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

Sachin Bhoite1, Chandrashekhar Himmatrao Patil*, 2, Harshali Patil3
1,2 Department of Computer Science and Application, Dr. Vishwanath Karad, MIT World Peace
University, Pune, India
3, Sri Balaji University, Pune, India
Email: sachintenjuly@gmail.com, chpatil.mca@gmail.com, hkarankal@gmail.com
Correspondence: chpatil.mca@gmail.com;

Abstract

The recent availability of powerful (SBC) Single Board Computing devices has facilitated edge computing at a level that was previously hard to deploy. This new shift, presented a gap, hitherto considered tough to implement in the industry with lower power consumption at the edge.

Generally, keeping in mind preventive maintenance intervention as the key purpose, a simple and quick implementation of federation in the industry had to be addressed. Industries need such predictions with data privacy and accuracy to take care of chronic spare replacements before things fail.

We were presented with an opportunity to suggest preventive maintenance procedures and make manufacturing decisions based on (IIoT) Industrial Internet of Things data from multiple sensors across the enterprise, from multiple similar machines in different shop floors in an industrial setup across a varied geography.

IIoT sensors chosen, ping the edge device at each location with sensor data at regular intervals. (MQTT) Message Queuing Telemetry Transport protocol was used [7] and the data reached the edge in (JSON) Javascript Object Notation format with a timestamp and sensor value. The SBC ensured low powered operation mode and was adequately cooled with a passive aluminium heat-sink and fans. This ensured that the edge server could be kept on for long periods of time, consuming only about an average of 15W of power.

We introduce a unique method of federation, specifically, using HDF5 model file transfer. The checkpoint file is then synchronized to the central server using timed file transfer scripts at the nodes achieving simple federation. 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. This was then networked to various edge devices in each shop floor across the industry. The individual machines and their sensor data were then captured into a named time series database at each edge device. The learning was done at each edge device and the model was then sent back to the central server without any actual sensor data for incremental learning. This learning model could be used at another similar deployment based on similar sensor data.

This is an implementation using Split Federation and Linear, DNN, CNN, RNN models.

Various sensor data was collected by the edge device in each of the industrial floors. We chose to base our first set of experiments on the time series voltage data relayed, since it was fluctuating at times from the power grid and had a seasonality pattern. Identifying periods of least fluctuations to run sensitive gauging machinery was the first step in forecasting preventive maintenance routines of sensitive equipment across the enterprise. This even ensured higher accuracy of the said gauging equipment. (FL) Federated Learning models were used to predict the sensor values and make decisions. The sensor data was stored and processed at the edge. The (ML) Machine Learning techniques only operated at the edge. The models were then synchronized to the central cloud server and back to other edge devices in the network. Results obtained lay a foundation for FL using a split learning paradigm in the IIoT space with SBCs consuming the least amount of power, for an enterprise spread across a diverse geography.

Data privacy is upheld and maintained, while at the same time reduced bandwidth requirements between the edge and the cloud make this a simple and easy first implementation of federation in the industry.

There are many social implications to this approach as well. 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.

The uniqueness in this approach is that the training checkpoints are saved. In case of any interruption, TensorFlow Keras callback ModelCheckpoint can continuously save the training model while training the model, and also at the end of the training.

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

1. Introduction

FL is a distributed machine learning technique wherein various devices on the edge, first train an ML model on their own data. They do not share their individual 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 [1], without compromising on 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 [2].

We were motivated to pursue this approach to create a solution that is quick to implement on the field.

We have implemented a split and federated learning approach wherein various participants train their models on their own 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 [3], [4].

The research had an objective to achieve a simple method of federation at scale using off-the-shelf and easily available components. It had to also 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 modeled 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. Prior to 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 ensure low latency. Online data flow paths are recommended to be implemented with a secure transport layer [21].

In Industry 4.0 and 5.0, federated learning can have the impact in 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 [5]. By training models on decentralized data, federated learning enables organizations to leverage data from multiple sources and make better use of their own data, ultimately leading to improved accuracy and performance in Industry 4.0 and 5.0 applications [4].

The original contribution here is the unique approach which can be implemented quickly with readily available components.

2. Related work

We briefly describe herein, some related work to this paper, in the following literature review.

Christopher Briggs, Zhong Fan and Peter Andras 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 [1].

Mojtaba Vaezi , Amin Azari et al. 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 have been discussed. Their impact in 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. And IoT devices have an ever increasing set of security issues.

Increase in reliability, along with increasing global spread, coupled with faster 5G networks, the deployment of the latest in AI methodologies, such as deep neural networks and FL, will be important in making this trend come true. 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 [2].

Yi Liu, Neeraj Kumar, Zehui Xiong, Wei Yang Bryan Lim et al. 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 [3].

IoT data generated from multiple devices used in sensing humidity in a study by Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda 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 [4].

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, Lina Xu and Gavin Mcardle 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 [5].

Abdul Rehman, Imran Razzak and Guandong Xu 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 [6].

Mahmoud Parto, Christopher Saldana and Thomas Kurfess 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 [7].

Mingqian Liu, Ke Yang, Nan Zhao, Yunfei Chen 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 [8].

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 [9]. 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 [10]. 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 [11].

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 [12].

Andrew Hard, Kanishka Rao et al. 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 [22].

Karim Gamal, Ahmed Gaber et al. 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 [23].

Faris F. Gulamali and Girish N. Nadkarni's 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 [24].

We see from these studies that there has been a rapid shift in paradigms in the industry. IIoT and the related ML techniques used to analyze data has 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.

3. Common techniques

Our Methodology was to attempt a single output forecast initially. Then for multiple sensor data, we also forecast for multiple outputs. After obtaining single time stepping and later multi time stepping, we created windows in the data in order for it to be reusable for Linear, DNN, CNN, RNN models.

With the advent of powerful SBC (Single Board Computing) devices in the market, the processing power available at the edge has grown tremendously in the past couple of years. We suggest various methods and techniques to use the IIoT sensor data collected over a period, and convert it into actionable intelligence to aid decision making on the shop floor.

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 analysed at the edge [7]. 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 best time of operation for sensitive machinery in the industry.

Algorithms enable the creation of models that make predictions based on data generated from multiple sources. This leads to improved efficiency, reliability, and security in industrial processes. Another approach uses deep-learning modeling, such as (RNN) Recurrent-Neural-Networks. They also explored (LSTM) long short term memory networks, and processed time series sensor data and make valid predictions. These models are trained on large amounts of historical sensor data and can make predictions based on the patterns and trends in the data [20][21].

Some commonly used federated learning algorithms for IIoT include [13] [14]:

Federated Averaging (FedAvg): This is a simple federated learning algorithm. The average value of the model parameters from a multitude of devices is computed. Post that computation, the global model is accordingly updated [19].

Split Learning: This is an algorithm that preserves data privacy. Data remains on the device and the computation only, is transferred to the server. We have used a technique in this type of FL which splits the learnings on various edge devices and transfers the models, aggregating learning across various edge devices. We used a combination of FedAvg and Split Learning in our experiment.

Differential Privacy: This is a methodology that preserves privacy. It does so by adding a noise layer to the data. The noise added, serves as a protection layer so that privacy is not compromised, and all the unique data is modulated with noise, still letting the model be aggregated and to get trained with data.

