Federated Learning for Autoencoder-based Condition Monitoring in the Industrial Internet of Things

Oliver Vincent Leon Stoll

A thesis submitted to the
Faculty of Electrical Engineering and Computer Science
of the
Technical University of Berlin
in partial fulfillment of the requirements for the degree
Bachelor of Computer Science

Berlin, Germany
August 04, 2022

Main supervisor:

Prof. Dr. habil. Odej Kao, Technical University of Berlin

Second supervisor:

Prof. Dr. habil. Sahin Albayrak, Technical University of Berlin

Student registration number:

382829

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

Zusammenfassung

Im *Industrial Internet of Things* (IIoT) führt die weitläufige Verbreitung von Geräten, die mit Funktionen zur Datenerfassung und -verarbeitung ausgestattet sind, zu einer neuen Dimension von existierenden Daten. Diese neuen Daten können auf vielfältige Weise genutzt werden, um industrielle Prozesse effizienter zu gestalten, z. B. für *Condition Monitoring* (CM), eine *machine learning* (ML) Technik, mithilfe welcher durch die gezielte Überwachung des Maschinenzustands Wartungskosten erheblich gesenkt werden können. Allerdings sind diese Daten nicht immer zugänglich, da die Sammlung der Rohdaten an einem zentralen Ort zu Problemen wie der Überlastung des lokalen Netzwerkes führen kann. Um diese Probleme zu umgehen, stellen wir in dieser Arbeit eine *Federated Learning* (FL) Implementierung für CM vor, einen typischer Anwendungsbereich des ML im IIoT, um Daten zu nutzen die über mehrere Geräte in einem IIoT-Anwendungsszenario verteilt sind. Um die Modell-Instanzen des FL auf typischerweise ressourcenbeschränkten IIoT-Geräten effektiv zu trainieren, wird eine ressourceneffiziente Anpassung einer modernen CM Methode für unsere Implementierung des föderierten Lernens erarbeitet. Die Ergebnisse der durchgeführten Evaluierung auf einem IIoT-typischen Datensatz rotierender Maschinen zeigen, dass unsere Implementierung in der Lage war, den sich verschlechternden Zustand einer Maschine genauso schnell vorherzusagen wie die moderne ressourcen-unbeschränkte CM Methode. Während unsere FL Implementierung eine vergleichbare Sicherheit in ihrer Vorhersage erreicht, ist es gleichzeitig möglich sie unter der Verwendung von maximal 25% einer 2.2 GHz Einzelkern-CPU mit einem Arbeitsspeicher von unter 2 Gigabyte anzuwenden, was sie für ressourcen-beschränkte Geräte des IIoT geeignet macht. Wir sind daher der Meinung, dass die Anwendung des FL auf CM in ressourcenbeschränkten Umgebungen wie dem IIoT vielversprechende Ergebnisse in Bezug auf die Fähigkeit zur Erkennung von Maschinendefekten bei gleichzeitiger Reduzierung der Belastung möglicherweise unzuverlässiger Netzwerkverbindungen zeigt.

Abstract

In the Industrial Internet of Things (IIoT), the widespread distribution of devices equipped with capabilities for data collection and processing leads to a new quantity of existent data. This new data can be used in a multitude of ways to make industrial processes more efficient, such as for condition monitoring, a machine learning technique that can significantly reduce maintenance costs by purposefully surveying a machine's condition. However, this data is not always accessible as the collection of raw data in a centralized location can create issues by burdening the local network. To circumvent these issues, in this thesis, an approach of a federated learning implementation for condition monitoring, a typical machine learning use case in the IIoT, is proposed to utilize data distributed across multiple devices in an IIoT use case scenario. To effectively train the federated learning model instances on typically resource-restricted IIoT devices, a resource-efficient adaptation of a state-of-the-art condition monitoring method is proposed for our federated learning implementation. The results of the conducted evaluation on an IIoT-typical rotating machines data set indicate that our federated learning implementation was able to predict the deteriorated condition of a machine as quickly as the state-of-the-art condition monitoring method. While our method achieved a comparable certainty about its prediction, it showed the ability to be trained on 25% of a single core 2.2 GHz CPU using under 2 gigabytes of memory, making it applicable for resource-restricted IIoT devices. This thesis therefore finds, that the application of federated learning to condition monitoring in resource-restricted environments such as the IIoT shows promising results in its capability to detect machine faults while at the same time reducing the load on possibly unreliable network connections.

Contents

List of Figures

List of Tables

1

Introduction

Since the emergence of Cloud Computing as the big computation-as-a-utility solution in the early 2000s, a lot has changed. The amount of existent data worldwide has grown, from 2 zettabytes (10^9 terabytes) in 2010, to 64 zettabytes in 2020, with an expectancy to grow to 181 zettabytes in 2025 for a 90-fold increase within 15 years [1].

One driving factor in this development of mass data availability is the ever-more widespread distribution of devices in the Internet of Things (IoT) [1]. In the IoT, many small footprint devices are distributed in the area of our everyday life, equipped with both a multitude of sensors to collect data accessible in their location and computational resources to process and communicate the collected data [2].

Especially in the subdomain of the Industrial Internet of Things (IIoT), also known as the Industry 4.0, the ever-increasing velocity and volume of data generated by devices, combined with sophisticated analysis jobs, can benefit the operational efficiency of the manufacturing process. Here, distributed devices equipped with real-time sensors and computational power provide a plethora of functionality for their industrial setting [3], for example supervising production, measuring manufacturing performances or detecting failures in a factory. Artificial intelligence is employed to fully utilize the abundance of raw data accessible by this distribution of sensors, most commonly in a central location such as the cloud [4]. Here, machine learning (ML) models with different purposes are trained or inferred on the data extracted by the sensors of IIoT devices.

In the IIoT, a major focus of these machine learning approaches are unsupervised machine learning techniques, such as the detection of anomalies. Unsupervised learning shows great promise for the wide-scale utilization of these newfound data sources. It avoids the costly process of labelling the data, which, requiring human interaction, is often not feasible. Thereby, unsupervised learning allows for many important use cases in the industrial production which would otherwise not be viable, one of them being condition monitoring. As modern industries heavily rely on extremely expensive and complex machines for manufacturing, a major cost factor of production is the maintenance and depreciation of such machines. Where traditionally, maintenance would be executed on periodic time schedules or following evident failures, the collected sensor data can be used to allow machine learning models to supervise machines

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

and predict times of necessary maintenance. Condition monitoring, the early recognition of machine failures or abnormal behaviour, can significantly cut running costs as otherwise costly defects can be detected without domain experts' permanent supervision of the relevant monitoring data streams [5].

Along with the distribution of remote devices such as in the IIoT, possessing both sources of new data and computational capabilities, came federated learning as a new machine learning technique. With federated learning, data distributed across multiple devices is not merged in a central location like a cloud server but utilized without extracting it from the local device. Instead, multiple instances of an ML model are directly trained on the devices holding the data. After each iteration of their training cycle, only the updated weights of the models are shared and aggregated in the central cloud server [6].

Federated learning allows for the conservation of network bandwidth, as only model weights need to be exchanged, and the much larger raw data sets remain on the local devices. Additionally, privacy for the data used is directly embedded into the system, as no central server gains access to the data itself, allowing the utilization of data that would otherwise be restricted due to data privacy regulations.

With raw data in an Industry 4.0 setting possibly containing valuable information about the business logic or personalized data, network and privacy issues by communicating the data to the cloud are mitigated by the utilization of federated learning [7]. Thereby, data on remote devices in the IIoT that would otherwise not be accessible can be utilized by the application of federated learning for many use cases in the Industry 4.0.

A remaining concern with the training of machine learning models directly on IIoT devices is their inherently resource-restricted nature. IIoT devices lack the full-fledged computational resources like GPUs, high-scale CPUs or extensive memory necessary for cloud-scale training. Due to this fact and the time-critical nature of use cases like condition monitoring, research needs to find ways to utilize the given resources efficiently, as they cannot be assumed abundant [8]. Even though there has been a substantial amount of work regarding Industry 4.0 specific use cases like condition monitoring and the federated learning technique itself, there has yet to be a great deal of research combining both the effectiveness of federated learning based condition monitoring and the resource efficiency crucial for direct deployment to IIoT devices.

For that reason, our work is concerned with the adaptation of a centralized and resource-unrestricted condition monitoring baseline method to a federated learning IIoT application. To this goal, we propose a resource-efficient version of this baseline implementation, which we will then use as the machine learning method of our newly implemented federated learning framework.

The subsequent chapters of this thesis are structured as follows: First, we give an overview of the theoretical methodology, after which we present the use case scenario our work is focused on, together with the baseline method for condition monitoring. Following, the contribution of this work is presented, in which we define requirements for our implementation as well as discuss our adaptation of the baseline method and the federated learning framework. In chapter five, we evaluate our approach to determine if the defined requirements of our implementation were met, while in chapter six, we compare our approach to other state-of-the-art approaches. Finally, in chapter seven, we summarize our findings and give an outlook on further possible research.

2
Methodology

This thesis is concerned with applying federated learning to condition monitoring for a use case scenario in the IIoT. Our work's machine learning aspect deals with adapting a long-short-term-memory autoencoder for time-series-data-based condition monitoring. In the following, we will introduce the related methodologies our work is based upon.

2.1 Industrial Internet of Things

The Industrial Internet of Things, as a subset of the IoT, covers the domain of industrial machine inter-connectivity, as well as the automated supervision and application of knowledge in an industrial context [9]. In contrast to the more common understanding of the IoT, also possibly titled the consumer IoT, [10] which is centred around human behaviour and improvement of day-to-day life, the Industrial IoT aims to improve highly expensive and fast-paced manufacturing processes. By the introduction of a large quantity of interconnected IoT devices to the manufacturing process, a new level of control is gained. New data sources, such as camera images or sensors for temperature, pressure or vibration data, are not only distributed but also leveraged to monitor and supervise aspects of production to a degree which are manually not possible [11],[12]. Using distributed devices of the IoT, even in the highly optimized industrial world of today, many improvements such as further automation of production processes, improving cost efficiency, the reduction of machine downtimes or additional worker safety are still to be made.

2.2 Condition Monitoring

Condition monitoring as a machine learning use case aims to meaningfully survey a machine's condition via its operating characteristics by utilizing real-time data streams of sensors observing the machine [13]. This is conducted with the goal of determining moments when a form of maintenance or manual intervention is necessary, either to improve the machine's health and durability or its productiveness. The data streams, consist of one or multiple sensory features

such as temperature, vibration or sound. These can be analyzed using machine learning to detect unexpected behaviour in the machine, signalling a need to intervene [14]. Such points of unexpected behaviour can be viewed as anomalies in the data, making condition monitoring a subclass of the anomaly detection problem.

As the goal of condition monitoring is to reduce the need for manual labour of domain experts, supervised learning approaches requiring fully labelled data are often not feasible in real-world applications. This absence of labelled data poses a significant challenge for machine learning models used in unsupervised condition monitoring, as they are not like traditional supervised machine learning architectures able to directly learn by comparison of the ground truth of the data. To mitigate this issue, unsupervised condition monitoring requires specialized unsupervised learning architectures designed to interpret unlabeled data in a meaningful way. One such model architecture often used for unsupervised anomaly detection is the autoencoder [15].

2.3 Autoencoder

An autoencoder is a neural network architecture used to learn efficient codings of unlabeled data [16]. Limited by an adjustable coding size, it learns to encode and decode significant features of the higher dimensional data, thereby learning which features of the data are important for representation and which are not.

Architecture

The architecture of the autoencoder consists of two smaller individual models, an encoder and a decoder, only communicating through a single passing of a low-dimensional vector, the coding [17]. Its overarching goal is to take an input which is of greater size, compress it down to a considerably smaller size and then efficiently reconstruct the input out of the small-sized compression. In this context, the compression of the input to the coding is completed by the encoder, while the decoder reconstructs the coding into an estimated representation of the original input. Both encoder and decoder can consist of a number of sequentially aligned layers, with a decreasing dimension for the encoder and an increasing dimension for the decoder to support the process of compression/decompression. A graphical depiction of the generic autoencoder architecture can be seen in Figure 2.1.

A global loss function is then applied to both encoder and decoder, which compares the output of the autoencoder, the representation of the input with the original input, measuring the loss by the difference between both input and representation. Encoder and decoder are trained together on the data that is to be reconstructed and thereby learn to collaboratively represent the input as closely as possible. This is achieved by the encoder learning to extract the most relevant features of the data and encode them into the small coding, while the decoder learns to interpret the features represented by the coding correctly and to approximate the input as closely as possible based on these most relevant features [18].

