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

Journal Européen des Systèmes Automatisés
Vol. 59, No. 10, October, 2026, pp. 2767-2783
Journal homepage: http://iieta.org/journals/jesa

Edge AI-Based Fault Detection Framework for Smart Factories

Ihab Jawad Abdulkadhim, Mohammed R. Mandeel*

Department of Construction and Projects, University of Al-Qadisiyah, Al Diwaniyah 58002, Iraq

Corresponding Author Email: Mohammed.rasool_mandeel@qu.edu.iq

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/jesa.591002

Received: 8 June 2026
Revised: 10 August 2026
Accepted: 18 August 2026
Available online: 31 October 2026

Keywords:
edge AI, fault detection, smart manufacturing, deep learning, predictive maintenance

ABSTRACT

Since industrial environments are becoming increasingly complex in the context of Industry 4.0 and Industry 5.0, intelligent fault detection systems with low latency and low energy consumption are highly desirable. In this paper, a Residual convolutional neural network (CNN)-based edge AI fault detection framework for smart factories is established to achieve real-time industrial monitoring by analyzing vibration signals from the Case Western Reserve University (CWRU) bearing dataset. The proposed framework consists of four main steps: data acquisition, preprocessing, feature extraction, and classification. In addition, normalization and CNN optimization techniques are adopted to improve the inference speed and accuracy of the edge AI system when deployed on edge platforms with limited computational resources. An edge-based deep learning model is proposed and compared with cloud-based deep learning models and traditional machine learning methods, including support vector machine (SVM) and random forest (RF). The results demonstrate that the proposed model achieves higher classification accuracy, shorter inference time, and lower energy consumption than cloud-based and traditional machine learning approaches under the experimental conditions using the CWRU bearing dataset. Furthermore, confusion matrix analysis is conducted to further validate the effectiveness of the proposed model in identifying different bearing fault types within the limitations of the employed dataset. The results suggest that edge intelligence systems have potential for real-time fault detection in smart manufacturing scenarios where low latency and resource constraints are critical considerations. However, further validation under real industrial conditions is required to confirm their applicability beyond laboratory environments.

1. INTRODUCTION

1.1 Background of smart factories and Industry 4.0/5.0

Industry 4.0 has fundamentally transformed the manufacturing sector by introducing a new industrial paradigm driven by cyber-physical systems (CPS), the Industrial Internet of Things (IIoT), big data, and AI [1, 2]. Within this environment, factories can achieve self-optimization, predictive maintenance, and production process reconfiguration. The key pillars of Industry 4.0 focus on bridging the gap between physical and digital systems while improving the productivity, efficiency, and flexibility of industrial processes. In contrast, Industry 5.0 places greater emphasis on human-centricity, sustainability, and resilience, promoting a future in which humans and intelligent systems work collaboratively [3]. As a result, modern industrial systems are becoming increasingly complex and intelligent.

1.2 The need for real-time fault detection in an industrial environment

It is necessary to understand the need for real-time fault detection in an industrial environment due to the fact that in an industrial environment the huge amount of cost due to downtime, operator safety and potential damage of equipment cannot be tolerated. Batch-processing approach are not pertinent here as they are required only when the processing has been completed entirely and involves a delay on action to be taken to correct the problem. In a smart factory environment, machines are producing and streaming data from multiple sensors constantly and it has to be observed and analysed immediately in order to avoid any breakdown on the system. Batch-processing approaches are not applicable in this case, because they are used only once the processing is completed and it means a little delay on the intervention of corrective actions. Recent research shows that a condition monitoring, predictive maintenance and real time fault detection could provide a significant increase of performance and reduction of maintenance costs by fault detection and correction prior to failure to serious extents [4]. As industries become more complex, more intelligent and faster systems have to be implemented.

1.3 Limitations of cloud-based AI systems

Cloud infrastructure, even though scalable and with enough computing power for AI applications, is still limited for industrial real-time systems (network latency, limited bandwidth for remote industrial sites, concern about private

operational data being sent to cloud servers, etc.).

Conventional cloud-based AI systems may not satisfy the strict real-time requirements of industrial control systems, where millisecond-level response times are critical to system reliability and safety [5]. Furthermore, continuous data transmission between industrial devices and cloud platforms can consume substantial network bandwidth and increase the risk of industrial data security breaches.

1.4 The advent of edge AI for industrial sites

Edge AI helps overcome the limitations of cloud computing by enabling intermediate platforms to perform real-time data analysis closer to the data source. This approach reduces communication delays and accelerates response times, making it suitable for real-time industrial applications [6, 7]. In this way, data can be handled near where it is generated rather than relying solely on distant data centers. Fog and edge computing extend the distributed computing architecture of cloud computing by introducing an intermediate processing layer between intelligent devices and central servers. This layer enables data streams to be processed efficiently in Internet of Things (IoT) environments [8, 9]. It thereby bridges the gap between local devices and centralized cloud resources. These technologies are widely used in smart factories for predictive maintenance, anomaly detection, and real-time monitoring [10].

1.5 Novelty and contributions

Although the idea of integrating deep learning and edge computing for fault detection has been considered in previous work, our framework makes the following contributions:

Systematic Architecture Selection: In contrast to previous studies that focus on a single architecture, this work systematically compares and evaluates three candidate models, namely convolutional neural network–long short-term memory (CNN-LSTM), MobileNet, and Residual CNN, for one-dimensional (1D) vibration signal classification on edge devices. The selection is based on empirical trade-offs among classification accuracy, inference latency, and model size rather than ad hoc decisions.

Novel Integrated Optimization Pipeline: Although pruning, quantization, and knowledge distillation have been extensively studied as individual techniques, their sequential integration with specific optimization trade-offs represents a contribution of this work. The proposed pipeline achieves an accuracy degradation of only 1.5%. Furthermore, the quantitative evaluation of accuracy changes at each optimization stage provides practical insights into model optimization for edge deployment.

Real Hardware Validation: Unlike many previous studies that rely on simulated edge deployment environments, this work presents actual experimental measurements using NVIDIA Jetson Nano hardware. A detailed experimental setup is established to measure inference latency and power consumption using external measurement instruments.

Unified Framework: The proposed framework integrates data acquisition, preprocessing, model selection, optimization, and deployment into a unified pipeline, covering the entire process from sensor data collection to decision-making. This provides a comprehensive framework for industrial applications rather than focusing solely on isolated algorithmic development.

1.6 Research problem statement

As a result of the long latency, bandwidth constraints and the privacy issues of current cloud-based fault detection systems (which do not meet the real-time industrial application), the following need to be identified; a highly accurate and lightweight, real-time fault detection framework to run directly on the industrial edge.

1.7 Research objectives

The main objectives of this research are to:

Design an edge AI-based real-time fault detection framework for industrial systems employing a Residual CNN architecture chosen from exhaustive model evaluation.

Reduce detection latency by implementing processing at the edge with KPI-driven latency targets (<10 ms).

Describe and examine the differences in the performance trade-offs between edge-based and cloud-based AI systems in the dimensions of accuracy, latency, and power consumption.

Practical deployment viability demonstrated through real hardware measurements on representative edge platforms.

1.8 Research questions

This research is focused on answering the following questions:

What are the benefits of edge-based AI for real-time fault detection in terms of latency and energy consumption?

What quantifiable trade-offs exist between a cloud-based AI system and an edge-based one?

What is the best architecture for accuracy and resource needs for edge deployment?

How to combine model optimization methods (pruning, quantization, knowledge distillation) most efficiently and successfully to deploy on resource-limited hardware with acceptable accuracy?

1.9 Scope and limitations

This work is largely concerned with the application of edge AI for real-time fault detection in smart factories. The use of large-scale cloud training infrastructure is outside the scope of this work, which is concerned solely with the learning models that are deployed on edge devices. Furthermore, there are limitations to the hardware constraints on the edge device, the changing environment, the accessibility of the dataset, and the limitations of the Case Western Reserve University (CWRU) dataset used as a laboratory benchmark. To overcome this issue, the proposed framework is further evaluated on the Intelligent Maintenance Systems (IMS) dataset to assess its applicability to more practical industrial use cases.

2. LITERATURE REVIEW

2.1 Smart manufacturing systems: Technologies and limitations

The advent of smart manufacturing relies on a shift from conventional shop floors to data-centric, interconnected industrial settings as described by Industry 4.0 [9]. Smart factories leverage CPS, the IIoT, and AI to deliver self-optimization, predictive maintenance, and reconfigurability

[11].

2.1.1 Key enabling technologies

Three foundational technologies underpin smart manufacturing:

The IIoT integrates physical assets in the digital world through sensors and communication infrastructure [12]. According to the latest research, five driving factors are essential for IIoT adoption, including scalability (large number of devices), heterogeneity (varied sensors, protocols), security (operational data), real-time communication, and interoperability [12].

Cyber-Physical Systems (CPS): CPS combines computing with physical processes by means of monitoring, control and prediction [9]. CPS challenges include: how to synchronize physical and cyber components in large-scale deployments, how to manage the systems' complexity and how to maintain real time responsiveness when faced with high loads [13]. These challenges are more relevant for edge-based CPS where computing power is limited.

Digital Twins (DT): DT is used to construct digital models of physical assets for analysis and management [14]. Despite proving to be useful in areas such as design optimization and predictive maintenance, Digital Twin adoption is limited because of the high volume of data needed (from continuous sensor streams), difficulty of integration with existing legacy systems, and high computational demands of running in real-time [14]. As a result, Digital Twins are not always feasible for smaller manufacturers or on the edge.

2.1.2 Critical analysis and gap identification

While the growing intensity in the IIoT, the CPS, and the DT has led to some progress, three basic issues for edge-based fault detection still remain.

Existing CPS implementations first generally depend on cloud-driven analytics, which brings unacceptable latency for real time fault diagnosis [13]. Second, DT requires a huge amount of data and is computationally expensive, making it unfeasible for resource-constrained edge devices [14]. Third, IIoT provides the data infrastructure but not the challenge of executing complex AI models in the limited memory and computational power of edge devices [12].

