FaultBit: Generic and Efficient Wireless Fault Detection Using the Internet of Things

Koustabh Dolui(✉), Ashok Samraj Thangarajan, Sergii Morshchavka, Zhaoyi Liu, Sam Michiels, and Danny Hughes

imec-Distrinet, Department of Computer Science, KU Leuven, Leuven, Belgium
{koustabh.dolui,ashok.thangarajan,sergii.morshchavka,zhaoyi.liu, sam.michiels,danny.hughes}@kuleuven.be

Abstract. The timely monitoring and maintenance of industrial machines is critical to prevent expensive disruptions and down-time. The Internet of Things (IoT) offers a solution to instrument production processes at lower cost and complexity. However, wireless IoT networks are bandwidth constrained, which precludes the transmission of high frequency signals such as vibration or electrical current. Prior approaches to tackling this problem require a high degree of application-specific re-engineering. In this paper, we argue for a new approach to fault detection that is accurate, efficient and applicable to a large class of fault detection problems. We propose FaultBit, an application independent toolkit for fault classification that can gather, compress and classify data using IoT networks. We evaluate FaultBit in two representative scenarios using current and vibration data. In both cases, FaultBit offers classification accuracy 99%, which is very close to application-specific classifiers, while requiring 512× less bandwidth, enabling a battery life of several years.

Keywords: Internet of Things · Wireless Sensor Networks · Machine Learning · Signal Processing · Energy awareness

1 Introduction

The real-time monitoring of industrial equipment prevents expensive machine failures and downtime by enabling *predictive maintenance*, which significantly reduces operational costs by more effectively targeting maintenance effort. For example, the failure of rolling bearings accounts for 45–55% of rotating machinery failures [24]. The emergence of low-cost IoT devices holds great promise in the domain of predictive maintenance, as it enables the wireless monitoring of large numbers of machines without the cost and complexity of routing dedicated

This research is partially funded by the Research Fund KU Leuven (ReSOS, C3/20/014).

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 593, pp. 97–118, 2024.
https://doi.org/10.1007/978-3-031-63989-0_5

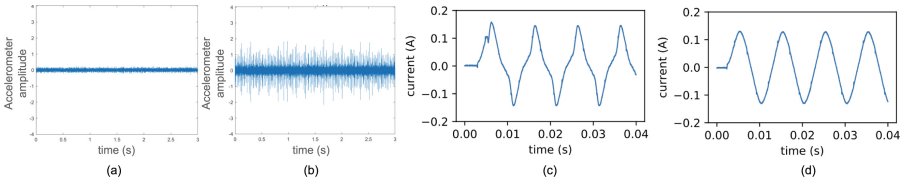

Fig. 1. Illustrative examples of raw fault detection signals taken from our use-cases. (a) shows accelerometer amplitude from a healthy bearing, while (b) shows accelerometer amplitude from a faulty bearing. (c) Shows electrical current signal passing through a healthy solenoid valve, while (d) shows electrical current signal passing through a faulty solenoid valve.

power and network cables [4,37]. Furthermore, wireless networking enables sensors to be embedded in difficult environments, such as dense machinery and on rotating equipment.

While the vision of IoT-enabled predictive maintenance is attractive, significant problems remain. Specifically, in order to deliver multi-year battery life, IoT devices use specialized low power networks such as BLE [5], 6TiSCH [12] or LoRaWAN [3], which offer throughput ranging from a few bps to a few kbps; far too slow to support real-time telemetry from audio, vibration or current sensors. While higher speed networks exist; the fundamental trade-off between bandwidth and energy precludes the combination of long life *and* high speed networking. Furthermore, the hardware design decisions that are required to ensure long battery life also lead to limited memory and processing capacity on the sensor devices themselves, which precludes the local execution of complex models. Neither networking nor computation constraints can be significantly relaxed without negatively impacting battery life and thereby increasing maintenance costs.

The need to reduce bandwidth requirements, while operating within strict memory, processing and power constraints has led to multiple streams of research. On the one hand, generic compression approaches [30,32] enable high frequency sensing by compressing data at its source. However, the bandwidth reductions that can be achieved by these approaches are limited due to their lack of application knowledge. On the other hand, embedded machine learning approaches [11,43] take an orthogonal approach, miniaturising the models themselves so that they can be executed partially, or fully on the IoT device itself. While this family of approaches can achieve dramatic bandwidth and energy savings, the complexity of models is limited and systems developed using this methodology are inherently application specific. This paper takes a different approach, investigating an energy-aware IoT fault detection system that is *generic* in the sense that it can use a wide variety of sensors and provide accurate results across a variety of application domains. FaultBit maximises the efficiency of sensing and compression through continual optimisation, while satisfying application goals such as device lifetime and desired model accuracy. FaultBit achieves that by using an ensemble of standard models that are commonly applied to fault detection and illustrates that faults can be classified efficiently without the need

for use-case specific and customized models. The scientific contributions of this paper are as follows:

1. The first self-adaptive classifier-in-the-loop fault detection system, wherein the classifier is responsible for (re)configuring and optimizing sensors and feature extraction in real-time.
2. An efficient embedded sensing and feature extraction toolkit that offers flexible reconfiguration of the sensor front end and feature extraction models, while operating within the resource constraints of conventional IoT platforms and preserving multi-year battery life.
3. Automated network-wide sensor reconfiguration techniques that empower engineers to easily trade off between fault identification latency and battery lifetime when applying standard fault classifiers.

We use a combination of lab-based benchmarks and open data sets [6] to evaluate FaultBit in two challenging and representative fault diagnosis applications: (i.) using vibration to evaluate bearing wear and (ii.) using electrical current to monitor the state of solenoid valves. In both cases, FaultBit detects faults with an average accuracy that is within 1% of state-of-the-art application-specific techniques, while reducing bandwidth by 512× in comparison to raw data transmission. The accurate classification of normal vs faulty devices facilitates maintenance and earlier interventions in comparison to periodic maintenance.

The remaining sections of the paper are structured as follows. In Sect. 2, we provide background information on fault detection at the edge using IoT devices and derive the requirements for FaultBit. The design of FaultBit is then described in Sect. 3, followed by a reference implementation in Sect. 4. In Sect. 5, we evaluate FaultBit's performance. In Sect. 6, we discuss related work and emphasize the gap that FaultBit fills. Section 7 concludes with discussion and future work.

2 Background

The objective of FaultBit is to detect incipient faults which develop over time in industrial equipment using sensors that are connected by wireless and resource-constrained IoT devices. In particular, we focus on performing fault detection on one dimensional data sharing a common timeline such as single axis acceleration, vibration, distance or electrical current measurements. This data will typically be generated by a large number of wireless sensors that may be distributed across large and complex environments such as factories, trains or warehouses.