Secure Multi Party Computation (MPC): This techniques preserves privacy by enabling different devices to compute functions in tandem on their data, whilst keeping their data private [18].

Federated Transfer Learning: This involves transfer of knowledge from a pre-trained centralized data model to that on the edge device. This reduces the quantum of edge data required to be sent to the centralized cloud server [17].

Multi-Party Computation (MPC): An FL algorithm that enables secure computation of machine learning models over multiple parties, while preserving data privacy.

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 :

(AR) Autoregressive Models: These type of models use the historic values in the series of data and suggest the next values.

(MA) Moving Average Models: These models use the averages of historic values in the series in order to suggest the next values.

(ARIMA) Autoregressive Integrated Moving Average Models: Models that combine AR and MA models to account for both the past values and trends in the sequence of data.

(SARIMA) Seasonal Autoregressive and Integrated Moving Average Models: When seasonality of data extends the ARIMA model, and supports the direct modeling of the seasonal component of the series is called SARIMA. Mainly, it takes into consideration the seasonal variant in univariate data.

(RNN) Recurrent Neural Networks: These are deep learning models. They are better suited for time-series data. They can capture patterns and dependencies over time.

(LSTM) Long Short Term Memory: These are a specific variant of RNN that can handle the long term co-relations and dependencies amid time series data and are found to be used in time series prediction tasks fairly often.

5. Experiment

The Central Server uses an Intel Xeon processor with 32GB RAM and an Nvidia GTX 1050 Ti GPU. Various machines at each shop floor are connected wirelessly as well as wired to the network to relay sensor data. These typically used ESP32 / ESP8266 based implementations in their architecture. Each one of the sensor relays communicated with an (MQTT) – Message Queue Telemetry Transport 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

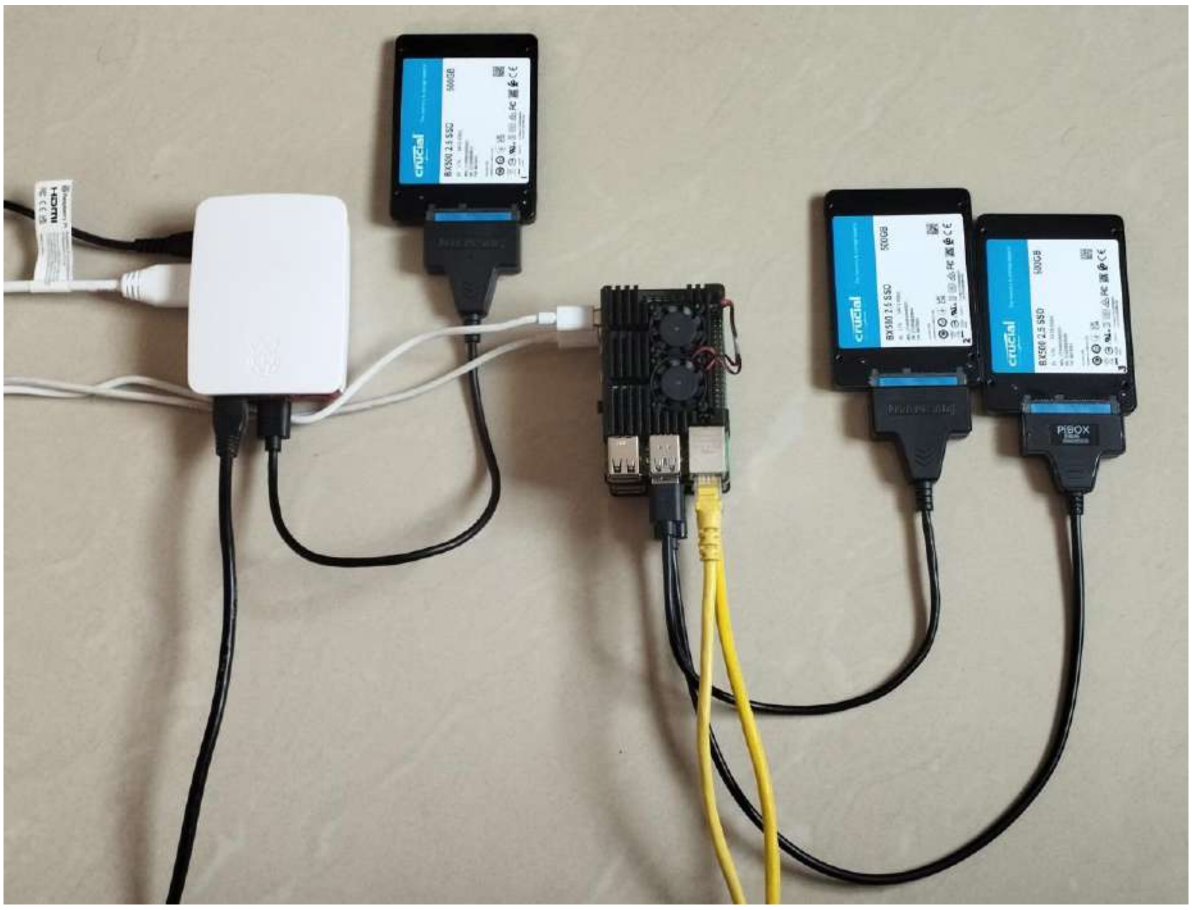
We used multiple Raspberry Pi 4 – 8GB boards along with a single Raspberry Pi 3B+ running as a storage controller, to run the open source software stack in this experiment setup. Each node relaying the data runs on either an STM32 or an ESP32 micro controller based board with custom coded embedded software 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 1a.

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

Fig 1a.



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

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 work flow that we implemented to get the cluster up and running at the edge.

1. 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 cards – 500GB was plugged in into the Raspberry Pi boards USB-3 slots as shown in Fig 1b.