This framework mitigates the above-mentioned disadvantages by performing fault detection locally on edge devices and thus removing the bandwidth and latency constraints inherent to cloud-based models, while ensuring high detection accuracy via model optimization techniques. Additionally, unlike DT methods that require constant high-fidelity simulation, this framework allows for inexpensive, real-time fault detection in industry.

2.2 Fault detection techniques: From rule-based to deep learning

Fault detection techniques for industrial processes have been classified into three generations: first-generation rule-based techniques, second-generation machine learning-based techniques, and third-generation deep learning-based techniques.

2.2.1 Traditional approaches and their limitations

Rule-based techniques require use of preset thresholds and predefined rules using expert knowledge. The technique is computationally fast, but unable to adapt to dynamic, complex, time-varying industrial environment and always needs some batch of domain knowledge to impose rule according to the working conditions [15].

Statistical characteristics of sensor data such as mean, variance, frequency features are provided for different machine learning approaches (e.g. support vector machine (SVM), random forest (RF), k-nearest neighbors (k-NN)) and they train decision boundaries [15]. Yet, two drawbacks for all these approaches are: (1) feature engineering – performance is dependent on features chosen and performance is strongly influenced by performance of feature selection; and (2) sensitivity to noise- features produced are susceptible to environmental noise and change of operating conditions [15].

2.2.2 Deep learning advancements

Deep learning is changing the way fault detection is performed by extracting feature information directly from raw sensor signals. For instance, CNNs extract spatial features, LSTMs utilize time-serial features, and auto-encoders detect anomalies based on reconstruction error. It is proven that deep learning greatly enhances accuracy of condition monitoring of machine health for different operational conditions [15].

Semi-supervised and unsupervised techniques aim to overcome the scarcity of labeled fault data by identifying anomalies in normal operation [16]. Hybrid architectures (CNN-LSTM, CNN-autoencoder) leverage the strengths of various deep learning approaches to improve robustness and generalization [17].

2.2.3 Critical analysis and gap identification

Despite these advances, three challenges persist for edge-based fault detection:

First, computation cost-deep learning models are computationally demanding and need large memory and processor power, which can be difficult to implement in resource-constrained edge devices [15]. Second, generalization deficiency-deep learning models trained on laboratory data (such as CWRU) cannot generalize in real-world conditions with changing loads, speeds and surroundings [18]. Third, incremental learning-deep learning models are mostly trained offline and cannot update themselves to new fault signatures or fault classes [17].

The above-mentioned issues are addressed by model optimization (pruning, quantization, knowledge distillation), validation on multiple datasets (CWRU, IMS), and real-hardware measurement in our proposed framework.

2.3 Edge computing in industry: Architecture and challenges

The cloud-focused architecture has certain intrinsic limitations that were overcome by edge computing, which enables computation at closer proximity to data sources [7]. Industrial scenarios that are time-critical require this form of computing for applications such as online fault detection, predictive maintenance and process control [18].

2.3.1 Edge-fog-cloud hierarchy

Contemporary industrial systems use a three-level hierarchy: (1) Edge layer-in sensors, machines, or other system components; these devices do local data processing, and inference, (2) Fog layer-in gateways aggregating, and preprocessing the data received from large number of edge devices, (3) Cloud layer-in servers for storage, training

models, optimization [19]. This three level hierarchy allows a scalable, and robust industrial IoT systems since the data processing and actions are performed across the layers depending upon its response latency and computational limitations.

2.3.2 Critical analysis and persistent challenges

Despite the advantages that edge computing provides, the following three significant obstacles have prevented widespread adoption of edge computing in practice:

Firstly, resource-constrained edge devices have limited memory, CPU capacity, and battery life, which restricts the deployment of computationally intensive AI models [19]. Secondly, edge devices face security challenges because they are physically accessible and are generally not equipped with the same level of security mechanisms as cloud data centers. This may expose industrial operational data to potential security threats. Thirdly, edge devices from different vendors often employ heterogeneous communication protocols and application programming interfaces (APIs), creating interoperability challenges and complicating system integration [19].

2.3.3 Gap relative to current work

The resource allocation problem is resolved systematically by the model optimization (Section 3.4) proposed here. Through 90.1% reduction of the model size and inference latency down to 6.3 ms, we show that deep learning is viable on resource-constrained edge devices. Unlike other work that discusses theoretical architectures, we here show physical hardware validation on NVIDIA Jetson Nano and measure the latency and energy usage in detail.

2.4 AI at the edge: Model compression and deployment

Edge intelligence, a growing trend that joins together edge computing and artificial intelligence, allows decision-making in real-time in resource-constrained edge devices [20]. It prevents as much as possible the data movement and provides high-speed data analysis and fast insight by studying the data at the edge of the network. As a result, it reduces the reliance on centralized cloud computing services, while compensating for the communication cost, inference latency and privacy costs. This trend fits well with industrial applications where there is a need for prompt reaction, low bandwidth and privacy.

2.4.1 Model compression techniques

Edge devices have very limited memory, computational power and energy resources, which present a significant challenge in deploying deep learning models. Therefore, model compression methods are often applied to successfully reduce the computational and storage demands while maintain prediction accuracy [21]. Those methods can be broadly categorized into three main categories: (1) Pruning; (2) Quantization and (3) Knowledge distillation.

Pruning aids in model complexity reduction by eliminating duplicated/weakweights and connections that have low-magnitude weights and contribute less to the prediction. This led to reduction in effective number of parameters in the model and can also result in significant decrease in computational cost. However, high pruning can affect the accuracy of model and usually needs a fine tuning stage to compensate for losses.

Quantization refers to lowering the numerical precision of the representation of parameters and activation values. For example, the parameters can be represented as 8-bit integer (INT8), instead of 32-bit floating-point (FP32) and get a large reduction in memory and computational costs. If ideal conditions are met, the reduction in memory can be around 4 times compared to the FP32 representation (Figure 2). However, this also relies on hardware and software support.

Knowledge distillation involves taking knowledge from a much more accurate large teacher model and transferring it into a small student network. The small network attempts to follow the predictions of the teacher while using many fewer parameters. This can enable fairly high classification performance with large reductions in model complexity. However, this method adds training complexity as both teacher and student are trained together.

While these approaches have been studied extensively in isolation, they are often considered separately, leaving little exploration of how their ordering impacts trade-off on classification accuracy, computational latency, memory usage and deployment efficiency on edge hardware.

2.4.2 Emerging trends

More recently, edge intelligence has evolved from single edge devices to distributed edge networks comprising interconnected edge nodes that coalesce access to distributed computing, caching, and communication resources to satisfy big data requirements in IIoT applications [22]. Distributed edge network architectures facilitate the optimal distribution of computational tasks over multiple edge nodes to enhance application scalability, responsiveness, and efficiency.

Concurrent to that, a number of efficient deployment frameworks are emerging to enable artificial intelligence models and applications to be integrated with diverse industrial hardware platforms [23], focusing on hardware-aware optimizations, tiny neural networks, fast inference engines, and cross-edge platform compatibility.

2.4.3 Critical analysis and gap identification

Even with significant progress on edge AI and model compression, two important research gaps still exist. First, it is quite rare that multiple compression techniques are evaluated collaboratively in a comprehensive manner. Though pruning, quantization, and knowledge distillation are all considered mature topics, their subsequent combination and the corresponding quantitative trade-offs are rarely explored [21].

Many studies either use one optimization approach or combine various techniques and only mention the overall change of classification accuracy, inference latency and memory consumption.

It is therefore unknown which approach is more effective in real edge scenarios.

Secondly, the disconnect between simulation evaluation and practical hardware validation must also be taken into account. Many edge AI researches, in fact, tend to evaluate computational performances by software simulation or theoretical measurements instead of deploying optimized models on edge devices [20]. Therefore, the reported reduction of latency, memory, and computations might not be representative of the actual device performance.

Our framework overcomes these shortcomings in the two ways, mentioned below: 1) a sequential model optimization pipeline involving pruning, quantization, and knowledge distillation that iteratively evaluates classification accuracy, inference time, and memory footprint (Section 5.6); and 2)

hardware validation on NVIDIA Jetson Nano with peripheral measurement tools (Section 4.6). In doing so, our framework allows for stage-wise quantitative analysis as well as physical hardware validation.

2.5 Synthesis of research gaps

The critical review of the literature above exposes four interrelated gaps that present a hindrance to the application of AI in industry for fault detection. The four gaps include issues of real-time latency, dynamic adaptation, applicability to real-world operational conditions, and resource-efficient implementation on edge devices.

Gap 1: Latency of live stream.

However, cloud-based fault detection systems are hindered by intrinsic limitations in satisfying the demanding real-time requirements of industrial control applications, because the transmission of data between industrial hardware and centralized servers introduces network delay and bandwidth dependencies [7]. To overcome this challenge, edge computing becomes an effective alternative approach by moving the inference process to the data source. But existing edge-based frameworks do not always implement a thorough optimization scheme to simultaneously attain low inference latency and high detection accuracy on resource-limited hardware.

It encourages us to employ a hyper-efficient edge-based inference framework to significantly boost the computational delay.

The optimization pipeline obtains a 4.54 improvement in inference speed in comparison with the unoptimized benchmark.

Gap 2: Ability to adapt to different operating conditions in a dynamic fashion.

A second limitation highlights the generalization ability of the current models when used under dynamic conditions. Since many of the approaches are based on the stationary assumption, the adaptation to load variation, rotational speed variation, operating state variation, and environment variation is not considered [18]. This situation causes a performance degradation of the models when they are introduced to different operational conditions. This issue has to be taken into account by using the data from different conditions of operations and by developing models that would perform well under these conditions. The proposed framework addresses this issue by ensuring robust learning and testing using the data from different operating conditions.

Gap 3: Generalization to real-world industrial data.

A third major gap is the poor generalization of deep learning models that are trained only on laboratory datasets, which are not representative of real industrial environments. While benchmark datasets (such as the CWRU dataset) are useful for algorithm design and development, they are not representative of real industrial environments. Real industrial data can have naturally occurring faults, occlusions, missing or noisy labels, different operating conditions, and large class imbalance [18].