2.1 Fault Data Characteristics

Faults in industrial equipment, such as motors and valves, are inevitable due to gradual deterioration caused by corrosion, high loads, and fatigue [40]. Timely monitoring and analysis are essential for predicting fault occurrence and targeting maintenance interventions. For example, predicting Remaining Useful Life (RUL) for solenoid valves can reduce plant shutdowns [25]. However, detecting

incipient faults is challenging because they begin as infrequent events, requiring continuous input data sampling. Constrained IoT devices cannot store raw data for periodic collection or stream it in real-time over low-power wireless networks. We explore these challenges through two representative industrial fault detection applications: vibration monitoring for motors and electrical current monitoring for solenoid valves, which will be used to evaluate FaultBit.

Vibration Monitoring for Motors. Vibration monitoring is widely used for structural health monitoring and fault diagnosis, as accelerometers capture faults earlier than other sensors. Figure 1(a) and 1(b) show contrasting accelerometer amplitude patterns from healthy and faulty bearings. Contemporary MEMS accelerometers sample at 6–26 KHz with a resolution of 1–4 bytes, resulting in a raw data bandwidth requirement of 53–820 kbps, which is orders of magnitude greater than the capacity of low-power IoT networks [19,26].

Solenoid Monitoring. Solenoid valve RUL estimation requires continuous monitoring of their electrical current consumption [25,38], sampled at rates as high as 50 KHz, resulting in data rates of up to 200 kbps per solenoid. This precludes continuous storage or transmission on constrained IoT devices and networks. High-frequency signal monitoring necessitates aggressive compression and filtering, while preserving sufficient data to identify subtle faults, as shown in the electrical current sensor output from a solenoid valve in healthy state and at end-of-life, faulty state in Fig. 1(c) and (d) respectively. Furthermore, in the case of a generic solution, re-configurable data processing algorithms are required to accommodate changing application requirements and environmental conditions. It should be noted that both examples involve cyclo-stationary signals [1], defined as signals having statistical properties that vary periodically with time [15]. A cyclo-stationary signal is illustrated in Fig. 1(c) and Fig. 1(d) showing the pattern of current passing through a solenoid valve over its lifetime. The signal from a faulty bearing in Fig. 1(a) and Fig. 1(b) also represents cyclo-stationarity, however, not as prominent as the solenoid, since the input current signal to the solenoid is cyclo-stationary itself and the solenoid switching is performed uniformly over time. We consider cyclo-stationary signals since this pattern is observable commonly in a wide range of industrial machinery like rotating machinery, gearboxes, piston pumps, bearings and valves [29,35]. As a result, FaultBit can be extended to a wide-range of applications beyond the two examples discussed in this paper.

2.2 IoT Resource Constraints

The limited resources of IoT devices pose challenges in several dimensions.

Sensors. As described in Sect. 2.1, the sample rate and resolution of many fault detection sensors is far beyond that which low-power IoT networks can support. The state of practice in data-driven analytics is to work with raw high

resolution data [42,44]. Likewise, it is extremely difficult to miniaturise fault detection models to the point that they can run directly on the IoT devices themselves. Hence, a generic sensor optimization framework is required together with optimal pre-processing and compression techniques [9]. This enables sensor data workloads to be reduced to the greatest extent possible while preserving full flexibility for fault detection models running in the back-end.

Networking. IoT networks are optimised for low-power operation in order to maximize battery life. This naturally comes at the expense of bandwidth. For example, Low Power Wide Area Network (LPWAN) technologies such as LoRa and Sigfox deliver long range transmissions on a low power budget, but offer a maximum theoretical bandwidth of 290 bps, which is further decreased by a factor of 10–1000× based on regional duty cycle limitations [3]. While higher speed networks are available, the fundamental trade-off between power, range and bandwidth precludes long life *and* high speed networks. This clearly makes the transmission of raw sensor data such as that described in Sect. 2.1 unfeasible. While technologies such as Bluetooth Low Energy (BLE) and 802.15.4 offer somewhat higher bandwidth, this is still insufficient to support any form of raw sensor data transmission. Hence, edge analysis must be performed on the device itself to optimize bandwidth use and transmit only the essential features that are required to train a failure detection model and classify collected data.

Energy. IoT devices may be powered by either batteries or harvested energy from the environment, either of which results in a constrained energy budget [4, 36]. Utilization of network bandwidth as well as local processing resources to perform on-device analysis depletes the battery. However, networking typically consumes the majority of available power. Compression or pre-processing of data on the device reduces transmitted data and thereby power consumption, however it may adversely affect the performance of the fault detection analysis itself. The key role for a generic IoT fault detection middleware such as FaultBit is thus to measure the relationship between data compression and model accuracy and co-optimise the two such that a sensor system achieves its required lifetime while also delivering fault diagnosis of sufficient accuracy.

2.3 Fault Modelling

The data collected for fault detection in various use cases may originate from different sensors with varying data distributions and modeling of such data. Data-driven techniques, such as the use of deep learning models, can solve this issue, however, prior work has typically required raw data to uncover hidden data representations for early fault identification [16,41]. Some prior work on the prediction of RUL for solenoids also utilizes multi-modal sensor data, such as temperature and pressure sensors along with the current measurement sensors, which necessitates either (i) deep learning approaches for data-driven analysis or (ii) considerable domain-specific expert knowledge to build physical models

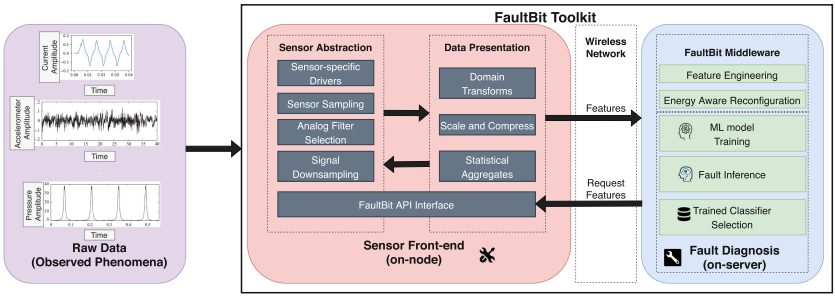

Fig. 2. FaultBit Architecture.

for fault detection. Physical model features also tend to be specific to the fault detection application, limiting the re-usability of these approaches. In contrast to data driven techniques, FaultBit enables deep configuration of sensors followed by generic on-device compression and filtering to minimize required bandwidth. The major challenge here lies in the consistent abstraction of (i.) sensors, (ii.) data compression and (iii.) filtering steps. In order to classify the sensor data as faults, FaultBit applies supervised learning techniques, such as Support Vector Machines (SVM), Random Forest (RF), and neural networks.