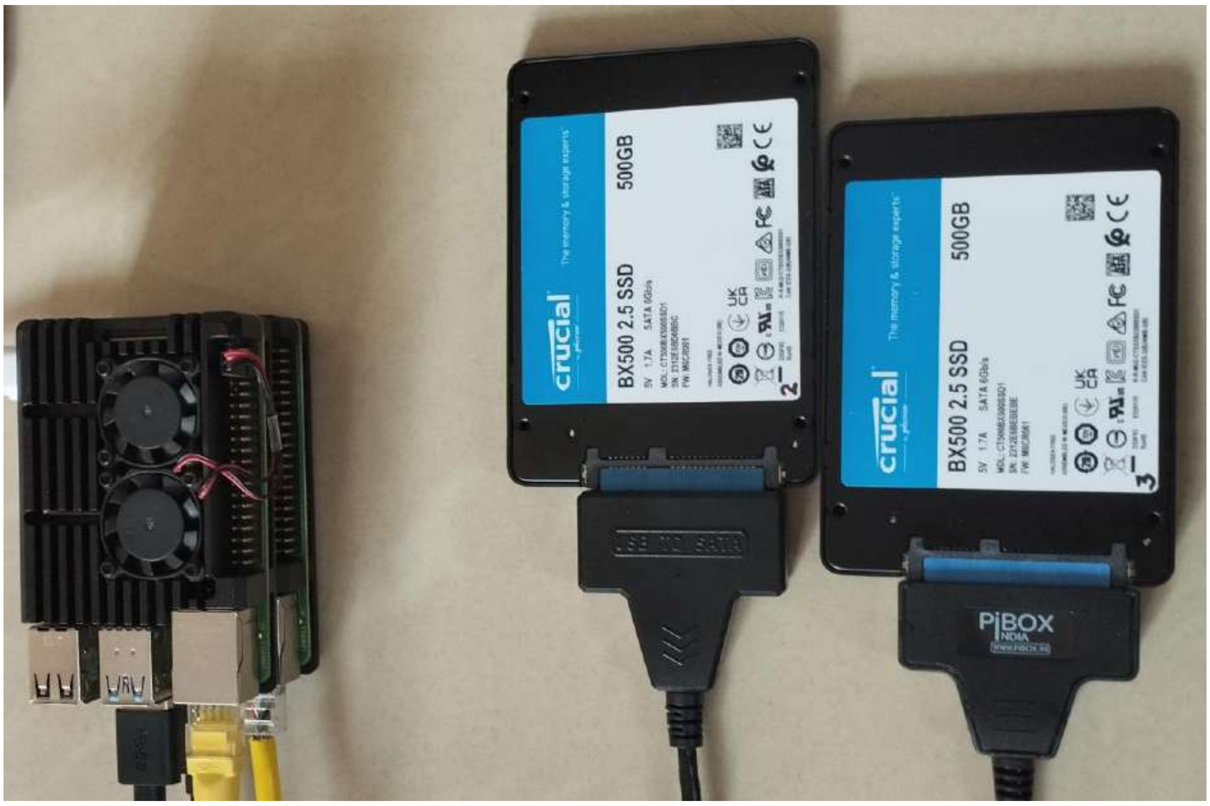


Fig 1b. USB SSDs ensure quick bootup and optimal edge performance with 300Mb/s throughput

2. 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.

3. 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.

4. 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.

5. 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>
```

6. 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>"
```

7. Power consumption:
   - We found that the typical power consumption of this edge cluster setup in Fig 1c. was around 45W at peak processing loads. The board temperatures averaged around 45 degrees centigrade.

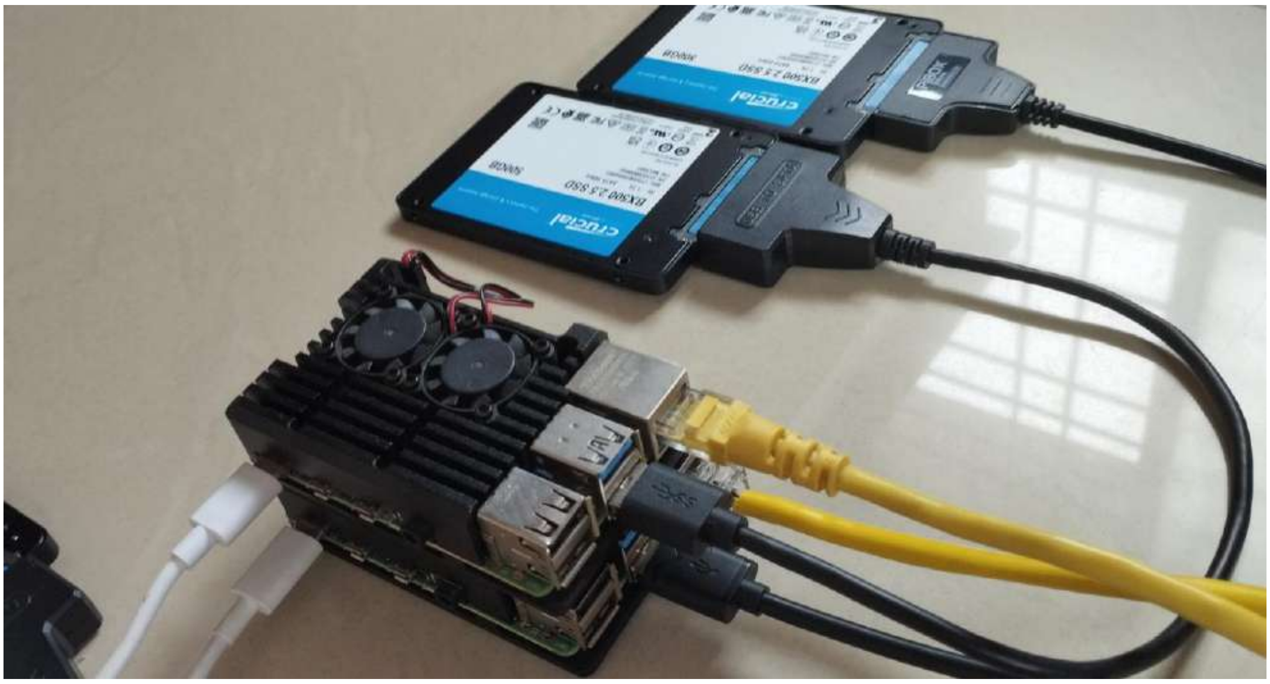


Fig 1c. Edge Cluster power usage is typically around 45W at peak processing demand

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 1d.

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.

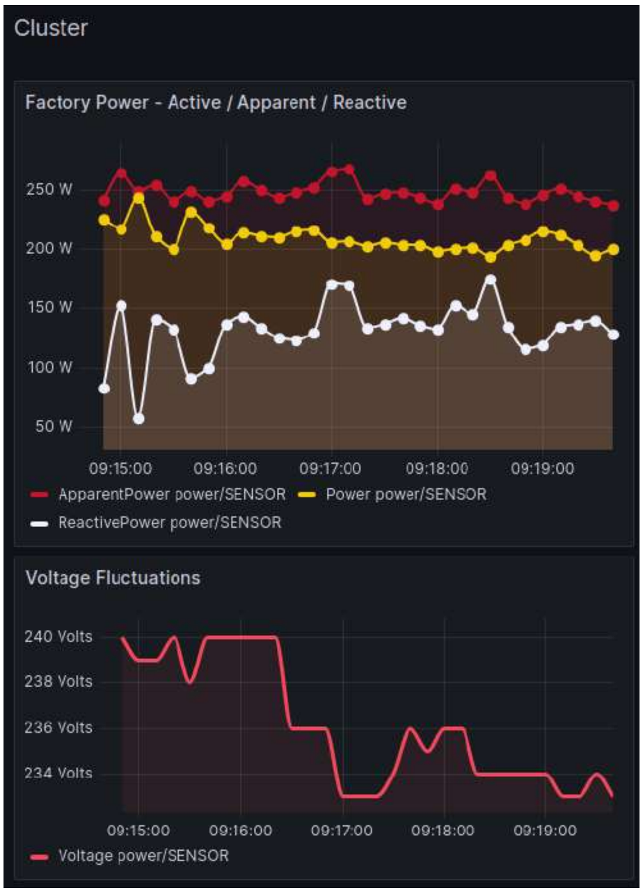


Fig 1d. A dashboard from the data stream

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.

