International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 16, No.1 (Sep-24)

http://dx.doi.org/10.12785/ijcds/1601100

Industrial IoT Sensor Data Federation in an Enterprise for an Early Failure Prediction

Sachin Bhoite[1], **Chandrashekhar H. Patil**[1,*] **and Harshali Patil**[2]

[*]***Corresponding author: Chandrashekhar H. Patil, chpatil.mca@gmail.com***
[1]*Department of Computer Science, Dr. Vishwanath Karad MIT World Peace University, Pune, MH, India.*
[2]*Sri Balaji University, Pune, India, MH, India.*

Received 28 Mar. 2024, Revised 2 Jun. 2024, Accepted 3 Jun. 2024, Published 19 Sep. 2024

Abstract: The recent availability of powerful (SBC) Single Board Computing devices has facilitated edge computing, filled a gap with lower power consumption at the edge. Preventive maintenance intervention in the industry is needed. These predictions with data privacy and accuracy to take care of chronic spare replacements before things fail. We are proposing preventive maintenance procedures based on (IIoT) Industrial Internet of Things data from multiple sensors installed in an industrial setup across a varied geography. The SBC ensured low powered 15W of power operation mode and was adequately cooled with a passive aluminium heat-sink and fans. We are proposing a unique method of federation, specifically, using HDF5 model file transfer. Preset cron jobs at the clients allow real-time federation as a quick solution using off-the-shelf hardware. The setup has a central server or alternatively a cloud server for fallback, in the monitoring station and is implemented using Split Federation and Linear, DNN, CNN, RNN models. Federated Learning (FL) models were used to predict the sensor values and make decisions. The Machine Learning (ML) techniques only operated at the edge. Data privacy is upheld and maintained. The quick and simple approach can help in a cheaper implementation in public service projects where site data needs to be private. Even the possibility of power cuts in rural areas will not affect the federation and decision making can happen even in the harshest of field situations. This has a lot of impact in decentralized decision making. Failure patterns can be identified and in general, an accurate model can be generated with limited resources.

Keywords: Synchronized models, Federated Learning, MQTT, Edge Computing, Single Board Computers, Industrial Internet of Things, Split Learning, Split Federation, Predictive Maintenance, Algorithmic preventive maintenance, Failure Prediction.

1. Introduction

FL is a distributed machine learning technique wherein various devices on the edge, first train an ML model on their data [1][2]. They do not share their data with other devices. This method is particularly relevant for Industry 4.0 and eventually in Industry 5.0, where there is a need to analyze vast amounts of data generated by various connected devices [3], without compromising data privacy and security. In Industry 4.0 and 5.0, Federated Learning will be implemented to train models on data collected from connected machines and devices, allowing for real-time analysis and improved decision-making in areas such as predictive maintenance, quality control, and process optimization [4] [5]. We were motivated to pursue this approach to create a solution having low cost and easy to implement.

We have implemented a split and federated learning approach wherein various participants train their models on their data which is local only to themselves. This is then shared with a central coordinator without including the localized data. All the learning is eventually aggregated and produces a global ML model [6] [7]. This research has the objective to achieve a simple method of federation at scale using off-the-shelf and easily available components. It also had to be quick to implement in rural industrial settings where the possibility of power cuts had to be factored in.

Privacy takes precedence when dealing with sensitive sensor data. Related IIoT device data can be modelled using the same algorithm. Among the sensors, those which generate sensitive data should be secured well. Different sensor data can be treated with varying levels of secure algorithms as per the application. Electronic devices generating data securely can be aggregated at the edge using an encryption layer. Ever-increasing data points can be resampled as per the forecast range in question. Before establishing a connection with the edge, the security protocols need to be decided upon. Requesting or publishing data to the broker can be done with a secure key or password. Along the data flow paths, a minimum number of hops ensures low latency. Online data flow paths are recommended to be implemented with a secure transport layer [8].

In Industry 4.0 and 5.0, federated learning can have an impact on improving collaboration and the sharing of learning among different organizations while maintaining data privacy and security. It can be used in various applications such as predictive maintenance, predictive quality control, and supply chain optimization [9]. By training models on decentralized data, federated learning enables organizations to leverage data from multiple sources and make better use of their data, ultimately leading to improved accuracy and performance in Industry 4.0 and 5.0 applications [7].

The current paper is organized into five sections. Introduction, motivation and applications are covered in the first section. In part two, a detail literature study was carried out. Section three is devoted to the adopted methodology, experimental work and setup of the experiment. In section four, the results and findings are discussed. In the final section, the investigation is concluded, and future scope is discussed.

2. Literature review

Christopher Briggs et.al. [3] suggest various FL strategies. This is then followed by a personalization step showing an improvement in model's performance. They show that FL can achieve this improvement by reducing the computation load when compared with localized learning. They provide information about aggregating predictions to be able to build private load forecasting applications. Modern distributed machine learning like FL trained load forecasting (LSTM) Long Short-Term Memory models. At the same time, it preserved the privacy of data in the field of power consumption at the customer's locations. It would enable a wider implementation of smart energy meters while being concerned of privacy at the same time. A comparison among different FL training methods and other benchmarks from regular training strategies was explored. The effect of the same on the level of forecasting was studied. Analysis of FL methods was done, comparing it to a variant of FL designed to perform well to the task of forecasting the loads. They also evaluated numerous efficiency issues in computation, in the FL system used in forecasting [3].

Mojtaba Vaezi et.al. [4] suggest how the paradigm will shift in the next decade is made. Integration of new technologies associated with each other, like (AI) Artificial Intelligence and networks which are not terrestrial, are detailed upon. The potential of implementing deep-learning methods alongside emerging techniques in FL has been discussed. Their impact on improving communication in IoT networks has been discussed. Even future research pathways going further than the current 5G technology in IoT networking are discussed. Digital transformation which was initiated by IoT have inspired trends across academia as well as the industry. It was identified that the use of IoT creates real global interconnect. AI controls the IoT enabled gadgets as well as the decisions taken by these gadgets. The so-called 'edge', increases its proximity to the cloud. The authors identified several issues in the privacy matters and security concerns in these situations. The increase in IoT security issues alongside the rapid global interconnect, makes IoT security threats and the related issues of data privacy concerning. This will need innovative and novel ideas to counter the threats in this domain in the near future [4].

Yi Liu et.al. [6] had conducted a research where Convolutional Neural Network (CNN) was used. The authors used the CNN-LSTM model. The finer subtleties were captured using units of CNN. This method maintains the benefits of LSTM to predict data from a time series. They used a gradient compression technique to yield results in predicting in the moment getting good results in detecting anomalies as well. Bandwidth needs were reduced and it even improved communication efficiency. It was noted that experiments based on datasets from the real world, had good results in detecting anomalies accurately using the framework proposed by them. It improved on traditional ways of doing the same reducing overheads in communication. Finally, they proposed an anomaly detection score. They normalized the time series sensor data collected [6].

IoT data generated from multiple devices used in sensing humidity in a study by Mohamed Amine Ferrag et.al. distances using ultrasonic methods, sensors detecting the level of water in applications, patient heart rate, flame, acidity level pH and moisture in soil etc., were collected. The study analyzed some attacks in the connectivity and communication protocols. (DoS) Denial of Services in network attacks, man-in-the-middle type of attacks, (DDoS) Distributed Denial of Services in attacks, gathering of private information were studied. (SQL) Structured Query Language Injection and other type of malware based attacks were also elaborated upon. Features extracted from varied sources like alerts, syslogs, resources, network data traffic, were studied and new features with high inter-relations were discovered. At the end of processing the above mentioned data set with security fall outs, the authors provided a preliminary exploration and then they analysed the sensor data. They evaluated different ML strategies in both modes : federated as well as central-server based learning [7].

Edge Computing (EC) is a scenario where sensors can process the data collected. Even stacked intermediate elements can process the data. The methods used in edge computing, reduce bandwidth and communication related costs. Processing speeds increase if the edge device is powerful enough. In one study, Taimur Hafeez et.al. explored IIoT methods to carry out (PM) Predictive Maintenance. They discussed how the collected sensor readings can be processed and where it can be analyzed. They presented sampling concepts along with techniques to reduce data communications. The quantity of data that was transmitted to the cloud, was diminished. Accuracy might be lost when ML algorithms have to deal with reduction in data. Alternative approaches move ML based algorithms nearer to the source of data and achieve a reduction in transmitted data. These techniques are categorized broadly into three types: Device & Edge, Edge & Cloud, Device & Cloud. The authors demonstrated an architecture in which edge computing can be implemented for sensor data reduction for preventive maintenance of equipment. In instances where the connection between the sensor node and the central server where the readings are processed, fails, there was a study which used ML and monitored how the devices performed. These methods were only edge dependent. The ML gets updated as soon as analytic processes running on the sensor information, locates anomalies at the edge layer. The edge connects to the cloud to push the update. The method used techniques like SNN, ANN. Some other methods like CNN, HOG etc. were also tried [9].

Abdul Rehman et.al. researched on an FL based framework that can aggregate models from contributions from different clients. The data set it used could train models on an individual basis using DNN (Deep Neural Network). Every client examines the results three times resulting in over 80% accuracy. It found that their framework could detect attack from alternate channels. They ensured that the system logs were purged of information that would compromise the privacy of individuals by anonymizing the data prior to training the model [10].

