Split Learning for Predictive Maintenance in Industrial Internet of Things

Haneen Hussein Adel[1], Osamah Fadhil Abdulateef[1] and Ali Hussein Hamad[2]
[1]*Automated Department of Automated Manufacturing Engineering, Al-Khwarizmi College of Engineering, University of Baghdad, 10071 Baghdad, Iraq*
[2]*Department of Biomedical Application, College of Artificial Intelligence, University of Baghdad, 10071 Baghdad, Iraq*
Hanin.Adel2404m@kecbu.uobaghdad.edu.iq, drosamah@kecbu.uobaghdad.edu.iq, ahamad@kecbu.uobaghdad.edu.iq

Keywords: Predictive Maintenance, Industrial Internet of Things (IIoT), Convolutional Neural Network (CNN), Sensors, SCINet.

Abstract: The need for intelligent predictive maintenance (PdM) systems that provide data privacy and operational efficiency has been made clear by Industry 4.0 and smart manufacturing. The scalability of traditional, centralized learning models is constrained by bandwidth, high communication costs, limited scalability, and data security, especially on Industrial Internet of Things (IIoT) platforms. This study addresses these challenges by proposing a lightweight and distributed predictive maintenance framework based on split learning (SL) that employs the sample convolution and interaction network (SCINet) to identify faults in distributed AC motor systems. In our system, each motor uses four sensors (vibration, current, temperature, and ambient) that capture real-time data, which is processed on-site, and only intermediate representations are sent to a cloud server where the model is trained. With the deep neural network (DNN) divided into edge devices and cloud, we can minimize the exposure of raw data, as well as the communication overhead. Experimental results exhibit that the proposed framework achieves 99.35 per cent accuracy as well as 0.0022 loss in the classification of the five types of failures (normal, overcurrent, misalignment, stop rotation, and heavy load), which is better than centralized training (99.04 per cent accuracy and 0.0216 loss), wh ile reducing communication overhead and computational burden on edge devices. The major contributions are: a privacy-conserving SL architecture that does not require the exchange of raw data, feature extraction using SCINet to better model temporal dependencies, and bandwidth saving through data transmission through compression. The experimental findings prove that SL not only improves fault detection but also meets the key IIoT requirements, namely, data security, computational efficiency, and real-time performance. The paper offers a scalable solution to intelligent maintenance in Industry 4.0 that can be extended further in the future by the introduction of hybrid federated-split learning to cross-factory collaboration.

1 INTRODUCTION

The rapid evaluation of the Industrial Internet of Things (IIoT) has significantly transformed the modern industrial systems by enabling continuous monitoring, intelligent automation, and data-driven decision making [1]. The Fourth Industrial Revolution (Industry 4.0) has introduced artificial intelligence (AI) and the Industrial Internet of Things (IIoT) to the industry. One of the keys to such transformation can be represented by predictive maintenance (PdM), which utilizes real-time sensor readings along with machine learning to predict prospective equipment breakdowns before they happen. In contrast to the traditional reactive maintenance or preventive maintenance, PdM maximizes the efficiency of operations, minimizes downtimes, and helps to increase the service life of machinery [2]. Nevertheless, there are urgent problems of the widespread deployment of PdM in IIoT systems: (1) the possibility of data privacy risks related to centralized cloud-based learning, (2) the high communication cost of transmitting raw sensor data over distributed networks [3], and (3) scalability becomes limited as the number of connected machines increases. More importantly, transmitting raw industrial data raises significant privacy and security concerns, particularly in distributed manufacturing environments. These limitations hinder the practical deployment of real-time predictive maintenance systems in industrial settings [4], [5].

To resolve these challenges, privacy-preserving alternatives have been proposed based on the distributed learning paradigm, including federated learning (FL) and split learning (SL). FL supports the network-wide shared training of models without the transmission of raw data, but it experiences high communication overhead and heterogeneity issues [6]. Conversely, SL splits deep neural networks into clients and servers, and only intermediate activations (so-called smashed data) are transmitted, which reduces the consumption of bandwidth and improves privacy, reduces communication cost, and lowers computational burden on resource-constrained edge devices [7].

Recent work on the Sample Convolution and Interaction Network (SCINet) [8] demonstrates superior performance in capturing multi-resolution temporal dependencies, making it ideal for IIoT fault detection. Combining model updates from several devices may be a long and resource-intensive procedure. Furthermore, it remains crucial to conduct further research to ensure the FL process is secure and resilient, especially against hostile attacks. High communication costs and potential security flaws at the split points pose additional difficulties for SL. Data privacy and computational efficiency are two benefits of split learning, which divides a machine learning model across many parties. Sensitive data stays localized by spreading model components, reducing privacy issues related to centralized data collection. Additionally, by dividing the training task among several devices, SL can increase computing efficiency. Conversely, federated learning focuses on using decentralized data to train models without sharing raw data. This method ensures data privacy while facilitating shared model creation. Nevertheless, Federated Learning may encounter increased communication costs and difficulties related to handling data that is unevenly distributed [9], [10].

This paper aims to propose a split learning system based on SCINet for predictive maintenance in real-time industrial Internet of Things (IIoT) systems. The key idea is that the combination of SCINet and the split learning environment is not intended to post hoc the novelty of the combination itself, but instead to take advantage of the hierarchical temporal property extraction ability of SCINet to enhance the fault detection accuracy and model efficiency with respect to privacy-constrained models. The main objectives of this study are:

1) Design a distributed lightweight Predictive Maintenance System: This is a predictive maintenance system that comprises several distributed benchmark machines. Four sensors (current, vibration, object temperature, and ambient temperature) are fitted in every client (AC motor) to allow monitoring the health and operating condition of the machine as well. The data is spread across three machines, and the split learning model based on SCINet is trained collaboratively, where at the client side, local feature extraction takes place, and the cloud server completes the model. This system design improves system privacy, scalability, and predictability.
2) Minimize communication overhead compared to centralized learning approaches.
3) Privacy-Preserving Architecture: A split deep neural network (DNN) is proposed, in which an edge device (AC motors) processes raw sensor data on-the-fly and only sends the compressed feature activation to the cloud server. This will avoid the exposure of raw data and will guarantee the confidentiality of data in the course of distributed training.

Experimental evaluation demonstrates that the proposed approach achieves high fault classification performance while significantly reducing communication load and maintaining real-time operation. The main contributions of the work can be summarized as follows:

1) Implemented a predictive maintenance algorithm on AC motor benchmark sensors.
2) Use a split learning (SL) architecture to integrate with IoT devices within the manufacturing environment to obtain lightweight, bandwidth-efficient, and improve the training and performance.
3) Comparative performance analysis against centralized approaches.