2.4 Requirements

The requirements for FaultBit, derived from the above challenges are as follows:

1. Deliver a consistent abstraction of the sensor layer that simplifies the reconfiguration of a variety of sensors.
2. Provide a generic suite of data processing, compression and filtering algorithms that reduce the bandwidth requirements of typical fault detection sensors.
3. Enable intelligent trade-offs between device lifetime, model accuracy and fault diagnosis latency.
4. Deliver self-adaptive software techniques to optimize sensor and edge analysis functionality to meet application requirements and environmental conditions.

3 Design of FaultBit

FaultBit strives to provide a generic deploy-and-forget approach to fault identification using resource-constrained wireless IoT sensors. This demands an end-to-end approach that continuously optimises the configuration of the IoT sensor nodes to minimise energy and bandwidth use, while maintaining a high level of accuracy in fault detection. To balance these contradictory requirements FaultBit adopts a two tier approach connected via a feedback loop. FaultBit includes a common sensor abstraction layer with low-level sensor specific drivers and a common networked API that can adapt key parameters of the driver to meet application-specific fault diagnosis demands.

The FaultBit toolkit operates in two phases (i.) the Training Phase and (ii.) the Fault Detection Phase. In the training phase, FaultBit analyses a range of different features from the acquired signal to identify the minimal yet sufficient *feature vector* that enables diagnosis of faults at sufficient accuracy and latency during the fault detection phase. Throughout both the training and fault detection phases, FaultBit operates in a self-adaptive feedback loop wherein the performance of the models is used to tailor data gathering functionality. The features that are selected for fault identification are identified through an iterative training process using increasing volumes of data. After the model is trained to identify faults, we show that the periodic transmission of very small feature vectors, often fitting within a single BLE or LoRa packet, is sufficient to diagnose a diverse range of faults. Specifically, we evaluate FaultBit on cyclo-stationary signals, defined as signals having statistical properties that vary periodically with time [15], which constitutes majority of signals observed from rotatory machines [1,8]. The training phase can be done with minimal data by imparting expert knowledge, drastically improving the lifetime of the sensor nodes. The block diagram of FaultBit is shown in Fig. 2 and explained in the remainder of this section.

3.1 Sensor Front-End

Contemporary digital sensors offer a wide range of configuration options which, in combination with the computation capabilities of IoT nodes, can be used to efficiently implement data compression and filtering [2,37]. FaultBit exploits this capability with a common device abstraction layer, which is used to configure sensor and data options. The commands are translated to sensor specific parameters and configurations for the algorithm adopted for on-node processing. An example sensor configuration for vibration monitoring may include: a spectrum with a frequency resolution of 1 Hz and a signal range of 0-400 Hz, with a 1-byte feature resolution. This is translated by FaultBit to a sensor sampling rate of 800 Hz and a 512 point FFT. This sensor front-end provides an array of parameters that form the first tunable knob in FaultBit. The configuration of the sensor node and the features extracted is controlled by the FaultBit middleware from the back-end through the aforementioned networked API interface (Table 1).

Sensor APIs are broadly divided into sensor node configuration and data representation. The sensor node processes received API calls as a stream, which allows them to be arbitrarily aggregated in order to improve energy efficiency. The use of a common API abstraction mitigates the need for back-end developers to understand low-level and platform-specific sensing details. Algorithm 1 shows the aggregate command translation of a sensor with a finite number of sampling configuration and frequency domain transform configuration.

3.2 FaultBit Middleware

FaultBit assumes that the signal under analysis is *stationary* (e.g. the frequency of an AC power supply or PWM motor control) or *cyclo-stationary* (e.g.

Table 1. APIs for Configuring FaultBit based Sensor

FaultBit Block	API	Parameter Values
Sensor Node Configs	set_range	r_{min}, r_{max}
	set_dimension	sensor_specific
	set_feature_resolution	data_resolution_in_bytes
	enable_impact_detection	enable/disable, no_of_peaks
	enable_frequency_domain_feature	enable/disable, no_of_peaks
	set_data_resolution	$res_{min}, res_{max}(bits)$
	set_monitor_signal_limit	two_byte_max_monitor_signal_limit
Data Representation	send_enabled_features	-
	config_bandpass_filter	f_L, f_S
	downsampling_coefficient	d
	set_fft_size	n_{fft}
	send_full_data	max_frequency, bin_resolution

accelerometer data from a bearing or electrical current measurements from a solenoid [1]). Based on the observed signal, the FaultBit middleware then optimizes pre-processing on the IoT device using the API described in Sect. 3.1. To accomplish this, during the training phase, a generic set of features are iteratively extracted from the raw data and used to train a family of generic models. The specific features that are extracted are tailored based upon feedback from the middleware during the training phase as described in Sect. 3.3. Once the training phase is complete and an adequate diagnosis accuracy has been achieved, FaultBit identifies a definite feature vector which will then be periodically transmitted by the embedded sensors.

Feature Engineering. In the case of cyclo-stationary signals, faults typically manifest in the frequency domain and consequently, spectrum analysis techniques such as Fast Fourier Transform (FFT) are applied to perform domain transformations. Due to its computational efficiency, FFT is well-suited for fault detection applications on constrained IoT devices. In addition, the FFT facilitates the extraction of a set of statistical features that can be utilized in a variety of applications. FaultBit also provides a configurable a band pass filter to extract signals of the relevant frequency for the application. The sample rate and resolution can often still be too high for IoT networks to support. Hence, the output signal is then down-sampled by a factor d depending on the feature resolution specified by the user.

The spectrograms obtained from the FFT are accumulated and averaged in the frequency domain to provide an output of the required resolution. For example, a 1024 bin FFT may be down-sampled in the frequency domain by averaging adjacent bins to form a 512 bin output, depending upon the required frequency resolution of the application. Peak detection is then applied, wherein peak amplitude and frequency pairs are extracted. Each amplitude-frequency pair serves as

Algorithm 1: Aggregate Query Translation.

Output: S_{src} := sensor sampling rate configuration
Output: B_{rc} := domain transform bin resolution configuration
Function *aggregate_query_translation(signal_range, feature_resolution)* **is**
    `// Finding the sampling rate configuration`
    `// Iterate through the available sampling rates of the sensor`
    for *iter* $\leftarrow 0$ *to length(sampling_rate[]) do*
        if $sampling\_rate[iter] \geq signal\_range * 2$ **then**
            $S_{src} \leftarrow sampling\_rate[iter]$
            break
        end
    end
    `// Finding suitable algorithm configuration for the sensor_sampling_rate_config`
    `// Iterate through the available algorithm resolution of the sensor`
    for *iter* $\leftarrow 0$ *to length(algo_resolution[]) do*
        $current\_config\_resolution = S_{src}/algo\_resolution[iter]$
        if $current\_config\_resolution \leq feature\_resolution$ **then**
            $B_{rc} \leftarrow algo\_resolution[iter]$
            break
        end
    end
    return S_{src}, B_{rc};
end

a feature for the fault detection models. The number of features transmitted is determined by the FaultBit middleware based on the requirements of the fault classification model, with priority being given to the highest amplitude features. The above steps are illustrated in Fig. 3. These steps represent the on-device processing that is applied across applications supported by FaultBit.