Mahmoud Parto et.al. mentioned a time based architecture for IoT. It mentioned techniques for ML at scale. The study focused on the processes of manufacture. They had a general architecture with three layers. They created an edge computing layer with sensors. Post that, they computed AI tasks in the fog, as they called the intermediate layer, and a central cloud server was in charge of federating the models. The whole setup was presented as a (FL) Federated Learning system with prior processing getting done at the edge, and the ML layer trained models incrementally in what was termed the fog layer. Aggregation of the models happened centrally in the cloud. High performance was achieved with this scalable architecture using hardware with fewer resources. Stacks of Raspberry Pi 3B devices were clustered and deployed, balancing minimum storage and computing power with better performance [11].

Mingqian Liu et.al. mentioned in a paper how the reduced the frequency and pre-treated the signal from the nodes. The representations of the signals were then trained with fusing the features. FL methods combined with deep learning were used to classify the signals at each node. The performance of the classification was found to be good when the results at each sensor were aggregated to the central aggregator [6].

Yung-Lin Hsu and Hung-Yu Wei saved a lot of energy by having the processes execute at the edge and also reduced loss in performance [12]. Xianyi Cheng et al. used fast performing classifiers that culminated in accurate results using stage classifiers. Decision trees were developed for each subset of sensors. The predictions were accurate when used along with state transitions [13]. Mi Wen et al. used deep learning to detect the theft of power on the grid. (TCN) Temporal Convolutional Networks were used and a lot of experiments resulted in a very accurate detection rate inspite of lower compute power. FL frameworks were deployed to quantify the footprint of energy usage and carbon emitted in a decentralized setting. The authors architected green designs for FL structures [14].

Overall, a lot of new paradigms of learning have evolved in the industrial settings recently. One such is the continuous learning or (RL) Reinforcement Learning paradigm by Stefano Savazzi et al. where the model is trained at the edge itself in IIoT devices involving a periodic repetition continuously. The process repeats as per the varying processes in the industry depending on a timely schedule [15].

Andrew Hard et.al. [16] Concluded that the federated algorithm, which facilitates model training on a more finely tuned and high-quality dataset tailored to this particular application, surpasses conventional server-based training in terms of predictive precision. The authors execute a comparative assessment between server-based training employing stochastic gradient descent and training carried out on individual client devices using the FederatedAveraging algorithm.

The federated learning framework empowers users through enhanced control over their data usage and streamlines the incorporation of privacy safeguards as an inherent feature through distributed training and data aggregation across a diverse array of client devices. Their primary aim is to enhance the predictive text functionality within a smartphone-based virtual keyboard [16].

Karim Gamal, Ahmed Gaber et al. [17] found that The empirical results derived from various model configurations indicate that, in either uncontaminated or adversarial scenarios, federated learning achieves similar performance to traditional centralized training when predicting emojis in multiple languages, even when the data comes from diverse sources with varying distributions.

To conduct this research, emoji prediction datasets were gathered from two sources: Twitter and the SemEval emoji datasets. These datasets served as the basis for training and evaluating different transformer model settings. Notably, the trained transformer models demonstrate superior performance compared to alternative techniques when tested on the SemEval emoji dataset. Furthermore, federated learning retains its inherent privacy and distributed advantages in this context [17].

Faris F. Gulamali and Girish N. Nadkarni et.al. results demonstrate that federated models outperformed their local counterparts, even when assessed on local data within the test dataset. Their performance was on par with models using pooled data. Federated learning presents a viable alternative to the conventional single-institution approach while circumventing the challenges associated with data sharing. Models are generated and updated on-site to achieve specific learning objectives. To illustrate its effectiveness, the authors provide a practical example involving COVID-19-associated AKI. In the context of cross-silo federated learning, data remains localized, and the raw data is retained at its source. Notably, the improvements in performance at individual hospitals showed an inverse relationship with dataset size, implying that smaller hospitals have substantial room for improvement with federated learning methodologies [18].

After a detailed study of literature, we concluded that there has been a rapid shift in paradigms in the industry. IIoT and the related ML techniques used to analyze data have been slowly moving from the central server to the edge as time goes by. This means more computation can happen at the edge, and the central server is relieved of the processing load. This also means more data security and added privacy in the future for big data analytics of industrial sensors to aid in decision-making in the industry. The research gaps identified in the literature are 1) cost 2) data security 3) computing at edge. These research gaps are fulfilled by this research.

3. The methodology

In experiment setup multiple sensor data forecast for multiple outputs. After obtaining multi-time stepping, developed windows in the data for it to be reusable for Linear, DNN, CNN, RNN models.

With the advent of powerful SBC (Single Board Computing), the processing power available at the edge has grown tremendously in the past couple of years. The suggested layers in the IIoT network with high-frequency sensor data are the centralized cloud, and the edge device further connected to the sensors on the machinery. The SBC Edge device uses an MQTT broker and for some IIoT devices, their Modbus layers to aggregate the sensor data in the IIoT network. This data is then analyzed at the edge [11]. The time-series data collected is processed at the edge itself and the central server applies federated learning methods to analyze, forecast, and create maintenance suggestions, and the best time of operation for sensitive machinery in the industry. This leads to improved efficiency, reliability, and security in industrial processes.

Some commonly used federated learning algorithms for IIoT include [19][20]: Federated Averaging (FedAvg), Split Learning. We have used a combination of FedAvg and Split Learning in our experiment. Time series sensor data prediction refers to the process of being able to predict the next values in a set of data generated by sensors over time. This is an important application in Industry 4.0, as it can help in forecasting equipment failures, detecting anomalies, and improving operational efficiency.

A few machine learning techniques used to make time-series data predictions are Autoregressive Models, Moving Average Models, Autoregressive Integrated Moving Average Models, Seasonal Autoregressive and Integrated Moving Average Models, Recurrent Neural Networks, Long Short Term Memory.

A. *Experiment Setup*

In an industry, various machines on each shop floor are connected wirelessly as well as wired to the network to relay sensor data to the central server. These typically used ESP32 / ESP8266 based implementations in their architecture. Each one of the sensor relays communicated with Message Queue Telemetry Transport (MQTT) broker running on the edge device at regular intervals, relaying sensor data which was processed only at the edge. The resulting models were transferred to the cloud, as part of the split and federated learning process. Our edge setup consists of multiple SBCs interconnected to form an edge cluster. This setup performs parallel processing at the edge and runs the code from the central server.

Hardware and connections: We have used multiple Raspberry Pi 4 – 8GB boards along with a single Raspberry Pi 3B+ running as a storage controller, to run the open source software stack in this experiment setup. Each node relaying the data runs on either an STM32 or an ESP32 micro controller based board with custom-coded embedded software that relays the telemetry data to the MQTT server. The embedded code was programmed with a telemetry period of 20 seconds to keep the data up to date for processing.

Cluster: Our SBC cluster was assembled using the following components as depicted in Fig 1.

Units utilized: Raspberry Pi 4 - 8GB boards – 2 units, Raspberry Pi 3B+ Storage controller – 1 unit. USB SSD boot drives – one for each SBC. Ethernet cables (for connecting the SBCs to the network switch / router). Network switch / router with 8 ports to accommodate the SBCs. Power adapters for the SBCs – capable of delivering 5.1V at 3 Amperes each. Cooling – solid aluminum heat sinks with fans to keep the boards at less than 45 degrees centigrade.

Open source software stack: We chose commonly available open source software components like the Raspberry Pi OS alongside a stack consisting of real time databases like InfluxDB. Open source observability software like Grafana, Message queueing using the Mosquitto MQTT layer, SSH clients like PuTTY for remote headless access to the SBCs and Ansible – an open source automation tool, were used to manage the cluster throughout the experiment.

Software flow for the cluster: This was our general workflow that we implemented to get the cluster up and running at the edge. Preparing each of the SBCs in the cluster: The latest version of Raspberry Pi OS was downloaded and flashed onto the SSD. Software used can be the Raspberry Pi Imager tool or Balena Etcher. The USB adapted SSD

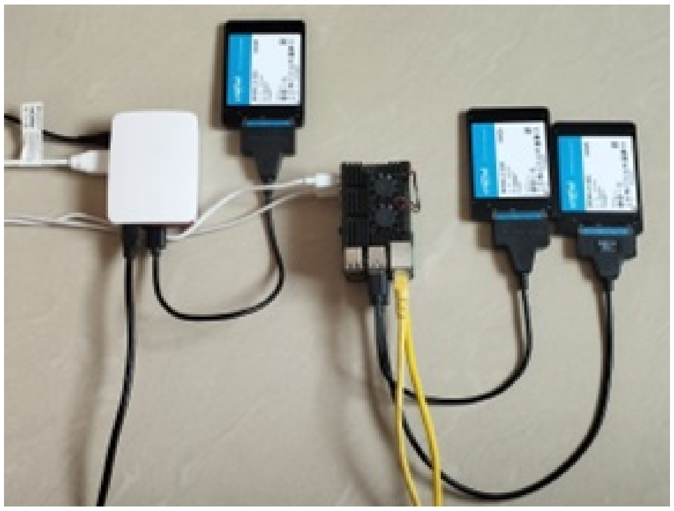


Figure 1. SBC cluster connections with two RPi 4 – 8GB and one RPi 3B+ Storage Controller

cards – 500GB was plugged in into the Raspberry Pi boards USB-3 slots as shown in Fig 2.