2 METHODOLOGIES

Split learning is a notable innovation in the field of federated learning, a collaborative machine learning method in which a local client and a centralized server jointly learn a global model while simultaneously safeguarding decentralized data. Local client is often used in this framework to refer to the user-owned data-generating devices, including Internet of Things (IoT) devices. The fundamental input of split learning is the ability to train the model using distributed data sources without necessarily transmitting the raw data, which improves data privacy and reduces communication overhead.

2.1 Split Learning

Split learning is an alternative to distributed deep learning that enables multiple parties to build ML models collaboratively without disclosing their raw data. Desegregation training: data terminals store all information on site and send model updates to other terminals to address privacy concerns and reduce data storage costs [11]. Split learning splits up a deep neural network into several parts, where each part is trained on different clients. In the meantime, the training set may exist on a supercomputing machine or may be separated and distributed among multiple clients for collaborative model training [12]. Then, the industrial partners that are training clients of the deep neural network cannot see each other's data. To push a deep neural network, there are ways of encoding the input into a different space and sending it up [13], [14]. The Privacy Protections and User Privacy are two cornerstones of Split Learning (SL). It does so via an interactive learning algorithm based on the fed-in local data and therefore without the necessity to share classifiers directly. In this context, security and privacy are one of the major challenges of implementing SL systems, as supported also by the previous literature [15], [16]. SCINet adopts the encoder-decoder architecture. The encoder is a pyramidal convolutional network that learns dynamic temporal connections across resolutions using multiple ranges of convolutional filters.

Its HTCBs could also enable the effective extraction of multi-scale temporal features of time-series T sensor data at the client side [17]. With the initial layers of SCINet running on edge devices, each client could contribute with local temporal patterns of raw sensor signals and generate compacted intermediate activations for transmission to the cloud server. This hierarchy structure favours the distributed split learning framework in that:

1) Capturing both short-term and long-term temporal dependencies, which improves fault detection accuracy.
2) Reducing the size of data transmitted to the server, thereby enhancing communication efficiency while preserving privacy.

It means that the cloud server can do more jobs of higher-level feature integration and classification and reduce the computational burden at the edge. In sum, the hierarchical organization of SCINet makes the distributed feature extraction efficient and effective, which allows the given split learning framework to deliver robust predictive maintenance with privacy protection and a low communication cost. As illustrated in Figure 1a, the fundamental component, SCI-Block, reduces the number of samples of an input data or a feature by down-sampling to two sub-sequences, then applying a bank of convolutional filters to each sub-sequence to identify and extract unique but useful features of each portion of the data. During down-sampling, we use interactive learning to recover lost information between the two sequences. The structure of SCINet implements multiple SCI-Blocks, which form a binary tree arrangement [18].

Multiple SCI-Blocks are implemented using the SCINet structure, creating a binary tree arrangement (Fig. 1b). The recovery of significant time points is facilitated by the local and global understanding of the time series given by each SCI-Block in this architecture. A fully connected network decoder merges the decoded information with the original data input and then predicts the new sequence shape formed after convolution operations, down-sampling processes, and sequence structure interaction. Stacked SCINet training takes multi-layer stacked networks as the starting step and then adopts intermediate 7 supervision for a network architectural design shown in Figure 1c.

As shown in Figure 1, the input feature F is split into two sub-sequences, F_{odd} and F_{even}, which interact iteratively within the SCI-Block. These sub-sequences do not convey all the information in the main sequence but are time-resolved less well. Next, we extract F_{even} and F_{odd} features by convoluting with different kernels: Because the kernels are independent, the retrieved characteristics would also form unique but helpful temporal relationships with improved representation skills. In order to minimize potential data degradation due to down-sampling, we suggest a novel interactive learning approach. The two-component sub-sequences can exchange information in both directions thanks to this tactic. This is accomplished by letting the sub-sequences discover affine transformation parameters among themselves. There are two stages to the interactive learning process shown in Figure 1a. Using two unique one-dimensional convolutional modules, the sequences Feven and Fodd are first projected to the hidden states. These concealed representations are then separately converted into formats appropriate for element-wise interaction. Finally, an element-wise product (by (1)).

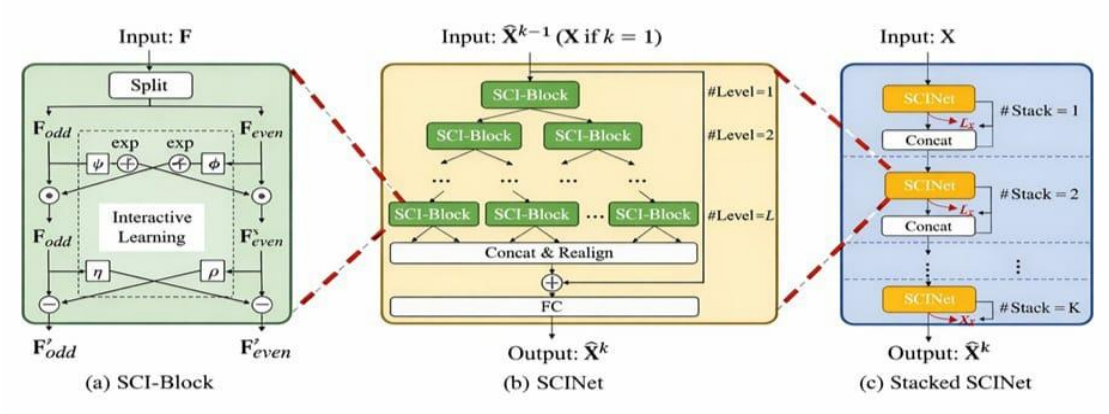


Figure 1: The overall architecture of the Sample Convolution and Interaction Network (SCINet).

$F^s_{odd} = F_{odd} \odot \exp(\phi(F_{even}))$, $F^s_{Even} = F_{even} \odot \exp(\psi(F_{odd}))$. (1)

$F^{\sim}_{odd} = F^s_{odd} \pm \rho(F^s_{even})$, $F^{\sim}_{even} = F^s_{even} \pm \eta(F^s_{odd})$, (2)

$F^{\sim}_{odd} = F^s_{odd} \pm \rho(F^s_{even})$, $F^{\sim}_{even} = F^s_{even} \pm \eta(F^s_{odd})$. (3)

F_{even} and F_{odd} are efficiently scaled by this method, in which learned neural network modules are used to extract the scaling factors from one another. The scaling features are expanded in the interactive learning module. Using the two 1D convolutional modules in accordance with (3), F_{even} and F_{odd} are transformed into two additional hidden states. Next, addition and subtraction operations are carried out to produce new features F_{even} and F_{odd}. The interactive learning module's final outputs yield two updated sub-features, F_{even} and F_{odd}. While executing each convolutional layer, the down-sampled convolve-interact architecture expands its receptive field beyond dilated convolutions, which serve as the foundation for TCN implementation. Important data gathered from the two down-sampled temporal sub-sequences that contain both local sequence viewpoints and global series comprehension are combined in the SCI-Block.