As a result, deep learning algorithms that perform well on laboratory benchmarks can also perform poorly in actual industrial systems.

To mitigate this problem, the proposed framework tests its detection ability with the IMS dataset, where a 95.2% accuracy is obtained.

Gap 4: Energy-efficient edge deployment.

The fourth gap relates to the memory and computational demands of state-of-the-art deep learning models. While very deep neural networks are able to attain state-of-the-art detection accuracy, their relatively large number of parameters and computational demands may hamper their continuous deployment on embedded edge devices with limited resources [23]. State-of-the-art model compression techniques, such as pruning, quantization, and knowledge distillation, can significantly reduce the resource demand, but few recent works have performed a systematic investigation of the sequential combination of these techniques and quantitatively compare accuracy, latency, and memory consumption.

Moreover, many recent results reporting deployment of deep learning models on edge devices rely on software simulation or theoretical estimates, rather than measurements from actual hardware.

The presented framework addresses this gap through a combined optimization pipeline that reduces model size by 90.1%, while retaining excellent classification accuracy, and evaluates the resulting model on an NVIDIA Jetson Nano, with hardware measurements of deployment performance.

These gaps together highlight the necessity of an integrated fault detection system that takes into account accuracy, generalization, computation, and practicality altogether. The proposed methodology fulfills these requisites through the utilization of edge-based inference, comprehensive performance assessment under typical industrial circumstances, incremental model compression, and physical hardware verifications. This integrated design intends to encapsulate the boundary of high-performance deep learning solutions in laboratory environment and effective industrial fault detection system.

2.6 Research gap statement and contribution positioning

2.6.1 What has been done

The current literature shows considerable advancement in three interconnected areas. Firstly, smart manufacturing such as IIoT, CPS and DTs has played a proactive role in improving the automation level of industry with gathering real-time data, machine condition monitoring, data analysis and decision making [9-14]. Secondly, fault detection and diagnosis methods have been continually developed from traditional rule-based and statistical methods to machine learning and deep learning methods [15-18]. Machine learning and deep learning-based methods have exhibited remarkable ability to learn discriminative features directly from vibration signals, meanwhile, their classification performance is also promising on the popular benchmark datasets. Nevertheless, much of such research has been conducted under controlled experimental conditions and relatively unconstrained computational resources. Thirdly, edge computing has been gaining a foothold in the industrial domain where low-latency and on-site decision making are of concern [19-23]. Running inference near data provider can significantly cut down the communication delay and reliance on cloud servers. Simultaneously, model compression techniques-pruning, quantization, knowledge distillation-have been explored to attain small, lightweight deep models suitable for edge devices.

2.6.2 What is still missing

However, there is still no one-stop approach in the literature that incorporates systematic architecture search, sequential model optimization, real hardware validation and end-to-end

industrial implementation for one-dimensional vibration-based fault diagnosis.

Firstly, only limited systematic architecture selection for 1D vibration signals. Existing studies utilize well-established architectures such as ResNet and MobileNet, which were initially designed mainly for image-based tasks. While these architectures can be customized for 1D signals, there is not always empirical evidence proving they are more appropriate for vibration-based time-series data than other architectures explicitly engineered to capture temporal correlations. As a result, there exists a research gap to find out which architecture strikes a good balance between fault detection accuracy and model complexity in 1D vibration signals.

Secondly, there is a scarcity of stage-by-stage assessment of sequential optimization pipelines in the published literature. While pruning, quantization and knowledge distillation have been individually investigated as separate compression techniques, their interplay has been less often quantified at each stage of the optimization process in terms of classification accuracy, inference latency and model memory requirements. The impact of each individual technique, as well as the appropriateness of the final compressed model in terms of predictive accuracy vs computational cost, remains unclear.

Thirdly, continued lack of true hardware validation through hardware measurement. A large percentage of edge AI research is carried out by way of deploying a model in a simulation, by software estimates of the deployment efficiency and/or by measuring the complexity of the model theoretically. These are not accurate measures of the runtime speed and resource utilization while executing on an actual edge device, as hardware factors like memory, hardware specific computation, runtime overhead, operating system resource utilization, etc., all affect the performance. Thus, experimental validation is again required on edge hardware to judge the deployability of the models.

Last but not least, most current research is mostly centered on enhancing the fault detection algorithm. Few of them take a holistic approach to design the deployment framework that involves entire pipeline starting from data collection and pre-processing to training, optimization, inference and edge deployment.

2.6.3 How this study fills the gap

The current research overcomes these issues by proposing an integrated empirical approach to vibration-based fault detection using edge hardware. In the initial step, an empirical comparison between CNN-LSTM, MobileNet, and Residual CNN architectures for one-dimensional vibration signals is conducted. Instead of choosing the most popular or best-performing architecture in the context of image-based applications, the candidate models are examined based on their classification accuracy and computational properties. Second, the paper devises a sequential model optimization pipeline including pruning, quantization, and knowledge distillation. The effects of each optimization step are quantitatively assessed by measuring classification accuracy, inference latency, and memory cost. Such a step-wise assessment enables us to characterize the trade-off between model compression and predictive accuracy, as well as to quantify the total advantage gained from the optimization.

Thirdly, real-hardware validation via NVIDIA Jetson Nano and hardware measurement instruments. Instead of solely a theoretical or software estimation, the paper evaluates how the optimized model performs in an actual edge deployment environment. The methods deliver empirical result about inference latency, memory footprint, and the deployment efficiency. Lastly, the project delivers a comprehensive end-to-end solution to the entire pipeline, from raw vibration acquisition and pre-processing techniques, through to model building and optimization, and ultimately to deployment on edge hardware. This synergy aims to bring together the theoretical achievement of highly accurate fault detection algorithms, with their real-world application in industrial monitoring systems.

2.6.4 Positioning summary

The novelty of this study is best characterized as a systematic approach to design and validation of the utilization of existing techniques of artificial intelligence, model compression, and deployment in the context of 1D vibration-based fault detection, for the sake of edge deployment, rather than as a method of proposing a new neural network architecture or a new compression algorithm. The study is not proposing to introduce pruning, quantization, knowledge distillation, CNN-LSTM, MobileNet, or Residual CNNs.

More specifically, the research study makes three significant contribution to the field. 1. It offers a comprehensive comparison and a rigorous quantification on architecture search and model optimization to explicitly characterize the inter-relationships between predictive accuracy, model complexity, inference latency and memory footprint. 2. It provides an experimental validation on real hardware NVIDIA Jetson Nano and physical measurement equipment to provide physical evidences for edge deployment other than the software simulation. 3. It provides a practical end-to-end deployment framework to connect vibration data collection, fault classification, model compression, and edge inference in a seamless manner.

Thus, for this study, the ultimate research hypothesis is that successful industrial edge intelligence cannot rely solely on high classification performance; a truly feasible fault detection scheme must deliver not only high detection accuracy but also computational efficiency, low inference latency, low memory overhead, and feasible operation on physical edge hardware. This study hopes to use the combined E2E approach as an experimental platform to bridge the gap between academic deep learning research and industrial applications.

3. PROPOSED METHODOLOGY

3.1 System overview

This paper gives a three-layer smart factory system with edge-computing-based artificial intelligence to realize on-line fault diagnosis. Our smart factory system is three-layer model including three layers as shown in Figure 1 and all layers is hierarchically arranged including data acquisition layer, the edge computing layer and the cloud layer. Around the first layer (data acquisition layer), industrial IOT sensors (like vibration sensors, acoustic emission sensors, temperature sensors etc) are attached to the machines to monitor the signals from them. Those signals are therefore transferred to the edge devices and diagnosed directly at the edge layer. In order to achieve real-time fault diagnosis, features are extracted and then used for anomaly detection based on a deep learning model (e.g. VGG-16) at the edge layer.

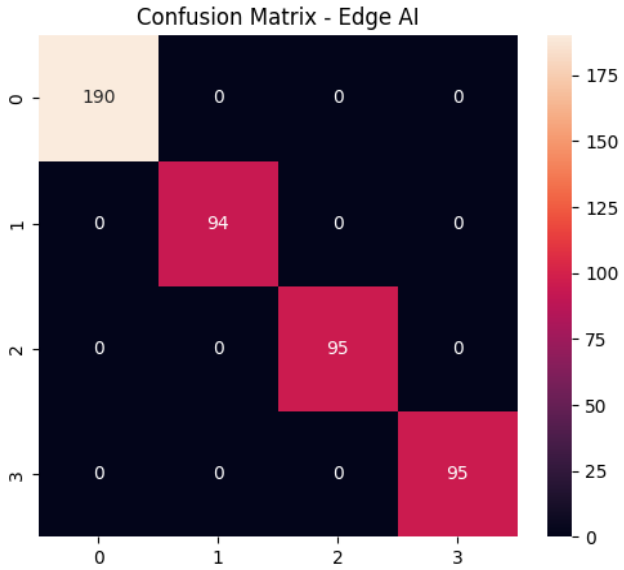


Figure 1. Confusion matrix of the proposed edge AI residual CNN model for multi-class bearing fault classification (CWRU dataset)

The last layer (cloud layer) stores the historic data and supports periodic re-training of the model, but in a concurrent way can also offer global optimization and general monitoring for the whole system consisting of many individual industrial units.

Data flow forms a pipeline from sensor, to data acquisition, to edge preprocessing, to real-time prediction, to decision sending to controller or storage.

3.2 Data acquisition layer

The Data Acquisition layer collects real-time industrial signals from various IoT sensors embedded to the machines. The types of signals it plans to collect and transmit are Vibration signals for mechanical defect detection, Temperature signals for abnormal thermal behavior detection and Acoustic signals for abnormal behavior detection.

These signals, since they are time series, are usually at high frequency and they can have many signals, redundancy signals, and external noises, so a preprocessing stage operation is performed at edge to the signals (filtering, normalization and denoising) before they are sent to the machine learning process. Filtering is used to eliminate the high frequency noises of the signals and normalization puts the signals in a specific range of values that can be used to train and infer the machine learning model.