As a well-sized coding does not nearly offer enough space to encode all information about each data point, less relevant and unreliable features such as natural occurring noise are effectively being ignored by the autoencoder's approximation of its input, and only important features to the data are retained [19]. This feature extraction characteristic of autoencoders is

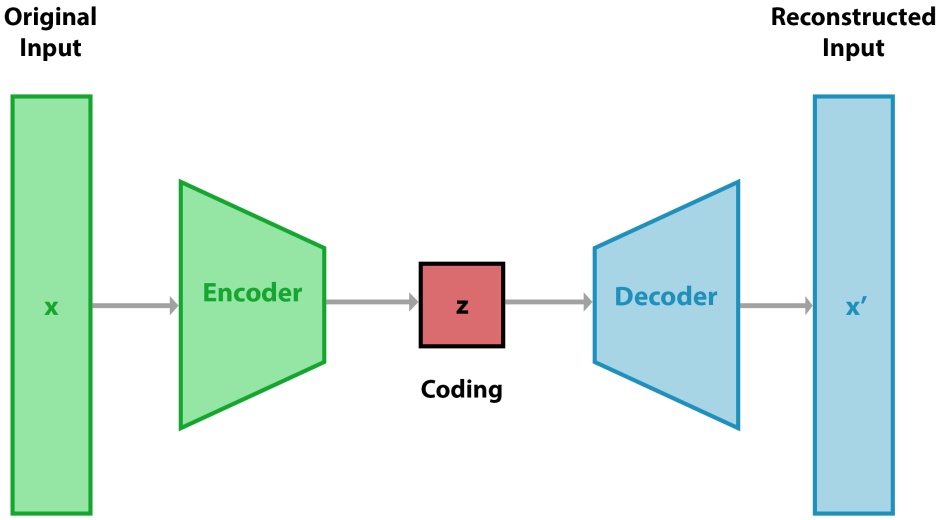

Figure 2.1: The architecture of an Autoencoder showing the input, coding and reconstructed output.

beneficial and a reason for the architecture seeing widespread use, as no labelled data is needed. To train an autoencoder efficiently, one just needs to provide data that is underlying a general structure the encoder and decoder can learn to represent.

Anomaly Detection

This correct interpretation of data features and, therefore, the reconstruction of the input with a low margin of error, also called reconstruction error, only works on the data the autoencoder was trained to recognize. Other types of data possess different features that the autoencoder does not know to interpret and therefore cannot adequately reconstruct without a sizeable reconstruction error. This discrepancy between well-known data with a low reconstruction error when fed into the autoencoder and yet unknown types of data with different characteristics and features, resulting in a high reconstruction error, is how autoencoders can be utilized for anomaly detection [20].

When applying an autoencoder for anomaly detection, the data on which the autoencoder is trained is assumed to be highly disproportionate in its data classes. Anomaly detection data sets typically consist of one big class of data that is considered normal behaviour and one or multiple small data classes, that represent the different anomalies that could occur. Where in many machine learning use-cases, this imbalance would be detrimental to the learning process, for unsupervised anomaly detection it is essential, as the autoencoder is trained on the data with the assumption that through this imbalance, it learns better to represent the 'normal behaviour' data class than the underrepresented anomaly data classes. This way, the autoencoder can be trained without any label. Only mostly normal data needs to be provided.

New data can now be used as input for the trained autoencoder, which is then reconstructed with varying degrees of reconstruction errors. Here, most normal data will show a baseline value of reconstruction errors as the approximation through the coding is never perfect, while

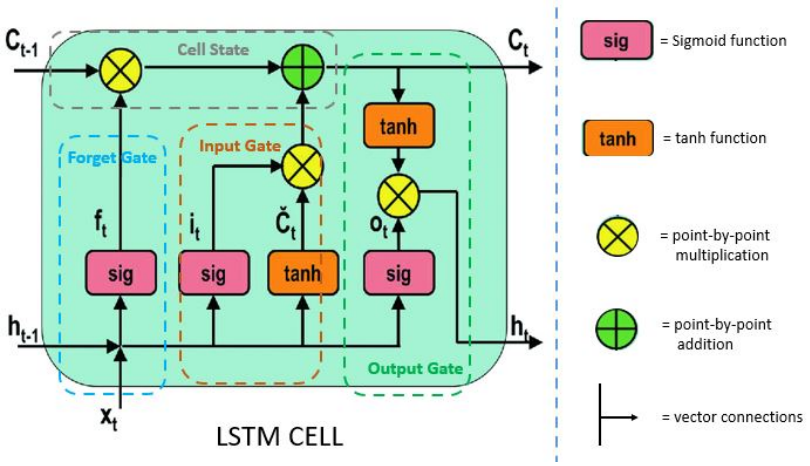

Figure 2.2: Architecture of an LSTM cell showing the different gates, as well as internal operations [1].

abnormal data will, with a properly trained autoencoder, show reconstruction errors in the manifolds of this baseline. An anomaly threshold can now be determined, either manually or in some other suitable way, which should aim to exclude most normal behaviour while still including all anomalous reconstruction errors. This threshold value or factors in its determination are hyper-parameter open for optimization. All data points with a reconstruction error higher than this anomaly threshold can subsequently be deemed detected anomalies of the autoencoder model.

2.4 Long Short Term Memory

As our work deals with the topic of condition monitoring on real-time data, there is a need for a neural network architecture that can comprehend one major source of information in real-time data: time correlation. Where traditional neural networks often struggle to understand the causal one-way relationship of temporal data, recurring neural networks (RNN) are specifically designed to mimic passing time [21]. One advanced variant of such RNNs, which we will be utilizing in our use-case as integral parts in both the encoder and the decoder to utilize knowledge about the temporal aspect of the data, is the long-short-term memory (LSTM) layer.

LSTM layers consist of multiple cells, which are as typical in RNN networks set in sequence, where each cell receives one piece of the input and the result of the last cells' computation. These cells each have three different inputs, the cell state, the hidden state and the original input. The cell state is passed through all cells, every time only slightly being altered by the combination of hidden state and input and by that acts as a form of long-term memory in the LSTM layer [22]. The hidden state is the output of the previous LSTM cell, passed into the current cell to be taken into consideration. This way, a direct short-term memory typical for RNNs is established. Last, the original matching input is received by the cell, where it is combined with the hidden state. The input can also consist of a data vector in the case that the data consists of multiple features.

These three inputs are passed through three different gates, small neural networks, mirroring steps in the human memory process, also depicted in Figure 2.2.

Forget Gate

First, in the forget gate, input and hidden state, further only called the combined input, are passed into a sigmoid neural network, which produces an output vector of values between 0 and 1. This output is then multiplied with the cell state, causing some parts of the cell state to be 'forgotten' depending on the combined input. This way, previously relevant information made obsolete by some new input can be discarded.

Learn Gate

Secondly, in the learn gate, the combined input is again fed into a sigmoid neural network, which produces a vector that represents which values of the cell state are to be updated. Following, a tanh function prepares possible value candidates for updating the cell state as a vector of values between -1 and 1. These are then point-wise multiplied with the sigmoid vector of values between 0 and 1, representing the updates strength. By this method, all cell state values can be fully updated. Using the learn gate, new understandings of the data by the LSTM layer can be preserved in the long-lived cell state to serve for future iterations.

Output Gate

Lastly, in the output gate, the cell state, transformed by a tanh function, is taken as the basis of the output. Again, the combined input is used by a sigmoid neural network to output a vector which is then point-wise multiplied with the tanh-compressed cell state. This filtered and compressed vector is then used as the output of the LSTM cell and propagated as the hidden state for the next cell in the LSTM layer. The output represents the interpretation of the LSTM layer of this exact data point.

2.5 Federated Learning

Federated learning is a machine learning technique that aims to build a joint machine learning model based on data located across multiple sites [24]. Multiple instances of the model are locally trained on the data sites, collaborating using their training results in the form of the current model state. These model states can be shared among the data sites, describing distributed federated learning, or with one central location tasked with the aggregation of all training results describing the more widespread centralized federated learning [25].

The collection of different model instances trained on different subsets of the data will then be collectively aggregated by an aggregation algorithm to construct one joined global model. Multiple training iterations can be performed by re-initializing the local model instances with the newly learned state of the global model to train in a continuous training cycle. Federated learning is thereby able to fully train a model on all subsets of the data without the need to share the original data from any of the data sites. In Figure 2.3 [23] we can see an illustration of one iteration of the training cycle of centralized federated learning.

[1]Singhal, G., Introduction to LSTM Units in RNN — Pluralsight. [online] Pluralsight.com. Available at: ⟨https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn⟩ [Accessed 3 August 2022].

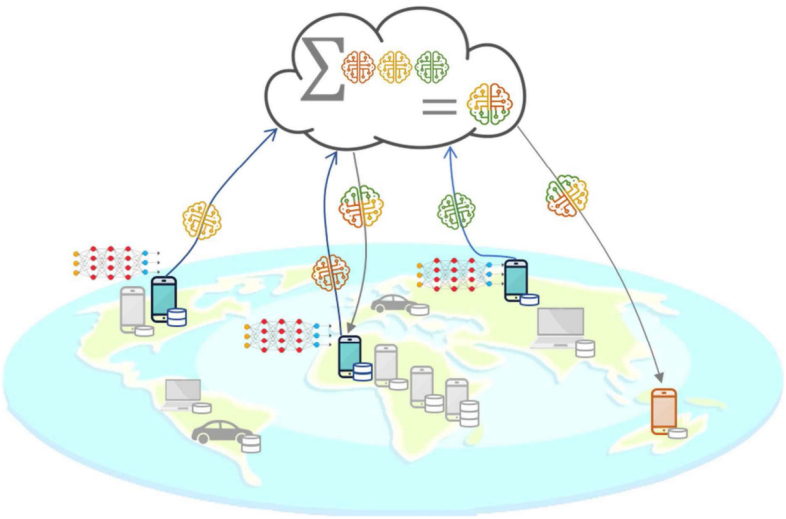

Figure 2.3: Depiction of the life-cycle of federated learning, showing local model training, aggregation and re-distribution [23].

In this thesis, we will focus on the application of centralized federated learning. Therefore, for the sake of simplicity, centralized federated learning will be referred to as federated learning.

3

Use Case

In the following chapter, we describe the use case our work is based upon. Furthermore, we present an IIoT-typical data set of rotating machines used for our evaluation. Finally, we introduce a state-of-the-art condition monitoring implementation for rotating machines that we use as a baseline method. This baseline method is the condition monitoring method our contribution aims to adapt for an application in the IIoT.

3.1 Use Case Scenario

The use case our work will be focused upon is condition monitoring for rotating machines in an Industry 4.0 scenario. A typical scenario for this would be a manufacturing plant containing multiple rotating machines of similar build, such as engines, pumps or turbines. In such a scenario, condition monitoring is applied with the goal of reducing costs and outages by minimizing preventable maintenance work, which is achieved by detecting meaningful opportunities for maintenance beforehand.

To monitor the condition of all rotating machines involved, sensors are installed that collect real-time data about the machines' current states by measuring significant features. For rotating machines, due to their constructive design, the most informative feature often proves to be their vibration, measured by accelerometers attached to or installed in the machines [26]. This real-time data is then gathered by IIoT devices, often by one IIoT device per rotating machine due to the proximity of the sensors, before being further processed. The gathered data of all rotating machines is then collected in a central location via an upload from the IIoT devices, where a centralized machine learning model learns and infers on the data to monitor the machine's condition and determine meaningful opportunities for maintenance. A graphic of the here described use case scenario is depicted in Figure 3.1

In the context of this initially centralized scenario, our work aims to circumvent the global upstream of raw data to the data aggregation server by deploying a federated learning model to every IIoT node instead. This local training and aggregation of model weights instead are depicted in Figure 3.2.

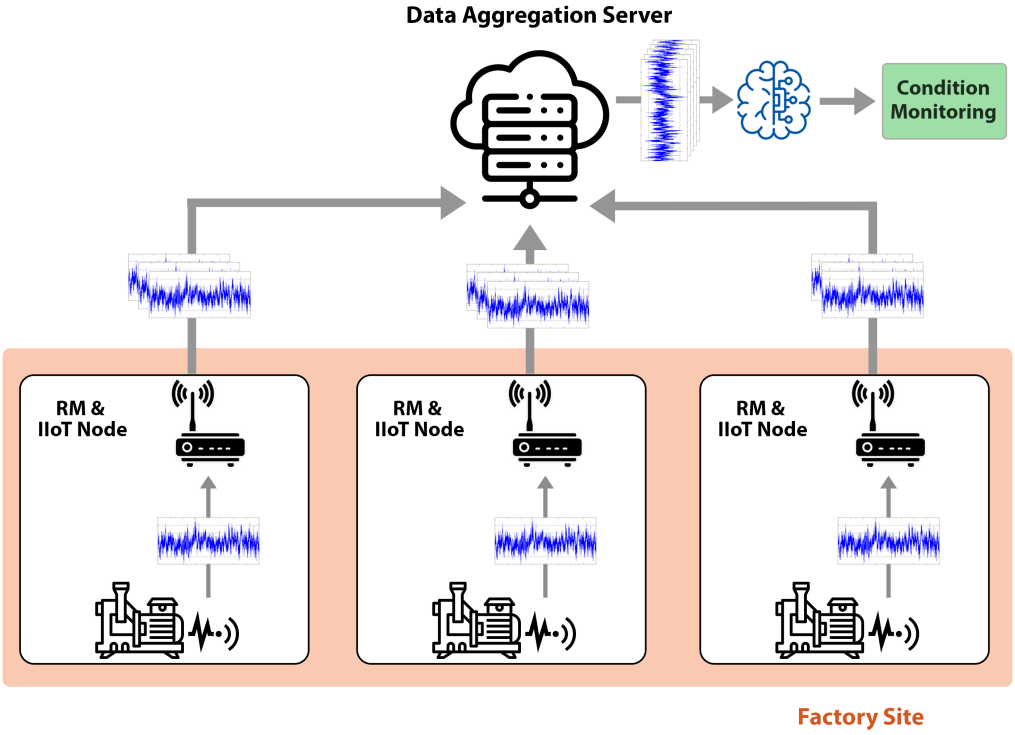

Figure 3.1: Graphic showing use case scenario, using a centralized condition monitoring approach with a global data aggregation and model training.