We can use the aforementioned SCI-Blocks to construct the SCINet by hierarchically arranging many SCI-Blocks. As seen in Figure 1b, this produces a tree-structured framework. There are 2^{ι} SCI-Blocks at the ι level, where $\iota = 1$, L is the index of the level, and L is the total number of levels. The input time series X (for k=1) or the feature vector $\check{X}_1^{k-1} = \{\check{X}_1^{k-1}, \ldots, \check{X}_\tau^{k-1}\}$ (for $k > 1$) is progressively down-sampled and processed by SCI-Blocks through various levels within the k-th SCINet of the stacked SCINet. This makes it possible to learn features effectively at different temporal resolutions. In particular, information from previous layers will gradually.

2.2 Deep learning Neural Networks

Table 1 shows the hyperparameters for the suggested DNN model. There are five layers in the model. The output layer is equal to the number of classes, which is also five, and the input layer is equal to the input feature [18].

Table 1: Hyperparameters of the proposed lightweight model.

Parameter name	Value
Number of neurons per layer	50
Batch size	64
Epochs	25
Loss function	Categorical cross-entropy
Activation function	ReLU, SoftMax (in output layer)
Optimizer	Adam
Number of classes (output layer)	5
Number of layers	5
Cut layer	32

In a DNN, ReLU is typically employed as an activation function for the hidden layers. To do this, we utilize the neural network's penultimate layer activation to backpropagate the ReLU classification layer's weight parameters ReLU formula is:

$$f(x) = \max(0, x). \quad (4)$$

When x is less than 0, this function yields 0, and when x is more than 0, it yields a linear function. The SoftMax function is typically used as an activation function (at the final layer) in deep learning models

for multiclass classification tasks [19]. The SoftMax formula is as follows.

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \quad (5)$$

A series of K real values is converted by the SoftMax activation function into a series of real values that add up to 1. Based on the function's output, which is always between 0 and 1, one can get a probability score. Since the output of the SoftMax activation and one-hot encoding have the same format, categorical cross-entropy (CCE) is frequently employed as the loss function in multiclass classification problems. The following method can be used to put all the components together [20]:

1) SoftMax activation: When a model has one neuron for each class in multiclass classification, SoftMax activation is frequently used. The output logits (raw scores) are converted into probabilities using SoftMax, where each score indicates the likelihood of a particular class, and the probability of all the classes is equal to 1. SoftMax performs well in concentrating on the most likely class with the highest probability in categorical classification problems, where we are trying to classify the single most likely class. Likelihood [21].
2) Encoding One-Hot: In multiclass jobs, one-hot is frequently used to encode labels. Each class is encoded as a vector using the one-hot encoding method, where a 1 indicates the proper class and a 0 indicates all other classes. This facilitates comparing the actual labels to the model's expected probability (SoftMax output).
3) CCE, or categorical cross-entropy: By comparing the projected probability by the SoftMax with the one-hot coded true labels, categorical cross-entropy determines the loss. As an illustration, CCE can be calculated mathematically as follows:

$$CCE = -\sum_{i=1}^{C} y_i \,.\log(p_i). \quad (6)$$

Where:

- y_i Is the one-hot encoded true label (1 for the correct class,0 otherwise).
- p_i Is the predicted probability for each class.

Since only the correct class is included in the loss (other. y yi), the model is trained to generate predictions with higher confidence in the correct class, since the loss is the negative log of the expected probability of the correct class. Lastly, the popularity of categorical cross-entropy can be attributed to its compatibility with SoftMax values and one-hot encoding. By attempting to penalize the model if it assigns a low chance of assigning the proper class, it also encourages the model to improve its accuracy in predicting multiclass classification issues [22].

2.3 Multi-Class Performance Metrics

One performance-measuring method that can be used to assess a classification model's accuracy is a confusion matrix. In supervised learning, where the real values of the data are known, it is very helpful. Instead of being a 2x2 matrix as seen above, a multiclass confusion matrix is an N x N matrix, where N stands for the number of classes. The confusion matrix used to assess models that divide examples into more than two classes is expanded upon in this type of confusion matrix. When there are more than two classes, this can be quite helpful. The frequency with which an example of class I was placed in class J is represented by c- (i, j) in each matrix element. A comprehensive set of metrics for the overall confusion matrix is:

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} TP(C_i)}{\sum_{i=1}^{N}\sum_{j=1}^{N} C_{i,j}}, \quad (7)$$

$$Recall(C_i) = \frac{\text{TP}(C_i)}{\text{TP}(C_i)+\text{FN}(C_i)}, \quad (8)$$

$$Pression\ (C_i) = \frac{\text{TP}(C_i)}{\text{TP}(C_i)+\ \text{FP}(C_i)}. \quad (9)$$

Where: TP ($.C_i$) True positive for class i, FN false negative for class i.

The loss function used in this work is mean square error (MSE), which is the most commonly used algorithm, where

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)2. \quad (10)$$

Where:

- $\hat{y}_i$ represent the actual value;
- y_i is the desired value;
- m is the sample size.

3 PRELIMINARIES

Predictive maintenance, which tracks equipment condition and anticipates issues using IIoT and data analysis, enables proactive maintenance. This study develops a machine learning and IIoT-based predictive maintenance solution.

3.1 Sensors

Four sensors have been used in the suggested PdM system. The ADXL345 is a 3-axis, 13-bit shaking sensor, making it the perfect tool for measuring motor vibrations. Its tiny form factor and low power consumption are among its appealing features. The ACS712 current sensor module is used to assess the motor current. It is a magnetically hysteresis-free current sensor that is completely enclosed. It runs on a single 5V power source and uses the Hall effect to sense both DC and AC currents. It also makes use of a low-resistance current conductor and an ADS1115 analog-to-digital converter. The MLX90614 infrared thermometer, the temperature sensor utilized in this investigation, needed the following particular crucial characteristics. It was selected because of its wide temperature range, excellent precision, and potential for non-contact temperature measuring. Due to its affordability and compact size, it is also a suitable option for this study. The outside temperature was also measured using an SHT21 digital humidity and temperature sensor.

3.2 Dataset Implementation

A current sensor, a non-contact temperature sensor, a temperature and humidity sensor, and a three-axis accelerometer sensor were among the sensors employed in the current study to collect the sensory data. When three sets of data are collected, the vibration sensor generates three sets of data (X, Y, and Z). A failure would happen every second of the sample time, and these data are shared in a column of the data set. Both online and offline sensor data are collected; in the latter case, a high density of data is produced in the following fine types of faults: normal, overcurrent, stop rotation, misalignment, and heavy load.