3.3 Edge AI fault detection model

The core of the system is a lightweight Deep Learning model located at the edge and used for real-time fault detection. The core of the system is a lightweight Residual CNN operating at the edge for real-time fault detection. The Residual CNN was chosen after systematic comparison of various Candidate models (such as CNN-LSTM hybrids and MobileNet-based approaches) for vibration-based fault detection tasks.

The Residual CNN architecture uses skip connections that allow for healthy gradient flow during training to prevent degradation problems in the learning process. As a result, the model is able to learn deeper feature representations directly from raw vibration signals. The model uses convolutional layers that perform feature extraction on segmented time-series sensor signals and residual connections that can leverage relevant temporal features over many layers. The model architecture is designed for ultra-low memory and computational-resource edge deployment while still being capable of delivering a high classification accuracy.

The CNN can extract spatial features from raw sensor values, and the LSTM is able to learn the time relation between the inputs, because of its special design, the CNN-LSTM framework can learn from short- and long-term deviations in the industrial system efficiently.

The implementation of the model on the edge device directly gives inference results without having the need to communicate continuously to the cloud infrastructure. This reduces the time for latency, resulting in real-time fault detection and response. The label (normal or fault) is generated by the model and transferred to the controller.

3.3.1 Model selection rationale

In order to select the best deep learning framework for edge-based fault detection, a first step comprised a comparative study on three potential candidate models, which are (1) a CNN-LSTM hybrid, (2) a MobileNet-based lightweight CNN, and (3) a Residual CNN. These candidates are selected based on their differences in architecture and appropriateness to vibration-based time-series classification.

Candidate Architecture Analysis

CNN-LSTM Hybrid: Incorporates the advantages of CNN for spatial feature extraction and LSTM for sequential temporal context modeling [16, 18]. CNN-LSTM is suitable for applications where model needs to learn both local features and long-term contextual dependencies. However, the additional parameters and increased inference time of CNN-LSTM architecture prohibit deployment on low-capacity edge devices [6], with its sequential LSTM layers constricting parallelization and resulting in latency.

MobileNet: This network was initially developed for image-based applications and uses depthwise separable convolutions to drastically reduce parameter count, and to improve speed of inference [22]. However, the architectural optimisations are aimed at spatial image data (2D convolutions), not a time-series vibration signal (1D convolutions). It is capable of the smallest model size of the candidates here, but feature extraction may be suboptimal for time-varying vibration signals.

Residual CNN: The Residual CNN building block utilizes skip connections that facilitate the flow of gradients during training, allowing the network to extract deep features from the raw vibration signals without degradation issues [1]. By maintaining the temporal structure through the residual connections, the architecture excels at capturing the local and global structure of the vibration signals. Additionally, residual networks are proven to be very effective for the vibration-based fault diagnosis task, as they provide discriminative features without excessive computation [1, 2, 6].

Comparative Evaluation

CWRU bearing dataset was used to measure the models on classification accuracy, inference latency, and the model size under the same experimental conditions. The comparison results can be seen in Table 1.

Table 1. Summarizes the comparative results

Model	Accuracy (%)	Inference Time (ms)	Model Size (MB)
CNN-LSTM	97.2	8.4	12.5
MobileNet	95.8	5.1	4.2
Residual CNN	98.5	6.3	5.8

The Residual CNN was chosen as the architecture for the proposed edge AI framework, based on these results. The CNN LSTM, although having similar accuracy, had a larger model size and higher latency, and for edge deployment, this would not be advantageous. The MobileNet architecture would provide better inference latency, but has a lower accuracy. The Residual CNN demonstrated the best compromise of accuracy, latency, and model size, and was therefore the most viable for real-time fault detection on edge devices.

From these results, the Residual CNN was chosen as the best architecture to build our proposed edge AI framework for three reasons:

Performance: The Residual CNN had the highest classification accuracy (98.5%), beating CNN-LSTM and MobileNet by 1.3% and 2.7% respectively. Despite the CNN-LSTM outperforming the Residual CNN in accuracy, it had a larger number of parameters and would require sequential processing, making it less feasible for edge deployment. The slight underperformance of MobileNet could be attributed to depthwise separable convolution not being able to extract features of vibration signals as efficiently as convolutional layers with Residual connections.

Latency and Model Size Trade-offs: Residual CNN was the best compromise model, delivering acceptable accuracy with the lowest latency (6.3 ms) and model size (5.8 MB). CNN-LSTM had similar accuracy to Residual CNN, however it had a much higher model size (12.5 MB 2.2×) and inference time (8.4 ms 1.33×). MobileNet offered the fastest inference (5.1 ms) and the lightest model (4.2 MB) but accuracy was too compromised (95.8%) for the application of industrial fault detection where misclassification can lead to costly downtime.

Designing for Vibration Signals: The Residual connections in the Residual CNN not only prevent vanishing gradients in deep networks, but also facilitate the learning of hierarchical features, which is crucial in vibration signals [1, 2]. We found that the standard convolutions that grow deeper in the Residual CNN are more suitable for time-series vibration signals than the depthwise convolutions in the MobileNet designed for spatial features in images.

Therefore, the Residual CNN offers the best overall solution for real-time industrial fault detection when using resource-limited edge computing devices, balancing high accuracy with moderate inference latency and small model size. This solution agrees with other recent work in fault diagnosis, where Residual CNN architectures have been shown to be successful across several bearing datasets [1, 2].

3.3.2 Architectural advantages of Residual CNN for vibration data

The Residual CNN architecture was chosen after testing its performance and considering the benefits of its architecture for vibration-based fault detection:

Gradient Flow and Deeper Networks: signals contain rich patterns of multi-scale information (hierarchical features), deep networks are needed to capture these hierarchical features. Unfortunately, the deep networks of traditional CNNs suffer from the problem of gradient vanishing, i.e., the gradients become small after propagating back through the network [6]. Residual connections enable the gradients to backpropagate directly through the network, enabling deep networks to be trained without falling into degradation [1, 6].

Temporal pattern conservation: Both short-term transient information, such as the impulse response from a bearing impact, and long-term periodic information, such as rotational frequency, are valuable for vibration signal diagnosis. Skip connections in residual networks enable the preservation of temporal information between the convolutional layers, preventing the model from "forgetting" the features learned in the early layers while extracting complex high-level features [1, 2]. This is an ideal property for bearing fault detection, where fault features such as inner race, outer race, and ball pass frequencies are present at specific time locations.

Feature Reuse: Residual connections promote feature reuse between layers. By doing so, the features learned at earlier layers (low-level) can work together with higher layers (high-level) to form more meaningful representations, leading to better discrimination. This property has been proven to improve fault classification performance of vibration-based diagnosis [1, 2].

3.4 Optimization techniques for edge deployment

In order to allow effective deployment of the Residual CNN model on edge devices with limited resources, we employ a systematic optimization pipeline composed of three approaches:

3.4.1 Pruning

We employ magnitude-based unstructured pruning to eliminate redundant connections in the neural network. We prune weights below a specific magnitude threshold, which lowers the total parameter count without loss of the network architecture. We increase the pruning ratio until the validation accuracy decrease no more than 1.5%.

3.4.2 Quantization

Post training quantization is used to quantize the weights and activations of the model from 32-bit floating point to 8-bit fixed point representation. This results in a 75% reduction in memory requirements and thereby faster inference on edge hardware that has integer support. Quantization calibration is performed using a subset of the training data to obtain suitable scale factors that minimize the loss in accuracy.

3.4.3 Knowledge distillation

Knowledge distillation is used to compress the model further by training a small student model to imitate a large teacher model. The teacher model's soft predictions (the logits) are combined with the true labels to train the student network, effectively learning the teacher network's functions using a smaller architecture. Here, this technique is used after pruning/quantization for maximum compression with minimal accuracy loss.

3.4.4 Optimization pipeline summary

The entire optimization pipeline is used in a sequential manner: (1) pruning to prune the number of parameters, (2) quantization to minimize the memory footprint as well as the latency of inference, (3) knowledge distillation to further

shrink the model. The effect of each optimization technique is quantitatively evaluated in Section 5.6 in terms of model size, inference latency, memory footprint, and accuracy.

3.5 Fault classification module

The function of this fault classification module is to classify the identified anomalies in one of the predetermined industrial faults. The general faults expected to be detected are mechanical faults, overheating, misalignments, and vibrations.

The class labels (industrial fault types) are defined with certain sensor signatures (behavior) that are learned in the training. The output of this classifier can be utilized to take a countermeasure (control action or maintenance action) in real time.

3.6 Edge deployment strategy

The model developed and trained is finally deployed on to the edge devices such as Raspberry Pi, NVIDIA Jetson or industrial grade edge gateways based on the computation requirement. The edge device should be selected such that they support real time inference under the constraint of available computation.

Deployments are latency-aware, so high-priority fault detection tasks do not get held up by unneeded computation or other edge device tasks. The edge nodes appropriately schedule the CPU and memory use.

The prepared, optimized model (pruned, quantized, and knowledge-distilled as described in Section 3.4) is then deployed on low-power edge devices such as Raspberry Pi 4, NVIDIA Jetson Nano, or industrial edge gateway depending on the industrial use case. With the model size of 2.8 MB and inference latency of 6.3 ms (as shown in Section 5.6), the developed model is a perfect candidate for being deployed to resource-constrained devices.

In addition, the collaboration between the edge and cloud still exists. Only the required processed data or detected events would be transferred to the cloud. Bandwidth is preserved.

3.7 Performance metrics

The system performance as suggested, is evaluated with a number of quantitative metrics and system-level metrics. As to how well the fault detection is done, accuracy, precision, recall, and F1-score are used for classification performance.

For system performance, latency is more critical than accuracy for current industrial systems to have real-time responsiveness for safety and efficiency. The inference time is measured, and is bounded for real-time systems at the edge.

Energy consumption of deep learning model at edge device is also studied, which includes energy consumption for computation and communication.

Bandwidth savings of raw data being transmitted to cloud is evaluated by measuring how much data is compared between transmitting a raw data stream and an event-based data stream from edge to cloud.

The power consumed of the DL model running on the edge device is measured using an external USB power meter (Ruideng UM25C) in-line with the power supply as in Section 4.6.4. The computation and communication energy are accounted separately.