Energy Awareness. The greatest sources of energy consumption for IoT devices are respectively: wireless communication, computation and sensing. The energy consumption of *sensors* varies significantly from low-power digital sensors to power-hungry analog devices and is proportional to the number of samples collected from the sensor. FaultBit supports both digital and analog sensors and enables energy-efficient data acquisition by sampling the sensor at the optimal rate required for fault detection in the application. *Computation* consumes significant energy, and therefore, FaultBit focuses on simple frequency transforms, filtering and data presentation routines that, when used in combination, can significantly reduce network load, while being feasible to host on embedded IoT devices. All models are trained on a cloud server, implying no computational load for the embedded devices. However, *communication* is inherently a power-hungry activity. FaultBit minimizes communication overhead in two key ways. Firstly,

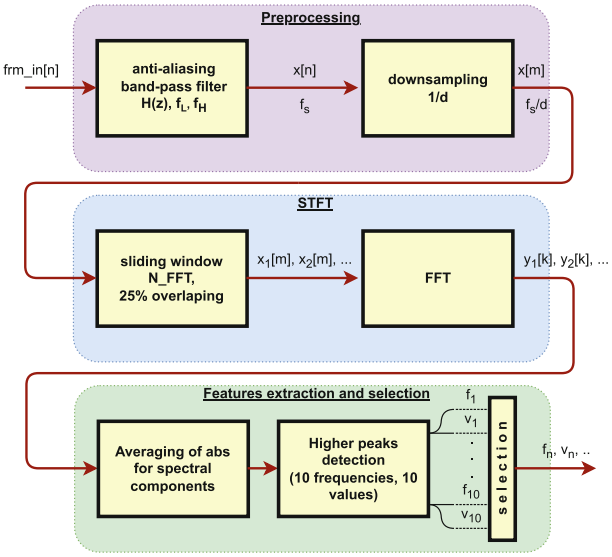

Fig. 3. On-device feature extraction from sensor

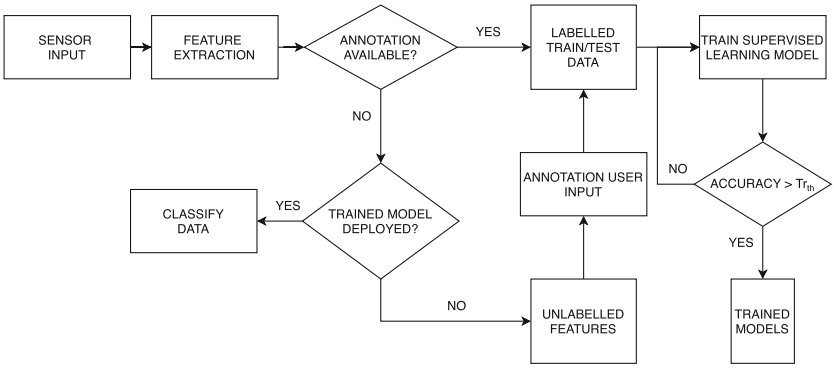

Fig. 4. The life-cycle of a FaultBit application

the data transmitted to the server from the device is compressed, reducing the bandwidth that is required to train the model. Secondly, during the inference stage of fault detection, the rate at which sensor nodes transmit these efficient features is throttled to the minimal rate that is guaranteed to meet the latency requirements of fault detection.

3.3 Fault Diagnosis

The FaultBit toolkit supports the entire life-cycle of model training and inference starting with data collection.

Data Collection and Training. In the data collection stage, the sensor is sampled and the FFT is computed as highlighted in Fig. 3. The entire FFT is transmitted from the device to enable visualization and analysis of the signals required to train various models on the server. The toolkit optimizes data collection at multiple resolutions and sample rates in specified frequency ranges to extract required FFTs for model training. For fault detection, we employ supervised learning, which entails training a model with labelled faulty and non-faulty data. The labelling of data is performed through annotations when periodic monitoring and maintenance of the equipment are performed.

Model Selection. A survey of the state of the art for predictive maintenance [28] reveals that the most popular models used in predictive maintenance include Support Vector Machines(SVM), Decision Tree (DT), Random Forest (RF) Artificial Neural Networks (ANN), Principal Component Analysis (PCA) and k-nearest neighbours (kNN) among others. We select an ensemble of supervised classification models namely Support Vector Machine (SVM), Random Forest (RF) and neural networks to train on the collected data.

Support Vector Machines: SVMs train unsupervised classification and regression models. They use hyperplanes to define class subspaces, and can classify failures as faulty or non-faulty in binary classification. Class overlap reduces accuracy, emphasizing class separability. SVMs are suitable for IoT sensor data, such as from accelerometers and electrical sensors, due to their resilience to noise and ability to perform with minimal training data. Additionally, SVMs use fewer supporting vectors than training instances, facilitating incremental learning [21].

Decision Trees: (DT) methodologies are renowned for their robustness in managing several forms of data, encompassing non-linear, mixed, and categorized data, which is aligned with the requirements of FaultBit. However, it is important to note that DT methods are susceptible to over-fitting and may not always exhibit stability in the presence of noisy data. Random Forest models based on DT approaches exhibits enhanced resistance to noise and a higher likelihood of mitigating over-fitting and in turn yield favorable outcomes in real-world scenarios [7,22]. However, DTs often falter on unbalanced datasets, including bias in data which is the case for CBM datasets due to the fact that the faulty data is typically minimal in real-world scenarios. Nevertheless, we select RFs for their versatility to be applied to various kinds of sensor data and to mitigate over-fitting.

Neural Networks: NNs are beneficial to extract patterns from data and tend to be useful where SVMs or DTs fail to perform or classify faulty data effectively. While SVMs and DTs perform well with limited amounts of data, NNs require larger amounts of training data and hence work efficiently when incrementally trained over a longer period.

Model Training. Once labelled data is available for all classes, the data is split into training and test sets for use with this ensemble of models. The models are

trained starting from only a single feature (i.e. frequency/amplitude pair) on the server with an increasing number of features until (i) the training accuracy meets the application requirements or (ii) the amplitudes of the peak frequencies fall to zero. When the required training accuracy is reached, the set of models satisfying the training accuracy requirement are chosen and their accuracy on the test data is measured. The models satisfying the accuracy requirement on the test data are then sorted by the number of features they use. The model which uses the fewest features while still meeting accuracy requirements is deployed for inference. These steps are shown in Algorithm 2. Each of these approaches can be successfully used with FaultBit, as will be shown in our evaluation.