In order to obtain the required dataset based on the required fault, these classes were applied to the motor. In order to mimic the vibration on the motor's three axes, an unbalanced mass has also been placed on the motor shaft. To simulate the rise and fall of the motor's current, a capacitor of various sizes is used. A mechanical braking system was used to simulate the stop and heavy load conditions. The quantity and number of data sets under each category are displayed in Figure 2. Table 2 displays examples of the data that was gathered.

The deep learning model uses this dataset to create a prediction model that is run on a cloud server to carry out real-time monitoring and anticipate any faults. Four categories are most frequently used to classify the collected data. The training dataset, which makes up about 60% of all the data gathered, is utilized to train the model. The test data (unseen data) makes up the remaining 30% of the validation percentage, which is utilized to assess the model's performance and adjust its hyperparameters.

Table 2: Class counts and sizes.

Failure type	File Size	Count Raws
Normal (No failure)	813 KB	20059
High-current failure	202 KB	4591
Vibration failure	301 KB	6562
Stop rotating failure	651 KB	16566
High-load failure	140 KB	2911

In this work, a deep neural network model has been used to classify the different types of proposed system failures. Before applying the DNN-based split learning algorithm, a dataset must be pre-processed, such as normalizing and labelling the different classes as follows:

1) Normalization: One major pre-processing step in machine learning is dataset normalization. It entails converting the dataset's features to common scales, usually between -1 and 1.
2) Preprocessing for dataset labeling includes converting categorical features into a numerical representation suitable for machine learning models. This transformation utilizes a one-hot encoder to generate a binary column (values of either 0 or 1) for every distinct category within the original data.
3) Data splitting: The dataset has been divided into 30% testing and 70% training. The training set, which typically contains the most data, is utilized to train the model. Testing is performed to evaluate the trained model's refined performance with untested data. This gives an approximation of the model's generalizability. The distribution of the gathered datasets for five different class types, normal, overcurrent, heavy load, stop, and misalignment, is displayed in Figure 2. The class is not evenly dispersed, as this image illustrate.
4) Class balance: The Synthetic Minority Over-Sampling Technique (SMOTE) is used in this work to address class imbalance in datasets. Emphasizing the minority classes in the training data gives a DNN model a more balanced perspective of the issue and increases the likelihood that it will learn to classify all the classes well. This contributes to confirming that the SMOTE process was effective in

guaranteeing that the class assignments improved equity. The five classes are displayed after balance in Figure 3, where SMOTE was applied only to the training set to address class imbalance, while the validation and testing sets retained their original distributions to ensure realistic performance evaluation

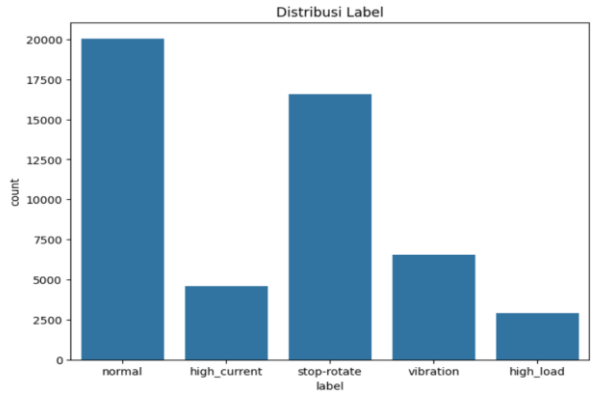


Figure 2: Dataset distribution.

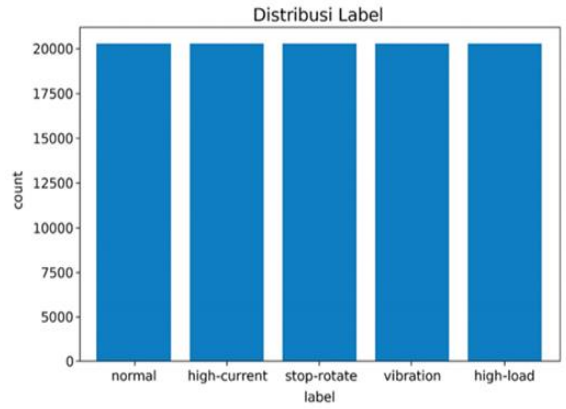


Figure 3: The distribution of the five datasets after balance.

4 PROPOSED SYSTEM DESIGN

The proposed system of predictive maintenance consists of three AC motors (220V, 0.5A), which were implemented and tested in a real-time environment to validate its feasibility and efficiency under Industrial Internet of Things (IIoT) conditions: Current, vibration, contact-less object temperature, and ambient temperature are the four sensors found in each motor. The sensors will be utilized to monitor the motor's condition while it is in flight. A Raspberry Pi 4 using the I2C bus communication protocol is used to gather the sensor's data. The Raspberry Pi is responsible for collecting and uploading the data to the cloud server. The 12C protocol, a two-wire interface with a key design to facilitate communication between several integrated circuits (ICs), is used to facilitate communication between sensors and the Raspberry Pi. Sensors are connected to the Raspberry Pi via two wires: the serial clock line (SCL) and the serial data line (SDA).

Controlling the supplied and received data via the sensors is efficiently handled by the Raspberry Pi 4. All client-side sensor outputs are transmitted to the cloud using the I2C protocol. The sensors SHT21, MLX90614, and ADXL345 are directly connected through built-in I2C interfaces, while the ACS712 current sensor is connected via an analog-to-digital converter (ADC, 0x48) to convert AC measurements into digital values. The deep neural network (DNN) deployed in the cloud consists of multiple layers, where the input layer size corresponds to the number of extracted features and the output layer size corresponds to the number of fault classes. During model training, the architecture is configured with client-side neurons (2), a cut layer representation (32), and an output layer on the cloud (5 classes), coordinated using a round-robin mechanism.

The SCINet-based model is divided into two primary segments under the split learning configuration. The client-side segment, deployed on the Raspberry Pi 4, includes the initial convolutional and interactive blocks responsible for temporal feature extraction from raw sensor signals. The cut layer is positioned near the end of the client-side model, enabling most feature extraction operations to be performed locally. This design significantly reduces the size of transmitted intermediate activations compared to raw data while keeping the computational load within the practical limits of the Raspberry Pi 4. The choice of the cut layer in split learning directly affects computation and data transmission. As shown in Table 3, each layer adds 2,500 parameters, from 0 in the first layer to 12,500 in the sixth. Shallow layers reduce edge computation but increase transmitted data, while deeper layers increase local load but send more compact activations. Consequently, the selected cut configuration achieves an effective balance between communication efficiency, edge resource utilization, and real-time predictive performance.

Client-side. Deployed on Raspberry Pi 4 edge devices, this segment includes the initial convolutional and interactive blocks responsible for temporal feature extraction. These layers process raw sensor signals locally and generate intermediate feature activations.