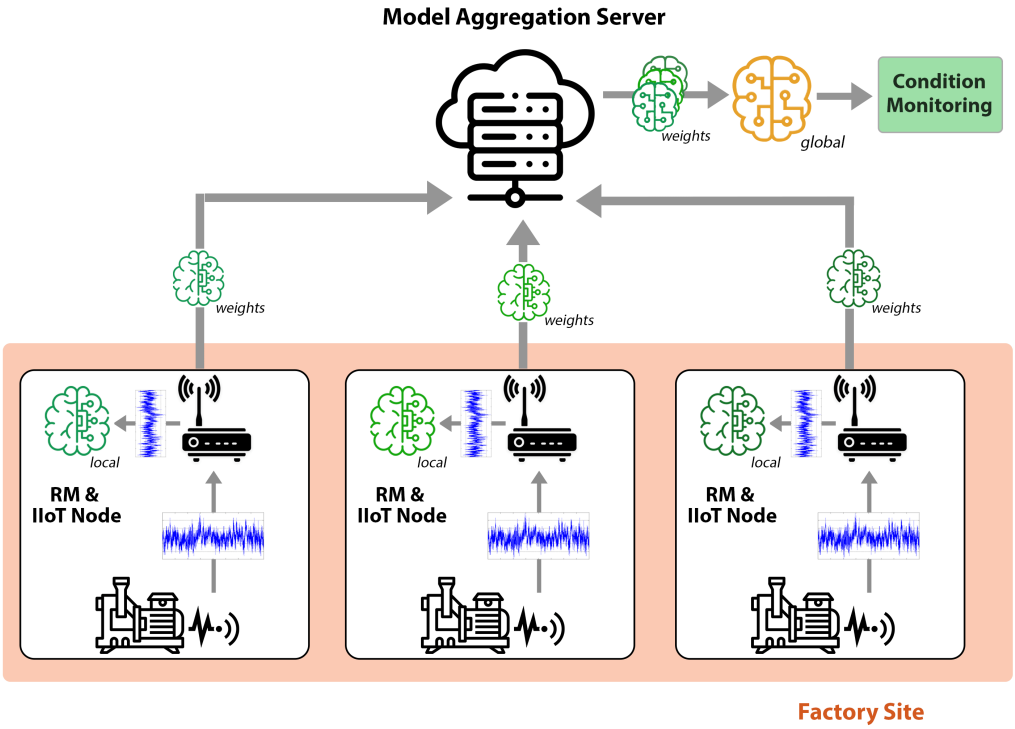

Figure 3.2: Graphic showing use case scenario, using a federated learning condition monitoring approach with localized training and a global model aggregation.

3.2 Data Set

The IMS bearing vibration data set [27], also known as the NASA bearing data set due to its affiliation with the NASA prognostics centre, is an anomaly detection data sets for rotating machines. In our further work, we will use this data set to evaluate our work it representing the use case scenario described in the previous chapter. As anomaly detection data sets, in general, consists of mostly normal data along with a number of anomalies.

The data set persists of vibration data gathered in three run-until failure experiments. In all three experiments, four bearings were mounted on one rotating axis while accelerometers were placed on each bearing, measuring its vibrations during the experiment. The bearings were then exposed to a radial load of 6000lbs (approx. 2700 kg) while the axis was rotated at a constant 2000 RPM by a motor. The experiments were then run until the failure of at least one bearing. As stated by the description offered by the authors of the experiment, this happened after the designed lifetime of 100 million revolutions for each bearing was exceeded. An overview of the number of measurement periods and bearing faults of the experiments is provided in Table 3.1.

During the experiment, the accelerometer was used to measure 1 second of raw vibration data every 10 minutes, with a frequency of 20480 Hz. Different failures were detected at the later stages of each experiment. Observing the raw data from the experiments using mean- and peak amplitude, some general observations can be made:

- ◇ With the occurrence of a defect in a bearing, the vibrations measured on the defect bearing increased
- ◇ Defects consequently expand over time, starting with a small irregular increase of measured vibrations and then escalating into multiple manifolds of original vibrations readings
- ◇ Defects on one bearing also affected the vibration measurements in the other bearings mounted on the same rotating axis. During the later stages of one bearings defect, the measurements of all bearings also showed irregularities in their vibration data, even though less substantial. This can be explained by the shared rotating axis the bearings are mounted on, transmitting additional vibrations from the defect bearing.

The data set is, by design, an unlabeled data set suitable for an IIoT use case scenario regarding unlabeled data. The ground truth provided by the data set only describes which bearing shows a fault and that these faults appear towards the end of the experiment, but not at which time exactly. As we cannot directly base our work on ground truth labels, it appeared sensible to utilize a baseline method previously shown to perform well on this exact data set for developing and evaluating our proposal.

3.3 Baseline method

To use as a baseline method for the application of condition monitoring for rotating machines, we will use a proposed method by S. Ahmad et al.[28]. This baseline method is a pure machine-learning focused proposal that aims to apply unsupervised condition monitoring to rotating machines, regardless of its use case setting.

Experiment	Measurement Periods	Sampling Rate [Hz]	Faults
Experiment 1	2156	20480	B3 & B4
Experiment 2	984	20480	B1
Experiment 3	6324	20480	B3

Table 3.1: Overview of different experiments of IMS bearing vibration data set.

The method is a good fit for our goal of condition monitoring in a distributed IIoT scenario, as it does not require explicitly labelled data of the anomalies present in the data set. On the contrary, the method proposes an LSTM autoencoder used for anomaly detection to implicitly learn the behaviour of normal vibration data by extracting the most relevant features. This fulfils our requirement for an unsupervised method for our use case, which aims to remove the need for domain experts. Furthermore, the method is able to utilize the autoencoder architecture without the usage of any pre-processing, feature selection or manual feature engineering. This further simplifies the application to our use case scenario, as the design of an efficient feature extraction process also requires direct knowledge of the data domain, thereby being less feasible.

To achieve this autoencoder-based condition monitoring, the baseline method trains the LSTM autoencoder on unlabeled time-series vibration data, where it learns to reconstruct the input data as best as possible. The so-trained model is then used to reconstruct data, calculating a reconstruction error from the difference between reconstruction and the real input. The baseline method evaluates every measurement period of the data in conjunction with multiple preceding measurement periods, a procedure creating a look-back period of twenty measurement periods. This look back-period helps in understanding the general trend of the machine's condition, as anomalies typically arise and develop over time.

The reconstruction error on a validation split is then used to determine a threshold, which can be used to classify all data points by their reconstruction error into either normal or abnormal behaviour. The assumption this method works under is that behaviour deviating strongly from the formerly learned normal behaviour with a high probability signifies an unhealthy state of the machine. The overarching goal of condition monitoring is thereby fulfilled, as classified anomalies are used as a signal of the need for maintenance.

In our following contribution, we aim to improve on this baseline method by implementing an IIoT-adapted version of this method to be used in a resource-restricted environment.

4
Contribution

To further the development of federated learning approaches for use cases in the IIoT, we aim to create and integrate a real-world condition monitoring method into an IIoT use case scenario. For a more detailed description of the use case scenario, refer to Chapter 3.1.

In the following chapter, our contribution is structured into three parts: First, we define a set of requirements for the application of a machine learning method to the IIoT using federated learning. Secondly, we describe our adaptation of a previously proposed LSTM-Autoencoder condition monitoring method for application in the Industry 4.0. Our adaptation is based on a state-of-the-art anomaly-detection-based condition monitoring method presented in Chapter 3.3. Thirdly, we create a federated learning framework with which we will apply this adapted condition monitoring method to our IIoT use case scenario.

4.1 Requirements

For our condition monitoring method to be applicable in a federated learning framework in the IIoT environment, it needs to be trainable on small footprint IIoT devices. As federated learning trains instances of the model on the remote data source itself, in our case the devices that collect the sensor data from their machines, training has to take place on these devices, which are traditionally severely limited in terms of computational resources. Therefore, the training process needs to be resource efficient, both in terms of CPU and memory usage [7]. Additionally, considering the heterogeneous environment of the IoT, our implementation should be easily deployable no matter the underlying device operating system or preinstalled dependencies.

For our proposal to be considered a success, we aim for our federated learning implementation to achieve condition monitoring results comparable to the results of the baseline method we are referencing. As our work is focused on the transfer of a previously effective method to a new, more challenging environment, we do not aim to achieve better results than the established baseline method but results that mirror its effectiveness as close as possible. Finally, we aspire for our federated learning implementation to achieve a generalized understanding of the rotating machines it will be monitoring. As federated learning creates a global model that is not the result of direct training but an aggregation process, a better separation of the models understanding from the data itself is possible and desirable [29]. Therefore, we include this

generalizability into the requirements for our federated learning infrastructure to be considered a success.

Considering the sensible deployment of this use-case to an Industrial 4.0 setting regarding its inherent properties, we subsequently summarize the following requirements:

- ◇ **Resource Efficiency:** Involved IIoT devices are not overburdened by the additional resource demand from continuous training
- ◇ **Model Effectiveness:** The condition monitoring results of our federated learning implementation are comparable to the results the centralized baseline method achieves
- ◇ **Generalizability:** The federated learning model gains a generalized understanding of normal and abnormal behaviour across all instances of the same machine type
- ◇ **Deployability:** The implementation is deployable to any IoT device, regardless of the heterogeneous nature of such devices environments

4.2 Implementation

For applying the use case of autoencoder-based condition monitoring to our Industrial 4.0 use case scenario, we first must obtain a suitable implementation of a method for the said use case. In this section, we will discuss the adaptation and implementation of the machine learning method, which will handle the actual condition monitoring. Our work will adapt a previously established state-of-the-art condition monitoring method, as described in Chapter 3.3, to improve its resource efficiency. This centralized adaptation method will also serve as a secondary comparison method for later evaluation of our federated learning implementation. Our work is implemented in Python, an open-source high-level programming language, with the use of TensorFlow, an open-source python library widely used for machine learning applications.

The code related to this work is publicly available on GitHub [1].

4.2.1 Architecture

Following the proposal of the baseline method, our adaptation of it will also be an LSTM autoencoder, which aims to learn the normal behaviour of all features in the data. The two smaller models of the autoencoder, the encoder and a decoder model, both consist of a variable number of sequentially aligned LSTM-Layers. As the autoencoder will be applied to time-series data, LSTM-Layers are used to provide the models with the capability for a general understanding of the sequential aspect of time-series data [30]. The autoencoder architecture allows for multidimensional input, including different features, such as vibration data on multiple axis or further temperature or pressure readings.