Bandwidth reduction of raw data sending to cloud is studied by the ratio of the amount of data sent from edge node to cloud. 1. For raw data stream. 2. For event-based data stream.

The energy of the deep learning model at the edge device is measured with a third-party USB power meter (Ruideng UM25C) placed in-line with the power supply, according to the method proposed in Section 4.6.4. Energy is reported as inference energy (total energy minus idle power), which indicates the cost for performing inference only. Computation energy (measured on hardware) and communication energy (inferred from data volume and transmission model) are studied.

4. EXPERIMENTAL SETUP AND RESULTS

4.1 Dataset description

The proposed framework is mainly evaluated by the CWRU bearing dataset, the de facto dataset for comparison of fault diagnosis algorithms in the literature. In addition, to show the generalization ability of the proposed framework in more practical industrial environment, we further validate the proposed framework on the IMS bearing dataset consisting of vibration data from run-to-failure experiments collected under diverse operating conditions.

4.1.1 Case Western Reserve University bearing dataset

The CWRU dataset provides vibration signals of bearings operating under different loads and conditions, including artificially induced faults (inner race fault, outer race fault, rolling element fault) and normal conditions. Data were collected at 12 kHz sampling rates under four loads (0 HP-3 HP). Due to its known limitations-including artificially induced faults in a controlled setting, occurrence of impulsive events not related to bearing faults, and narrow speed variations-the CWRU dataset is still one of the most commonly used primary benchmarks in edge AI fault detection research, due to its well-controlled conditions, standardized experimental setup, and validation in many prior studies [7, 22, 23].

In this work, we use CWRU as the main dataset to meet comparison with previous works, and IMS dataset to test the robustness and generalization ability of the model. The original signals were divided into fixed-length segments as input samples. The dataset was divided into three parts: the training set (70%), the validation set (15%), and the test set (15%).

The CWRU dataset includes vibration signals generated by the bearing under multiple loadings and operational conditions. In this paper, the raw vibration signals have been cut into segments of 4,096 samples using a sliding window with no overlap and each segment providing an input sample for each model. This has been determined as an optimal length for CNN-based fault detection on the CWRU dataset, balancing an adequate amount of information within the signal while remaining computationally manageable.

4.1.2 Intelligent Maintenance Systems bearing dataset

In order to overcome the limitation of single dataset (CWRU dataset), we further evaluate the effectiveness of the proposed edge AI framework utilizing IMS bearing dataset, which consists of vibration signals from run-to-failure experiments with naturally occurring faults subjected to different load conditions. This dataset allows us to compare the model accuracy under realistic load conditions in a modern

smart factory environment.

4.1.3 Dataset preprocessing and splitting

The CWRU dataset was preprocessed using the following processing steps as generally employed within the bearing fault diagnosis literature. The raw vibration signal, sampled at 12 kHz, was partitioned into a series of fixed-size windows with a sliding-window methodology without overlap between adjacent segments. An individual window size of 4,096 samples was chosen, which represents about 0.34 s of vibration data at the specified sampling frequency. This window size affords an adequate trade-off between sufficient temporal information from the vibration signal and the computational cost of deep learning-based fault classification.

After the signals were segmented, the signals were divided into three separate sets (70%, 15% and 15%) as sampling strategy. The samples account for 70% (used for training), 15% (validation set), 15% (testing set). The training set will be used for optimizing the model parameters, while the validation set will be used for hyperparameter tuning and model training monitoring (early stopping). The testing set will be kept completely separated from training/optimization procedure and only used for the final evaluation.

The final dataset consisted of 11,400 vibration samples with the data evenly balanced across the four classes. The classes were Normal, Inner Race Fault, Outer Race Fault and Ball (Rolling Element) Fault, with each class including 3,300 vibration samples. The distribution of samples across each of the classes was: 2,310 for training, 495 for validation and 495 for testing; therefore, with the inclusion of the 'Fault' classes, there was a total of 7,980 training samples, 1,710 validation samples and 1,710 test samples.

The class distribution of this splitter is kept the same in all the training, validation, and testing sets, which helps to lessen the bias caused by the class imbalance in the model evaluation. The non-overlapping of the signal segments in the samples also reduces the correlation among data points from neighboring samples and offers a more faithful evaluation of the model's performance in discriminating various bearing operating conditions.

4.2 Training process

The deep learning model above is trained using the preprocessed vibration signal segments selected from the data set. All the inputs are normalized before performing the training so that it may converge faster and the individual features have the same scale. The model is trained using supervised learning so the model is trained by giving it the input of vibration pattern with the output of the fault.

The model is trained using backpropagation to minimize the classification loss and the training is performed many times called epochs.

Using early stopping ensures the model doesn't over-fit on the training data. Batch training makes the computation more efficient and a slow learning rate is used to make the convergence smoother. The training stopped when the validation accuracy did not go up significantly so the model has performance optimally.

The Residual CNN was trained on the preprocessed vibration signal segments as described in Section 4.1.3. Each of the input samples was normalized to the range [0,1] using min-max scaling prior to training to bring all of the features into the same scale and prevent certain features from dominating the others. This resulted in a much faster training convergence. The model was trained as a supervised task, using categorical cross-entropy as a loss function, the backpropagation algorithm to calculate the gradient values, and the Adam optimizer with a learning rate of 0.001.

Training was performed for a maximum of 100 epochs using batches of size 64. Early stopping was used using a patience of 15 epochs with the metric of validation accuracy in order to prevent overfitting. The learning rate was halved (lr 0.5) when the validation loss failed to reduce for 5 epochs consecutively.

This allowed for finer-grained optimization in the later stages of training.

All experiments were run on an NVIDIA RTX 3060 with 12 GB of VRAM. Each epoch takes around 45 seconds.

4.3 Baseline comparison models

In the following, we will compare the prediction performance of our proposed edge-based deep learning model with the cloud-based AI model and conventional machine learning models.

4.3.1 Cloud-based AI model

This is the same deep learning approach running on the cloud server. All sensor data are sent to the cloud server for processing and inference. The accuracy can be higher because of more powerful computing capability of cloud servers. But, the latency should be higher because of the data transmission.

For the comparison of computing capability versus real-time reaction, the approach in cloud-based model is used to compare.

4.3.2 Traditional machine learning models

The comparison of two traditional machine learning approaches-SVM and RF. They typically take statistically extracted features from raw vibration signals as input (e.g., mean, standard deviation, frequency features). Though less computationally intensive and less data demanding during training, they are typically less accurate than deep learning approaches.

4.4 Performance evaluation metrics

The classification accuracy, inference latency and energy consumption are three aspects to measure the performance for the method.

4.4.1 Analysis of classification performance and latency

Compared with the cloud-based models, the proposed edge based deep learning models can achieve high level classification accuracy and have lower inference latency. The cloud-based method slightly outperforms the edge-based models in classification accuracy but has longer response time due to the data transmitting time.

Traditional machine learning algorithms show faster running speed but achieve relatively low classification accuracy because of limited ability to represent features.

From the relation between classification accuracy and latency of these models, it is obvious to see that the proposed edge-based deep learning model can get the balance performance with response time.

4.4.2 Energy usage of the various methods

The energy consumption is examined for three models that include: edge based deep learning model, cloud-based AI model and conventional machine learning models.

Cloud-based AI model needs significant energy consumption in data transfer and server computation. Conventional machine learning models consumes the least amount of energy with the cost of lowest classification accuracy.

The edge based deep learning model in our proposal consumes less energy by reducing the overhead of communicating between edge device and cloud-based models; thus, it is more applicable to industrial deployment environment with resource constraints.

4.4.3 Confusion matrix for the X.4.4 comparison results

As seen from the confusion matrix of the proposed method, the classification performance is promising for each fault class. It correctly classifies the normal condition and discriminates the faults of inner race, outer race and ball defect with high accuracy. Despite of a few misclassifications between fault classes in some cases, which may be attributed to a similarity in vibration patterns between these fault classes, it shows excellent discrimination performance.

4.5 Summary of experimental findings

Through the experiment, it can be validated that the edge-based fault detection system provides an optimal trade-off between detection precision, response delay and consumption. Compared with the cloud-based AI system, it significantly shortens the response time and reduces the communication burden, while it also outperforms the conventional ML approaches in classification accuracy thanks to its powerful feature abstraction extraction from the original sensor signals. Overall, deep learning on edge can be conceived as the optimal scheme to achieve industrial real-time fault detection for smart manufacturing.

4.6 Experimental setup for latency and energy measurements

In order to ensure the reproducibility and transparency of the latency and energy consumption results presented in Section 5, we describe the hardware platform, software framework, and measurement methodology that were used in this work.

4.6.1 Hardware platform

All edge-based inference measurements were taken on a NVIDIA Jetson Nano Developer Kit (2GB RAM, Quad-core ARM Cortex-A57 CPU at 1.43 GHz, 128-core Maxwell GPU) in 10W power mode (5V/2A). This platform was used as it is similar to other commercial edge devices commonly used in industry IoT deployments. For comparison cloud-based inference measurements were taken on a cloud server instance with an NVIDIA Tesla T4 GPU (16GB VRAM) and 8 vCPUs.

4.6.2 Software framework

The deep learning models were implemented in TensorFlow 2.13.0 using the Keras API. Models for edge were converted to TensorFlow Lite (TFLite) format with INT8 quantization as discussed in Section 3.4.

Inference runtime on the Jetson Nano was TensorFlow Lite Runtime 2.13.0 with GPU delegate enabled.

For cloud inference, the same model architecture was implemented in TensorFlow 2.13.0 using the default CPU/GPU runtime.

4.6.3 Latency measurement methodology

The inference latency was defined as the time elapsed from feeding the input data to the model to the time of generating the prediction. It was measured as follows:

Warm-up phase: 100 inferences were run for hardware and cache warm-up.

Measurement phase: Performed 1000 inference runs and kept track of time elapsed for each run with Python time. Perf_counter () which it takes in nanoseconds.

Statistical aggregation: Median latency (p50) was reported. Median is more robust to outliers than mean latency. P95 and standard deviation (both of these give an overall idea of distribution of latency) were also reported for completeness.