Figure 2. USB SSDs ensure quick bootup and optimal edge performance with 300Mb/s throughput

Setting up networking in the cluster: Each SBC is connected to the network switch / router using Ethernet cables. It is ensured that all SBCs and the terminal computer are on the same local network.
Powering on the SBCs: The power adapters to each SBC is connected and turned on, one by one. Each SBC in the cluster takes about 2 minutes to boot up completely. Once booted, the cluster is ready for use at the edge.
Configuring the SBCs: The terminal computer is connected to the same network as the cluster. DHCP IP addresses assigned to each SBC is got from the router's configuration. An IP scanner tool can be used alternatively or even running nmap over SSH, works well. Each SBC can be accessed via SSH using the IP address and the SSH client such as PuTTY.

Configuring the cluster: On each SBC, Ansible is installed by running the following command: sudo apt update sudo apt install ansible An inventory file listing the IP addresses or hostnames of all the SBCs in the cluster is created. It is then saved for use in scripts later. An Ansible playbook was created to define the desired configuration for the cluster, such as software installations or system settings. The Ansible playbook was executed using the following command: ansible-playbook -i ¡IP Addresses¿ ¡playbook¿
Testing the cluster: Once the above configuration is complete, each cluster can be tested by executing parallel tasks or distributing workload among the SBCs. Parallel commands can be executed across all the SBCs using the Ansible command: ansible all -i ¡IP Addresses¿ -a "¡user commands¿"
Power consumption: We found that the typical power consumption of this edge cluster setup in Fig 3. was around 45W at peak processing loads. The board temperatures averaged around 45 degrees centigrade.

Figure 3. Edge Cluster power usage is typically around 45W at peak processing demand

Data flow: The C code in the embedded devices creates a JSON formatted message that is relayed to the MQTT server daemon running in the SBC cluster. The MQTT broker relays data in the JSON formatted message. This is processed using Telegraf and then is stored into the time series database InfluxDB.

Data aggregation: The TICK stack we chose is a set of open-source tools for managing and processing time-series data. The Telegraf module is a plugin-driven server agent that collects, processes, and delivers metrics and events from multiple sources like sensors into the chosen database. The InfluxDB database is scalable and is a very high-performance time-series database which is designed specifically to handle extremely large volumes of time-stamped data. We used this to store the sensor stream data. Chronograf is a web-based user interface (UI) provided by the TICK stack that helped us visualize and explore the data stored in the time-series database. This provides the tools to create dashboards and helps in managing alerts based on preset conditions as depicted in Fig 4. Kapacitor is a processing engine in the stack that enables real-time streaming along with real-time data processing. It allowed us to define and execute tasks on the data which included aggregations, downsampling of the data, anomaly detection in the data stream, and also alerted us based on preset conditions.

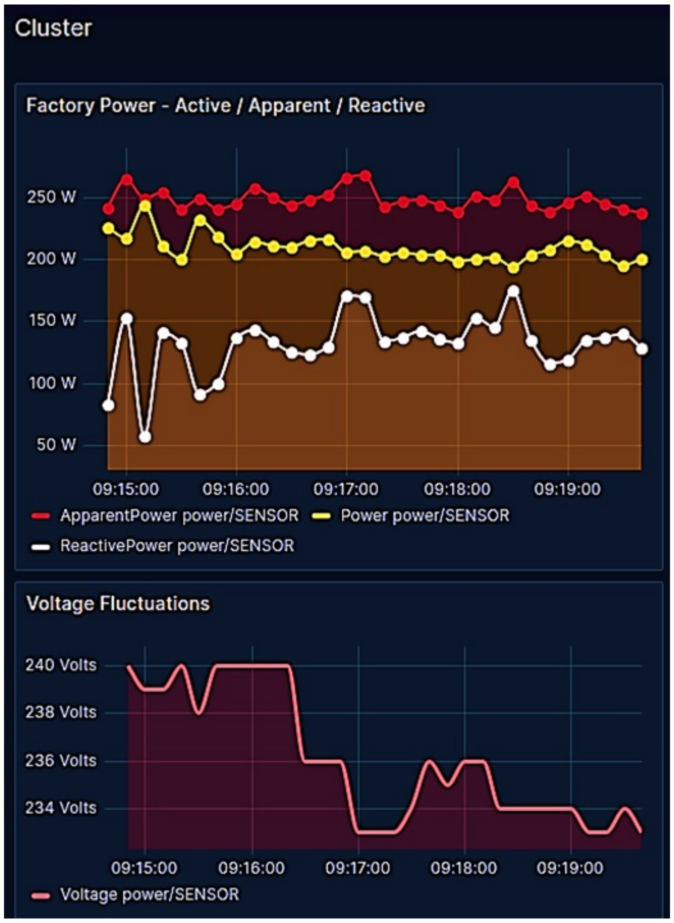


Figure 4. A dashboard from Data stream

To aggregate our sensor stream data using the TICK stack described above, we followed the steps as described below:

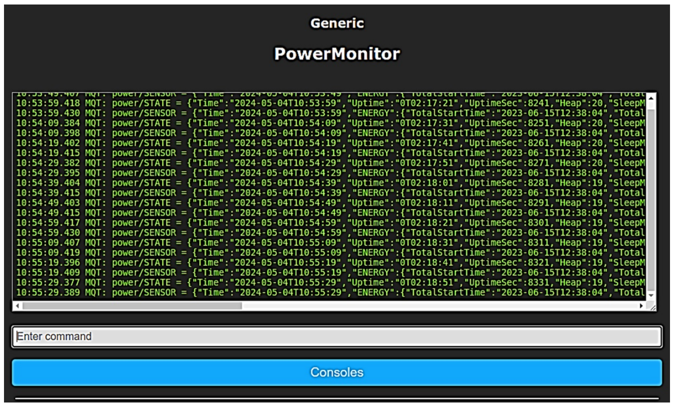
We first installed and configured Telegraf. We set up Telegraf on the cluster at the edge where the sensors are located. Telegraf provides various input plugins to collect data from different sources, such as MQTT, SNMP, or our custom written scripts. We configured Telegraf, specifically the telegraf.conf file, to collect data from the sensors over the wi-fi network and then send it ahead into an InfluxDB bucket via MQTT.

We then installed and configured InfluxDB. Specific care was taken to configure the InfluxDB bucket to receive and store the incoming sensor data sent by Telegraf. We define a bucket with retention policy to always keep data forever, since we would need it for comparisons later. To verify data ingestion by the database, we ensured that Telegraf is successfully sending data to InfluxDB. This was done by checking the InfluxDB bucket using the web interface, to confirm that the data is being stored correctly.

Next we explored and visualized the data using Chronograf. The Chronograf web user interface was used to visualize the sensor data stored in InfluxDB. We created dashboards and configured queries to display the data in meaningful ways using the Flux query language. Since Chronograf provides various visualization options, such as line graphs, bar charts, and scatter plots, it was easy to get a real time view of the data of different time slices on the fly.

We could process the data using Kapacitor. Performing real-time calculations or analysis on the sensor data, we configured Kapacitor by defining tasks in Kapacitor using its Domain Specific Language (DSL) to perform aggregations, downsampling the data, detecting anomalies, and other data transformations. We also set up different alerting rules in Kapacitor and generated notifications based on predefined conditions.

Data Collection: Data was collected for 4 gauging machines connected in a single shop floor at a factory in an Industrial Estate at Hadapsar in Pune. The machines were supplied power through the power sensor which is calibrated to send data to an MQTT every 20 seconds. The console of the power monitor and its data logging setup is shown in the Fig. 5 : The dataset regarding voltage, machines information, timestamp, sensor information, sensor data is stored in the CSV file. Total 14000 records are stored and the classifier is trained on the dataset.



```
10:53:49.407 MQT: power/SENSOR = {"Time":"2024-05-04T10:53:49","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:53:59.418 MQT: power/STATE = {"Time":"2024-05-04T10:53:59","Uptime":"0T02:17:21","UptimeSec":8241,"Heap":20,"SleepM
10:53:59.430 MQT: power/SENSOR = {"Time":"2024-05-04T10:53:59","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:09.384 MQT: power/STATE = {"Time":"2024-05-04T10:54:09","Uptime":"0T02:17:31","UptimeSec":8251,"Heap":20,"SleepM
10:54:09.398 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:09","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:19.402 MQT: power/STATE = {"Time":"2024-05-04T10:54:19","Uptime":"0T02:17:41","UptimeSec":8261,"Heap":20,"SleepM
10:54:19.415 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:19","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:29.382 MQT: power/STATE = {"Time":"2024-05-04T10:54:29","Uptime":"0T02:17:51","UptimeSec":8271,"Heap":20,"SleepM
10:54:29.395 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:29","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:39.404 MQT: power/STATE = {"Time":"2024-05-04T10:54:39","Uptime":"0T02:18:01","UptimeSec":8281,"Heap":19,"SleepM
10:54:39.415 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:39","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:49.403 MQT: power/STATE = {"Time":"2024-05-04T10:54:49","Uptime":"0T02:18:11","UptimeSec":8291,"Heap":19,"SleepM
10:54:49.415 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:49","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:54:59.417 MQT: power/STATE = {"Time":"2024-05-04T10:54:59","Uptime":"0T02:18:21","UptimeSec":8301,"Heap":19,"SleepM
10:54:59.430 MQT: power/SENSOR = {"Time":"2024-05-04T10:54:59","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:55:09.407 MQT: power/STATE = {"Time":"2024-05-04T10:55:09","Uptime":"0T02:18:31","UptimeSec":8311,"Heap":19,"SleepM
10:55:09.419 MQT: power/SENSOR = {"Time":"2024-05-04T10:55:09","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:55:19.396 MQT: power/STATE = {"Time":"2024-05-04T10:55:19","Uptime":"0T02:18:41","UptimeSec":8321,"Heap":19,"SleepM
10:55:19.409 MQT: power/SENSOR = {"Time":"2024-05-04T10:55:19","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
10:55:29.377 MQT: power/STATE = {"Time":"2024-05-04T10:55:29","Uptime":"0T02:18:51","UptimeSec":8331,"Heap":19,"SleepM
10:55:29.389 MQT: power/SENSOR = {"Time":"2024-05-04T10:55:29","ENERGY":{"TotalStartTime":"2023-06-15T12:38:04","Total
```

Figure 5. Console for Data Gathering

Data analytics: The real time stream of the telemetry data from the shop floor is presented in customized dashboards reflecting the current situation at the factory floor. Similarly, predicted values, after processing the telemetry data stream are also presented in real time projected-timeline dashboards shown in fig 6.