Server-side. This segment, which is hosted on the cloud server, receives compressed activations from every client and completes the remaining SCINet layers to carry out fault prediction and classification. In this case, the model trains the data up to the cut layer (the final layer on the client side) and generates an intermediate result for its connected model, where sensor data is gathered from each motor and uploaded to the cloud server. The cloud server then multiplies all of this output before sending backpropagation to the machine's sensory data. The servers complete the training phase and calculate the required gradients after receiving the client's partially trained or intermediate models.

These computed gradients are subsequently sent back to the clients, facilitating their contribution to the following round of training. During training, each client performs a forward pass on its local data up to the cut layer, producing intermediate activations that are transmitted to the cloud server. The server aggregates the activations from all clients, computes the loss, and performs backpropagation, sending the resulting gradients back to the clients. The whole process is repeated until the global model has been fully trained. These gradients enable clients to update model segments in their local device without transmission of raw sensor data. A round-robin process guarantees that all the clients make iterative contributions until the global model converges. The communication protocol is also applied as an established I2C-based synchronization scheme, which guarantees efficient data transfer with minimum latency. On a live system, the system was observed to have attained an average end-to-end latency of between 1.8 and 2.3 seconds, including local inference, communication delay, and cloud computation. Such performance is found to be sufficient in near-real-time predictive maintenance applications, where the task is to identify anomalies at an early stage and not to act in the moment. These findings indicate that the designed system can have a stable operation and constant communication with minimal computational cost, making it scalable and flexible to implementations at the industrial level. (Fig. 4) shows the entire architecture of the proposed system, client-server interactions, cut layers, middle activations, and data and gradient flow during the training process.

All the sensors share the same lines but with varying addresses, where the ADXL345 sensor is employed to detect any abnormal vibrations that could be a sign of mechanical problems like misalignment or unbalance in motors and rotating parts. The ACS712 sensor is a useful device that measures abnormal changes in current consumption and provides early warning of electrical overload or internal flaws in the electrical system. Instead, the MLX90614 allows direct temperature measurements via infrared technology, which makes it an appropriate tool in the detection of cooling system malfunctions early or a significant amount of friction that can cause thermal damage. The environmental sensor SHT21 is also utilized to check the ambient temperature and humidity, high values of which may also cause internal corrosion or insulation decay with time, particularly in sensitive industrial settings (as indicated in Table 4). The benefit of using a different address on each sensor is to prevent packet collisions.

5 RESULTS AND DISCUSSION

By comparing the predicted labels with the actual labels of each class, the confusion matrix offers a comprehensive assessment of the model's performance. The confusion matrix was displayed as a heatmap at work, with the predicted label (x-axis) and the number of occurrences of each true label (y-axis) in each cell. The off-diagonal elements show incorrect classifications, while the diagonal objects show correctly classified occurrences (true positives and true negatives). Figure 5's confusion metrics (a) without SL and b) with SL demonstrate this.

The different metrics have been calculated using equations 7, 8, and 9, and the comparative analysis of the two confusion matrices (a: without SL; b: with SL) shows the impact of introducing Split Learning (SL) on classification performance. In case a (without SL), the classifier achieved high accuracy in detecting the normal (5944/5944) and stop-rotate (5045/5045) classes, while misclassifications were mostly concentrated between high-load and vibration. For instance, 92 samples of high-load were incorrectly classified as vibration, and the vibration class exhibited minor misclassifications across other categories (29 to high-current, 15 to high-load, and 5 to normal).

Table 3: Number of parameters per layer in the neural network and its implications for cut layer selection.

No.	Parameters
One layer	0
Two layers	2,500
Three layers	5,000
Four layers	7,500
Five layers	10,000
Six layers	12,500

Table 4: Sensor specifications.

Sensor	Definition	Fault Type	Measurement Range	Accuracy
ADXL345	3-axis accelerometer, high accuracy, low power consumption	Misalignment detection via axial vibrations	±2g, ±4g, ±8g, ±16g	0.004g
ACS712	Current sensors (measure AC and DC currents accurately)	Overload, electrical leakage	±5A, ±20A, ±30A	66mV/A (±5A
MLX90614	Non-contact temperature sensor	Motor overheating, excessive friction	-70°C to 380°C	0.02°C
SHT21	Digital humidity and temperature sensor, high accuracy, fast response	Excessive humidity, insulation issues, and environmental effects	0% to 100% humidity, -40°C to 125°C	0.3°C for temperature, 2% for humidity
ADC	A device that converts analog signals into digital values.	Used with multiple sensors for signal analysis	0-5V	±1% to ±0.1%