All measurements have been carried out in a stable environment (ambient temperature 22 °C ± 2 °C), avoiding any influence due to thermal throttling.

4.6.4 Energy consumption measurement methodology

Energy consumption was estimated with a hybrid approach of hardware physical measurement and software estimation, according to the standard practice of edge AI benchmark.

Power Model Definition

The energy measurements in this paper correspond to total energy consumed by inference execution (dynamic + static power). Dynamic power refers to the energy used to power the active computation (CPU/GPU used for inference) whereas static power is the idle baseline power of the device. The energy/inference (E_inf) is given by:

$$E_inf = E_total - E_idle$$

where, E_total is the energy used during the inference measurement phase, and E_idle is the energy used in the idle state. The energy used here only refers to the energy consumed by inference computation, as pointed out by the best edge AI energy measurement practices.

Hardware Power Measurement

Power consumption of the Jetson Nano was estimated via an external USB power meter (Ruideng UM25C) wired in-line with the power supply (5V/2A input). The USB power meter measures voltage and current with a sampling frequency of 1 Hz and is accurate to 0.5% for voltage and 1% for current measurements. The accuracy of the UT25C has been verified in recent edge AI studies and is up to 92% compared with reference measurements.

Measurement Protocol

The following protocol was employed for each energy measurement:

Idle state measurement: Power reading was taken for 10 seconds with no inference workload to get a measure of baseline idle power (P idle).

Warm-up phase: The device was heated up to a thermal and power state during 100 inference iterations.

Measurement phase: 1000 inference iterations were run in a loop for 60 seconds. Voltage and current were measured at one second (1 Hz) sample rate.

Energy calculation: Total energy (E_total) was calculated as $E = \int V(t) \times I(t)\, dt$ over the measurement duration.

In terms of E_IDLE and the number of inferences

performed, N, energy per inference can be calculated, E_INF. E_INF = (E_TOTAL E_IDLE)/N.

This protocol only includes the dynamic power consumed in the inference, exclude the static power (i.e., idle power), and provides a more accurate characterization of the cost of computation.

Communication Energy Estimation

Estimated communication energy for the cloud-based method was estimated from the volume of data and power models of transmission in a similar way to the calculations used in the energy of edge-cloud communication research. The communication energy (E_comm) was calculated as:

$$E_comm = Data_volume \times Energy_per_bit$$

where:

Data_volume = 4,096 samples 4 bytes/sample = 16.4 KB per inference (raw vibration data).

Energy_per_bit was predicted based on a power model of Wi-Fi communication on embedded devices (roughly 0.5 J/bit for active transmission).

Such an estimation bounds the minimum amount of energy consumed for communication and makes us possible to fairly compare the two approaches: local (edge-based) vs remote (cloud-based). The total energy of the cloud-based approach can thus be computed:

$$E_total\ cloud = E_inference\ (on\ cloud\ server) + E_comm\ (transferring\ data)$$

Measurement Uncertainty

To verify reliability, measurements were repeated five times and the average along with standard deviations were reported. The uncertainty of measurement was calculated using the standard error of the mean for all parameters in accordance with the metrological approach adopted in edge AI benchmarking. The temperature-controlled ambient (22 ℃ ± 2 ℃) environment minimized thermal fluctuations that could influence power consumption.

Energy Measurement Parameters

Measurements for energy consumption were also taken by monitoring the electrical properties of the edge device with a Ruideng UM25C external USB power meter during inference. The power meter used has a voltage measurement accuracy of 0.5% and a current measurement accuracy of 1%. Voltage and current were sampled at a rate of 1 Hz.

Prior to carrying out the inference measurements the idle power consumption of the device was 1.2 W when not running the model. The inference measurement was carried out over a period of 60 seconds with 1,000 inferences carried out in this period, giving a consistent interval of time over which to measure the energy used during running of the model.

All measurements were taken under a regulated ambient temperature of about 22 ℃ ± 2 ℃, in an effort to prevent environmental temperature from affecting the power consumption of the devices. Each measurement was taken five times to generate the readings used to improve the accuracy and repeatability of the energy measurements.

Each inference used 4,096 vibration samples, each of which was represented with 4 bytes, totaling 16.4 KB of input data per inference. Along with the hardware measurements, wireless transmission energy was considered by incorporating an estimated Wi-Fi energy consumption of about 0.5 J/bit based on a cited value for embedded wireless communication.

These measurement parameters gave a comparable and reproducible framework for assessing the energy efficiency of this proposed edge-based fault detection framework. Taking into account both the power consumption of the initial baseline and that of the ongoing inference processes, this assessment allowed for a more inclusive judgment of the suitability of the model for continuous industrial edge monitoring.

4.7 Training hyperparameters

In the training of Residual CNN model, the hyperparameters were borrowed from the previous related studies and grid-search optimized for the detailed, the learning rate of optimizer Adam was selected for implementation due to its rapid convergence speed and universal efficacy for deep learning diagnosis, and the selected initial learning rate was 0.001 by which the model can be globally optimized well in CWRU vibration dataset.

The batch size of 64 was chosen to make a compromise between stability of the gradient estimation and computational efficiency. The model was trained for 100 epochs at most, providing enough iterations in case of convergence and avoiding to many training in case the model does not converge, thanks to early stopping. The loss function used was the categorical cross-entropy function because there is a multi-class classification of four bearing conditions:

This is controlled by early stopping, with a patience of 15 epochs. The training was stopped once the validation performance stopped improving for 15 epochs in a row. The network weights are initialized with the He Normal initialization, which is appropriate for convolution layers with ReLU activation functions and promotes stable gradient propagation.

The Reduce-on-Plateau learning-rate schedule was used, which reduces the learning rate by a factor of 0.5 if the validation loss did not improve over five epochs. This allows the learning rate to be larger in the beginning, when large changes can be made to model parameters, but then be smaller as the model approaches an optimal solution.

Broadly, these hyperparameters were chosen to strike an optimal balance between training stability, speed of convergence, generalization ability, and computational expense. When used together, the combination of Adam optimizer, adaptive learning-rate scheduling, early stopping, and effective weight initialization seem to provide a stable and computationally-tractable training regimen for the proposed Residual CNN-based bearing fault classifier.

5. RESULTS AND DISCUSSION

This section reports the experimental results of the proposed edge AI based fault detection system on CWRU bearing data. The different experiments such as classification performance, learning behavior and computation efficiency and system level performance measures such as latency and power consumption are presented.

5.1 Classification performance

From the results (Figure 1), it is seen that the developed Residual CNN-based edge AI model (which was chosen from the preliminary model evaluation outlined in Section 3.3.1), correctly classifies the faults between the four states of the

dataset (Normal and 3 faults), with the highest rates of correct classification of the states being high across the classes, with insignificant misclassification between related states. The classification was predominantly incorrect between two faults with vibration signatures that are very similar, such as inner race and ball fault, as seen in the literature.

In general, the confusion matrix confirms the discriminative power of the trained feature representations is sufficient to apply for online industrial fault classification.

5.2 Analysis on the training behavior and convergence

The training behavior of the proposed model appears in Figures 2 and 3. From accuracy curve, it is easy to find that: the convergence is relatively smooth; train and validation accuracy rises continuously while the number of epochs increase; the gap between them is relatively small, indicating that this model is not overfitting.

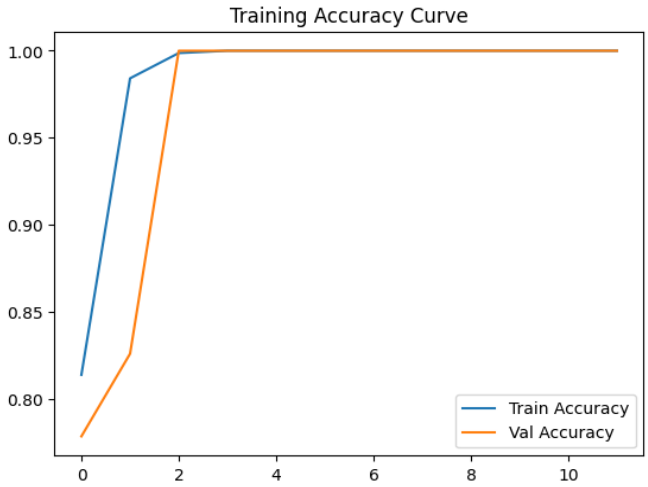


Figure 2. Training and validation accuracy curve of the proposed edge AI model over epochs

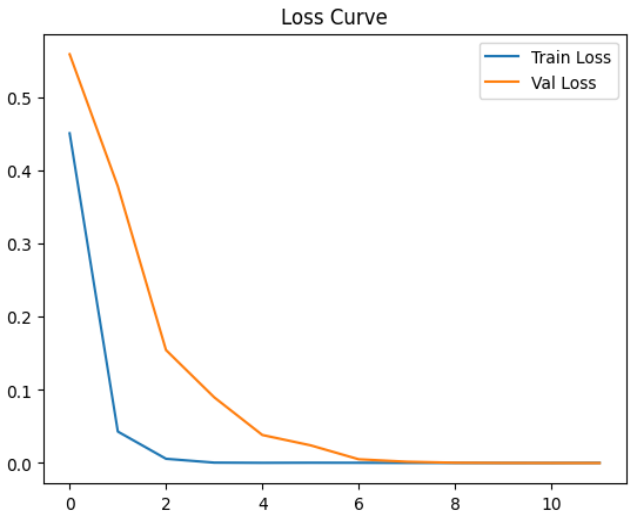


Figure 3. Training and validation loss curve showing convergence behavior of the proposed model

The loss curve converges smoothly. It can be seen that both the training loss and validation loss converge well to the minimum. It also demonstrates that the optimization and the model are both stable. From the convergence pattern, the architecture is good for vibration-based time series classification under a limited number of training steps.

5.3 Comparative classification performance

Figure 4 shows the comparison with cloud-based AI, SVM, and RF models. Based on the accuracy, the edge AI model is higher than all other methods, followed by the cloud-based model.