Together, the encoder and decoder learn to represent and reconstruct a given measurement period of vibration data points from our data set, limited by the variable small size of the coding, also referred to as the hidden layer, through which the encoder passes a representation of the data to the decoder. Contrary to the baseline method, our implementation does not utilize a look-back period of multiple preceding measurement periods but only the current measurement

[1] https://github.com/OliverStoll/bachelor_implementation

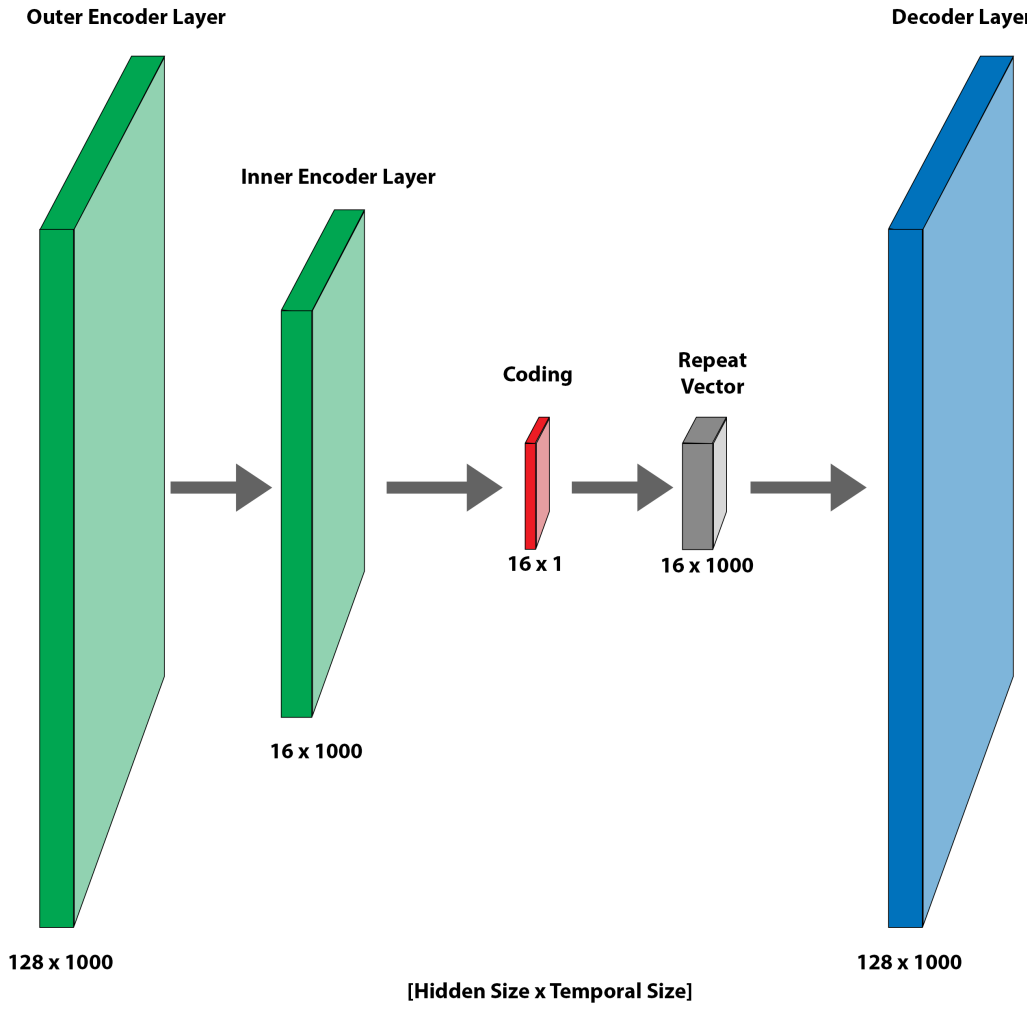

Figure 4.1: Model architecture of our adapted method with number and size of LSTM layers as determined by hyper-parameter optimization.

period. This change was adapted as the utilization of a look-back period has a significantly higher memory usage than the evaluation of only a single measurement period. Internally, the LSTM layers of both encoder and decoder pass their entire output sequence to the following layer to provide them with their full interpretation of the data. For the encoder, a final inner LSTM layer encodes the last LSTM-layers output into the reduced coding size. For the decoder, the output of the last LSTM-layer is passed into a final time-series layer using a dense layer, which formats this information to mirror the inputs' format for direct comparability.

For the specific architecture, the baseline proposed two outer LSTM layers for both the encoder and the decoder of the size of 600 and 250 cell state units respectively, the order of the decoder layers being reversed. Our work experiments with different layer sizes and amounts than the baseline method, using hyper-parameter tuning as described in Chapter 4.2.4.

Furthermore, in our adapted architecture, each layer except the readout layers uses the Rectified Linear Unit (ReLU) activation function, which shows promising results in computational efficiency as only positive values lead to the activation of neurons, whereas negative values are capped by a true zero [31]. Additionally, we use a small L2 regularizer value of $1e^{-7}$ to regu-

larize activity and the kernel of all layers in the model. This regularisation penalizes the model for using excessively large weights or learning every aspect of the data. The kernel regularization is applied with hopes of preventing the exploding gradients problem, whereas the activity regularization incentivizes the model to sparse learning.

For testing, the prediction of the model is compared against the original input to compute the reconstruction error (RE). This RE will be used to evaluate the normality of the related time sequence in comparison to others. To determine anomalies in the data, an anomaly threshold is defined for both methods, which will then be used to discriminate between normal and abnormal behaviour.

4.2.2 Data

For training and evaluation of our models, we use the IMS bearing vibration data set, previously described in Chapter 3.2.

For training, only a partial subset of the data set is used to exclude anomalies from the learning process. This is done as the core goal of the autoencoder-based method is to learn the normal behaviour of features in the data. Including anomalies in the training data could hinder the models from discriminating normal from abnormal behaviour, as it would learn to interpret and reconstruct abnormal behaviour correctly as well. Anomalies are, by definition, less common than the average behaviour. Therefore the models could still learn to distinguish both but would be severely impeded in their accuracy.

As we work with an unlabelled data set of run-until-failure experiments, we assume that the first 70% of the experiments measurement periods contain a minimal amount of anomalies for our split of training data, as abnormal behaviour caused by machine failure is less likely to occur at the beginning of the designed lifetime of the tested bearings. In a real-world condition monitoring setting, it would be feasible to ensure model training only takes place on normal behaviour. One could either generally restrict training only to a sensible portion of the life expectancy of the monitored machines, or one could externally monitor the machines to ensure no anomalies occur and revert the training process done if anomalous behaviour occurred.

In the following, we describe how the data was processed differently than by the baseline method we are adapting. The data, including the training split, is down-sampled from 20480 to 1000 data points per measurement period. This is done for multiple reasons: Firstly, to achieve a reduction of existing noise in the data. Measurements of higher sampling rates are prone to recording more data and thereby include more noise, as most relevant information is already existent in the lower dimensional sampling rate of the feature. The applied layers themselves cannot easily distinguish between relevant and irrelevant information, therefore hindering the training process. Secondly, we use down-sampling due to its reduction in computational costs. As resource efficiency is a defined requirement for our implementation, and learning the inherent time correlations of each measurement using LSTM layers is computationally complex, a reduction of the data dimensionality offers a valuable cutback of computational complexity for our training.

To compute the down-sampling of the data, we divide each measuring period into 1000 buckets each containing 204/205 data points. Following, the mean of each bucket is computed to gain the new down-sampled measuring period consisting of 1000 data points. This represents the vibration as a sensor would have measured it with a lower sampling frequency of 1000 Hz. Thereby, our method also gains the benefit of not requiring the availability of a high-frequency

accelerometer sensor, which makes it more easily accessible.

After down-sampling, the data is normalized to a mean value of 0 and a standard deviation of 1. This is shown to ensure a better convergence of training [32].

Finally, again in contrast to the baseline method, a sliding window technique is applied to the data used by our model to improve our model's understanding of the down-sampled data [33]. Here, a sliding window with a size of 100 data points or 1/10 of the measuring period is sampled with a 50% overlap, creating 20 splits for each measuring period, respectively. By this, we hope for our model to focus on understanding the general oscillations in the vibration data.

4.2.3 Training

For the process of training our model, we use the Adam optimizer as proposed by D. P. Kingma and J. Ba [34] Previous research has shown that the Adam optimizer, while being computationally efficient also achieves good results compared to other optimizers [34].

In conjunction with the previously described kernel regularization of our LSTM layers, we use gradient clipping to prevent the exploding gradient problem [35]. To prevent excessively large gradient vectors both in total and in one specific dimension, we use a clipping norm of 1 and a clipping value of 0.5, limiting the absolute norm as well as any singular value.

We use an exponential learning rate scheduler which applies a 1% reduction of the learning rate each epoch [36]. The learning rate itself will be determined by hyper-parameter tuning.

For our mini-batch size, we chose the value 64. For purposes of a consistent evaluation of resource efficiency for both memory and computational resources, we use the chosen batch size of the state-of-the-art comparison baseline method presented in Chapter 3.3. This will display our findings from a conservative perspective, as further optimization of the mini-batch size to our model would still be possible.

The loss function used for both training and evaluation of the RE is the *mean squared error* loss function. It penalizes prediction errors with an increasing scaling in severity. This incentivizes the model to prioritize minimizing large prediction outliers, learning to predict the general oscillation of the vibration data, as small prediction errors are, due to the nature of the autoencoder architecture, unavoidable.

4.2.4 Hyper-parameter Tuning

To optimize our model's performance, we use empirical hyper-parameter tuning to test the performances of different hyper-parameter combinations. For hyper-parameters to optimize, we choose the learning rate of our optimizer, as well as the number of outer LSTM layers for our model, the size of these LSTM layers and the coding size. The tuning is done by fully evaluating all possible combinations of parameter candidates. To fairly evaluate slower but possibly better converging layer combinations, we trained all models created using these parameter combinations for a total of 25 epochs. Models were then evaluated using a validation loss of a period of the data, not including anomalies.

By this method, we found that a learning rate of 0.001, as well as one outer LSTM-Layer with a size of 512 and a hidden layer size of 16, as optimal. For our training, we choose the learning rate, LSTM-Layer amount and hidden size according to these findings, an exception being made only for the layer size of the outer LSTM layer. Here we chose 128, as it achieved

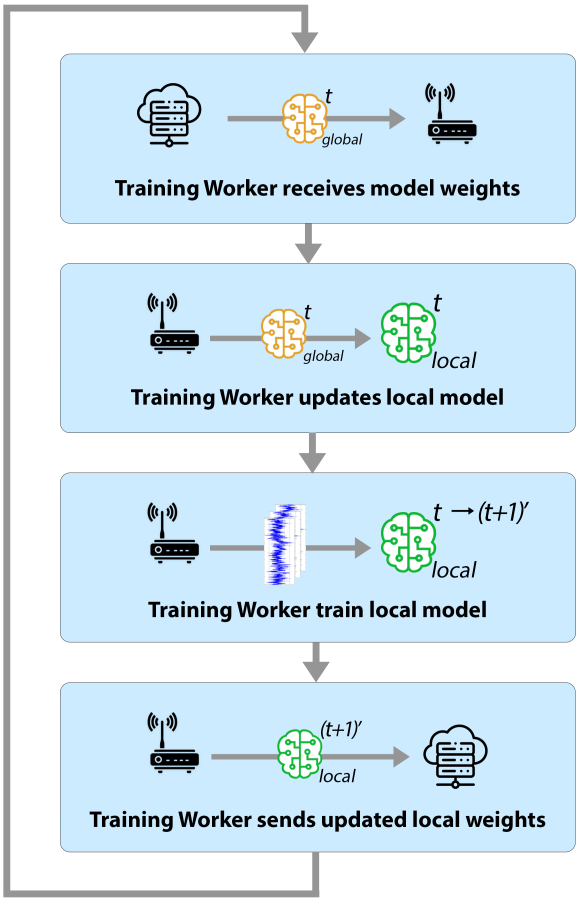

Figure 4.2: Training Cycle of a training worker instance.

almost equivalent results while being computationally more efficient. A depiction of our final model architecture can be found in Figure 4.1.

4.3 Integration to IIoT

In the third part of our contribution, we describe the implementation of the federated learning framework used for the integration of our previously single-model adaptation of the baseline method to our IIoT use case scenario. This is done by the creation of localized worker scripts, handling the distributed learning of multiple instances of the model, as well as communicating and aggregating the learning data in the form of model weights by a central aggregator instance. To this extent, we implement two worker instances, further referred to as *training worker* and *aggregation worker*, which respectively manage the localized training and the centralized aggregation of communicated model weights.

4.3.1 Training Worker

The training worker is a local instance that controls both the training process on an IIoT device as well as the communication with the global aggregation server. It mounts the directory of the local machine for utilization of its data, thereby gaining access to the data island it represents.

Here, it monitors the real-time output of the IIoT device's sensor and manages the integration of this data into the training process.

Each training worker instance is initialized with knowledge of the global aggregation server's IP address. At initialization, it registers with the server to display availability for the federated learning process. It then receives the current state of the global model from the aggregation server for model initialization. Communication between the training worker instances and the global aggregation server is implemented low-level using TCP-Sockets passing the model weights as a byte stream. We chose TCP as it does not rely on unnecessary overhead such as HTTP headers for communication but ensures the transmission of the data.

During federated learning, the training worker trains the local model following the data pre-processing and training process described in Chapter 4.2.2 and 4.2.3. After initialization and registration with the global aggregation server, the training worker starts the training cycle, which consists of four different steps, also shown in Figure 4.2:

- ◇ First, the training worker receives the new set of weights from the aggregation worker
- ◇ The training worker uses the received weights to update its current model's state
- ◇ Then, the training worker trains a single epoch on the model using the unlabeled training data
- ◇ Finally, by the training cycle updated weights of the model are collected and sent to the global aggregation server for aggregation.

Whenever the training worker instance receives new weights from the aggregation server, a new training iteration is started. As federated learning uses multiple training nodes, numerous instances of training workers exist and work in parallel, each training an instance of the same model architecture.

4.3.2 Aggregation Worker

The aggregation worker acts as the controlling instance of the global aggregation server. It manages multiple training worker instances as clients of the aggregation process, where newly connected worker instances are provided with the current state of the globally aggregated model, also referred to as the global model. After completion of each round of training, the global aggregation server receives all different model weights from the training workers and aggregates them as the next iteration of the globally aggregated model. The aggregation of the client's model weights is done via the FedAvg algorithm, as proposed by McMahan et al. [37]. It is chosen as recent research has shown the FedAvg algorithm to show good results between most state-of-the-art federated aggregation algorithms [38].