Data federation: Tensorflow running on the same SBC cluster would have ensured that processing could have happened on the edge. But since Tensorflow does not yet

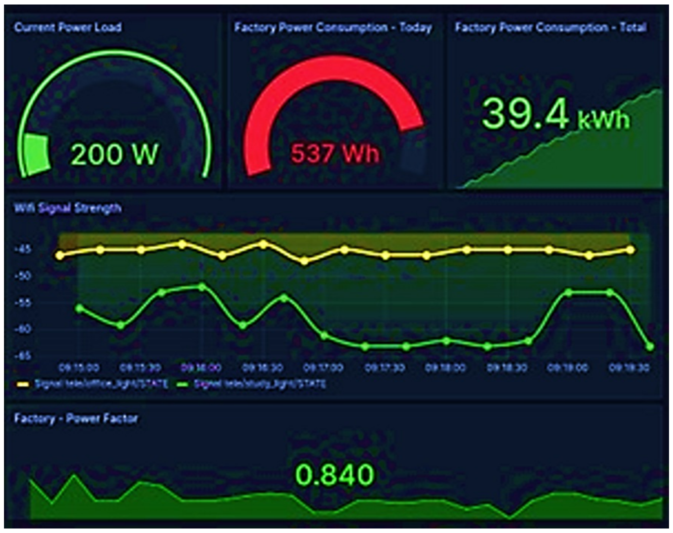


Figure 6. Edge Cluster power usage is typically around 45W at peak processing demand

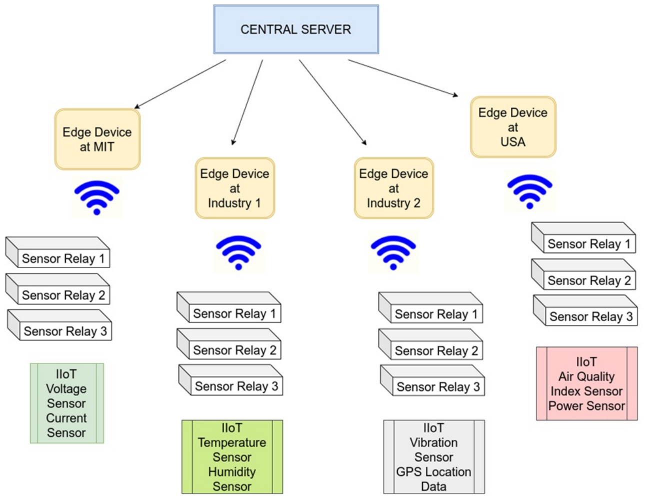

CENTRAL SERVER
Edge Device at MIT
Edge Device at Industry 1
Edge Device at Industry 2
Edge Device at USA
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Voltage Sensor Current Sensor
IIoT Temperature Sensor Humidity Sensor
IIoT Vibration Sensor GPS Location Data
IIoT Air Quality Index Sensor Power Sensor

Figure 7. A flow of the Enterprise Server and Edge Devices with IIoT sensor relay

exist for this particular variant of ARM64 processor, Jupyter Notebook was installed at the edge to run the process. The files were synchronized using rsync and a cron job managed it on a schedule.

Predictive analysis: The learning process on each SBC cluster transported through rsync. Eventually when Tensorflow can be installed at the edge SBC cluster on these ARM64 boards, the actual code can run at the edge without having the rsync. Distributed computing frameworks like Kubernetes or Apache Spark to utilize the cluster's capabilities for distributed processing or running containerized applications could be explored in the future.

Fig.7 below, shows the general structure of the central server federating data at the edge devices and their individual sensor relays at the edge. Various time-series data like voltage, current, power, temperature, vibration, air quality, humidity etc. are relayed to the edge device. A few sensor nodes were wired to the network using ethernet cables, but most sensor devices relayed the data and operated over wireless wi-fi connections.

B. *Experiment Flow:*

The edge devices at each location do not relay the sensor data to central cloud. The reading information collected from the sensors is processed at the edge device itself, thereby maintaining the data privacy at each location.

The edge has become increasingly powerful, with most analytical processes completed in this layer. Businesses requiring new solutions on the periphery, combined with a rapid increase of data from these sensors, have begun migrating to process most computation on the edge. Some edge nodes collect lots of private data which is modeled at the edge itself. Software plays an important role in managing the edge data layer with (API) Application Programming Interface access, locally served dashboards combining integration with automation and controls, and delivery to alerting systems. All these systems in tandem with the cloud native analytical software at the head quarters, are used to make real-time business decisions, both at the edge as well as in the wholistic level. Edge and cloud data, have strict privacy policies at times. There are many considerations when choosing the edge or cloud for storage and compute. Finally developers want to ensure these two data layers work together in an efficient way, ensuring privacy and delivering centralized business insights in near real-time. Federated Learning has an important role to play in this type of situation.

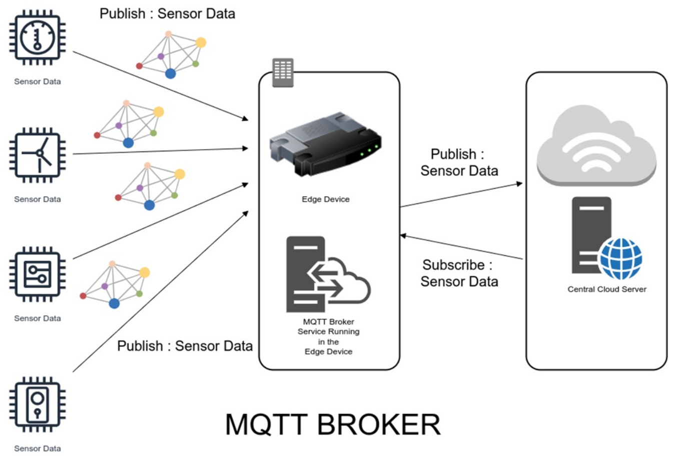


Figure 8. MQTT – Publish and Subscribe

In HTTP, the data is transferred between the client and the server. In (MQTT) – Message Queue Telemetry Transport, the exchange of data is facilitated through publish-subscribe communication protocols as depicted in Fig.8. This was initially developed by IBM and is now popular in the IIoT space [21][21]]. MQTT addresses the need for rapid communication in IIoT and stays distributed at the same time. Another popular protocols for sensor data exchange is Data Distribution Service (DDS). In addition to that, there exist alternatives like XMPP, RabbitMQ etc. This experiment uses MQTT and HTTP for the IIoT gateways, sensor devices and edge devices .

Since we are experimenting with a research FL simulation , we have not implemented TFF functions as yet. TFF is not yet primed for installation on Raspberry Pi 4 which is the edge device of our choice [22]. The code is always executed and tested eliminating any tff.* reference in the code. This can be used again without the presence of TFF. Client training loops in FedAvg is deployed as shown in fig 9.