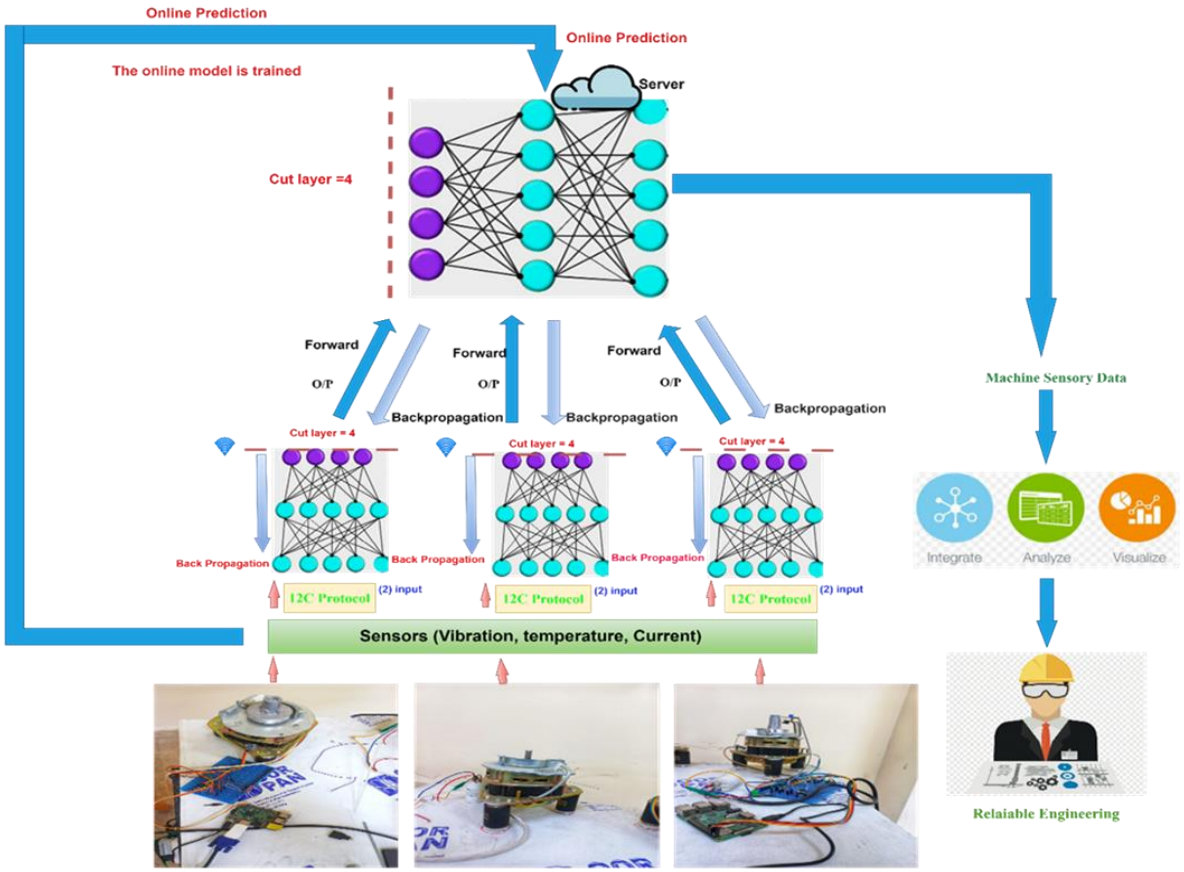
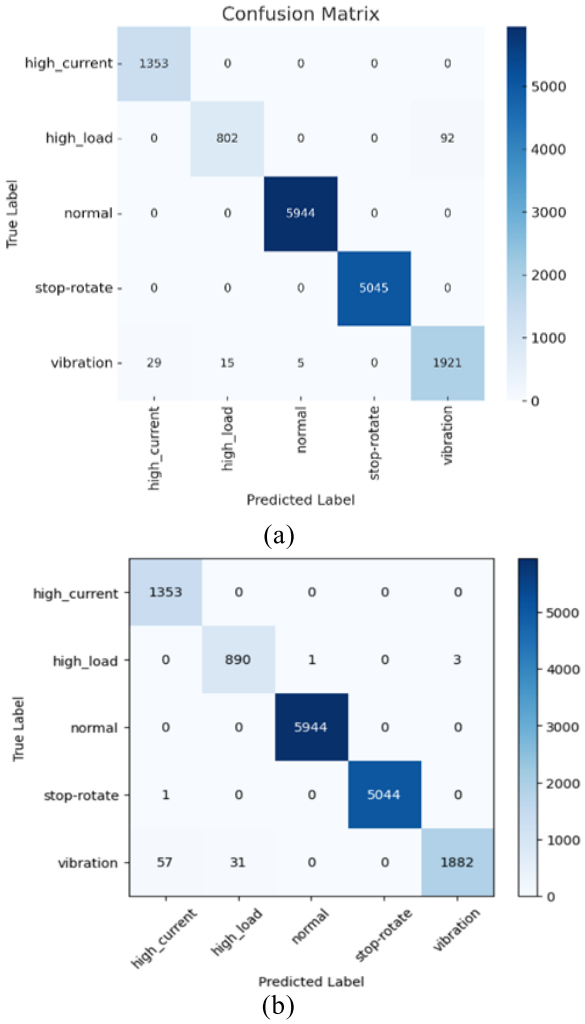


Figure 4: The design and implementation of the proposed system.

In contrast, case b (with SL) shows that SL improved the classification of the high-load class (from 802 correctly classified samples in case a to 890 in case b), thereby reducing the misclassification rate for this class. Similarly, normal and stop-rotate maintained consistently high performance, with negligible differences between the two cases. However, the vibration class exhibited more confusion with high-current and high-load under SL (57 and 31 misclassifications, respectively), compared to the distribution of errors in case a.

In general, with the introduction of SL, the strength of the classification in the case of the high-load class, which was more susceptible to errors earlier, slightly deteriorated, and the line between vibration and other operational conditions became unclear. This indicates that SL adds value to the overall system performance, especially in the reduction of systematic errors to certain types of faults, although its mechanism needs to be refined further to minimize overlaps of the vibration type. Table 5 displays the classification report of various measures with and without SL.

These results are grounded in the accumulation of data from all clients. The local model that was trained on the whole dataset without split learning had a decent performance on test (0.9904) and train (0.9917), and this indicates that the model can learn the data and predict what is right. However, a centralized strategy has drawbacks regarding privacy, security, generality, and bandwidth. This strategy effectively enhanced the model's performance in terms of accuracy, recall, and precision, while addressing some of its main challenges, including overfitting and generalization. Split learning minimized the risk of overfitting and improved privacy and data security. As raw data was always Local to the edge devices and only the intermediate representations (compressed data) were sent, the risk of sensitive operational data information exposure was minimized. Moreover, the interaction between the client and server was encrypted, which is an added protection in the case of a possible cyber threat. Split Learning had an apparent advantage in bandwidth efficiency.



(a)

(b)

Figure 5: Confusion metrics: a) without SL, b) with SL.

Table 5: Classification report a-without SL, b-with SL.

a. Classification	Precision	Recall	support	b. Classification	Precision	Recall	Support
High-current (0)	0.98	1.00	1353	High-current (0)	0.96	1.00	1353
High-load (1)	0.98	0.90	894	High-load (1)	0.97	1.00	894
Normal (2)	1.00	1.00	5944	Normal (2)	1.00	1.00	5944
Stop-rotate (3)	1.00	1.00	5045	Stop-rotate (3)	1.00	1.00	5045
Vibration (4)	0.95	0.98	1970	Vibration (4)	1.00	0.96	1970
accuracy		0.99	15206	accuracy		0.99	15206
macro avg	0.98	0.97	15206	macro avg	0.98	0.99	15206
weighted avg	0.99	0.99	15206	weighted avg	0.99	0.99	15206

Table 6: Compare results between the client model and the split model.

Model	Accuracy		Precision		Recall		Loss	
	Train	Test	Train	Test	Train	Test	Train	Test
Client Model	0.9917	0.9904	0.9919	0.9907	0.9917	0.9904	0.0276	0.0216
Split Model	0.9897	0.9935	0.9914	0.9940	0.9910	0.9934	0.0037	0.0022