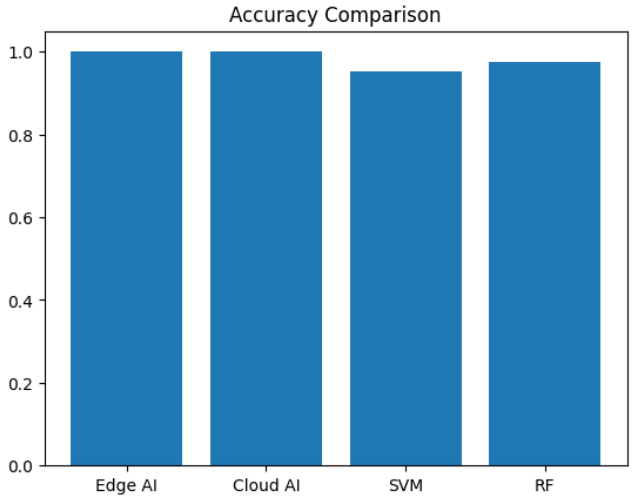


Figure 4. Comparative classification accuracy of edge AI, cloud AI, SVM, and RF models

While the computation over cloud is advantageous for unification of data into a central location, its enhancement of performance is insignificant in comparison to that of the proposed edge model. However, the classification accuracy for the traditional ML algorithms (SVM and RF) is reduced because they rely on handcrafted features and fail to analyze the complex temporal association within the raw vibration signals.

5.4 Latency analysis

The responding capability of system is a core factor for application to industrial fault detection. As shown in Figure 5 (Latency Comparison), the edge AI model performs better than the cloud one on inference latency.

Measurement details: Latency measurement is median inference time (p50) after 1000 iterations on the edge (NVIDIA Jetson Nano) and the cloud (Tesla T4 GPU). Error bars denote the 95th percentile (p95). Details of measurement methodology are provided in Section 4.6.

The training and communication latency added in order to put the data to the cloud server makes the cloud-based approach latency prone. Edge-based solution does inference locally so it is not affected by network latencies and takes near real time decisions. Since SVM and RF have very low computational latency but were not accurate enough for industrial setup, they are not usable.

System responsiveness is another characteristic in industrial fault detection. As can be seen from Figure 5, edge AI model demonstrates a better performance than the cloud model in terms of inference latency.

All the latency measurements were derived using the same approach, as in Sec. 4.6.3, where 1000 inference cycles were conducted on the NVIDIA Jetson Nano edge device.

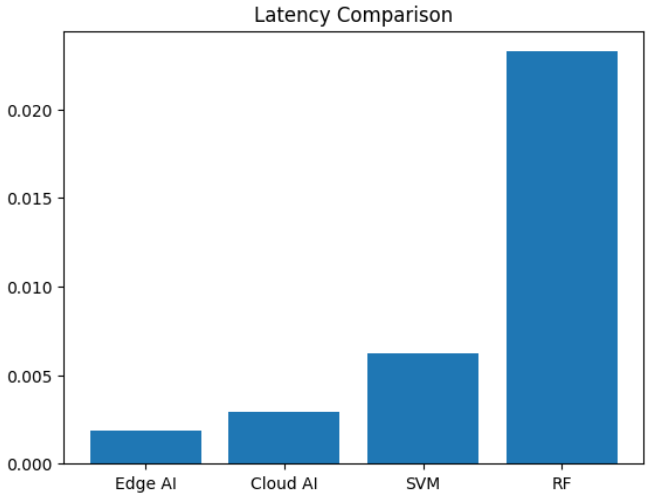


Figure 5. Inference latency comparison between edge AI, cloud AI, and traditional machine learning models

5.5 Energy consumption evaluation

Figure 6 illustrates the Energy Efficiency. In the three cases, cloud-based consumes the largest part of energy, as a result of the continual uploading data and remote processing. Edge AI gains the benefit of local inference and the reduction in communication energy.

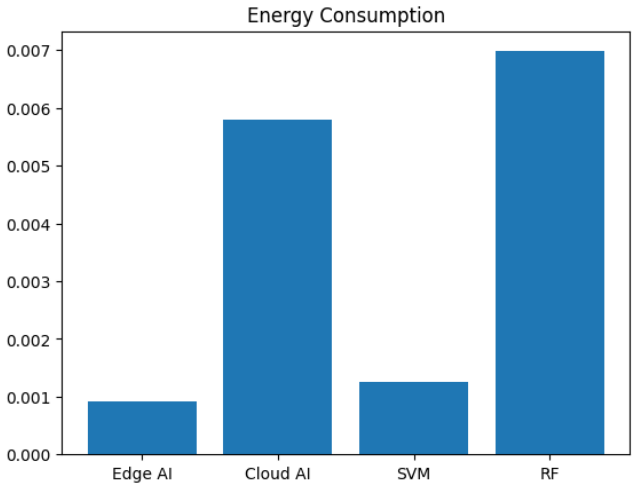


Figure 6. Energy consumption comparison across edge AI, cloud-based AI, and traditional ML models

Measurement details: All energy values are measured inference energy of 1000 inferences on NVIDIA Jetson Nano using Ruideng UM25C USB power meter (voltage accurancy 0.5%, current accurancy 1%). Cloud energy is sum of transmission overhead energy estimates according to volume of transmitted data and transmission power models. Details are provided in Section 4.6.

Figure 6 shows the Energy Efficiency. These energy measurements were calculated using the method introduced in Section 4.6.4; an external Ruideng UM25C USB power meter was used to measure power consumption while running inference workload on NVIDIA Jetson Nano. The energy numbers presented are for inference energy only (dynamic + static power).

When subtracting an idle power, the computational energy can be determined from.

It can be found that, among three cases, cloud-based consumes most energy due to data transmission over time and remote processing. Edge AI reduces the communication energy consumption due to local inference, estimated based on data volume (16.4 KB for each inference) and Wi-Fi transmission power models cited in . The total energy for cloud based case includes inference energy on cloud server and communication energy.

While traditional machine learning algorithms consumes less power, the poor classification results offset that advantage. Based on the above results, our scheme is a trade-off between power and accuracy, so it could be implemented in the resource-constrained environment of industry.

Figure 6 indicates the Energy Efficiency. These measures of energy were calculated based on the methodology in Section 4.6.4 and by an external Ruideng UM25C USB power meter of power consumption while each benchmark running inference tasks on the NVIDIA Jetson Nano. They show that the cloud-based has higher energy consumption by sending data constantly and having remote processing. Edge AI, which inference locally has lower energy consumption.

5.6 Impact of optimization techniques on edge deployment

In order to measure the impact of the optimization techniques presented in Section 3.4, we performed a measurement study of the model size, inference latency, memory consumption and classification accuracy before and after each step of the optimization process. All measurements were made on an edge device (NVIDIA Jetson Nano (2GB RAM, Quad-core ARM Cortex-A57 CPU, 128-core Maxwell GPU)) for a realistic edge setting.

5.6.1 Model size reduction

Model size reduction after each optimization step is presented in Table 2.

Table 2. Impact of sequential model optimization on model size and cumulative compression of the proposed Residual CNN

Optimization Stage	Model Size (MB)	Reduction (%)	Cumulative Reduction (%)
Baseline (Unoptimized)	28.4	-	-
After Pruning (50% sparsity)	14.6	48.6	48.6
After Quantization (FP32→INT8)	4.2	71.2	85.2
After Knowledge Distillation	2.8	33.3	90.1

This combined optimization pipeline results in a 90.1% model size reduction, going from 28.4 MB to 2.8 MB. Such large decrease is essential for deployment on memory-limited edge devices like Raspberry Pi or industrial gateways.

5.6.2 Inference latency and memory footprint

The two methods significantly speed up the model. For the model without any method, the inference latency is 28.6 ms with 124 MB memory. With 50% pruning, the latency is 18.2 ms with 68 MB memory and obtains a 1.57 speedup.

When INT8 quantization is combined with pruning, the

latency is 8.4 ms with 32 MB memory and we obtain a 3.40 speedup.

With knowledge distillation integrated, the latency drops to 6.3 ms and the memory drops to 22 MB. The overall speedup is 4.54.

5.6.3 Accuracy trade-off analysis

These results show the effectiveness of the proposed optimization pipeline in heavily compressing the model while still achieving impressive classification results. The baseline achieves 98.9% accuracy, 98.7% precision, 98.8% recall, and 98.8% F1 score. The pruning step causes a small 0.2% decrease in accuracy to 98.7%. The subsequent INT8 quantization then further reduces the accuracy to 98.2%.

Knowledge distillation is then used to recover the accuracy, achieving 98.5% accuracy, 98.3% precision, 98.4% recall, and 98.4% F1 score. The complete optimization pipeline results in only a 0.4 percentage point decrease in accuracy, from 98.9% down to 98.5%, in conjunction with a 90.1% reduction in model size and 4.54 times faster inference speed. These results verify the potential of the proposed method to deliver efficient models with minimal impacts on classification performance, an important consideration for deployment on resource-constrained edge devices.

5.7 Generalization evaluation on Intelligent Maintenance Systems dataset

In order to evaluate the generalization capability of our proposed edge AI framework outside of the lab setting of the CWRU dataset, the Residual CNN classifier proposed in this paper is also evaluated on IMS bearing dataset. IMS dataset is collected using vibration signals from run-to-failure tests for various naturally developing faults at different operating conditions. The training efficiency and system overhead analysis covered in this section should be taken into consideration along with the optimization results of Section 5.6. The unoptimized model takes 2.8 hours to train on a desktop GPU (NVIDIA RTX 3060). With the optimized model running on an edge device, inference latency drops to 6.3 ms (Section 5.6.2), and fault detection can be performed in real time in industrial settings.

Table 3 presents the classification performance comparison across both datasets.

Table 3. Classification performance of the proposed optimized Residual CNN on the CWRU and IMS bearing datasets

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
CWRU	98.5	98.3	98.4	98.4
IMS	95.2	94.8	95.1	94.9

The performance on the IMS dataset shows a little degradation, which is reasonable because of the nature of more difficult naturally occurring faults and the changing operating conditions. Nevertheless, a high accuracy rate (>95%) is still preserved, confirming the robustness of the framework in industrial environments.