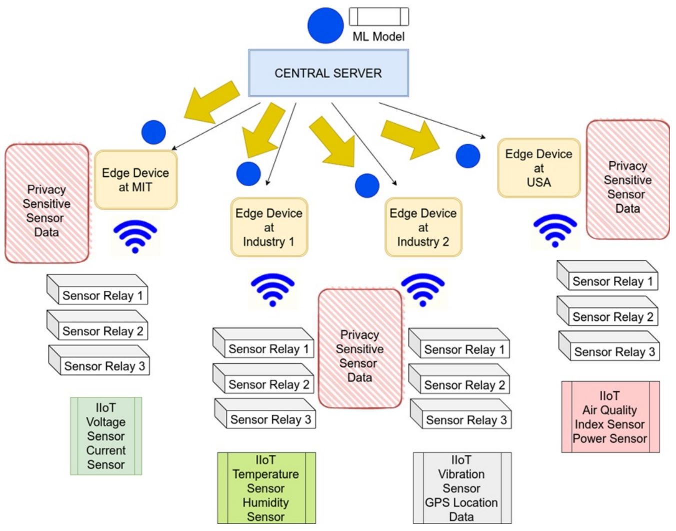
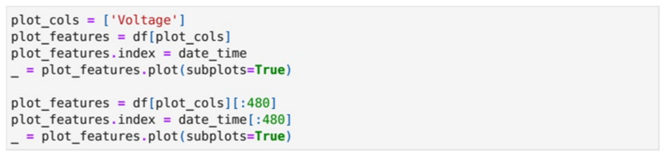

ML Model
CENTRAL SERVER
Privacy Sensitive Sensor Data
Edge Device at MIT
Edge Device at Industry 1
Edge Device at Industry 2
Edge Device at USA
Privacy Sensitive Sensor Data
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
Privacy Sensitive Sensor Data
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Voltage Sensor Current Sensor
IIoT Temperature Sensor Humidity Sensor
IIoT Vibration Sensor GPS Location Data
IIoT Air Quality Index Sensor Power Sensor

Figure 9. The Federated Learning Model

FL is an ideal technology for IoT applications as its efficiency and protection model make it well-suited to this type of application. Some notable examples of these applications include wearable devices, self-driving cars, and smart intelligent homes. This turns complex with plenty of data waiting to be collected at each step in order to operate smoothly. The limited bandwidth availability of FL makes it difficult for these devices to relay all the data at regular intervals, which can result in slower response times or decreased performance [23]. Here, we have collected the voltage fluctuations at a location in our IIoT network. The industrial unit where this was deployed exists in an agricultural zone with unreliable power supply in a rural zone. The intent is to study the fluctuations over a periodic cycle, and study seasonal variance of power-cuts and times in the day when maximum fluctuations occur. Sensitive equipment can be prevented from operating during times prone to unreliable voltage input from the power grid. With the analytic information, it should be able to suggest best times of operation of sensitive gauging equipment which can be damaged due to such fluctuation and cuts in supply voltage.


```
plot_cols = ['Voltage']
plot_features = df[plot_cols]
plot_features.index = date_time
_ = plot_features.plot(subplots=True)

plot_features = df[plot_cols][:480]
plot_features.index = date_time[:480]
_ = plot_features.plot(subplots=True)
```

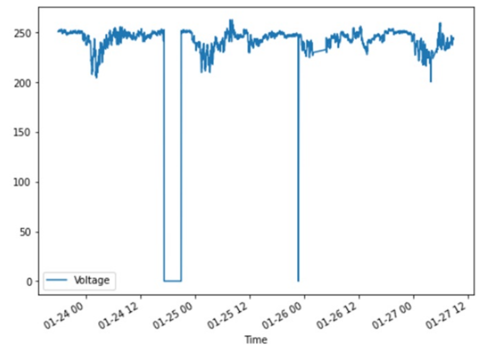


Figure 10. The voltage sensor data across 3 days

The voltage data was published to the broker every 20 seconds. The edge device subscribed to the sensor message queue which contained the sensor readings. This was done over a period of a month. For this study, the data from three days was used to perform the experiment of creating a model at the edge and sending it across to the central server as shown in Fig 10. The seasonal daily variations in the grid supply voltage, the periods of heavy fluctuations and stable supply slots can be studied in this sample. This model ends up being vital in taking decisions on the best time to run sensitive equipment which are vulnerable to heavy fluctuations.
Baseline performance for comparison: Prior to building a model to train, it is recommended to have a baseline performance, to compare with the incremental models we would build in the future. Initially, we have to be able to predict the voltage an hour ahead, when we have the present values of each feature. We initialize with a model that would give us the present voltage as a forecast, thereby forecasting no change at all. Since the voltage from the power grid fluctuates around the 230V mark, and is 0V during power cuts, this can be considered a fair baseline to start with.

Long tail end data in the figure is the power cut part of the normalization. After the normalization step, we get voltage sensor data that is now as shown in Fig 11. We created windows across the data: The number of time steps, width of the input along with the windows labels. For the time gap between the windows, voltage is used as inputs along with the labels.

C. Model Configuration:

We attempted a single output forecast initially. Then for multiple sensor data, we also forecast multiple outputs. After obtaining single time stepping and later multi time stepping, we created windows in the data in order for it to be

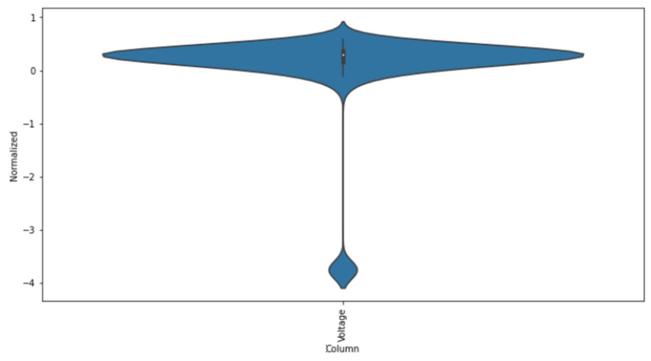


Figure 11. Voltage sensor data normalization

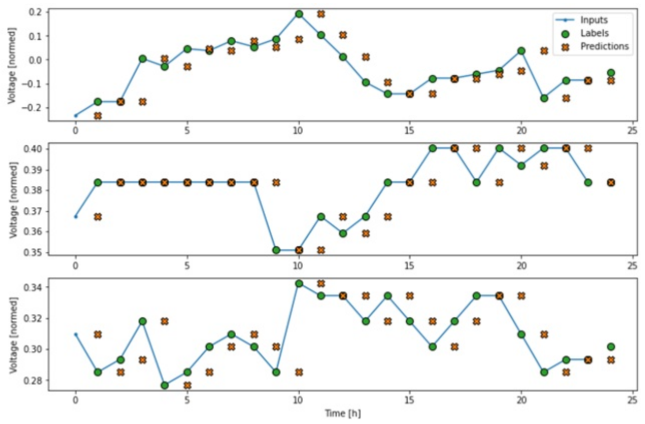
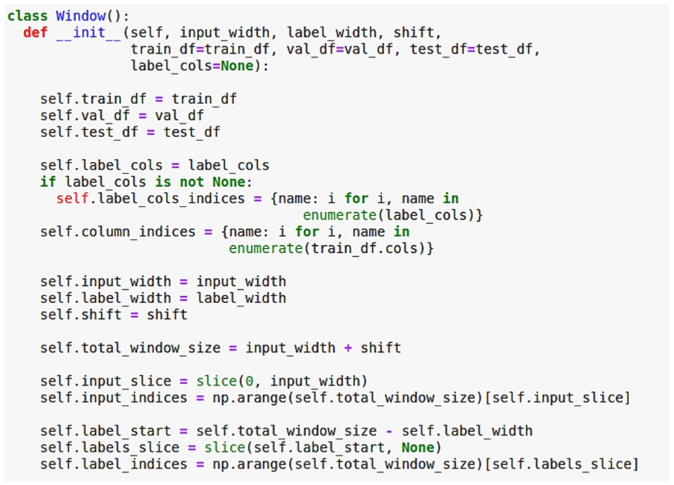


Figure 13. Voltage sensor data inputs with forecasting

reusable for Linear, DNN, CNN, RNN models. TensorFlow data is a bunch of arrays [21]. At the outer index through out all examples, lies the batch. At the middle, the time or space indices exist as width and height and the indices at the core are called as features. We took a set of three nine-time step windows, using five features at each step of time. This then obtains a single time step single featured label. The label has a single feature since the window got initialized as label_cols=['Voltage']. We built a model that forecasts labels with a singular output shown in fig 12.



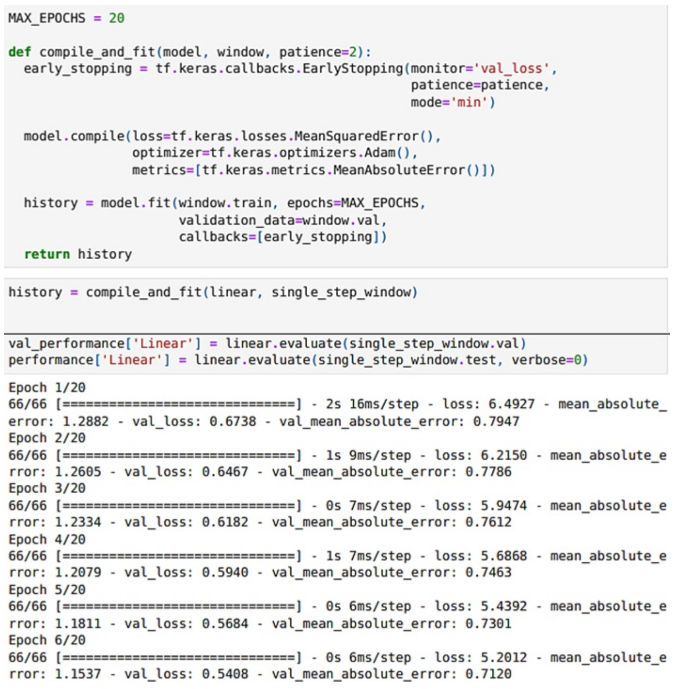
```
class Window():
  def __init__(self, input_width, label_width, shift,
               train_df=train_df, val_df=val_df, test_df=test_df,
               label_cols=None):

    self.train_df = train_df
    self.val_df = val_df
    self.test_df = test_df

    self.label_cols = label_cols
    if label_cols is not None:
      self.label_cols_indices = {name: i for i, name in
                                 enumerate(label_cols)}
    self.column_indices = {name: i for i, name in
                           enumerate(train_df.cols)}

    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift

    self.total_window_size = input_width + shift

    self.input_slice = slice(0, input_width)
    self.input_indices = np.arange(self.total_window_size)[self.input_slice]

    self.label_start = self.total_window_size - self.label_width
    self.labels_slice = slice(self.label_start, None)
    self.label_indices = np.arange(self.total_window_size)[self.labels_slice]
```

Figure 12. Windows in the voltage sensor data

Single stepping model:

The initial model we created with data predicts a singular feature value. One step of time (an hour ahead) using only present conditions of the voltage. The models were built forecasting voltages an hour ahead. A window object to create these singular-step input and label combinations is as in Fig 13.