In total, the aggregation cycle comprises four different steps, which are also depicted in Figure 4.3:

- ◇ First, training worker instances receive the current version of the global model, where they train their local instances of the model.
- ◇ Secondly, the training worker instances transmit the weights of their local model instances back to the aggregation server.

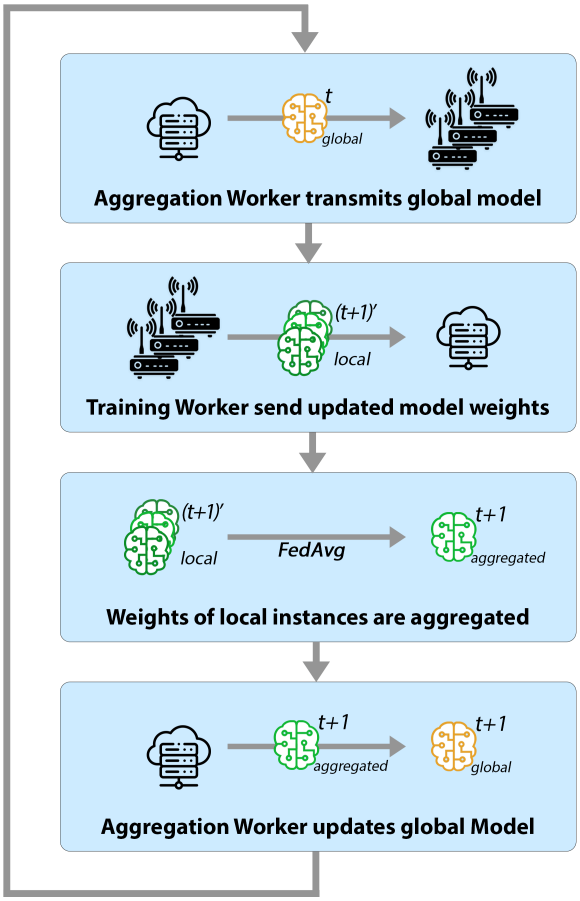

Figure 4.3: Aggregation Cycle of the global aggregation worker.

◇ When all connections have reported their weights, the aggregation worker then aggregates all weights via the FedAvg algorithm.

◇ Finally, the global model is updated, using the aggregated weights.

This way, every training worker instance begins each iteration with an identical, globally aggregated model. After training the next round, these local model instances only differ in their model weights by the difference in their training data used in this iteration. This continuous re-integration and aggregation of the local model instances allows for the general training to jointly converge against the shared global optimum observed over the entirety of the training data available on the different IIoT nodes.

4.3.3 Containerization

To realize the defined requirement of deployability independent from the underlying system environment, which is an important functionality for any IIoT deployment, container technology [2] is used to solve the issue of heterogeneous system environments in the IIoT.

As previously discussed, the context of a use-case implementation in any IoT environment brings up several difficulties. Of these, one major issue is the heterogeneity of the present

[2] https://www.docker.com/

devices. [39] One cannot assume that all IoT devices possess the same software environment, with many choices, such as the choice of the operating system often not being unified.

Therefore, to avoid the laborious and error-prone process of preparing every device's software environment to include the necessary compiler and additional software packages such as Python and TensorFlow, the worker applications were containerized.

Both training worker and aggregation worker code were each packaged together with all their necessary software requirements to create a so-called container image, also referred to as an image, which can be completely self-sufficiently integrated, no matter the operational system. Container images not only comprise all necessary software required for code execution but also provide an entire virtual environment for the executed code. By this, the full image can easily be downloaded as a singular file onto any IIoT device. Such images can then be executed as a container, an instance of the blueprint image. Containers are run in isolation, possessing their own virtual file system and network, not knowing the outside context, which can then be exposed to only the necessary ports and mount only necessary files from the local machines' file system [40].

The only requirement for this is the installation of a local daemon provided by a containerization platform. This way, the previously defined requirement of deployability can be fulfilled, as such an installation and therefore, the usage of our framework is feasible for any number of heterogeneous IIoT devices [41].

5
Evaluation

In the following chapter, we will conduct multiple experiments to evaluate our federated learning implementation against the baseline model presented in Chapter 3.3 and our IIoT optimized adapted method. Ensuring a consistent terminology, our centralized IIoT optimized adaption of the baseline model and our federated learning implementation will also be referred to as the centralized model and the federated model, respectively. To simulate a real-life use case scenario for our evaluation, the IMS bearing vibration data set presented in Section 3.2 will be used to represent an IIoT scenario where multiple rotating machines are being monitored. Both the effectiveness of the models, as well as their respective resource efficiency, will be tested to evaluate if our goal to create a feasible and effective IIoT federated learning method was achieved.

5.1 Experiments

In this section, we will describe the experiments conducted to evaluate both the machine learning aspect of our contribution, as well as the networking aspect revolving around the resource efficiency of our implementation.

5.1.1 Set-Up

To ensure clear reproducibility, as well as to guarantee unbiased measurements, we will use a test-bed of google cloud virtual machines (VM) for our experiments. Every virtual machine was a `E2-highmem-4` instance with four virtual CPUs and 32GB of memory. As Google defines virtual CPUs as one of two hyper-threading cores of a CPU, four virtual CPUs are equivalent to two CPUs using hyper-threading as described by the manufacturer [1]. The CPU's were `Intel(R) Xeon(R) CPU` with a base frequency of 2.2 GHz.

Every instance ran with Ubuntu 18.04 as the primary boot disk, with a sufficient size of 50 GB. As the experiments were conducted with swap memory, the disk size had no influence

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

on the results. Otherwise, all default settings regarding security, management and sole tenancy were kept. Docker version 19.03 was installed on every instance to manage and execute the previously built docker images for all three models. Additionally, the Google Cloud Monitoring Agent was installed to measure the CPU and memory usage on each instance. For the federated learning framework, a virtual network was established by the Virtual Private Cloud network of Google Cloud, where all instances used one network interface to communicate. The external IP address of the instance hosting the aggregation worker was fixed and passed to all training worker instances for communication.

5.1.2 Condition Monitoring

In this section, we describe our experiments to evaluate the effectiveness of both our centralized adaption of the baseline model as well as our federated model in regards to their ability to correctly detect anomalies. The evaluation will be conducted by training all three different models, the baseline model, our centralized model as well as our federated model, on the same experiment of the IMS bearing vibration data set. As previously discussed in Chapter 3.2, this data set emulates vibration machines of the use case scenario by vibration data of four bearings mounted onto a shared rotating axis.

The baseline model, as well as our centralized model, are directly trained on the training split of the data of all four bearings combined, where the federated learning framework is trained using four different training worker instances, each holding the training data of one bearing. The training itself is concluded using the hyperparameter configuration described in Chapter 4.2.3 and 4.2.4. Each model is trained for a total of 25 epochs. This relatively low number is chosen to represent realistic training on real-time data, as one epoch already represents all data points from a whole run-until-failure experiment on four different rotating machines. We deemed this number of epochs a sensible compromise between a feasible data assembly and the convergence of the models, as the waiting periods of ten minutes between measurement periods could still be reduced to accommodate such an increase in training data.

Following the training process, the trained models are used to predict the entire data sequence of all bearings. The reconstruction error (RE) is then calculated by comparing the prediction and the actual data sequences by the mean squared error loss function, which is the same loss function used during the training process. Every measurement period is assigned a collective RE value that represents the reconstruction error of the model over the entire period.

For one measurement period $\mathbf{j}$ with original input and prediction vectors $\mathbf{o}, \mathbf{p} \in \mathbb{R}^d$ where d is the dimension of a measurement period, we calculate the RE value for this measurement period as

$$RE_j = \frac{\sum_{i=1}^{d} (r_i - p_i)^2}{d},$$

The RE is applied to each measurement period and not to each data point individually, as the differentiation of normal and abnormal behaviour inside of a single measurement period of one second proves of little worth in the context of long-term condition monitoring. To put this into perspective, the run until failure experiments were conducted for a time period of several days each. Additionally, due to the noisy nature of high-frequency time series data, an evaluation of reconstruction errors of singular data points would be inherently susceptible to data outliers.

The validation split of each data stream for one bearing is then used to calculate a threshold value T for this bearings data, using the mean and the standard deviation of the RE values. For $RE \in \mathbb{R}^v$ where v is the number of measurement periods in the validation split, we calculate:

$$\overline{RE} = \frac{1}{v}\sum_{i=1}^{v} RE_i$$

$$T = \overline{RE} + \delta\sqrt{\frac{1}{v-1}\sum_{i=1}^{v}(RE_i - \overline{RE})^2}$$

As the validation split is believed to contain no anomalies, the assumption here is that a threshold value multiple standard deviations higher than the mean RE is sure to be greater than the RE value of most normal measurement periods. In contrast, to detect anomalies, we assume that abnormal behaviour in a measurement period causes a RE for this period to differ significantly from such a standard deviation interval around the mean RE of normal data. To determine a threshold which can be used to confidently predict anomalies while at the same time being noise-robust, a practical factor δ for the standard deviation offset needs to be chosen. For our experiment, we chose the factor $\delta = 4$, as it effectively excludes the statistical possibility of truly normal behaviour to show a RE value greater than the determined threshold. For comparison, the possibility of a value following a normal distribution to be greater than the mean plus four standard deviations of this distribution is about 0.003%. The threshold factor of the baseline implementation was chosen in accordance with the work of [28], as we did not want to influence the baseline performance by introducing our own thresholding strategy.

Using these threshold values compared against the RE for each bearing, the behaviour recognition and, therefore, anomaly predictions of the models on all four bearings were evaluated. Our centralized and federated model, in contrast to the look-back period utilized by the baseline model, evaluate each measurement period individually and are therefore more susceptible to noisy measurement periods. To reduce such noise in the predictions, we utilize a small rolling minimum approach of three measurement periods, as anomalies in this use case are assumed to develop over time. For every measurement period, the smaller RE of the last periods or itself was used to just predict abnormal behaviour, not quickly reverting back to normal behaviour. This period size is a trade-off between the amount of false positive predictions and anomalies being predicted in a reasonable time frame.

Additionally, we evaluate the ability of our federated learning implementation to learn generalizable normal behaviour to predict anomalies on a device's data not seen before. To test this, we train our federated model exclusively on the data of three bearings, excluding the anomalous bearing to be tested. The data of this anomalous bearing will then be tested using the same procedure described previously in this section. By this, we hope to show if our federated learning implementation is not only able to learn the normal behaviour of data streams shown to it but to extract an understanding of the type of rotating machine used to the point of broad generalizability.

5.1.3 Resource Efficiency

To evaluate our second defined requirement, resource efficiency, which is crucial to the deployment in a resource-restricted IIoT scenario, we will compare the resource usage of all three models during training. To evaluate this, we log the start and finishing times of the training

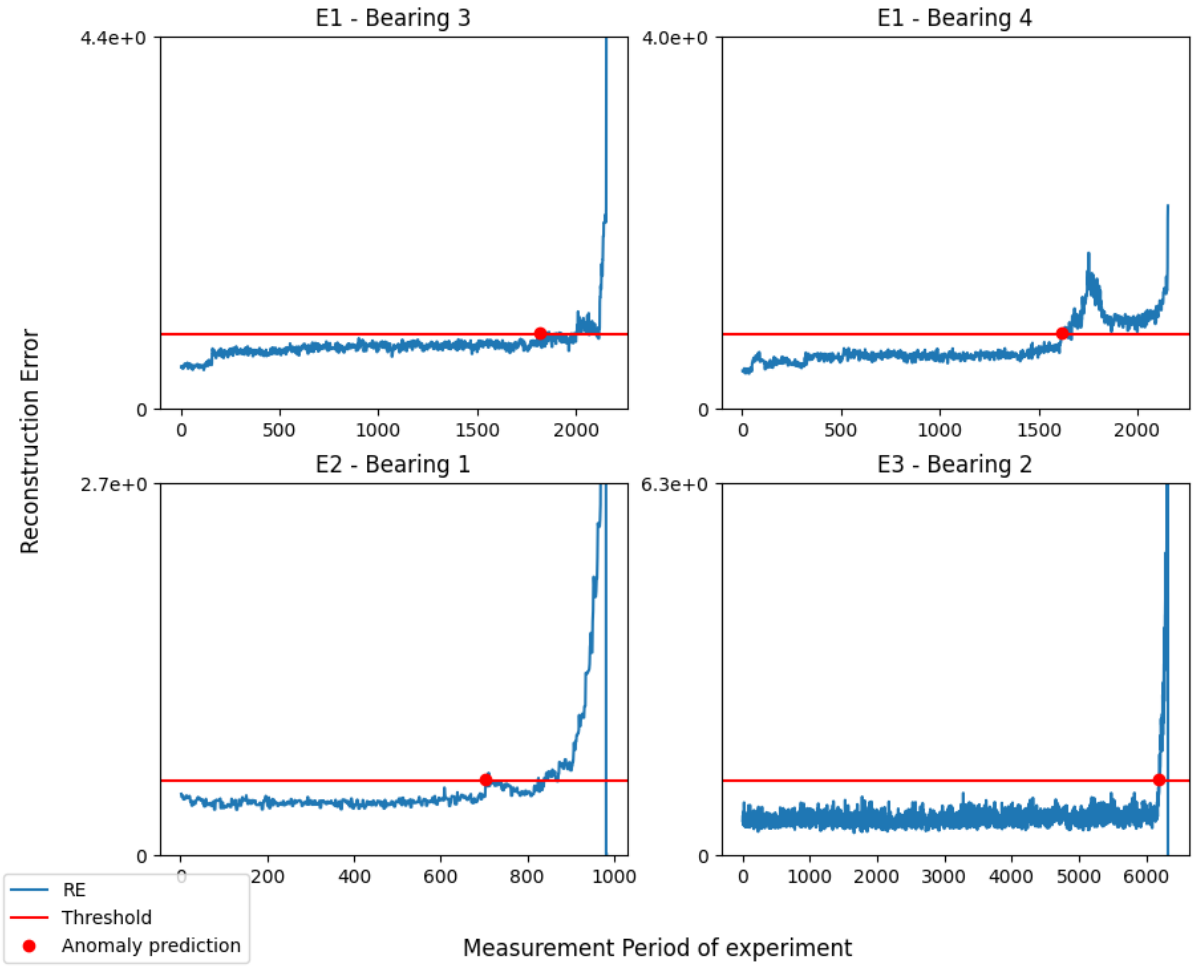

Figure 5.1: Anomaly predictions by the federated learning model for all different bearing faults.