Inference. The deployed model is used for classification of new incoming data from the device and, as a result, only extracted features are transmitted from the device for inference. FaultBit classifies a device to be faulty based on which maintenance cycles can be performed rather than predicting the time at which the device will fail. During a state of transition from one state to another, data can get mis-classified. Hence, a change of state is considered only when the model consecutively shows a new state for 5 samples of incoming data. The inferred labels cannot be validated in real time. However, when the periodic monitoring of equipment is performed, the ground truth is available and the inference error can be re-calculated. As with sensor configuration, the rate at which sensors send features is configurable and can be tuned to minimise network and energy use while respecting application latency requirements.

Incremental Training. The transmission of the FFT during data collection requires more energy than the transmission of features for inference. Once a model is deployed, data collection is still performed for continuous training of the model. However, during this phase, the full FFTs are transmitted intermittently along side the extracted features. We define the number of full FFT transmissions among the total number of transmissions in a given time window as the *training duty cycle*. In the data collection stage, the duty cycle is set to 1 and is gradually reduced in the inference stage to improve energy efficiency while still retraining the model. However, when the test accuracy of the trained model fails to meet application requirements or the inference error is above threshold, the duty cycle may be increased. In this way, FaultBit maintains the trade-off between device lifetime and model accuracy while also allowing models to evolve to changing environmental conditions. Unlike deep learning models mentioned earlier which classifies faults based on additional input data such as humidity and temperature, FaultBit incrementally learns the pattern of data factoring in deviations resulting from environmental changes. The life-cycle of a FaultBit application is shown in Fig. 4.

Algorithm 2: Deployment Model Selection

Input : Set of starting models M_{start}, train and test accuracy requirements Tr_{th}, Te_{th}, FFT data D, labels D_L, classes C
Output: Chosen model for deployment m_{sel}, Set of trained models $M_{trained}$

Initialize trained model set $M_{trained}$ and deployable model set M_{dep} to *null*
Set $n_{features} := 1$
Function *aggregate_query_translation(signal_range, feature_resolution)* **is**
  while *(*$|D| > 0$ *&* $|D_L| > 0 \forall c \in C$*) &* *(—*$M_{start}| > 0$*)* **do**
    features := $n_{features}$ from D
    if *(any feature is null)* **then**
      break;
    end
    Train models from M_{start} over D
    if *(*Acc_{train} *for* $m_i \geq Tr_{th}$*)* **then**
      $M_{trained} \leftarrow m_i$
      Remove m_i from M_{start}
    end
    if $|M_{trained}| > 0$ **then**
      Calculate Acc_{test} $\forall$ $m_i \in M_{trained}$
      if $Acc_{test} > Te_{th}$ **then**
        $M_{dep} \leftarrow m_i$
      end
    end
    `// Select model with sufficient test accuracy and fewest features`
    if $|M_{dep}| > 0$ **then**
      Sort M_{dep} in ascending order by number of features
      $m_{sel} \leftarrow m_0$ from M_{dep}
    end
    `// No model suffices test accuracy requirement; pick model with best test accuracy`
    else
      Sort M_{train} in descending order of Acc_{test}
      $m_{sel} \leftarrow m_0$ from M_{train}
    end
    $n_{features} += 1$
  end
  return m_{sel};
end

4 Implementation

The FaultBit toolkit is implemented on a representative embedded IoT device; the Nordic nRF52840 MCU [31], which offers a 64 MIPS processor, 512 kB RAM, 2MB flash and built in Bluetooth Low Energy (BLE) radio. For vibration

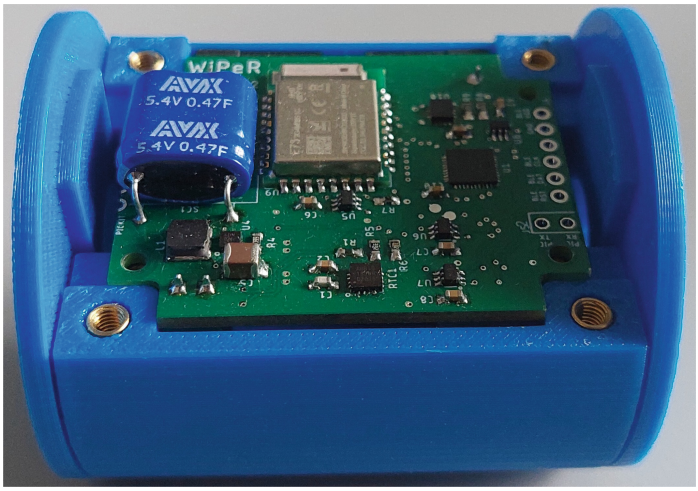

Fig. 5. Device for monitoring bearing faults implementing the FaultBit toolkit

measurements in the bearing monitoring use-case, the accelerometer sensor Kionix KX132-1211-e [19] is used, while the current sensor LEM CTSR 0.3-P is used for the solenoid valve monitoring use-case [20]. The IoT device is powered by a 3 V 620 mAh battery. All energy measurements were performed using a precision desktop multi-meter in a lab environment. The representative node used for deployment in the bearing monitoring application is illustrated in Fig. 5. We employ a class of supervised algorithms to classify the sensor data as faults including Support Vector Machines (SVM), Random Forest (RF) and neural networks. The models are trained using the sklearn Python library while a Flask server is deployed to perform incremental training and inference as discussed in Sect. 3.3.

5 Evaluation

In this section we evaluate the ability of FaultBit to detect and classify faults in two different and representative fault detection tasks; vibration monitoring of motor bearings and current monitoring of solenoid valves.

Vibration Monitoring of Bearing Motors: The vibration monitoring data used in the experiments are acquired from the Case Western Reserve University (CWRU) bearing data set [6]. The data was obtained at a sample rate of 12KHz for three fault conditions (inner race, outer race, and ball fault) and four rotation speeds (1730, 1750, 1772 and 1797 RPM). The bearing motors are first classified to be healthy and faulty. The faulty bearing motors are further classified into 6 classes based on the health of the motor and the root causes of failure namely,

no defects, defects on balls, defects on inner race, defects on outer race: centered, orthogonal and opposite to the load zone.

Solenoid Monitoring: The data from the solenoid monitoring use-case is provided by an industrial partner. The data collection was performed over a solenoid endurance test setup to monitor degradation of 48 closed solenoid valves. The degradation was performed by switching the valves at a rate of 1 Hz with an input voltage of 110 V AC at 50 Hz over a duration of 6 weeks. The solenoid valves undergo three stages in its lifetime, healthy, faulty and End of Life (EOL) which represent the three labels for the data. We assume that data transfer is performed with a standard BLE packet size of 32 bytes for all experiments. The training of the models are performed by randomly dividing the data using the holdout method into 75% training data and 25% validation/testing data. The target test accuracy for both application is set to 99%.

