security in Networking Conference (CSNet)*; IEEE: New York, NY, USA, 2024; pp. 33–37.
26. Lamdjad, B.; Chaiter, A. AI-Powered Predictive Maintenance and Prognostic Health Management Using Edge-Based Predictive Algorithms for Industrial Operations. *Algorithms* **2026**, *19*, 286. [\[CrossRef\]](#)
27. Padella, A. Process and Resource-Aware Responsible Recommender Systems (Extended Abstract). In *Proceedings of the Joint Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum at BPM 2025 Co-Located with 23rd International Conference on Business Process Management (BPM 2025)*, Seville, Spain, 31 August–5 September 2025; Reijers, H.A., Marrella, A., del-Río-Ortega, A., Rinderle-Ma, S., Depaire, B., Rehse, J., Santoro, F.M., Zerbato, F., Márquez-Chamorro, A.E., Beerepoot, I., et al., Eds.; CEUR Workshop Proceedings; CEUR-WS.org: Aachen, Germany, 2025; Volume 4032, pp. 6–10.

28. Schwartz, R.; Dodge, J.; Smith, N.A.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [CrossRef]
29. Lannelongue, L.; Grealey, J.; Inouye, M. Green algorithms: Quantifying the carbon footprint of computation. *Adv. Sci.* **2021**, *8*, 2100707. [CrossRef]
30. Dodge, J.; Prewitt, T.; Borchardt, R.; Luccioni, A.S.; Knickerbocker, P.; Balch, E.; Yuan, B.; Roy, S.; Roy, S.; Smith, N.A.; et al. Measuring the carbon intensity of AI in cloud instances. In Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, 21–24 June 2022; pp. 1877–1894.
31. NASA. IMS Bearings Dataset. NASA Open Data Portal. 2025. Available online: <https://data.nasa.gov/dataset/ims-bearings> (accessed on 12 September 2025).
32. You, D.; Chen, L.; Liu, F.; Zhang, Y.; Shang, W.; Hu, Y.; Liu, W. Intelligent Fault Diagnosis of Bearing Based on Convolutional Neural Network and Bidirectional Long Short-Term Memory. *Shock Vib.* **2021**, *2021*, 7346352. [CrossRef]
33. OpenLogic. Apache Kafka Stream Processing Use Cases. OpenLogic Blog. 2024. Available online: <https://www.openlogic.com/blog/kafka-stream-processing> (accessed on 12 September 2025).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.