process of all three models to determine the exact training period. In conjunction with this knowledge of the training periods for each model, we use the resource metrics of the installed Google Cloud Monitoring Agent to conduct a realistic resource usage survey of all container images hosting the model's training. Using the Cloud Monitoring Agent, we only use the resource metrics of the process on the VM handling the model training to exclude other processes on the machines, such as the docker daemon, from our model evaluation.

The average CPU usage over the run time, the mean and maximum memory allocation during the training as well as the run times themselves were compared.

5.2 Results

In the following section, we present the results of the described experiments. We will evaluate both the machine learning aspect of the work regarding the new federated learning implementation's ability to perform condition monitoring, as well as the required resource efficiency when

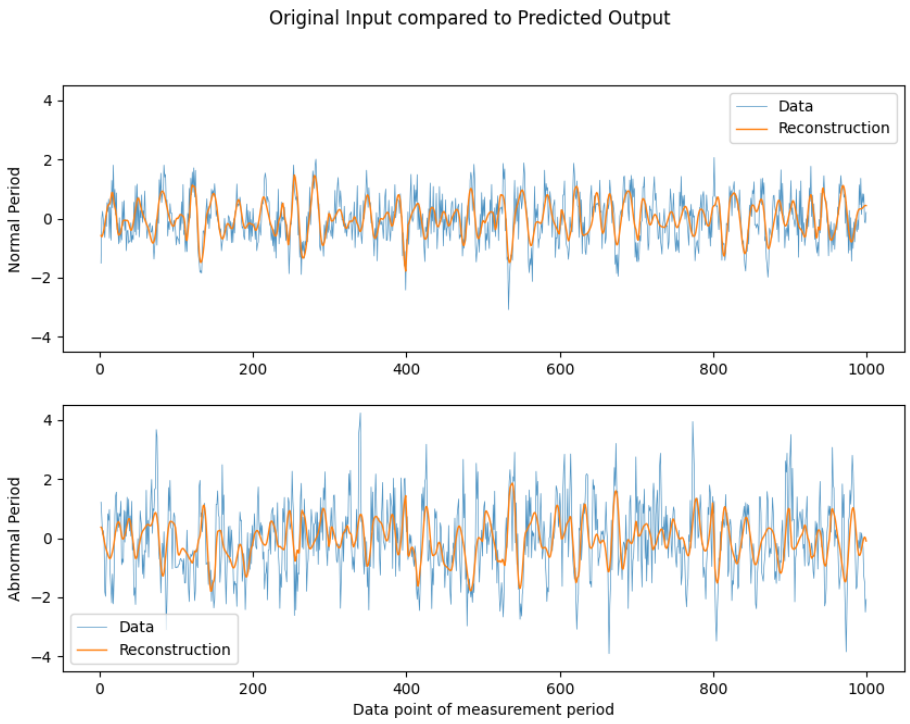

Figure 5.2: Input and reconstruction of healthy and defective bearing in experiment 2.

deploying it to an IIoT scenario.

5.2.1 Condition Monitoring

In Figure 5.1, we can see the reconstruction error values and, therefore, predictions of the condition of the bearing for all faulty bearings of the experiment compared. For every bearing, the federated learning model predicted a defect in the final stages of the experiment, with a strong increase in the RE. Simultaneously, the early stages of the experiment were predicted as normal up until the time of the detected anomaly. No clear false positives were predicted in the early stages of the experiments, for which we know that no defects were present. These predictions can be interpreted as functional condition monitoring results for bearings inhibiting a defect, to be expected for a run until failure experiment. For our following evaluation, we will focus on a single experiment, experiment 2, so as not to introduce too many dimensions into our comparison to provide better comparability.

In Figure 5.2, we visualize two measurement periods for its original data and the reconstruction, both for a late measurement period on a bearing labelled as faulty and normal. Here we find that our expectation of the way the autoencoder works was achieved for our federated learning implementation. Both vibrations are of different amplitude, with the anomalous data having a maximum amplitude of about two times the normal amplitude. The vibration in the normal period is followed approximately by the prediction of the autoencoder, while for the abnormal data, the autoencoder is not able to reconstruct the higher amplitudes successfully. This leads to a higher reconstruction error that makes a classification possible.

To gain an understanding of when the models predict the bearing to be in a faulty state, we

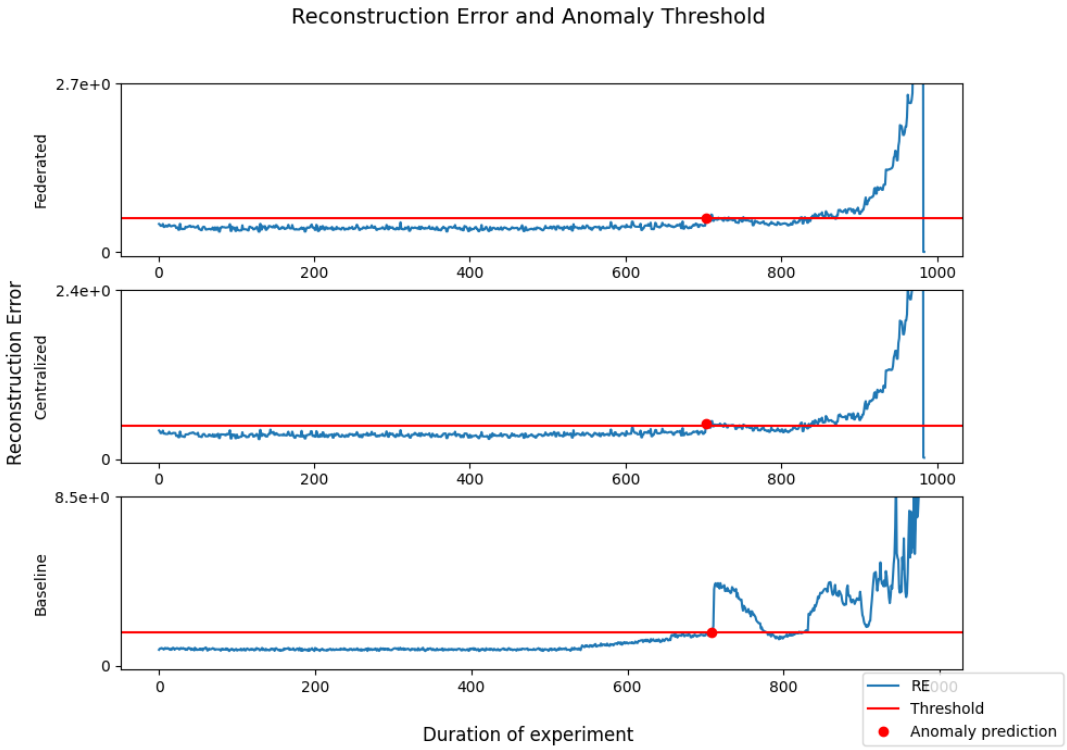

Figure 5.3: Reconstruction Error of models on the faulty bearing.

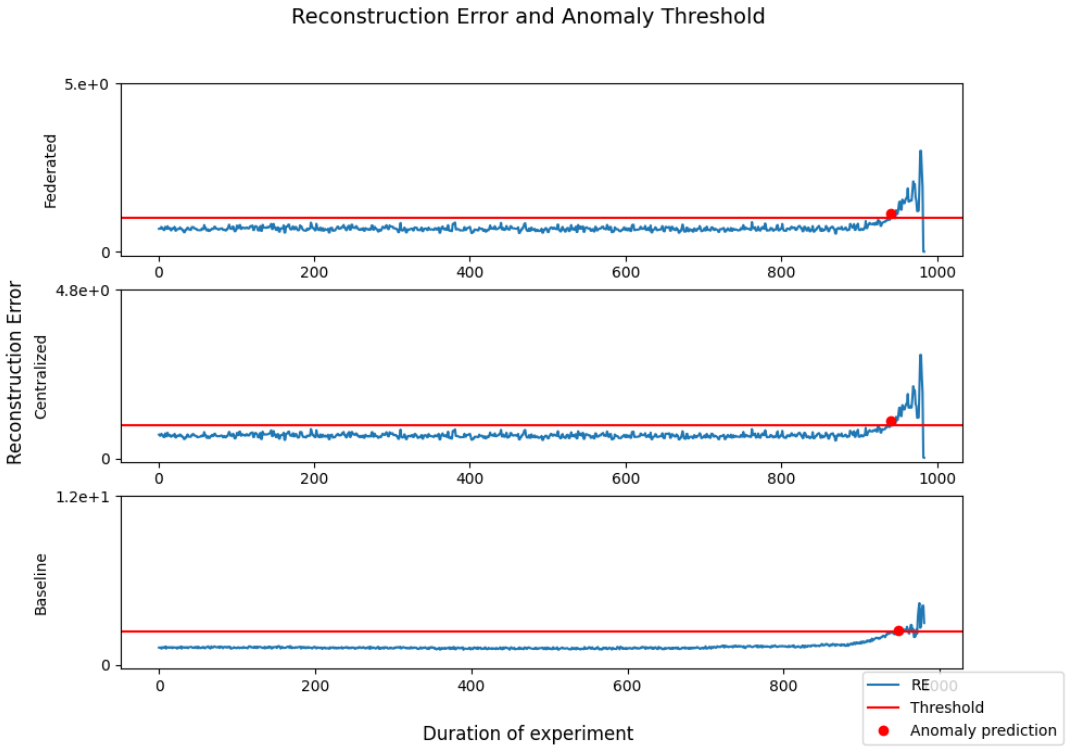

Figure 5.4: Reconstruction Error of models on a healthy bearing.

Model	Federated	Centralized	Baseline
First Anomaly Period	703	703	709

Table 5.1: Comparison of first anomaly predictions by measurement period for experiment 2.

compare the RE in relation to the calculated threshold value for all three models. In Figure 5.3 and Figure 5.4, we see the plotted values for the faulty bearing and a healthy bearing of the second experiment. The blue graph describes the RE values gained from each model. The red horizontal line indicates the threshold limit, while the red dot describes the first measurement period classified as anomalous. To ensure comparability, the y-axis of the plots was scaled to five times the threshold value, thereby showing the extent to which the RE values surpass the threshold.

In the Figures, we can see the clear distinction between the prediction of an anomaly between the faulty and the healthy bearing. All models predicted an anomaly for the faulty bearing at about the same time, with a strong increase of RE values towards the end of the experiment. In the comparison for the bearing labelled as healthy by the ground truth of the data set, we can see that all three models predict a faulty state with increasing RE values only later in the last 5% of the experiment. Notably, here is that the predictions follow the general trend of the fault predictions for the bearing experiencing the defect in a less magnified way. This circumstance could be explained by the experiment design, in which all bearings were mounted on one shared rotating axis. As noted in Chapter 3.2, this could possibly allow for the transmission of vibrations from the defect bearing to other healthy bearings.

In both comparisons, we can see that the RE values for the centralized and the federated model are very similar. This attests to the ability of our federated learning model to learn equally well from its distributed model instances on different data islands, as the centralized model was able to learn on the collected data directly.