5.1 Accuracy of Fault Analysis

As discussed in Sect. 2.3, FaultBit utilizes generic feature extraction techniques and adapts to the application specific requirements when performing on-device feature extraction. The on-device feature extraction pipeline illustrated in Fig. 3 is used for both types of sensor data.

Vibration Monitoring of Bearing Motors: The accuracy of the trained models are shown in Table 2 for the bearing motors. The columns represent the number of features used to train the model, 'F' representing the peak frequency of the feature while 'A' representing the amplitude at that frequency. It is observed that FaultBit is able to classify the bearing motors as faulty or healthy at 100% accuracy with only 2 features. Furthermore, reusing the same FFT bins, FaultBit classifies the faults to 6 classes by the health or root cause of the fault extracting the required number of features from the FFT. The accuracy for this classification is shown in Table 3. As can be seen from the tables, FaultBit is extremely accurate in classifying faults and their root cause despite requiring the transmission of very little data.

Solenoid Monitoring: FaultBit classifies the data from the solenoid valves with the accuracy shown in Table 4. The accuracy of classification for the solenoid valve data using a state-of-the-art Convolutional Neural Network (CNN) is 99% [38]. FaultBit is able to classify solenoid faults close to the same accuracy with only 20 extracted features. In this case, since the test accuracy threshold is not satisfied, the model with the highest accuracy is chosen and deployed for inference.

5.2 Energy Efficiency

FaultBit functions in two phases, (i) the data collection and training stage and (ii) the inference stage. In the data collection and training phase, for each frame of data, the full FFT is transmitted to the server for training the model. This

Table 2. Accuracy of supervised models trained on bearing motor data to classify if a bearing motor is healthy/faulty with varying number of frequency/amplitude features and $Te_{th} = 99\%$

	1F,1A	1A	1F
Fine Gaussian SVM	**100**	97.2	100
Wide NN	**100**	97.3	97.2
Random Forest (RF)	**100**	96.5	100

Table 3. Accuracy of supervised models trained on bearing motor data to classify the root cause of fault in the bearing motors with varying number of frequency/amplitude features and $Te_{th} = 99\%$

	10F	1F,1A	1A	1F
Fine Gaussian SVM	97.8	96.1	92.6	90.4
Wide NN	99.2	97.1	91.2	82.8
Random Forest (RF)	**99.7**	97.3	91.3	93.9

Table 4. Accuracy of supervised models trained on data collected from solenoid valves to classify valves into three health classes with varying number of frequency/amplitude features and $Te_{th} = 99\%$

	10F,10A	10A	10F	6F,6A	4F,4A	2F,2A	1F,1A	1A	1F
Fine Gaussian SVM	98.1	98.5	91.2	98.1	97.4	97.5	96.5	95.3	73.4
Wide NN	98.4	98.4	91.0	98.2	97.6	97.6	96.6	95.3	73.2
Random Forest (RF)	**98.7**	98.6	92.2	98.5	97.7	97.4	95.0	92.7	75.8

maximises the data available for training at the cost of battery life of the device due to heavy transmission loads. Once a model is trained and is ready for deployment as discussed in Sect. 3.3, the duty cycle of data collection is reduced to only collect data for re-training the model periodically while transmitting only features required for classifying the incoming data.

Figure 6 illustrates the battery life of the device when monitoring for each use-case. We use a nRF52840 processor for this experiment with BLE being the mode of communication between the device and server. The average consumption for the device in the data collection and training phase is 33 µA for solenoid monitoring and 10 µA for bearing motor monitoring. The data collection phase starts with duty cycle = 1 and is reduced when inference phase starts. For example, in the inference stage for a duty cycle of 4%, only 1 full FFT is transmitted for training for 24 set of features transmitted for inference. Hence, as the duty cycle of transmission is reduced the life-time of the device improves drastically in both use-cases as shown in Fig. 6. Reaching the desired level of accuracy in the solenoid monitoring application requires computation of the FFT over more number of windows to the cover the entire frequency spectra resulting in higher

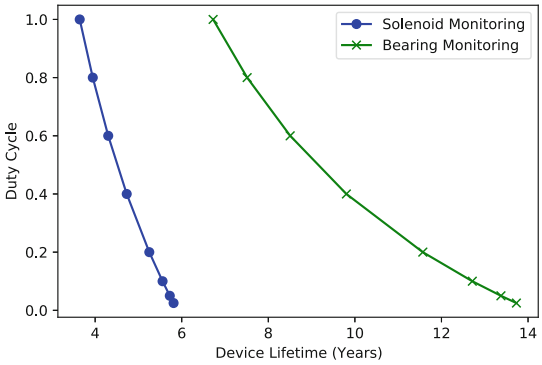

Fig. 6. Device lifetime for fault diagnosis with varying duty cycle of data collection deployed with 620 mAh battery

data transmission in comparison to bearing motor monitoring. This translates to an increased lifetime of the device for the bearing monitoring use-case.

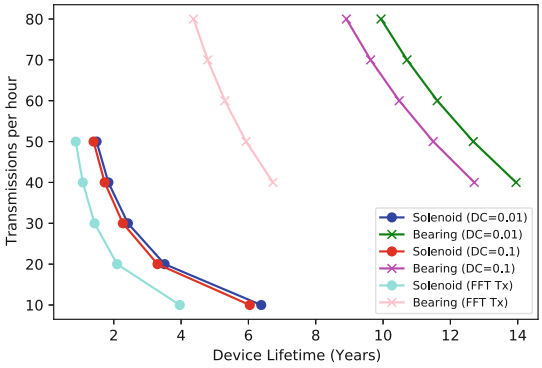

Fig. 7. Device lifetime for fault diagnosis with varying transmissions per hour for fixed duty cycles deployed with 620 mAh battery

We can also illustrate the difference in lifetime caused by transmission of the data at differing intervals for a fixed duty cycle as shown in Fig. 7. The difference between each plotted line in a particular use-case shows the improved lifetime that can be achieved for the same number of transmissions per hour and differing duty cycles. The lifetime significantly drops when we transmit only full FFTs as performed in the data collection and training stage throughout the device lifetime. The transmission of raw data only causes device lifetime to drop far below 1 year and is not suitable for representation in this plot.

5.3 Network Efficiency

The efficiency of network usage is illustrated by comparing the transmission of only the features required to train models and classify data against the raw data transmission as used to train deep learning models over IoT networks.

Vibration Monitoring of Bearing Motors: The accelerometer sensor sampled at 1024 Hz and a 512 byte FFT produces an output of 5 FFT frames per sampled signal. The entire FFT for each sample amounts to 1280 bytes, while the extracted features amounts to 9 bytes of transmission for 2 features. In the data collection phase we see a reduction of 68.75% in network usage while a reduction of 99.8% for the inference phase.