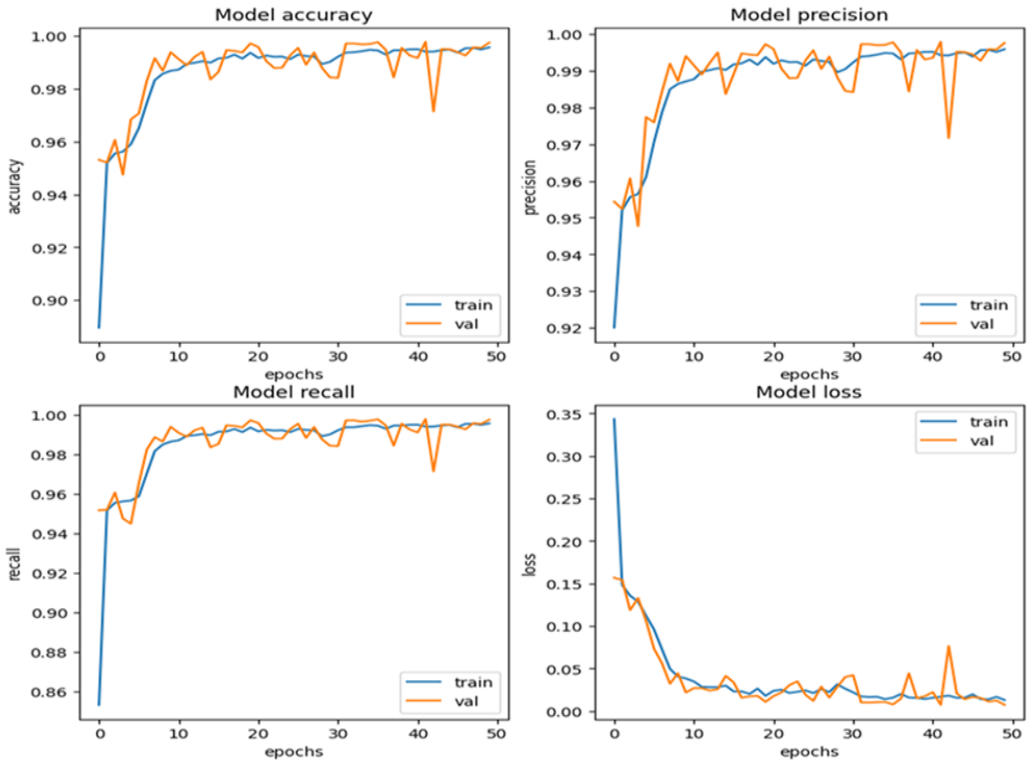


Figure 6: Performance metrics without split (without SL): a) Accuracy, b) precision, c) recall. d) loss.

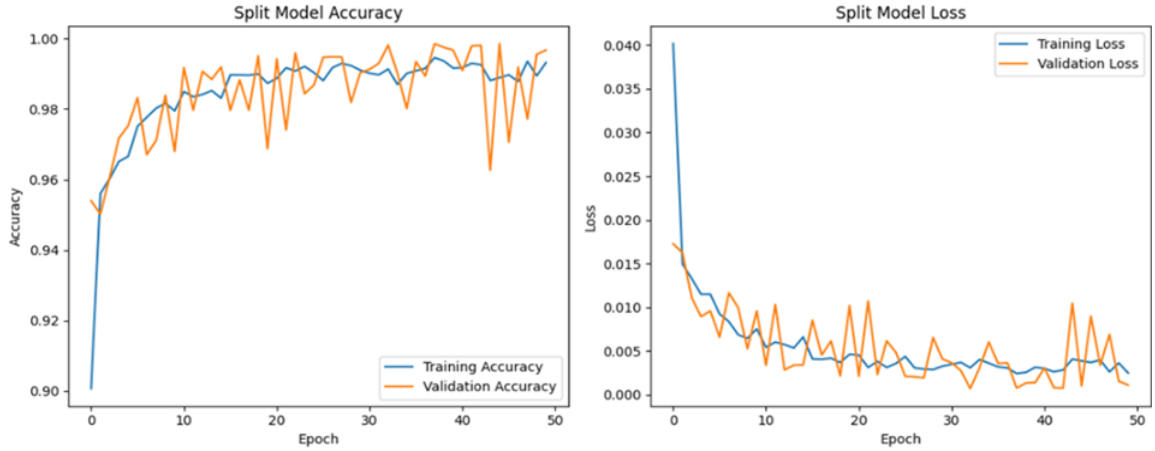


Figure 7: Performance metrics with a split (with SL). a) Accuracy b) Loss.

The proposed approach minimized network overheads based on the transmission of full model updates as compared to federated learning, unlike which full model updates are frequently transmitted. This especially suited the system to the Industrial Internet of Things (IIoT) environment. The Split Learning method proposed has a reasonable trade-off between computational load, communication cost, and data privacy. The split method also showed the smallest loss (0.0022), meaning that it was more robust and was able to learn effectively based on the disseminated data. The split model demonstrates greater understanding of the underlying relationships and patterns of the data and minimizes the loss.

A comparison has been made between the training and testing results of the client model and the split model, as indicated in Table 6. Figure 6 demonstrates the performance indicators of the model in the case when no split learning (SL) was used and includes: a) Accuracy, b) Precision, c) Recall, and d) Loss. These measures depict the behavior of the system at baseline when the model is trained in a centralized way without distributed feature extraction. The findings suggest that the model does not perform as well as it can, but the absence of split learning can restrict data privacy and efficiency of communication in comparison to the suggested SL framework. This base comparison shows the benefits of the SL methodology in enforcing privacy and similar or better predictive performance. Figure 7 shows how the split model is trained, combining the efforts of the three client models. In this configuration, the final layer (output layer) was the cut layer, whereby the activations of all the clients were summed on the cloud server to complete the forward pass. The performance measures presented in the figure reveal that the split model attains stable and high accuracy, and low training and validation loss on all the clients. It proves the fact that the distributed split learning framework is able to combine local feature representations of various clients to obtain a properly trained split model, which can be used to detect faults reliably. The figure also shows how effective the round-robin mechanism and hierarchical feature extraction are in ensuring similar convergence among clients.

The suggested SCINet-based Split Learning (SL) scheme demonstrated a significant enhancement in the efficiency of communication with a reduction of intermediate activations (instead of full model weights) sent. Such a design cut down the communication overhead by about 35%, and the end-to-end latency was kept constant at work in real time. These findings show that the SCINet-SL design can strike a balance between computation and communication as well and is therefore very appropriate in real-time predictive maintenance in the IIoT setting.

6 CONCLUSIONS

This study introduced a lightweight and distributed predictive maintenance based on SL for IIoT environments. Through edge devices and a central server, the proposed system was able to provide high classification accuracy, significantly reducing communications overhead and low training and validation loss, which implies good fault detection and good model generalization. Through transmitting intermediate activations instead of raw sensor data, the framework effectively maintains data privacy and supports real-time operations in resource-constrained edge environments. These results confirm the suitability of the proposed model for industrial practice.

The major contribution of this work relates to the development of a lightweight distributed architecture tailored for IIoT-based predictive maintenance. Unlike conventional centralized approaches, the proposed framework integrates split learning into a real-world multi-sensor monitoring system. The implementation using Raspberry Pi devices connected to vibration, current, and temperature sensors demonstrates the feasibility of deploying privacy-preserving deep learning solutions in realistic industrial settings. Furthermore, a comparative evaluation against centralized learning highlights the efficiency, scalability, and communication advantages of the proposed method.