As we can see in the comparison for the faulty bearing, the RE graph for both the federated model and the centralized model predicts the bearing to be of healthy condition up until about 72% of the duration of the experiment, where both exhibit a jump in RE values, indicating a change in data behaviour and therefore a defect. The baseline exhibits a jump in RE values at the same time as the other two models, even though more significant. But in contrast, the baseline model predicts a continuously degrading state of the bearing earlier, starting at about measurement period 550 of the experiment's duration. This earlier increase in RE values for the baseline model could be interpreted as the baseline method showing a greater sensitivity for the underlying data, which would be expected regarding its higher data complexity and utilized look-back period.

When we analyze the exact measurement period of the predicted fault shown in Table 5.1, we find both the federated model and centralized model detect an anomaly at measurement period 703 of 984, but the baseline model only predicts the fault of the machine six measurement periods later. In our experiment, this is a delay of 60 minutes, which the adapted method was able to predict the anomaly earlier. This delay in prediction can be attributed to the mentioned look-back period of 20 epochs which the baseline model used in comparison to the smaller rolling minimum window of 3 measurement periods used by the other models. We can therefore conclude that while being less sensitive to the data, our adaptation approach is able to predict strong anomalies faster by not relying on a contextual analysis of the data stream.

In Figure 5.5 we compare each model's certainty of an anomaly over the course of the

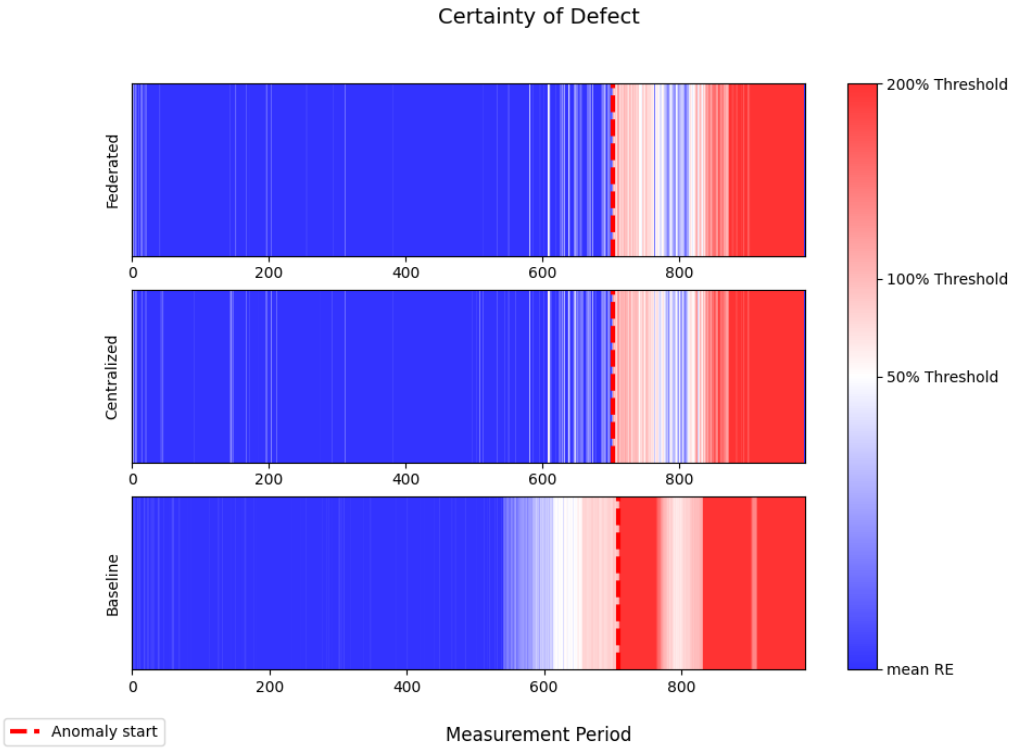

Figure 5.5: Model certainty about failure compared for faulty bearing in experiment 2.

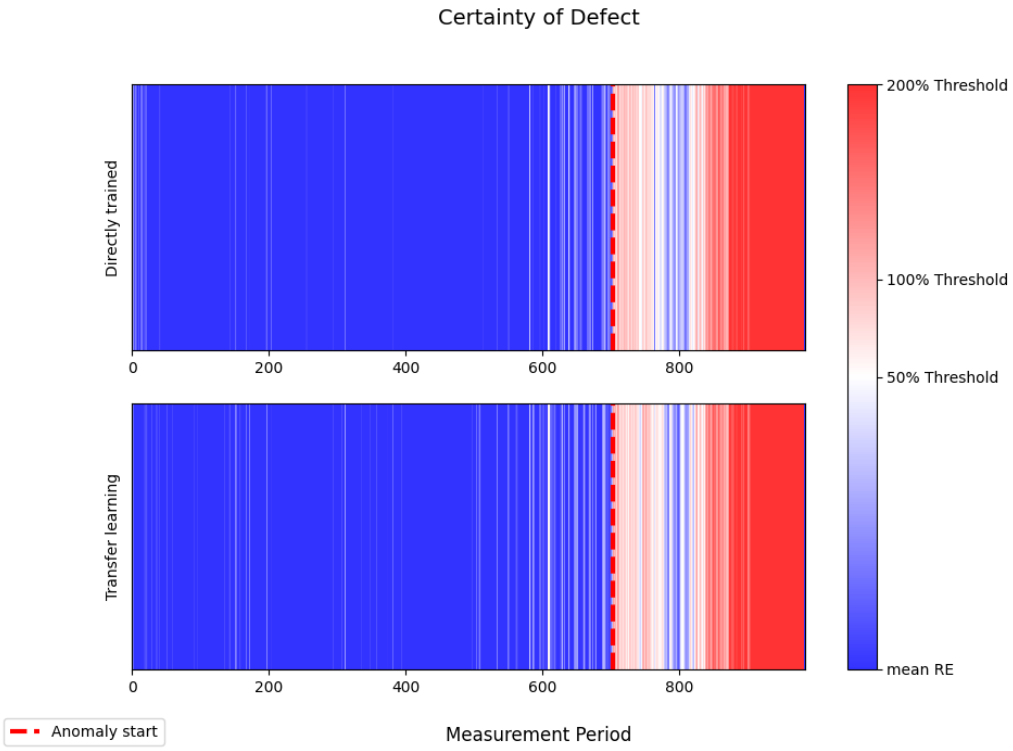

Figure 5.6: Federated learning models certainty about failure compared for the faulty bearing, one model fully trained and one model only trained on other bearings.

experiment. The certainty is measured by the difference of the RE to the calculated mean RE value. RE values only insignificantly larger than the mean are depicted in blue to show the prediction of a healthy condition. In contrast, RE values exceeding the threshold are shown in red to signal the certainty of a faulty condition.

Here we can see that the baseline model shows fewer signs of uncertainties about the bearings state during the early course of the experiment while predicting the faulty state with a higher certainty than the centralized and federated model. Between the federated and the centralized model, we can only see minimal differences, with the centralized model showing more small signs of uncertainty as well as showing a higher certainty about its prediction of the defect. Notably, both also showed an increase of uncertainty about the healthy condition of the bearing starting around measurement period 600, but only for individual measurement periods. In contrast to this, the baseline demonstrated a steadily decreasing certainty of a the healthy condition. This difference can most likely also be attributed to the contextual approach of the baseline method.

Finally, in Figure 5.6 we compare the two different versions of our federated learning model with a second one being trained on all bearings except the faulty bearing. Here we find that the transfer learned model, which was not trained on the faulty bearings' data directly, was slightly less certain about the healthy condition during the early phase of the experiments as well as about the faulty condition after it predicted an anomaly. Nevertheless, the transfer learned model was able to predict the anomaly at the same time as the directly trained model and not predict any false positives beforehand.

Therefore, we can conclude that the federated learning model is able to achieve a more generalized understanding of normal behaviour for the rotating machine in question, in this case, bearings. As in an Industry 4.0 manufacturing scenario, machines of the same built are often used for the same purpose, this marks a great practical quality of our proposed implementation.

5.2.2 Resource Efficiency

In the second part of our results, we compare the required resources for training all models in Figure 5.7. For our evaluation, the model training of the worst performing federated learning instance was compared. Therefore, it is to note that each federated model instance only trained on 1/4 of the data compared to the centralized and baseline model. For a conservative run time comparison, we assume approximately four times the run time given the same amount data. We find that for the run time of the training, the baseline model, in comparison to both other models, performed best. This result seems unexpected, as the baseline method works using a more complex data sampling rate. One explanation for this could be the fact that the baseline model does not train the lstm-layers of its autoencoder over the course of the measurement periods, but over the look-back period it utilizes, which means the LSTM layers of the baseline process entire measurement periods at a time. In terms of computational efficiency, this appears to be significantly more efficient, most likely since the layer only needs to iterate over 20 vectors instead of 1000 smaller ones.

Comparing the memory usages, our centralized and federated model inhibit a significantly better memory efficiency. In both, the data set appears to be only one factor in their memory allocation size, as the difference between a federated model with only 1/4 of the data uses about 15% less memory than the centralized model. For the baseline method instead, memory usage is a strong factor, with it differing from the federated & centralized model by a factor of ten to

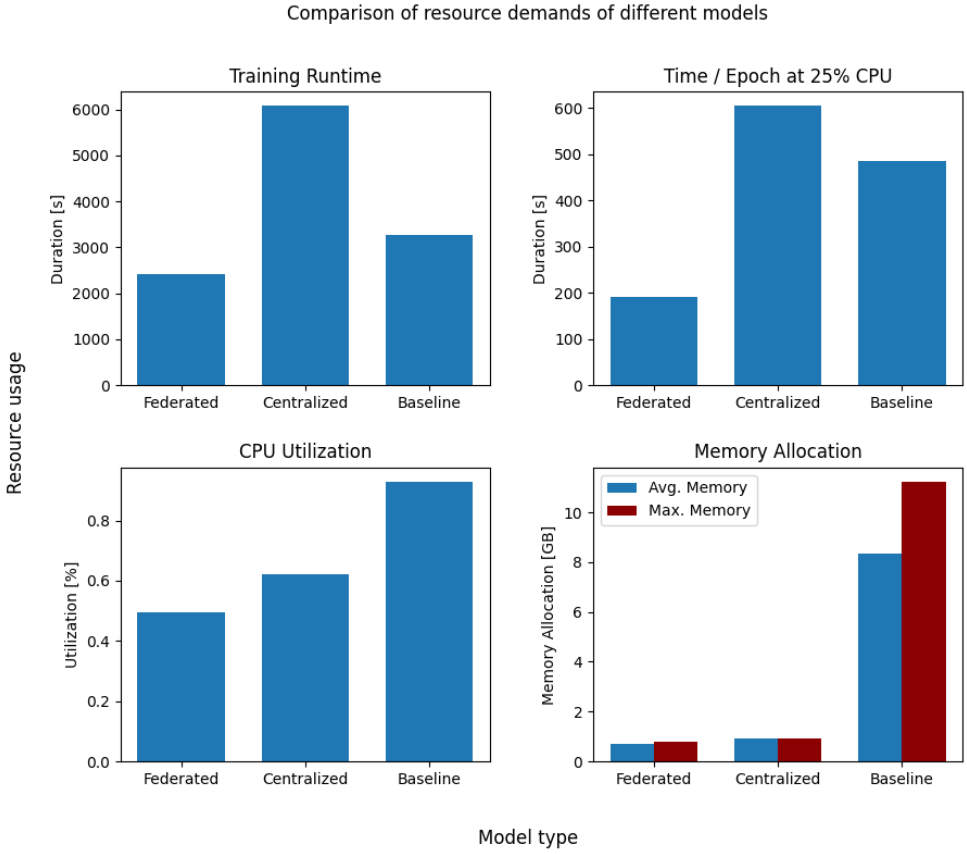

Figure 5.7: Resource Usages compared for the centralized, the baseline and the worst performing instance of the federated model.

one. This is to be expected, as the look-back technique and the higher data complexity used by the baseline method require more memory.

Also, we can safely determine that CPU utilization is no issue for training our model, as the training of one epoch could use about 200 seconds for our federated learning model but represents the collected data of about 984 measurement periods or 590400 seconds of the experiment. This implies that even training on a continuous data stream could be possible, as the combined data would span 984 seconds. All these calculations assuming a CPU cap of 25%, enabling the IIoT device to continue other work uncompromised. Therefore, even lightweight IIoT devices with a single core, low processor speed and a small memory of 2 GB are able to host our federated learning model.

Furthermore, when comparing the potential network burden for our federated learning implementation and the baseline method, we find that per iteration our federated learning implementation transferred 2.3 MB of model weights from all training-worker instance to the global aggregator. For comparison, the uncompressed raw data used for training the baseline method one iteration was 893 MB of size. As this is a more than 300-fold reduction, we find that our federated implementation is able to significantly reduce the network burden of the training process.

6
State of the Art

There has already been a great deal of work done related to the research fields of condition monitoring, federated learning and the combination of both for various Edge, IoT and IIoT scenarios. These works, in relation to the IoT, can subsequently be divided into their different research fields:

6.1 Federated Learning

Federated learning, even though a relatively new topic in the space of machine learning, has already seen a great deal of research regarding both general improvements and specific use cases.