The blue line depicts voltage for every step of time. All sensor data is received by the model, but the plot above only maps the voltage. Labels is green dots depict the forecast values of the target. Dots are plotted at the times of forecast. This causes the label range to shift a single step ahead as compared to the inputs. The forecast is depicted as orange crosses at each step of time. In case of a perfect forecast, the crosses would coincide with the labels as in Fig 14.



```
MAX_EPOCHS = 20

def compile_and_fit(model, window, patience=2):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                    patience=patience,
                                                    mode='min')

  model.compile(loss=tf.keras.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=[tf.keras.metrics.MeanAbsoluteError()])

  history = model.fit(window.train, epochs=MAX_EPOCHS,
                      validation_data=window.val,
                      callbacks=[early_stopping])
  return history
```

```
history = compile_and_fit(linear, single_step_window)
```

```
val_performance['Linear'] = linear.evaluate(single_step_window.val)
performance['Linear'] = linear.evaluate(single_step_window.test, verbose=0)
```

```
Epoch 1/20
66/66 [==============================] - 2s 16ms/step - loss: 6.4927 - mean_absolute_
error: 1.2882 - val_loss: 0.6738 - val_mean_absolute_error: 0.7947
Epoch 2/20
66/66 [==============================] - 1s 9ms/step - loss: 6.2150 - mean_absolute_e
rror: 1.2605 - val_loss: 0.6467 - val_mean_absolute_error: 0.7786
Epoch 3/20
66/66 [==============================] - 0s 7ms/step - loss: 5.9474 - mean_absolute_e
rror: 1.2334 - val_loss: 0.6182 - val_mean_absolute_error: 0.7612
Epoch 4/20
66/66 [==============================] - 1s 7ms/step - loss: 5.6868 - mean_absolute_e
rror: 1.2079 - val_loss: 0.5940 - val_mean_absolute_error: 0.7463
Epoch 5/20
66/66 [==============================] - 0s 6ms/step - loss: 5.4392 - mean_absolute_e
rror: 1.1811 - val_loss: 0.5684 - val_mean_absolute_error: 0.7301
Epoch 6/20
66/66 [==============================] - 0s 6ms/step - loss: 5.2012 - mean_absolute_e
rror: 1.1537 - val_loss: 0.5408 - val_mean_absolute_error: 0.7120
```

Figure 14. Performance of the Federated Learning Model on the Edge Devices

Algorithm: The R edge devices are indexed by r S is the batch size, is the learning rate and Ep is the number of epochs on the edge device as in Fig 14.

Server executes this algorithm in the experiment :

```
1. Initialize W0
2. for each iteration t = 1,2,3,... do
max ← max (C · R, 1)
```

```
St ← (set of max edge devices)
for each edge device in n  St do
```

$$\omega^r_{(t+1)} \text{¸} UpdateEdge(r,\ \omega_t) \tag{1}$$

$$\omega_{(t+1)} \text{¸} \sum_{r=1}^{R} \frac{n_r}{n} \omega^r_{(t+1)} \tag{2}$$

```
    UpdateEdge(r,w) : // Execute at edge device r
3. S ← (split into sets with size S)
4. for each edge epoch k from 1 to Ep do
for set s  S do
```

$$\omega \text{¸} \omega \ - \ \nabla l(w; b) \tag{3}$$

```
    5. return to central cloud
```

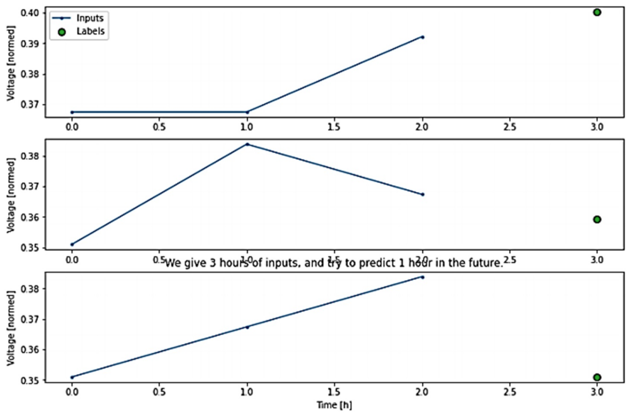


Figure 15. Three hours of inputs given, and a single hour of forecast

As a test, we gave 3 hours of input data and tried to predict 1 hour in the future as shown in Fig 15. This method yielded fairly reasonable predictions for the short-term. Linear and LSTM models were accurate in the preliminary tests.

Split Federation for Data Privacy - Model file transfer:The model was saved in the HDF5 format. The model files are synchronized to the cloud server using rsync after every training process as in Fig.16. Then for incremental modeling, it is transferred back to another edge device. This keeps the data only at the edge, and allows for incremental modeling at the edge thereby federating the learning across the IIoT network edge. tf.keras was used to build and to train the models using TensorFlow at the edge.

The training checkpoints can be saved as depicted in Fig.17. In case the training at the edge is interrupted, the TensorFlow Keras callback ModelCheckpoint can continuously save the training model while training the model, and also at the end of the training.

This Naïve Bayesian Classifier is used for quick learning. As long as B definitely occurs, then the probability of A occurring is denoted by p(A/B) :

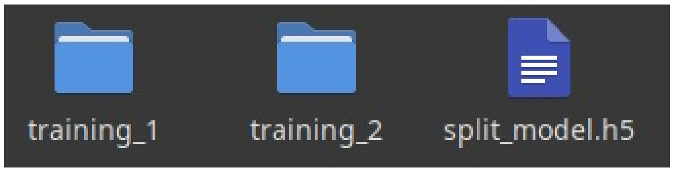


Figure 16. Trained model at the edge - HDF5 format

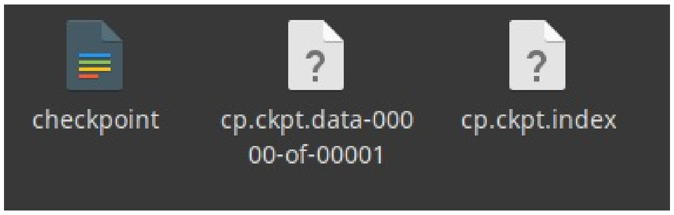


Figure 17. Training checkpoints that can resume on interruption

$$p(A|B) = \frac{p(B|A)p(A)}{\sum_a p(B|A = a)p(A = a)} \tag{4}$$

(IGNB) Incremental Gaussian Naïve Bayes Classifier, considers the Naïve Bayes equation and can be used to calculate the mean deviation incrementally, as opposed to logging all sensor data for the data-distribution :

$$p(C_k|x) = \frac{p(C_k)\pi^n_{i=1} p(x^i|C_k)}{p(x)} \tag{5}$$

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma^2_k}} e^{-\frac{(v-\mu_k)^2}{2\sigma^2_k}} \tag{6}$$

The Bayes method can be used by slicing the sensor data-sets or by considering their distributions in the data-set. Deviations from the mean are considered during slicing and resampling sensor data.

We can see that using simple Bayesian classification at the edge device running a power-efficient SBC can offer reasonable analytics and forecasting abilities at the edge. The aggregated model so created, can be used on different similar locations, to be able to forecast and take decisions related to preventive maintenance at different shop floors. Data remains local to each shop floor and only the learning model migrates around, ensuring privacy.

We have demonstrated a practical and simple implementation that is quick to get off the ground, based on open source technologies only. This approach ensures that it is easy to kick-start FL in IIoT in an industrial setting with a quick turn-around time. This approach emphasizes on being able to get reasonable results with least resources and with minimum power consumption at the edge, though resource-heavy processes are running on the edge devices in real-time.

Details of the implementation :

One layer refers to the architecture of a Recurrent Neural Network (RNN). An RNN typically consists of multiple layers, but in this case, we're only using one layer. Each layer in an RNN processes information sequentially, taking into account the current input as well as the information it has learned from previous inputs. We used SimpleRNN: This is a type of RNN architecture that's relatively straightforward compared to more complex variants like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit). SimpleRNN processes sequential data by taking the current input and the previous time step's output to generate the current output. ReLU (Rectified Linear Unit): This is the activation function used in the SimpleRNN layer. ReLU is defined as f(x)=max(0,x). It's a simple non-linear activation function that outputs the input directly if it's positive, and zero otherwise. ReLU has become very popular in neural networks due to its simplicity and effectiveness in preventing the vanishing gradient problem. Putting it all together, a ”one layer SimpleRNN with ReLU” architecture means we have a recurrent neural network with a single layer that uses the SimpleRNN architecture for processing sequential data, and the Rectified Linear Unit (ReLU) activation function to introduce non-linearity to the network.