Solenoid Monitoring: The current sensor down-sampled to 5120 Hz and a 512 byte FFT produces an output of 37 FFT frames per sampled signal. The entire FFT for each sample amounts to 9472 bytes of transmission while the extracted features amount to 40 bytes of transmission for 10 features. In the data collection phase we see a reduction of 53.75% in network usage while a further reduction of 99.9% for the inference phase. These readings show that FaultBit is extremely efficient for the network under test compared to sending raw data. During the inference phase, this would enable orders of magnitude more devices to be supported before network bandwidth is exhausted. At the current time, we do not know what re-training duty cycle will be required in real-world deployments. Investigating this is the subject of our future work.

5.4 Discussion

In contrast to traditional methods, which involve the selection of a subset of features from a given pool, FaultBit compresses the data into a collection of features that possess both minimalistic and universal features, enabling their utilization by multiple models. This allows FaultBit to select the most suitable model from a collection of models, while simultaneously minimizing the use of features. FaultBit has been evaluated on two distinct datasets: one derived from an electrical current signal and the other derived from an accelerometer signal. The results show that FaultBit achieves high accuracy comparable to deep learning models, while transmitting only a fraction of data from the IoT device. Overall, FaultBit is a promising approach for fault classification on IoT devices. It is particularly well-suited for low-power and low-bandwidth networks, where it can achieve high accuracy while minimizing data transmission.

6 Related Work

6.1 Machine Learning-Based Industrial IoT

Machine learning techniques have been widely used for industrial IoT applications [23]. Data-driven approaches in fault detection bypass the need for feature extraction from the raw sensor data. The sensor data itself is provided as an

input to a deep neural network where the features are extracted and the signal is classified. The authors of [27] propose a two-layer artificial neural network to classify faults in bearings. Deep neural networks such as [18] utilize three to five layers for fault diagnosis in roller bearings with improved accuracy. Authors of [10] proposed deep neural network-based approaches for segmentation and classification tasks in IoT sensor streams with no or limited labels. In [34], the authors presented a model incorporating ML for low-power transmission among IoT devices using LoRa. Specifically, they introduced a Channel State Information (CSI) learning mechanism in LoRa edge devices, which are then used to control the entire network's power. In [33], Soother et al. used a soft real-time fault diagnosis system for edge equipment based on domain adaptive training strategy using long- and short-term memory networks for remote and real-time detection of bearing faults from vibration data to reduce faults. Many of these approaches are complementary to FaultBit, in the sense that they could be integrated into the collection of available back-end models. However, none of these approaches can be directly applied to the problem of industrial fault detection using resource-constrained IoT devices.

6.2 Fault Detection in IoT Networks

Data compression techniques collect raw sensor data and perform feature extraction on-device to reduce network load. Thangarajan et al. [37] proposed such an approach to compress accelerometer sensor data to perform bearing fault diagnosis while improving battery life and reducing latency. The authors of [39] perform data reduction on a magnetic sensor and an accelerometer on-device for bearing fault diagnosis in a permanent magnet synchronous motor. This results in a 95% reduction in transmitted data and hence optimizes energy usage of the device. IoT devices are resourceful enough for feature extraction on the device [4] along with fault identification on-device itself [17]. This results in a massive reduction of energy expenditure due to data transmission. However, the data resolution is also affected and traditional fault detection algorithms cannot be enabled using this paradigm [14]. The devices are tailored to an application specific deployment which makes it quite static over its lifetime [13,36]. FaultBit addresses this problem supporting multiple applications at run-time through continuous data collection and model training.

7 Conclusion and Future Work

This paper proposed FaultBit, an application independent toolkit for fault detection using IoT devices and machine learning. Faultbit enables deep reconfiguration of the IoT system to enable efficient data collection and compression followed by classification over bandwidth limited IoT networks. Evaluation illustrates that FaultBit performs well on two diverse use-cases, achieving accuracy close to raw data, while offering multi-year battery life and significant network bandwidth savings.

In the future, we will extend our evaluation of FaultBit to support more sensors by conducting empirical research in other fault types and datasets with uni-dimensional data sharing a common timeline. FaultBit requires labelling to be performed via a human-in-the-loop approach. In future work, we will explore (i) methods to weakly label the data using unsupervised learning approaches, (ii) automation of model switching based on model performance while preserving the energy consumption and limited bandwidth usage of the current approach and (iii) methods to output a time-window in which the fault is predicted to occur, i.e. predict the RUL of devices. Since FaultBit models are incrementally trained, they are susceptible to over-fitting. We currently address this issue by selecting models which are resistant to over-fitting. In future work, we will consider using more complex model structures or introducing regularization methods and other techniques address the problem of over-fitting.

References

1. Rolling element bearing diagnostics—a tutorial. Mech. Syst. Signal Process. **25**(2), 485–520 (2011)
2. LSM6DS3: iNEMO inertial module always-on 3D accelerometer and 3D gyroscope (2016). ST Microelectronics. https://www.st.com/resource/en/datasheet/lsm6dsl.pdf. Accessed 20 Oct 2019
3. Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., Watteyne, T.: Understanding the limits of LoRaWAN. IEEE Commun. Mag. **55**(9), 34–40 (2017)
4. Afanasov, M., et al.: Battery-less zero-maintenance embedded sensing at the Mithræum of circus maximus. In: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 368–381 (2020)
5. Baert, M., Rossey, J., Shahid, A., Hoebeke, J.: The Bluetooth mesh standard: an overview and experimental evaluation. Sensors **18**(8), 2409 (2018)
6. CWRU Bearing Data Center: CWRU vibration dataset with faulty bearings. https://engineering.case.edu/bearingdatacenter/download-data-file
7. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
8. Chesnes, J., Kolodziej, J.: An application based comparison of statistical versus deep learning approaches to reciprocating compressor valve condition monitoring. In: Annual Conference of the PHM Society, vol. 13 (2021)
9. Chi, Y., Dong, Y., Wang, J., Yu, F.R., Leung, V.C.: Knowledge-based fault diagnosis in industrial Internet of Things: a survey. IEEE Internet Things J. (2022)
10. Chowdhury, T.: Towards reducing labeling efforts in IoT-based machine learning systems: Ph.d. forum abstract. In: Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021), pp. 416–417 (2021)
11. Dolui, K., Thangarajan, A.S., Claes, T., Michiels, S., Hughes, D.: Towards on-board learning for harvested energy prediction. In: Proceedings of the 6th International Workshop on Embedded and Mobile Deep Learning, pp. 7–12 (2022)
12. Dujovne, D., Watteyne, T., Vilajosana, X., Thubert, P.: 6TiSCH: deterministic IP-enabled industrial Internet (of Things). IEEE Commun. Mag. **52**(12), 36–41 (2014)