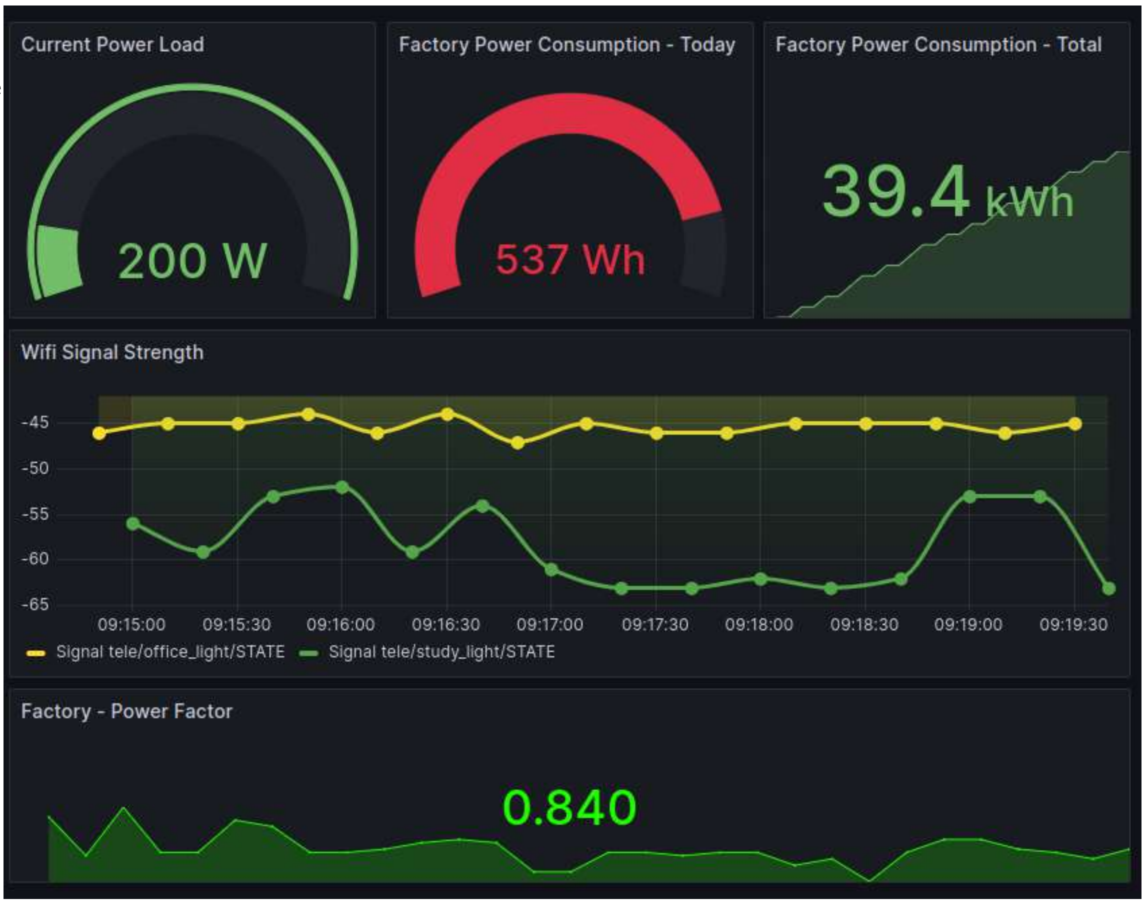
Next we explored and visualized the data using Chronograf. The Chronograf web user interface was used to visualize the sensor data stored in InfluxDB. We created dashboards and configured queries to display the data in meaningful ways using the Flux query language. Since Chronograf provides various visualization 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 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.

Fig 1e. Edge



Cluster power usage is typically around 45W at peak processing demand

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 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.2 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.

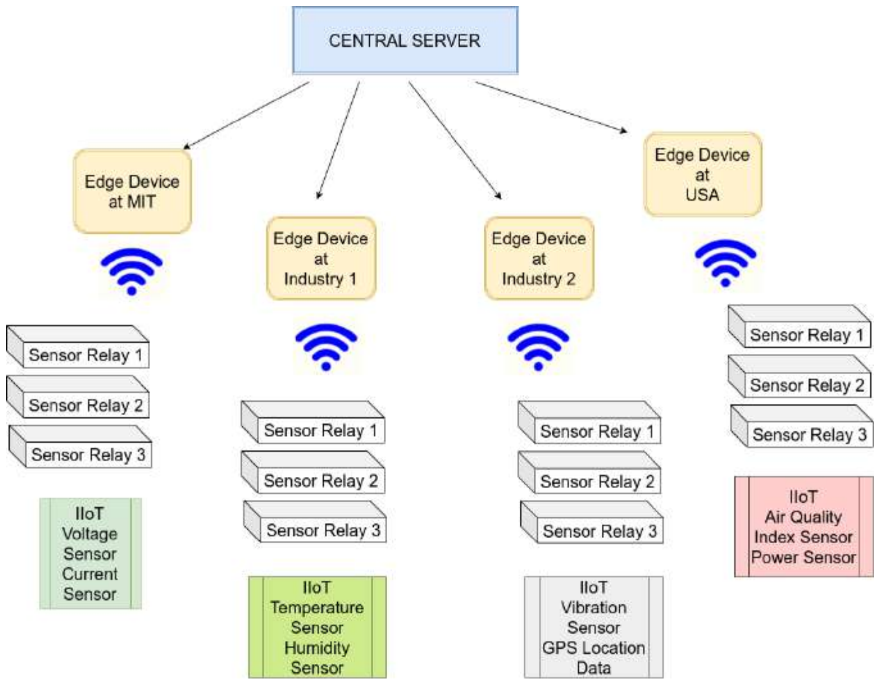

CENTRAL SERVER
Edge Device at MIT
Edge Device at Industry 1
Edge Device at Industry 2
Edge Device at USA
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Voltage Sensor Current Sensor
IIoT Temperature Sensor Humidity Sensor
IIoT Vibration Sensor GPS Location Data
IIoT Air Quality Index Sensor Power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

Fig 2. A flow of the Enterprise Server and Edge Devices with IIoT sensor relay

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.

6. 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.

MQTT

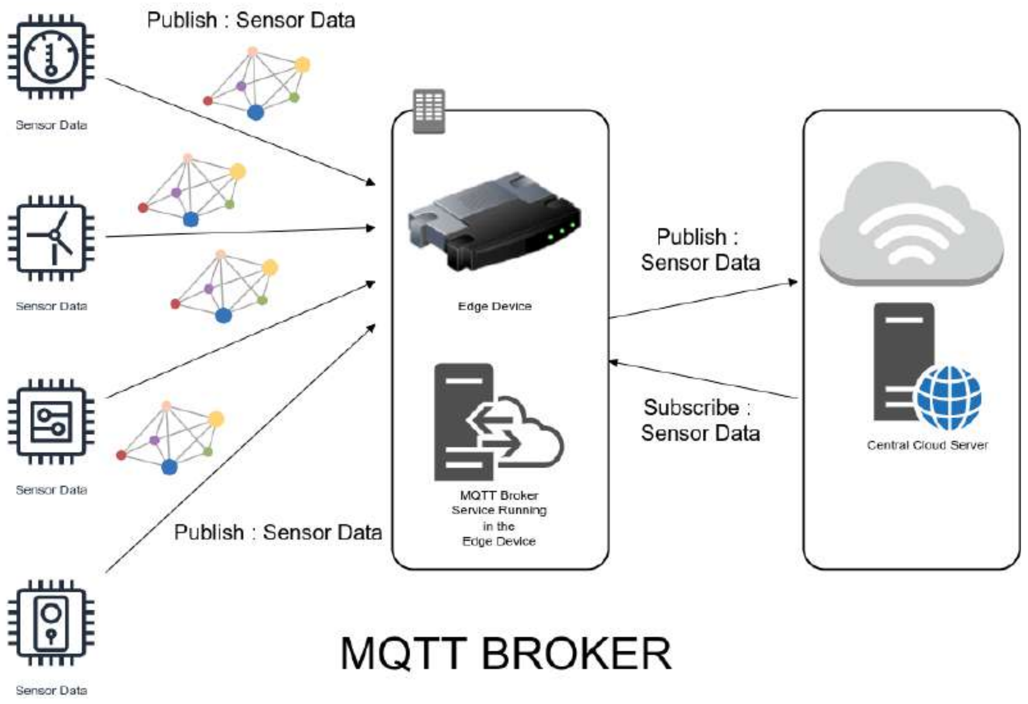


Fig.3 MQTT – Publish and Subscribe

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.3. This was initially developed by IBM and is now popular in the IIoT space [17]. 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 .