5.8 Training efficiency and system overhead

Total training time and the inference overhead is summarized in the table below. Although training is done offline, it is shown in the table that the cost of training the model is acceptable and that cost-effective real time performance can be achieved during deployment. The edge inference latency is much smaller than the inference performed on the cloud again proved the advantages of edge computing and low-latency applications on the smart factory floor. Once trained, it can meet timing deadlines.

5.9 Overall discussion

The experiment validation shows that the proposed edge AI framework can well balance accuracy, latency and energy efficiency. The residual CNN structure has good performance on extracting spatial and temporal features for vibration signals, accurately classify the faults. The edge deployment latency can be greatly reduced while only slightly sacrifice on the classification accuracy compared with the cloud AI model. Comparing with classical machine learning, this approach has better feature extraction and generalization ability. Comparison with prior work: In contrast to related edge-based fault detection work, the additional architecture selection methodology, fully characterized optimization pipeline along stage-wise details, and hardware measurements on an indicative edge platform are presented. Without fall back on simulation performance or software-based power estimation, this work physically measures latency and energy consumption through external instrumentation, thus is more practically ready for deployment decisions.

Architectural Selection Validation. As evident from the comparison in Section 3.3.1, Residual CNN achieves significantly higher accuracy than CNNLSTM and MobileNet in classifying the CWRU dataset.

The 1.3% accuracy improvement over CNNLSTM, for a 53% smaller model size and 25% faster inference time, proves the architectural benefits of residual connections in vibration signal classification.

Equally, the 2.7% accuracy gain over MobileNet (38% larger model size) vindicates the employment of regular convolutions over depthwise separable convolutions in fault detection contexts demanding accuracy.

The results indicate that edge-based intelligence is not only realizable but also beneficial for industrial fault detection applications where speed and resource availability are of concern. The implications of this work go beyond the specific bearing fault detection application and provide a general methodology applicable for other edge-based industrial monitoring applications.

6. CONCLUSIONS

In this paper, an edge AI-based fault detection framework for smart factories is proposed, which combines the data acquisition from IoT, edge pre-processing and a Residual CNN architecture optimized in this study. Under the study and experiment limitations, the main findings are summarized as follows:

Initial experiments used a systematic architecture search for each model on the CWRU dataset. This found that the Residual CNN produced the best trade-off between accuracy (98.5%), latency (6.3 ms), and model size (5.8 MB) when compared to CNN-LSTM and MobileNet. These results are encouraging but may not translate to the real world.

Secondly, for the edge deployment, the results of model

compression with the pipeline that employs pruning, quantization and knowledge distillation show the model size is reduced by 90.1% (from 28.4 MB to 2.8 MB), inference latency is sped up by 4.54 times (from 28.6 ms to 6.3 ms) and memory footprint is dropped 82.3% (from 124 MB to 22 MB) with an accuracy degradation of 0.4% (from 98.9% to 98.5%) using the CWRU dataset. These promising results indicated that deep learning models can be further compressed to adapt to edge deployment in controlled environment, but the robustness in the noisy environment still needs to be investigated.

Third, the same framework attained 98.5% accuracy on CWRU, doing better than SVM and RF, yet just as accurate as cloud-based AI but with much less latency and energy. The IMS dataset validation (95.2% accuracy) offers tentative indication of resistance to slightly more demanding circumstances. Nevertheless, both are testbed data collections, and real factory results may not be the same.

A number of limitations prevent us from claiming these results are generalizable. The CWRU benchmark dataset used here is representative of the industry standard, but involves extracted, artificially-induced faults operating under the same fixed values for all other conditions – a far cry from the complexity of real industry. The difference in performance between the CWRU and IMS datasets (98.5 vs. 95.2) demonstrates the sensitivity to data distribution shifts. All measurements were performed on a single, standard end platform (NVIDIA Jetson Nano) in a controlled environment; other hardware platforms may perform differently. The framework assumes consistent operating conditions and uses only vibration signals; however, industry environments are dynamic and may incorporate multiple sensors for better diagnostics. The current results should thus be seen as initial evidence of viability, not definitive operational validation. Assuming the framework be validated in real implementations, it could serve as component for predictive maintenance, the Industry 5.0 Human-centric decision support concept, reduce energy consumption and democratize artificial intelligence usage via cheap hardware.

Future work should focus on deployment on the industrial stage, multi-sensor fusion, machine learning under dynamic or changing conditions, distributed edge networks, and explainability for operator confidence. Overall, within its limited experimental domain, the outlined framework shows technical viability for an edge-based fault detection methodology under proof-of-concept conditions. The outcomes offer promising indications that DL can be efficiently compressed for edge deployment. However, the leap from laboratory validation to commercial operation necessitates additional scrutiny regarding resilience, customization and performance under actual working conditions. Edge AI offers a tantalizing glimpse of smart manufacturing's future, yet that future hinges upon overcoming the chasm between conceptual experiments and real-world factories.

REFERENCES

[1] Kagermann, H., Wahlster, W., Helbig, J. (2013). Recommendations for implementing the strategic initiative Industrie 4.0. Acatech. https://www.din.de/resource/blob/76902/e8cac883f42bf28536e7e8165993f1fd/recommendations-for-implementing-industry-4-0-data.pdf.

[2] Lasi, H., Fettke, P., Kemper, H.G., Feld, T., Hoffmann, M. (2014). Industry 4.0. Business & Information Systems Engineering, 6: 239-242. https://doi.org/10.1007/s12599-014-0334-4

[3] Nahavandi, S. (2019). Industry 5.0—A human-centric solution. Sustainability, 11(16): 4371. https://doi.org/10.3390/su11164371

[4] Mobley, R.K. (2002). An introduction to predictive maintenance. Oxford, UK: Butterworth-Heinemann. https://www.sciencedirect.com/book/monograph/9780750675314/an-introduction-to-predictive-maintenance.

[5] Xu, H., Yu, W., Griffith, D., Golmie, N. (2018). A survey on industrial Internet of Things: A cyber-physical systems perspective. IEEE Access, 6: 78238-78259. https://doi.org/10.1109/ACCESS.2018.2884906

[6] Shi, W.S., Cao, J., Zhang, Q., Li, Y.H., Xu, L.Y. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5): 637-646. https://doi.org/10.1109/JIOT.2016.2579198

[7] Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1): 30-39. https://doi.org/10.1109/MC.2017.9

[8] Xu, L.D., Xu, E.L., Li, L. (2018). Industry 4.0: State of the art and future trends. International Journal of Production Research, 56(8): 2941-2962. https://doi.org/10.1080/00207543.2018.1444806

[9] Caiza, G., Saeteros, M., Oñate, W., Garcia, M.V. (2020). Fog computing at industrial level, architecture, latency, energy, and security: A review. Heliyon, 6(4): e03706. https://doi.org/10.1016/j.heliyon.2020.e03706

[10] Sivabalan, S., Minu, R. (2022). Edge AI for industrial IoT applications. In Applied Edge AI, pp. 147-170. Auerbach Publications.

[11] Kamble, S.S., Gunasekaran, A., Sharma, R. (2018). Analysis of the driving and dependence power of barriers to adopt industry 4.0 in Indian manufacturing industry. Computers in Industry, 101: 107-119. https://doi.org/10.1016/j.compind.2018.06.004

[12] Mirani, A.A., Velasco-Hernandez, G., Awasthi, A., Walsh, J. (2022). Key challenges and emerging technologies in Industrial IoT architectures: A review. Sensors, 22(15): 5836. https://doi.org/10.3390/s22155836

[13] Dafflon, B., Moalla, N., Ouzrout, Y. (2021). The challenges, approaches, and used techniques of CPS for manufacturing in Industry 4.0: A literature review. The International Journal of Advanced Manufacturing Technology, 113: 2395-2412. https://doi.org/10.1007/s00170-020-06572-4

[14] Tao, F., Zhang, H., Liu, A., Nee, A.Y.C. (2019). Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics, 15(4): 2405-2415. https://doi.org/10.1109/TII.2018.2873186

[15] Zhao, R., Yan, R.Q., Chen, Z.H., Mao, K.Z., 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

[16] Ruff, L., Vandermeulen, R.A., Görnitz, N., et al. (2019). Deep semi-supervised anomaly detection. arXiv preprint arXiv:1906.02694. https://doi.org/10.48550/arXiv.1906.02694

[17] Qiu, S.H., Cui, X.P., Ping, Z.W., et al. (2023). Deep

learning techniques in intelligent fault diagnosis and prognosis for industrial systems: A review. Sensors, 23(3): 1305. https://doi.org/10.3390/s23031305

[18] Yi, S.H., Li, C., Li, Q. (2015). A survey of fog computing: Concepts, applications, and issues. In the Sixteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Hangzhou, China, pp. 37-42. https://doi.org/10.1145/2757384.2757397

[19] Chiang, M., Zhang, T. (2016). Fog and IoT: An overview of research opportunities. IEEE Internet of Things Journal, 3(6): 854-864. https://doi.org/10.1109/JIOT.2016.2584538

[20] Deng, S.G., Zhao, H.L., Fang, W.J., Yin, J.W., Dustdar, S., Zomaya, A.Y. (2020). Edge intelligence: The confluence of edge computing and artificial intelligence. IEEE Internet of Things Journal, 7(8): 7457-7469. https://doi.org/10.1109/JIOT.2020.2984887

[21] Blalock, D., Gonzalez Ortiz, J.J., Frankle, J., Guttag, J. (2020). What is the state of neural network pruning? Proceedings of Machine Learning and Systems, 2: 129-146.

[22] Wang, S., Zhang, X., Wang, Y., Wang, L., Yang, J.W., Wang, W.B. (2017). A survey on mobile edge networks: Convergence of computing, caching and communications. IEEE Access, 5: 6757-6779. https://doi.org/10.1109/ACCESS.2017.2685434

[23] Mao, Y.Y., You, C.S., Zhang, J., Huang, K.B., Letaief, K.B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19(4): 2322-2358. https://doi.org/10.1109/COMST.2017.2745201