13. Dutta, P., Grimmer, M., Arora, A., Bibyk, S., Culler, D.: Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In: Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, pp. 497–502. IEEE (2005)
14. Entezami, M., Roberts, C., Weston, P., Stewart, E., Amini, A., Papaelias, M.: Perspectives on railway axle bearing condition monitoring. Proc. Inst. Mech. Eng. Part F: J. Rail Rapid Transit **234**(1), 17–31 (2020)
15. Gardner, W.A., Napolitano, A., Paura, L.: Cyclostationarity: half a century of research. Signal Process. **86**(4), 639–697 (2006)
16. Hoang, D.T., Kang, H.J.: A survey on deep learning based bearing fault diagnosis. Neurocomputing **335**, 327–335 (2019)
17. Hou, L., Bergmann, N.W.: Novel industrial wireless sensor networks for machine condition monitoring and fault diagnosis. IEEE Trans. Instrum. Meas. **61**(10), 2787–2798 (2012)
18. Jia, F., Lei, Y., Lin, J., Zhou, X., Lu, N.: Deep neural networks: a promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data. Mech. Syst. Signal Process. **72**, 303–315 (2016)
19. Kionix: KX132-1211 tri-axis digital accelerometer specifications. https://kionixfs.azureedge.net/en/datasheet/kx132-1211-e.pdf
20. LEM: Current Transducer CTSR series. https://www.lem.com/sites/default/files/products_datasheets/ctsr_0_3-p_ctsr_0_6-p_v8.pdf
21. Li, H.X., Yang, J.L., Zhang, G., Fan, B.: Probabilistic support vector machines for classification of noise affected data. Inf. Sci. **221**, 60–71 (2013)
22. Li, J., Liu, L., Liu, J., Green, R.: Building diversified multiple trees for classification in high dimensional noisy biomedical data. Health Inf. Sci. Syst. **5**, 1–10 (2017)
23. Li, L., Ota, K., Dong, M., Borjigin, W.: Eyes in the dark: distributed scene understanding for disaster management. IEEE Trans. Parallel Distrib. Syst. **28**(12), 3458–3471 (2017)
24. Mao, W., Liu, Y., Ding, L., Safian, A., Liang, X.: A new structured domain adversarial neural network for transfer fault diagnosis of rolling bearings under different working conditions. IEEE Trans. Instrum. Meas. **70**, 1–13 (2021). https://doi.org/10.1109/TIM.2020.3038596
25. Mazaev, G., Ompusunggu, A.P., Tod, G., Crevecoeur, G., Van Hoecke, S.: Data-driven prognostics of alternating current solenoid valves. In: 2020 Prognostics and Health Management Conference (PHM-Besançon), pp. 109–115. IEEE (2020)
26. ST Microelectronics: LSM6DSOX datasheet. https://www.st.com/resource/en/datasheet/lsm6dsox.pdf
27. Prieto, M.D., Cirrincione, G., Espinosa, A.G., Ortega, J.A., Henao, H.: Bearing fault detection by a novel condition-monitoring scheme based on statistical-time features and neural networks. IEEE Trans. Ind. Electron. **60**(8), 3398–3407 (2012)
28. Ran, Y., Zhou, X., Lin, P., Wen, Y., Deng, R.: A survey of predictive maintenance: systems, purposes and approaches. arXiv preprint arXiv:1912.07383 (2019)
29. Roshanmanesh, S., Hayati, F., Papaelias, M.: Utilisation of ensemble empirical mode decomposition in conjunction with cyclostationary technique for wind turbine gearbox fault detection. Appl. Sci. **10**(9), 3334 (2020)
30. Sadler, C.M., Martonosi, M.: Data compression algorithms for energy-constrained devices in delay tolerant networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006, pp. 265–278. Association for Computing Machinery (2006)
31. N Semiconductor: NRF52840 product specification v1.1. https://infocenter.nordicsemi.com/

32. Shao, C., Nirjon, S.: ImageBeacon: broadcasting color images over connectionless Bluetooth LE packets. In: Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, IoTDI 2017, pp. 121–132. Association for Computing Machinery (2017)
33. Soother, D.K., Ujjan, S.M., Dev, K., Khowaja, S.A., Bhatti, N.A., Hussain, T.: Towards soft real-time fault diagnosis for edge devices in industrial IoT using deep domain adaptation training strategy. J. Parallel Distrib. Comput. **160**, 90–99 (2022)
34. Suresh, V.M., Sidhu, R., Karkare, P., Patil, A., Lei, Z., Basu, A.: Powering the IoT through embedded machine learning and LoRa. In: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 349–354. IEEE (2018)
35. Tang, S., Yuan, S., Zhu, Y.: Cyclostationary analysis towards fault diagnosis of rotating machinery. Processes **8**(10), 1217 (2020)
36. Thangarajan, A.S., Yang, F., Joosen, W., Hughes, D.: Deterministic 40 year battery lifetime through a hybrid perpetual sensing platform (hyper). In: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8 (2020)
37. Thangarajan, A.S., Yang, F., Joosen, W., Michiels, S., Hughes, D.: ReFrAEN: a reconfigurable vibration analysis framework for constrained sensor nodes. In: 2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 124–131 (2021). https://doi.org/10.1109/DCOSS52077.2021.00033
38. Tod, G., Mazaev, G., Eryilmaz, K., Ompusunggu, A.P., Hostens, E., Hoecke, S.V.: A convolutional neural network aided physical model improvement for AC solenoid valves diagnosis. In: 2019 Prognostics and System Health Management Conference (PHM-Paris), pp. 223–227 (2019). https://doi.org/10.1109/PHM-Paris.2019.00044
39. Wang, X., Lu, S., Huang, W., Wang, Q., Zhang, S., Xia, M.: Efficient data reduction at the edge of industrial internet of things for PMSM bearing fault diagnosis. IEEE Trans. Instrum. Meas. **70**, 1–12 (2021)
40. Zhang, B., Sconyers, C., Orchard, M., Patrick, R., Vachtsevanos, G.: Fault progression modeling: an application to bearing diagnosis and prognosis. In: Proceedings of the 2010 American Control Conference, pp. 6993–6998. IEEE (2010)
41. Zhang, S., Zhang, S., Wang, B., Habetler, T.G.: Deep learning algorithms for bearing fault diagnostics-a comprehensive review. IEEE Access **8**, 29857–29881 (2020)
42. Zhang, W., Peng, G., Li, C., Chen, Y., Zhang, Z.: A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals. Sensors **17**(2) (2017). https://doi.org/10.3390/s17020425. https://www.mdpi.com/1424-8220/17/2/425
43. Zhao, Y., et al.: Towards battery-free machine learning and inference in underwater environments. In: Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications, pp. 29–34 (2022)
44. Zheng, H., Yang, Y., Yin, J., Li, Y., Wang, R., Xu, M.: Deep domain generalization combining a priori diagnosis knowledge toward cross-domain fault diagnosis of rolling bearing. IEEE Trans. Instrum. Meas. **70**, 1–11 (2021). https://doi.org/10.1109/TIM.2020.3016068