Future research will focus on extending the architecture toward fully decentralized Split Learning to further eliminate central dependency and improve fault tolerance. Additional investigations will include expanding the experimental setup to a larger number of machines, integrating additional sensor modalities such as acoustic signals, and enriching the dataset with more diverse fault conditions to improve model generalization. Moreover, optimizing the model to enable more processing directly at the edge device level represents an important direction for achieving greater autonomy and reduced latency in IIoT predictive maintenance systems. Funding: “This research received no external funding.” Conflicts of Interest: “The authors declare no conflict of interest.”

REFERENCES

[1] P. Karuppusamy, "Machine learning approach to predictive maintenance in the manufacturing industry comparative study," Journal of Soft Computing Paradigm (JSCP), vol. 2, no. 4, pp. 246-255, 2020, [Online]. Available: https://doi.org/10.36548/jscp.2020.4.006.

[2] A. R. Zarzoor, N. A. S. Al-Jamali, and D. A. A. Qader, "Intrusion detection method for the Internet of Things based on the spiking neural network and decision tree method," International Journal of Electrical and Computer Engineering, vol. 13, no. 2, pp. 2278-2288, 2023, [Online]. Available: https://doi.org/10.11591/ijece.v13i2.pp2278-2288.

[3] O. Alfarraj, "Internet of things with bio-inspired co-evolutionary deep-convolution neural-network approach for detecting road cracks in smart transportation," Neural Computing and Applications, vol. 37, no. 35, pp. 29061-29071, 2025, [Online]. Available: https://link.springer.com/article/10.1007/s00521-020-05401-9.

[4] M. Cakir, M. A. Guvenc, and S. Mistikoglu, "The experimental application of popular machine learning algorithms on predictive maintenance and the design of an IIoT-based condition monitoring system," Computers & Industrial Engineering, vol. 151, p. 106948, 2021, [Online]. Available: https://doi.org/10.1016/j.cie.2020.106948.

[5] M. Sohaib, S. Mushtaq, and J. Uddin, "Deep learning for data-driven predictive maintenance," in Vision, Sensing, and Analytics: Integrative Approaches, pp. 71-95, Cham: Springer International Publishing, 2021, [Online]. Available: https://doi.org/10.1007/978-3-030-75490-7_3.

[6] Z. Li, Q. He, and J. Li, "A survey of deep learning-driven architecture for predictive maintenance," Engineering Applications of Artificial Intelligence, vol. 133, p. 108285, 2024, [Online]. Available: https://doi.org/10.1016/j.engappai.2024.108285.

[7] Q. Duan, S. Hu, R. Deng, and Z. Lu, "Combined federated and split learning in edge computing for ubiquitous intelligence in the internet of things: State-of-the-art and future directions," Sensors, vol. 22, no. 16, p. 5983, 2022, [Online]. Available: https://doi.org/10.3390/s22165983.

[8] Z. Zhao, Y. Mao, Y. Liu, L. Song, Y. Ouyang, X. Chen, and W. Ding, "Towards efficient communications in federated learning: A contemporary survey," Journal of the Franklin Institute, vol. 360, no. 12, pp. 8669-8703, 2023, [Online]. Available: https://doi.org/10.1016/j.jfranklin.2022.12.053.

[9] S. M. S. Mohammad Abadi, S. Zawad, F. Yan, and L. Yang, "Speed Up Federated Learning in Heterogeneous Environments: A Dynamic Tiering Approach," IEEE Internet of Things Journal, 2024, [Online]. Available: https://doi.org/10.1109/JIOT.2024.3487473.

[10] R. W. Abdalah, O. F. Abdulateef, and A. H. Hamad, "A Predictive Maintenance System Based on Industrial Internet of Things for Multimachine Multiclass Using Deep Neural Network," Journal Europeen des Systemes Automatises, vol. 58, no. 2, 2025.

[11] A. Jalali, M. Soltany, M. Greenspan, and A. Etemad, "Learning Time-Series Representations by Hierarchical Uniformity-Tolerance Latent Balancing," arXiv preprint arXiv:2510.01658, 2025, [Online]. Available: https://doi.org/10.48550/arXiv.2510.01658.

[12] Z. Hu, T. Zhou, B. Wu, C. Chen, and Y. Wang, "A review and experimental evaluation of split learning," Future Internet, vol. 17, no. 2, p. 87, 2025, [Online]. Available: https://doi.org/10.3390/fi17020087.

[13] S. Guo, X. Zhang, F. Yang, T. Zhang, Y. Gan, T. Xiang, and Y. Liu, "Robust and Privacy-Preserving Collaborative Learning: A Comprehensive Survey," arXiv, arXiv:2112.10183, 2021, [Online]. Available: https://doi.org/10.48550/arXiv.2112.10183.

[14] Z. Hu, T. Zhou, B. Wu, C. Chen, and Y. Wang, "A review and experimental evaluation of split learning," Future Internet, vol. 17, no. 2, p. 87, 2025.

[15] G. H. Lokesh, G. S. Hukkeri, N. Z. Jhanjhi, and H. Lin, Split Federated Learning for Secure IoT Applications: Concepts, Frameworks, Applications, and Case Studies, 2024, [Online]. Available: https://doi.org/10.1049/PBSE025E.

[16] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6G edge networks," IEEE Wireless Communications, vol. 31, no. 4, pp. 170-176, 2024, [Online]. Available: https://doi.org/10.1109/MWC.014.2300319.

[17] J. Ryu, D. Won, and Y. Lee, "A Study of Split Learning Model," in IMCOM, pp. 1-4, Jan. 2022, [Online]. Available: https://janicejihyeon.github.io/files/IMCOM2022_Jihyeon.pdf.

[18] M. Liu, A. Zeng, M. Chen, Z. Xu, Q. Lai, L. Ma, and Q. Xu, "Scinet: Time series modeling and forecasting with sample convolution and interaction," Advances in Neural Information Processing Systems, vol. 35, pp. 5816-5828, 2022.

[19] J. Zhang, C. Zhang, S. Xu, G. Liu, H. Fei, and L. Wu, "Remaining life prediction of bearings based on improved IF-SCINet," IEEE Access, 2024, [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3355978.

[20] K. M. Hosny, A. Magdi, A. Salah, O. El-Komy, and N. A. Lashin, "Internet of things applications using Raspberry Pi: A survey," International Journal of Electrical & Computer Engineering, vol. 13, no. 1, pp. 902-910, 2023, [Online]. Available: https://doi.org/10.11591/ijece.v13i1.pp902-910.

[21] A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, "Machine learning and deep learning approaches for cybersecurity: A review," IEEE Access, vol. 10, pp. 19572-19585, 2022, [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3151248.

[22] R. H. Abood and A. H. Hamad, "Multi-Label Diabetic Retinopathy Detection Using Transfer Learning Based Convolutional Neural Network," [Online]. Available: https://doi.org/10.54216/fpa.170221.