(TFF) TensorFlow Federated

Since we are experimenting 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 [18].

In the documentation for TensorFlow it has been mentioned that isolated blocks of TensorFlow code, consisting tf.functions, that capture the logic running at a single location such as our edge device. 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 such.

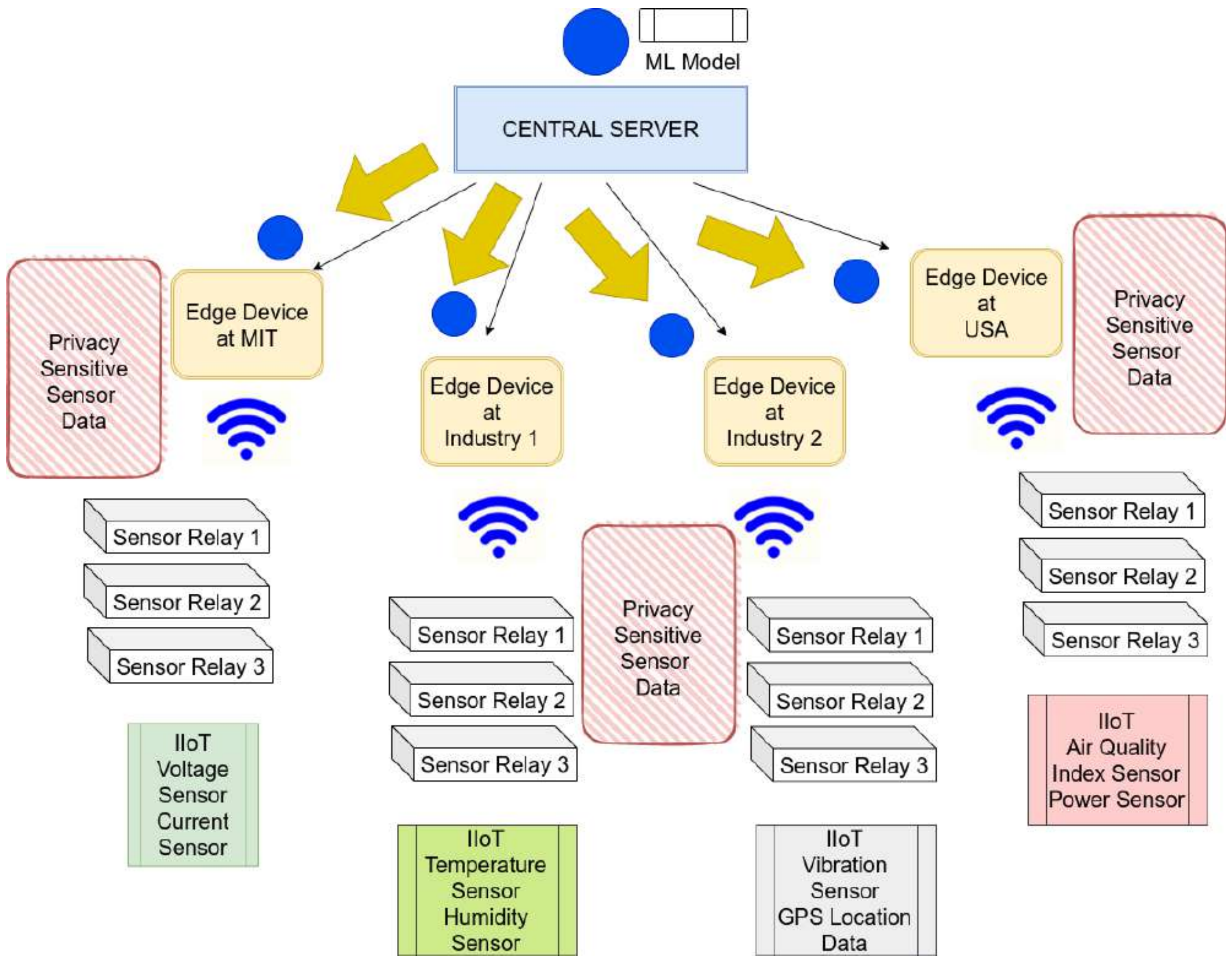

ML Model
CENTRAL SERVER
Privacy Sensitive Sensor Data
Edge Device at MIT
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Voltage Sensor Current Sensor
Edge Device at Industry 1
Sensor Relay 1
Sensor Relay 2
Sensor Relay 3
IIoT Temperature Sensor Humidity Sensor
Privacy Sensitive Sensor Data
Edge Device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

Fig 4. 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 [19].