In this setup, we're utilizing a single layer within a Recurrent Neural Network (RNN), specifically the SimpleRNN variant. Unlike more intricate RNN types, like LSTM or GRU, SimpleRNN operates with a straightforward mechanism, processing sequential data by considering both the current input and the output from the previous time step. Within this layer, we apply the Rectified Linear Unit (ReLU) activation function. ReLU is a widely used non-linear function in neural networks. It essentially outputs the input directly if it's positive, otherwise, it outputs zero. This choice of activation function helps address issues like the vanishing gradient problem, making it a popular choice for neural network architectures.

In split learning, the model is divided into two parts: the client part and the server part. The client part resides on factory floor, and the server part resides on a central server. This split allows the sensitive data to remain on the client side, ensuring privacy. In model collaboration the client part processes the local data and generates a compressed representation. This compressed representation is then sent to the server part. In aggregation, the server part receives the compressed representations from multiple clients. It aggregates these representations and updates the global model accordingly. This update process typically involves techniques like model averaging or gradient aggregation. For deployment when the global model is updated, it is sent back to the clients. The updated model can then be deployed on the client devices for further local training or inference. In federated learning, split learning provides an additional layer of privacy by ensuring that raw data never leaves the client devices. Only the compressed representations are exchanged between the clients and the central server, reducing the risk of exposing sensitive information.

Our method touches on making the setup scalable using readily available parts from the market. Cheap Small Board Computers make the federation process very economical at scale.

Analysis methodology and Federation :

```
def split_sequence(sequence, n_steps_in, n_steps_out):
X, y = list(), list()
for i in range(len(sequence)):
find the end of this pattern
end_ix = i + n_steps_in
out_end_ix = end_ix + n_steps_out
check if we are beyond the sequence
if out_end_ix ¿ len(sequence):
break
gather input and output parts of the pattern
seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
X.append(seq_x)
y.append(seq_y)
return np.array(X), np.array(y)
```

Prediction :

```
model.add(SimpleRNN(256, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse', metrics=[r2_metric])
```

The accuracy in the predictions were found to be within a variance of 8% of the actually observed values, over a period of 3 days. The comparision of actual and predicted voltages after training the model at a specific location is shown in the comparision table below. This is used for predicting the times when machines could fail due to operating at over and under voltage conditions. The times avoidable to run a machine whenever the voltage drops below a specific treshold or raises above a threshold, the delicate machinery could be stopped from being deployed at the factory floor.

4. Results and discussions

In the course of our experiment of the split learning FL process, the models we created at the edge were transferred to the centre and were incrementally trained after being transferred back to another edge device. Modeling happened

Table I: Comparison with earlier research

Year	Reference	IIoT at Edge	ML at edge	power efficiency at edge
2020	7	YES	NO	NO
2021	8	YES	YES	NO
2022	1	YES	YES	NO
2023	Proposed work	YES	YES	YES

at the edge and the results got transferred again to the central server without transferring the actual data. The aggregated model was formed in the central or the cloud server. The computations were performed on the Raspberry Pi 4 with lower computational power than the central server or the cloud server for an industrial setting. FL models were aggregated on the central server without any data being transferred.

Results confirm that the trained models of the various different shop floors across geographies can be aggregated with success, at the cloud. Privacy is maintained at all times. Once the model from the central server was trained, we were able to forecast voltage fluctuations at each site fairly reasonably.

Table II. Results: Actual Voltage and Predicted Voltage

Time	Actual Voltage	Predicted Voltage
2023-01-28T17:55:40Z	250	250.6
2023-01-28T17:56:00Z	250	250.4
2023-01-28T17:56:20Z	250.5	250.6
2023-01-28T17:56:40Z	250.5	250.6
2023-01-28T17:57:00Z	251	251.3
2023-01-28T17:57:20Z	250.5	251
2023-01-28T17:57:40Z	250.5	250.6
2023-01-28T17:58:00Z	251	251.3
2023-01-28T17:58:20Z	250.5	251
2023-01-28T17:58:40Z	251	251.6
2023-01-28T17:59:00Z	251	251.6
2023-01-28T17:59:20Z	251	251.6
2023-01-28T17:59:40Z	251.5 252	
2023-01-28T18:00:00Z	251	251.3
2023-01-28T18:00:20Z	251.5	251.9
2023-01-28T18:00:40Z	252.5	252.6
2023-01-28T18:01:00Z	253	253.1
2023-01-28T18:01:20Z	252.5	252.8
2023-01-28T18:01:40Z	252.5	252.8
2023-01-28T18:02:00Z	251.5	252
2023-01-28T18:02:20Z	252	252.1
2023-01-28T18:02:40Z	252	252.1
2023-01-28T18:03:00Z	252	252.1
2023-01-28T18:03:20Z	252	252.6
2023-01-28T18:03:40Z	252	252.6
2023-01-28T18:04:00Z	252	252.5
2023-01-28T18:04:20Z	252	252.2

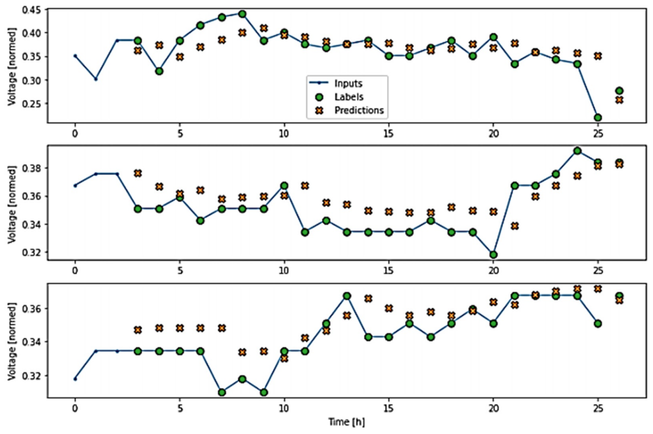
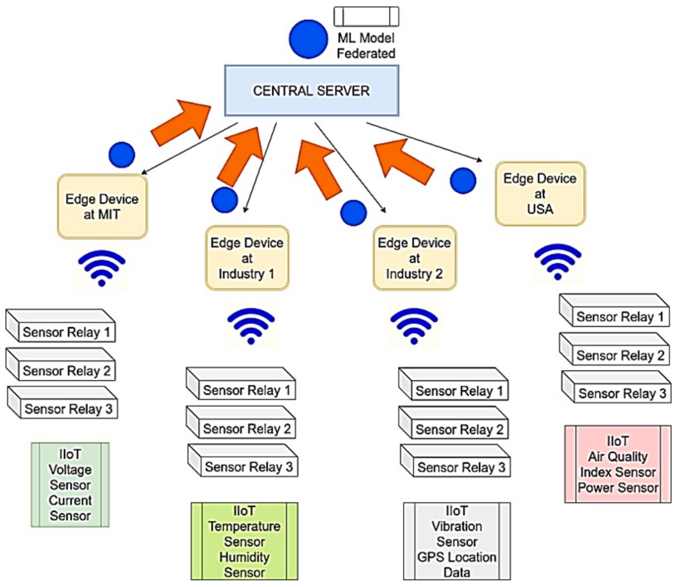


Figure 18. Forecast – Yellow crosses

Based on the forecast as shown in Fig.19, simple flows were designed to give recommendations on the best times to run sensitive machinery without compromising on safety and accuracy. FL algorithms such as LSTM was proven to establish that such IIoT frameworks can be deployed in an enterprise, with higher privacy and scalability.


ML Model Federated
CENTRAL SERVER
Edge Device at MIT
Edge Device at Industry 1
Edge Device at Industry 2
Edge Device at USA
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Voltage Sensor Current Sensor
IIoT Temperature Sensor Humidity Sensor
IIoT Vibration Sensor GPS Location Data
IIoT Air Quality Index Sensor Power Sensor
Figure 19. Federated Learning models sent back to the central server

The rsync of the model files combined with the use of power efficient SBCs was experimented with, and eventually was found to be a unique attempt at learning at the edge. The migrating of the model files between edge devices for incremental learning, was an approach different from that used earlier by other researchers. The use of cron jobs to sync files distributed the learning model among the edge devices modeling the same sensor data at different locations, without needing to send data across to headquarters. This approach can be viewed as an alternative where edge resources needs to be saved for more important processes. Failure prediction and accuracy :

The accuracy in the predictions were found to be within a variance of 8% of the actually observed values, over a period of 3 days. The comparision of actual and predicted voltages after training the model at a specific location is shown in the comparision table II below. This is used for predicting the times when machines could fail due to operating at over and under voltage conditions. The times avoidable to run a machine whenever the voltage drops below a specific treshold or raises above a threshold, the delicate machinery could be stopped from being deployed at the factory floor.

A few methods we studied and based our current experiment on, are summarized in the table I :

5. Conclusions and Future Work

The Federated Learning concept (Fig.17) is best suited for situations where sensor data privacy is important. Split Learning can be implemented for such scenarios. This study has shown how such a setup can be conceived and implemented across a wide geography. The recent improvements in SBC power has enabled intensive processes like TensorFlow to be able to run on the edge device itself. In edge devices where TensorFlow Federated cannot be run, one can federate learning through the split learning paradigm. This decreases communication traffic between the headquarters and results in lower computational load on the cloud server at the cental location. It also reduced bandwidth requirements between the edge and the cloud. Shared ML models can be created and incremental learning can take place with the models aggregated with other training elements. FL is best for privacy where local data is not transferred to the cloud. This can be integrated into IIoT architectures in the industry as part of Industry 4.0 and 5.0 initiatives to utilize best practices in efficient edge-cloud implementation for the modern enterprise.