X. Wang et al. [42] and Y. Yunfan et al. [43] developed two generalized frameworks, "In-Edge AI" and "EdgeFed", for improving federated learning at the edge. Exploring different approaches respectively of reducing the system communication load and offloading resource-intensive work to the cloud to archive a more effective aggregation. Q. Wu et al. [44] proposed a framework for mitigating issues of strong heterogeneity by the usage of personalized federated learning methods. These works dealt with the general optimization of federated learning in resource-restricted environments, while our work focused on the application of a specific use case. M. Dhada et al. [45] used a federated learning approach for determining the remaining useful life of engines on the turbofan data set [46], a topic strongly related to the basic idea of condition monitoring also using rotating machines. Our work, in contrast, dealt with the application of such a use case regarding the resource limitation of the IIoT.

6.2 Condition Monitoring

S. Sakib et al. [47] achieved low-latency condition monitoring inference by deploying pre-trained models to IIoT devices T. V. Hahn and C. K. Mechefske [48] utilized a self-supervised approach of Condition Monitoring using a Variational-Autoencoder on the NASA milling data set [49]. A. Mostafavi and A. Sadighi [50] proposed a condition monitoring framework which is fully located on IIoT devices. Their framework, like ours, uses an LSTM-autoencoder for

anomaly detection while being resource-restricted. Finally, S. Ahmad et al. [28] proposed the approach which was utilized by our work as its baseline method, of an LSTM-Autoencoder, detecting anomalies on unlabeled rotating machines' vibration data.

Often, the relevant data of the IIoT network, to which the model is supposed to be applied to is either too great in size and therefore would prove too costly to fully transfer to a centralized server where the model operates, or underlying privacy concerns prevent the data from being shared and collected in such a centralized location. [] To utilize distributed and otherwise inaccessible data, our work differs from the mentioned research above by utilizing federated learning to achieve this condition monitoring.

6.3 Federated Learning for Anomaly Detection

There has also been a lot of research into the combination of both anomaly detection and federated learning for an application to resource-restricted environments such as the Industrial Internet of Things. As condition monitoring heavily relies on the use of anomaly detection, we will use anomaly detection for this combination of related work to provide a broader overview. This research can be further divided into three parts after the type of learning which is applied:

6.3.1 Supervised learning

A. Qayyum et al. [51] proposed an effective federated learning approach for anomaly-detection-based covid-19 diagnosis at the edge on both X-ray and Ultrasound images, utilizing clustered federated learning for better coping with non-IID data. W. Zhang et al. [52] developed a blockchain-based federated learning solution for condition monitoring in the IIoT. Their approach included a novel weighted federated averaging algorithm to mitigate the data heterogeneity issue and the utilization of smart contracts for client incentives. The research does mostly show the highest performance metrics for their respective use cases. This, however, is counteracted by the requirement that for training, fully labelled data is needed. As fully and correctly labelled data for anomaly detection is often only obtainable by the use of manual work of domain experts who need to determine for every single data-point if it shows signs of anomalous behaviour and how it needs to be classified, this fully labelled data is costly to obtain. [] Our work, therefore, focuses on the application of unsupervised learning techniques lowering the barriers to feasibility.

6.3.2 Unsupervised learning

As in our work, there has been research trying to utilize a combination of federated learning and anomaly detection without the usage of any labelled data for training in the IoT. These approaches showed the most promising attributes for general feasibility, as they have the least preconditions their work was based upon.

T. Huong et al. [53] detected cyberattacks in an IIoT setting, using federated learning with an autoencoder architecture, treating attacks as anomalies in the network's behaviour while at the same time proving the feasibility of its deployment by evaluating CPU- and memory usage. D. Preuveneers et al. [54] used federated learning for autoencoder-based network anomaly detection in the IoT. Their approach utilized blockchain technologies for accountability and

auditing of machine learning models while at the same time limiting their performance impact on the network. Y. Liu et al. [55] utilized federated learning in conjunction with a combination of convolutional neural networks and LSTM layers for deep anomaly detection, solving the feature extraction using attention-based mechanisms, therefore reducing the communication overhead. W. Zhang et al. [56] proposed a deep-federated-learning-based method using dynamic validation for machine fault diagnosis in the IIoT.

While these works focused on many techniques regarding federated learning for unsupervised anomaly detection in the resource-restricted IoT and IIoT, our work differs in its direct comparison of a centralized, resource unrestricted baseline method and a resource restricted federated learning adaptation for a specific IIoT-typical use case.

7
Conclusion

In our work, we proposed a new, resource-efficient adaptation of an established state-of-the-art autoencoder-based condition monitoring method, which we then used in the implementation of a resource-efficient federated learning framework for a resource-constrained IIoT use case scenario. In our evaluation, we found that regarding its model's effectiveness, our federated learning implementation was not precisely as certain of a defect as the baseline method. Nevertheless, the federated learning implementation was able to detect all defects while not showing any false positive predictions. Additionally, our federated learning implementation was able to determine the beginning of a defect slightly faster than the baseline method, as it did not rely on a contextual understanding of multiple measurement periods. Furthermore, our implementation achieved the required efficiency of resource usage. It was effectively trained using only small amounts of computation power and memory for it's training, enabling even continuous training on a data stream while requiring less than 25% of a single core 2.2 GHz CPU and a maximum of 2 gigabytes of memory, which even most resource-constrained devices such as in the IIoT can provide.

We, therefore, conclude that our proposed federated learning implementation is applicable to the IIoT environment and provides an effective condition monitoring method without the need for a data transfer from the IIoT devices themselves. This makes it possible to apply this use case while at the same time preserving data privacy, utilizing otherwise inaccessible data and preventing network issues caused by the full transfer of data streams.

As our work researched the application of federated learning to a specific use case, the question stands if our findings are also applicable to other use cases in the IIoT. Further research could evaluate our findings on other industrial data sets to evaluate the generalizability of our work to both other machines than rotating machines or other use cases in general.

Bibliography

[1] O. Salman, I. Elhajj, A. Chehab, and A. Kayssi, "Iot survey: An sdn and fog computing perspective," *Computer Networks*, vol. 143, pp. 221–246, 2018.

[2] S. Balaji, K. Nathani, and R. Santhakumar, "Iot technology, applications and challenges: a contemporary survey," *Wireless personal communications*, vol. 108, no. 1, pp. 363–388, 2019.

[3] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in industry*, vol. 101, pp. 1–12, 2018.

[4] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: A survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[5] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors—a review," *IEEE transactions on energy conversion*, vol. 20, no. 4, pp. 719–729, 2005.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[7] Q.-V. Pham, K. Dev, P. K. R. Maddikunta, T. R. Gadekallu, T. Huynh-The, *et al.*, "Fusion of federated learning and industrial internet of things: A survey," *arXiv preprint arXiv:2101.00798*, 2021.

[8] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[10] M. R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of things in the 5g era: Enablers, architecture, and business models," *IEEE journal on selected areas in communications*, vol. 34, no. 3, pp. 510–527, 2016.

[11] J. Pizoń, G. Kłosowski, and J. Lipski, "Key role and potential of industrial internet of things (iiot) in modern production monitoring applications," in *MATEC Web of Conferences*, vol. 252, p. 09003, EDP Sciences, 2019.

[12] A. Banerjee, A. R. M. Forkan, D. Georgakopoulos, J. K. Milovac, and P. P. Jayaraman, "An iiot machine model for achieving consistency in product quality in manufacturing plants," *arXiv preprint arXiv:2109.12964*, 2021.

[13] Y. Han and Y. Song, "Condition monitoring techniques for electrical equipment-a literature survey," *IEEE Transactions on Power delivery*, vol. 18, no. 1, pp. 4–13, 2003.

[14] P. A. Higgs, R. Parkin, M. Jackson, A. Al-Habaibeh, F. Zorriassatine, and J. Coy, "A survey on condition monitoring systems in industry," in *Engineering Systems Design and Analysis*, vol. 41758, pp. 163–178, 2004.

[15] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, JMLR Workshop and Conference Proceedings, 2012.

[16] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[17] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.

[18] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069*, 2018.

[19] W. H. L. Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," in *Machine learning*, pp. 193–

208, Elsevier, 2020.

[20] H. Nguyen, K. P. Tran, S. Thomassey, and M. Hamad, "Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management," *International Journal of Information Management*, vol. 57, p. 102282, 2021.

[21] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[22] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.

[23] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5476–5497, 2020.

[24] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[25] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[26] R. B. Randall, "State of the art in monitoring rotating machinery-part 1," *Sound and vibration*, vol. 38, no. 3, pp. 14–21, 2004.

[27] J. Lee, H. Qiu, G. Yu, and J. Lin, "Rexnord technical services (2007) ims, university of cincinnati."," *Bearing Data Set". NASA Ames Prognostics Data Repository (http://ti. arc. nasa. gov/project/prognostic-data-repository), NASA Ames Research Center, Moffett Field, CA*, 2007.

[28] S. Ahmad, K. Styp-Rekowski, S. Nedelkoski, and O. Kao, "Autoencoder-based condition monitoring and anomaly detection method for rotating machines," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 4093–4102, IEEE, 2020.

[29] H. Yuan, W. Morningstar, L. Ning, and K. Singhal, "What do we mean by generalization in federated learning?," *arXiv preprint arXiv:2110.14216*, 2021.

[30] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The performance of lstm and bilstm in forecasting time series," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 3285–3292, IEEE, 2019.

[31] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[32] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on nuclear science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[33] T. G. Dietterich, "Machine learning for sequential data: A review," in *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pp. 15–30, Springer, 2002.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," *arXiv preprint arXiv:1905.11881*, 2019.

[36] Z. Li and S. Arora, "An exponential learning rate schedule for deep learning," *arXiv preprint arXiv:1910.07454*, 2019.

[37] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[38] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the second workshop on distributed infrastructures for deep learning*, pp. 1–8, 2018.

[39] Y. B. Zikria, S. W. Kim, O. Hahm, M. K. Afzal, and M. Y. Aalsalem, "Internet of things (iot) operating systems management: Opportunities, challenges, and solution," *Sensors*, vol. 19, no. 8, p. 1793, 2019.

[40] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[41] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, "Orchestration of containerized microservices for iiot using docker," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1532–1536, IEEE, 2017.

[42] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[43] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu, "Edgefed: Optimized federated learning based on edge computing," *IEEE Access*, vol. 8, pp. 209191–209198, 2020.

[44] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent iot applications: A cloud-edge based framework," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 35–44, 2020.

[45] M. Dhada, A. Parlikad, and A. S. Palau, "Federated learning for collaborative prognosis," *COPEN 2019*, 2020.

[46] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *2008 international conference on prognostics and health management*, pp. 1–9, IEEE, 2008.

[47] S. Sakib, M. M. Fouda, Z. M. Fadlullah, and N. Nasser, "Migrating intelligence from cloud to ultra-edge smart iot sensor based on deep learning: An arrhythmia monitoring use-case," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 595–600, IEEE, 2020.

[48] T. V. Hahn and C. K. Mechefske, "Self-supervised learning for tool wear monitoring with a disentangled-variational-autoencoder," *International Journal of Hydromechatronics*, vol. 4, no. 1, pp. 69–98, 2021.

[49] A. Agogino and K. Goebel, "Best lab," *UC Berkeley, Milling data set. NASA Ames Prognostics Data Repository. http://ti. arc. nasa. gov/project/prognostic-data-repository, NASA Ames Research Center, Moffett Field, CA*, 2007.

[50] A. Mostafavi and A. Sadighi, "A novel online machine learning approach for real-time condition monitoring of rotating machines," in *2021 9th RSI International Conference on Robotics and Mechatronics (ICRoM)*, pp. 267–273, IEEE, 2021.

[51] A. Qayyum, K. Ahmad, M. A. Ahsan, A. Al-Fuqaha, and J. Qadir, "Collaborative federated learning for healthcare: Multi-modal covid-19 diagnosis at the edge," *arXiv preprint arXiv:2101.07511*, 2021.

[52] W. Zhang, Q. Lu, Q. Yu, Z. Li, Y. Liu, S. K. Lo, S. Chen, X. Xu, and L. Zhu, "Blockchain-based federated learning for device failure detection in industrial iot," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5926–5937, 2020.

[53] T. T. Huong, T. P. Bac, D. M. Long, T. D. Luong, N. M. Dan, B. D. Thang, K. P. Tran, *et al.*, "Detecting cyberattacks using anomaly detection in industrial control systems: A federated learning approach," *Computers in Industry*, vol. 132, p. 103509, 2021.

[54] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, "Chained anomaly detection models for federated learning: An intrusion detection case study," *Applied Sciences*, vol. 8, no. 12, p. 2663, 2018.

[55] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6348–6358, 2020.

[56] W. Zhang, X. Li, H. Ma, Z. Luo, and X. Li, "Federated learning for machinery fault diagnosis with dynamic validation and self-supervision," *Knowledge-Based Systems*, vol. 213, p. 106679, 2021.