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)
```

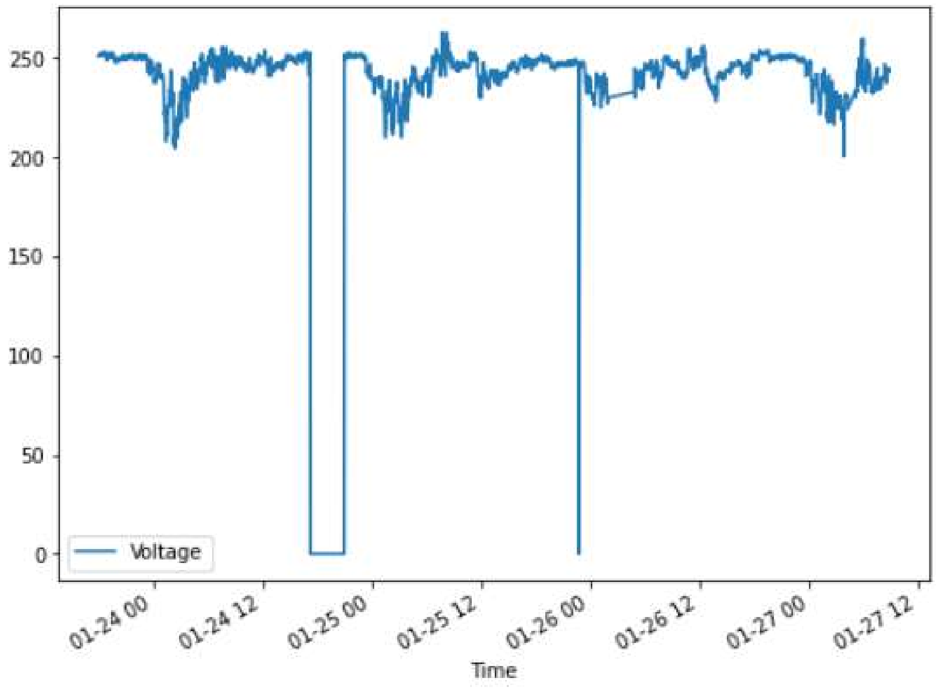


Fig 5. The voltage sensor data across 3 days

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. 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.

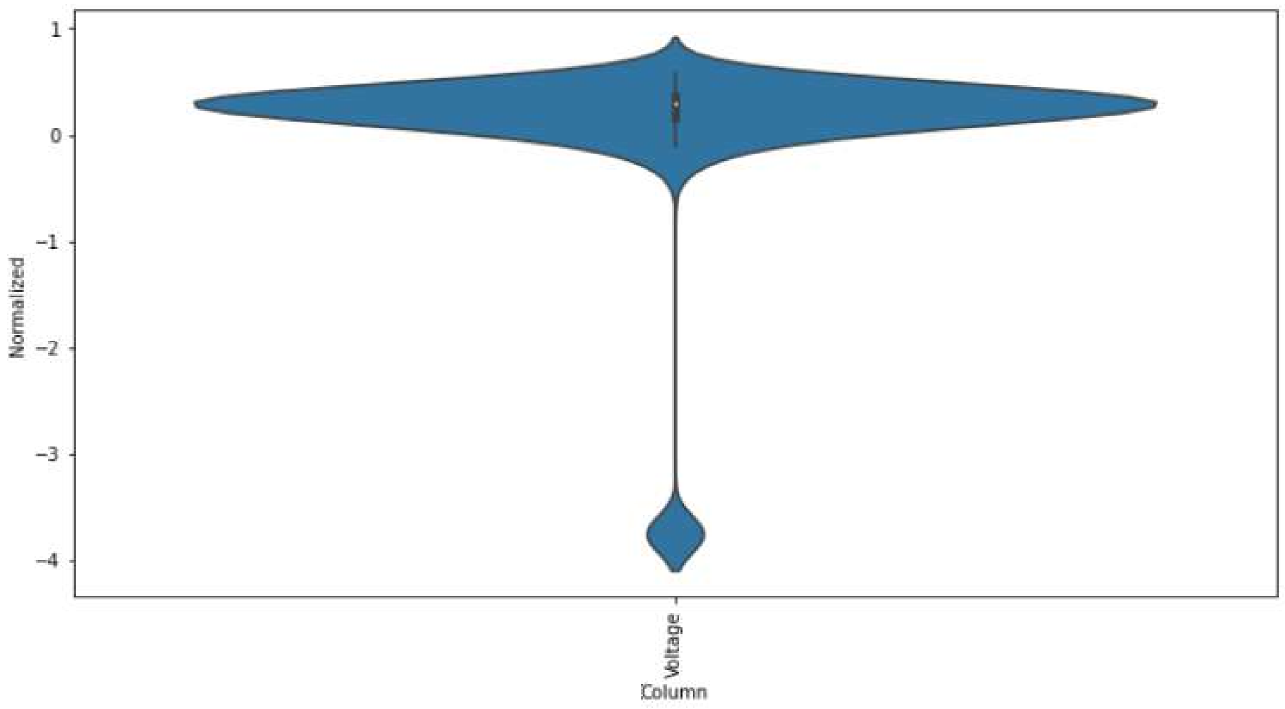


Fig 6a. Voltage sensor data normalization

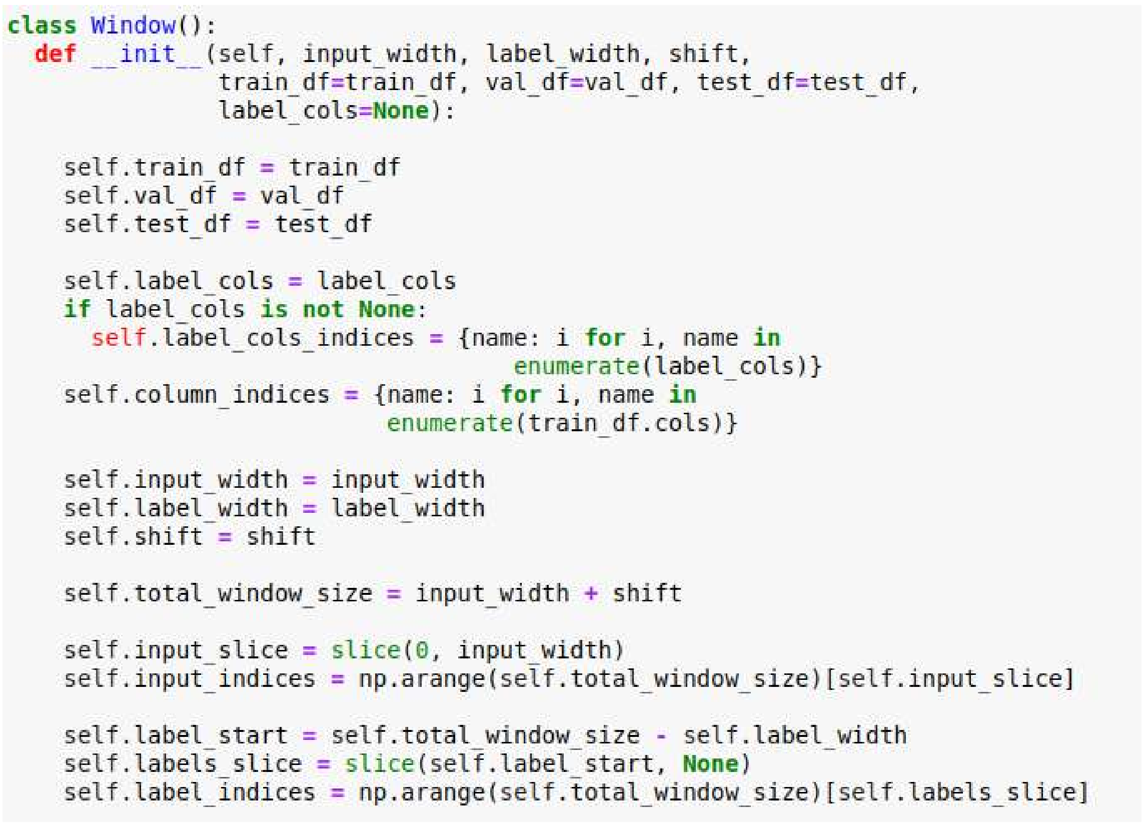
Long tail end data in the figure is the power cut part of the normalization. After the normalization step, we get voltage sensor data that is now as shown in Fig 6a.

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.

Methodology

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 reusable for Linear, DNN, CNN, RNN models.

TensorFlow data is a bunch of arrays [17]. 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.