This method of federating is dependent on the energy efficient 28nm FinFet based CPU driven SBCs being available in the market. Currently, due to the chip shortage in early 2023, at the time of conducting this experiment, we found that these SBCs are in short supply. There is a wait list over 24 months from original equipment manufacturers and it is found that in the alternative local market, there is an inconsistent supply and even when available, the SBCs are marked up to almost double the actual suggested price. We expect the supply to be streamlined by the end of the year (2023). The future holds tremendous potential for federating at the edge in IIoT and it all depends on the supply chain stabilizing in the post-pandemic era. Once the fab facilities get their act together and assure a regular supply at reasonable rates, we are sure that more research in IIoT will be accomplished with different techniques and different needs. The adoption of energy-saving technologies creates opportunities for job growth and economic development in the renewable energy and energy efficiency sectors. Industries involved in manufacturing, installing, and maintaining energy-efficient machinery require skilled workers, leading to job creation and investment in training and education programs. Additionally, energy savings free up capital for businesses to invest in innovation, expansion, and job creation, contributing to overall economic growth. By reducing energy consumption, running machinery with energy-saving technologies helps mitigate environmental impacts associated with energy production, such as air and water pollution, habitat destruction, and climate change. This contributes to improved public health and environmental quality, benefiting communities and ecosystems locally and globally. Energy-saving machinery helps conserve natural resources, such as fossil fuels and water, by reducing the demand for energy and raw materials used in manufacturing processes. This promotes sustainable resource management and reduces the environmental footprint of industrial activities, preserving natural habitats and ecosystems for future generations. Energy-saving technologies can improve energy access and affordability for underserved communities and developing regions by reducing energy costs and expanding access to clean and reliable energy sources. This promotes social equity and reduces energy poverty by ensuring that all individuals and communities have access to essential energy services for heating, lighting, cooking, and communication.

Authors' contributions: Sachin Bhoite, Chandrashekhar H. Patil, and Harshali Patil have equally contributed.

References

[1] Kairouz, McMahan, Avent, Bellet, Bennis, Bhagoji, Bonawitz, Charles, Cormode, Cummings, and D'Oliveira., "Advances and open problems in federated learning. foundations and trends® in machine learning," vol. 14, pp. 1–210, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:209202606

[2] M. Wes, "Python for data analysis," 2012.

[3] C. Briggs, Z. Fan, and P. Andras, "Federated learning for short-term residential load forecasting," *IEEE Open Access Journal of Power and Energy*, vol. 9, pp. 573–583, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:252118900

[4] M. Vaezi, A. Azari, S. R. Khosravirad, M. Shirvanimoghaddam, M. M. Azari, D. Chasaki, and P. Popovski, "Cellular, wide-area, and non-terrestrial iot: A survey on 5g advances and the road toward 6g," *IEEE Communications Surveys & Tutorials*, vol. 24, pp. 1117–1174, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID: 235755492

[5] A. P. Singh and S. Chaudhari, “Embedded machine learning-based data reduction in application-specific constrained iot networks,” *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:214694129

[6] M. Liu, K. Yang, N. Zhao, Y. Chen, H. Song, and F. kui Gong, “Intelligent signal classification in industrial distributed wireless sensor networks based industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 17, pp. 4946–4956, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221969570

[7] M. A. Ferrag, O. Friha, D. Hamouda, L. A. Maglaras, and H. Janicke, “Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning,” *IEEE Access*, vol. PP, pp. 1–1, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246367315

[8] K. Tange, M. D. Donno, X. Fafoutis, and N. Dragoni, “A systematic survey of industrial internet of things security: Requirements and fog computing opportunities,” *IEEE Communications Surveys & Tutorials*, vol. 22, pp. 2489–2520, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226591908

[9] T. Hafeez, L. Xu, and G. Mcardle, “Edge intelligence for data handling and predictive maintenance in iiot,” *IEEE Access*, vol. 9, pp. 49 355–49 371, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233177880

[10] A. Rehman, I. Razzak, and G. Xu, “Federated learning for privacy preservation of healthcare data from smartphone-based side-channel attacks,” *IEEE Journal of Biomedical and Health Informatics*, vol. 27, pp. 684–690, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:248504256

[11] M. Parto, C. Saldana, and T. R. Kurfess, “A novel three-layer iot architecture for shared, private, scalable, and real-time machine learning from ubiquitous cyber-physical systems,” *Procedia Manufacturing*, vol. 48, pp. 959–967, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219983134

[12] Y.-L. Hsu, C.-F. Liu, H.-Y. Wei, and M. Bennis, “Optimized data sampling and energy consumption in iiot: A federated learning approach,” *IEEE Transactions on Communications*, vol. 70, pp. 7915–7931, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253348122

[13] X. Cheng, Z. Jia, A. Bhatia, R. M. Aronson, and M. T. Mason, “Sensor selection and stage & result classifications for automated miniature screwdriving,” *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6078–6085, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54203838

[14] M. Wen, R. Xie, K. Lu, L. Wang, and K. Zhang, “Feddetect: A novel privacy-preserving federated learning framework for energy theft detection in smart grid,” *IEEE Internet of Things Journal*, vol. 9, pp. 6069–6080, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:240503647

[15] S. Savazzi, V. Rampa, S. Kianoush, and M. Bennis, “An energy and carbon footprint analysis of distributed and federated learning,” *IEEE Transactions on Green Communications and Networking*, vol. 7, pp. 248–264, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249889151

[16] A. S. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, “Federated learning for mobile keyboard prediction,” *ArXiv*, vol. abs/1811.03604, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53207681

[17] K. Gamal, A. Gaber, and H. Amer, “Federated learning based multilingual emoji prediction in clean and attack scenarios,” *ArXiv*, vol. abs/2304.01005, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257913614

[18] F. F. Gulamali and G. N. Nadkarni, “Federated learning in risk prediction: A primer and application to covid-19-associated acute kidney injury,” *Nephron*, vol. 147, pp. 52 – 56, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250560012

[19] H. Q. Le, Y. Qiao, L. X. Nguyen, L. Zou, and C.-S. Hong, “Federated multimodal learning for iot applications: A contrastive learning approach,” *2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 201–206, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:262970920

[20] W. Y. B. Lim, J. S. Ng, Z. Xiong, D. T. Niyato, and C. Miao, “Federated learning over wireless edge networks,” *Federated Learning Over Wireless Edge Networks*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252569863

[21] A. Alatram, L. F. Sikos, M. Johnstone, P. Szewczyk, and J. J. Kang, “Dos/ddos-mqtt-iot: A dataset for evaluating intrusions in iot networks using the mqtt protocol,” *Comput. Networks*, vol. 231, p. 109809, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258792762

[22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, “Tensorflow: A system for large-scale machine learning,” in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[23] S. Saha and T. Ahmad, “Federated transfer learning: concept and applications,” *Intelligenza Artificiale*, vol. 15, pp. 35–44, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:225103353

Sachin Bhoite a highly accomplished academician and researcher, holds a Ph.D. in Computer Science specializing in Machine Learning, Ensemble ML, and Deep Learning from Bharati Vidyapeeth (Deemed to be University), Pune, India. Currently serving as a Senior Grade Assistant Professor and Ph.D. Coordinator at Dr. Vishwanath Karad, MIT World Peace University, Pune, he brings over 23 years of teaching experience and expertise in diverse areas such as Data Mining, NLP, Data Science, and Artificial Intelligence. Dr. Bhoite's extensive research contributions, including publications in reputed conferences and journals, alongside his recognition as a Ph.D. Guide in Computer Science, reflect his dedication to academia. Additionally, he has received accolades such as Google Scholar Citations of 115, an H-index of 6, and an i10-index of 3, and secured an Indian patent for an IoT-based Intelligent Parking Management System. His commitment to education is further evident in his administrative roles, highlighting his dedication to fostering academic excellence and research-driven learning.

Chandrashekhar H. Patil is an Associate Professor at the Department of Computer Science and applications, Faculty of Science, MITWPU, Pune, India. His fields of research interest include Digital Document Processing, Optical Character Recognition, Biometrics,Word spotting, Video Document Analysis, Signal processing etc. He has published 69 research papers in international journals like Elsevier, springer, and calendar conference proceedings. He is a recognized PhD guide in computer science at MITWPU, Pune. Currently 3 PhD. Scholars are working under him. He has been a reviewer of the International Journals of Elsevier, springer etc and Calendar conferences. He has 35 Scopus indexed research publications, 8 patents granted, 2 book chapters, 1 book and 1 minor research project at his research credentials. The current publication citations of his research are more than 230+ with an h-index of 7 as per Google Scholar. He can be contacted at email: chpatil.mca@gmail.com.

Harshali Patil is an Associate Professor at Sri Balaji University, Pune, where she has 13 years of teaching experience to MCA students. She received her doctoral degree in Computer Application from Dr. Babasaheb Ambedkar Marathwada University, Aurangabad in 2018, under the faculty of Commerce and Management. Her areas of research interest include technology acceptance, AI, Image processing, AR VR. She has attended and published research papers in many faculty development programs, conferences, workshops, seminars, and Scopus indexed international conferences. Her published research papers and book chapters are indexed in Scopus, Web of Science, Taylor and Francis, and Wiley. She can be contacted at email: hkarankal@gmail.com.