```
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]
```

Fig 6b. 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 6b.

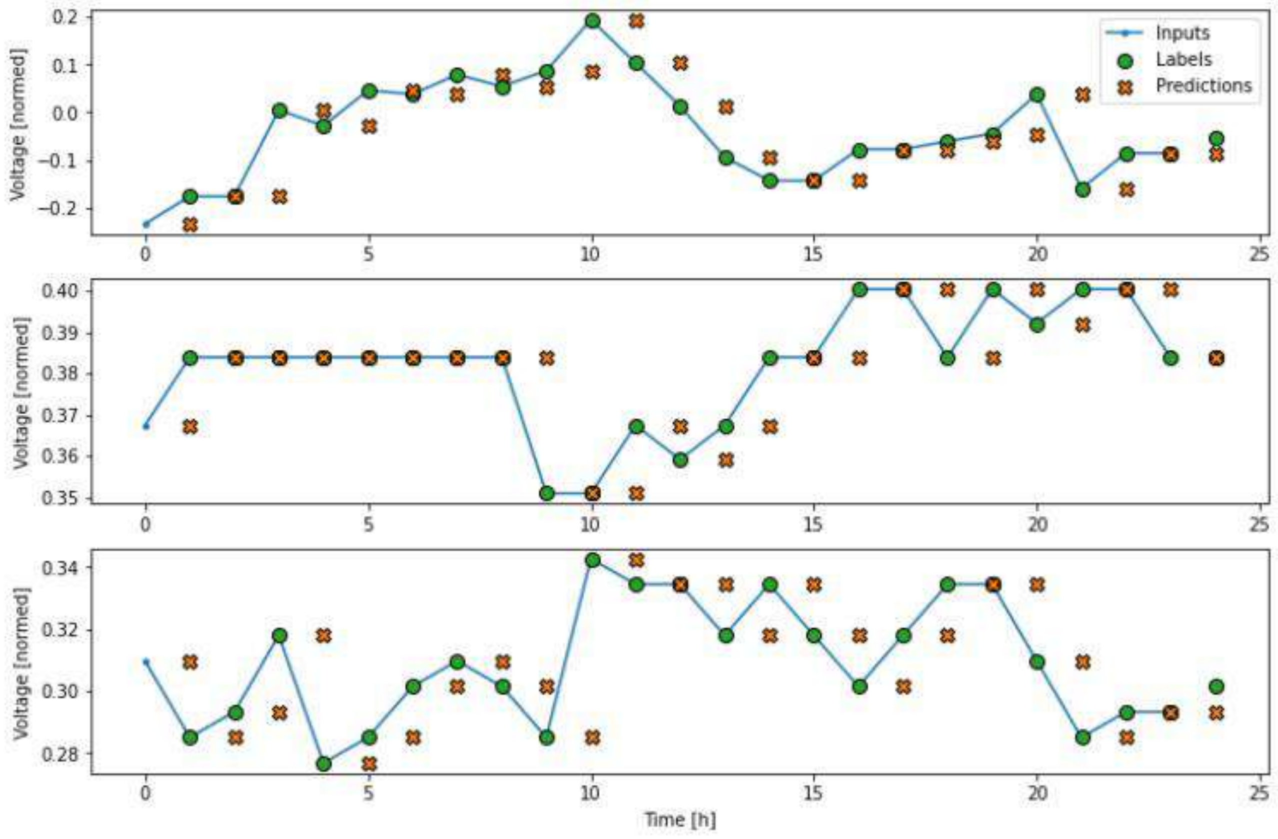
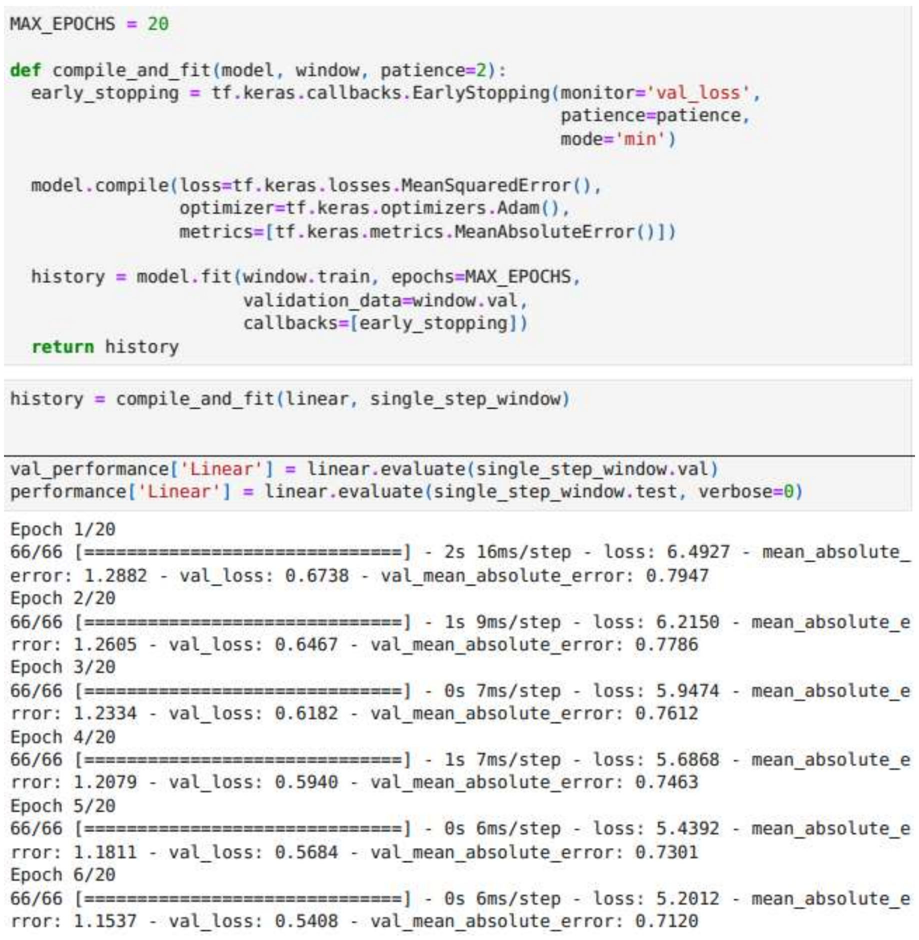


Fig 7. Voltage sensor data inputs with forecasting

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 7.



```
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
```

Fig 8. Performance of the Federated Learning Model on the Edge Devices

---

Algorithm: The R edge devices are indexed by r; *S* is the batch size, η is the learning rate and Ep is the number of epochs on the edge device as in Fig 8.

```
Server executes this algorithm in the experiment :
Initialize $w_0$
for each iteration t = 1,2,3,... do
  max ← max (C · R, 1)
  $S_t$ ← (set of max edge devices)
    for each edge device in $n \in S_t$ do
       $w^r_{(t+1)} \leftarrow \mathrm{UpdateEdge}(r, w_t)$
       $w_{(t+1)} \leftarrow \sum_{r=1}^{R} \frac{n_r}{n} w^r_{(t+1)}$

$\mathrm{UpdateEdge}(r, w)$:  // Execute at edge device r
  S ← (split $P_r$ into sets with size S)
  for each edge epoch k from 1 to Ep do
    for set $s \in S$ do
    $w \leftarrow w - \eta \nabla l(w; b)$
return $w$ to central cloud
```

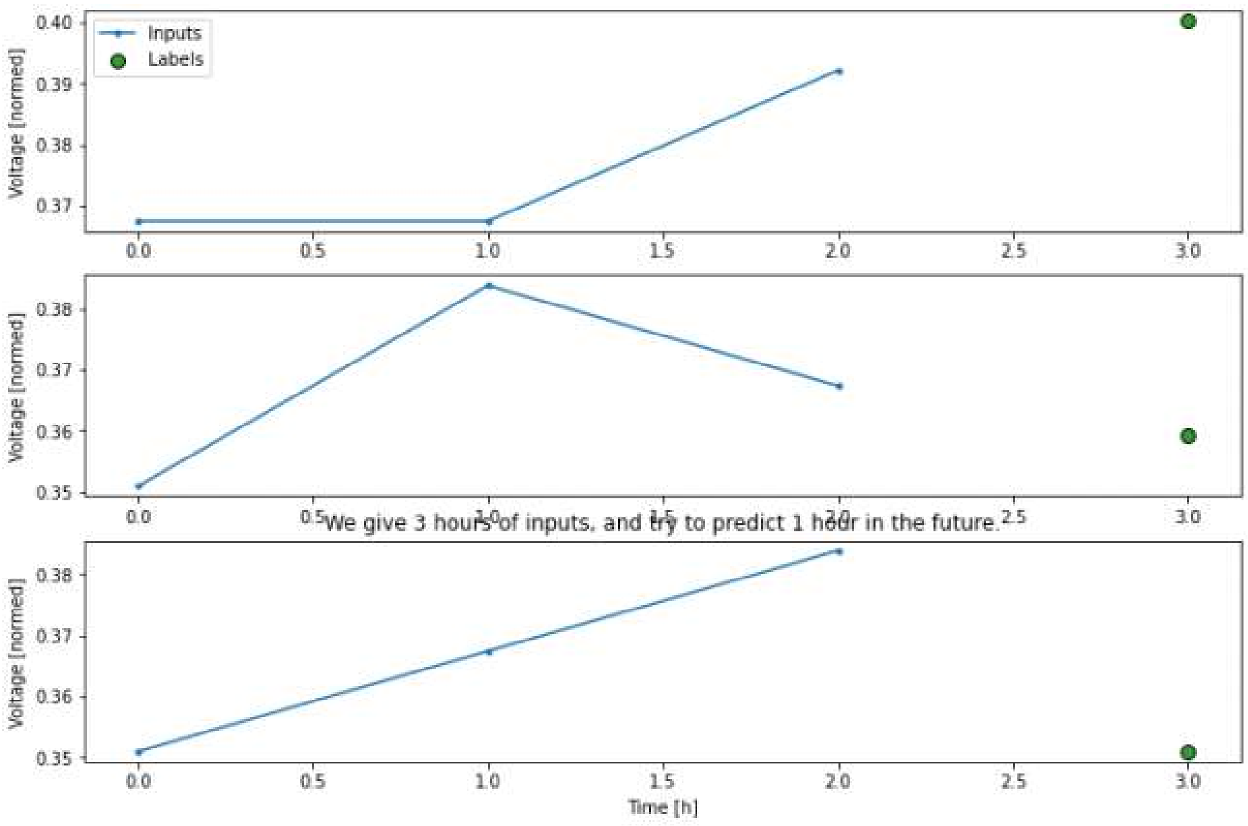


Fig 9. Three hours of inputs given, and a single hour of forecast

As a test, we gave 3 hours of input data and tried to predict 1 hour in the future. 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.10. 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.

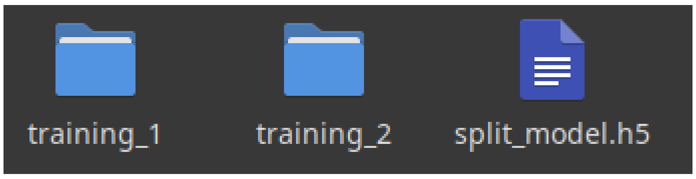


Fig.10 Trained model at the edge - HDF5 format

The training checkpoints can be saved as depicted in Fig.11. 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.

Fig.11 Training checkpoints that can resume on interruption

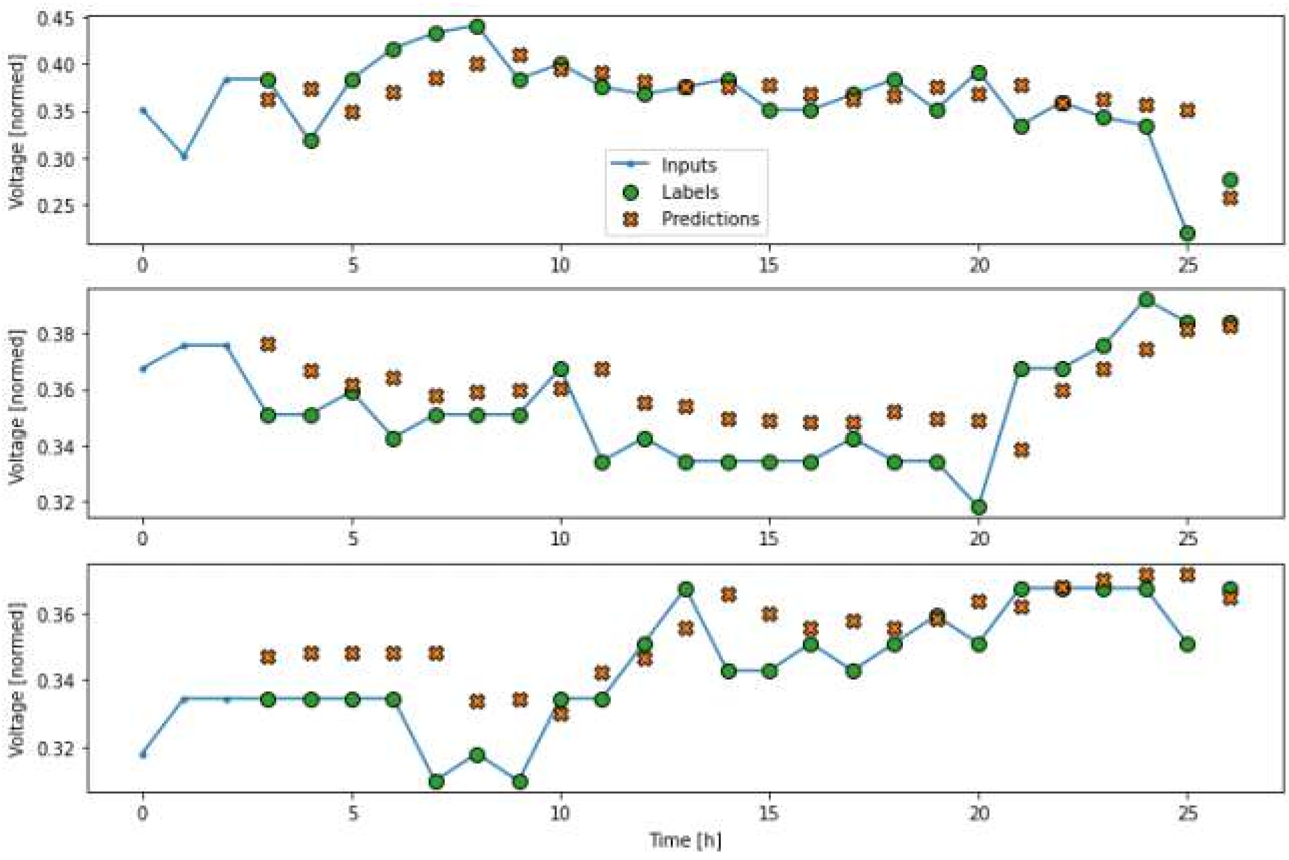


Fig 12. Forecast – Yellow crosses

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) :

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

(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 :

$$p(C_k|x) = \frac{p(C_k)\prod_{i=1}^{n} p(x_i|C_k)}{p(x)}$$

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

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.

7. Contribution

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.

8. Results and Discussion

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 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.

Based on the forecast as shown in Fig.12, 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.

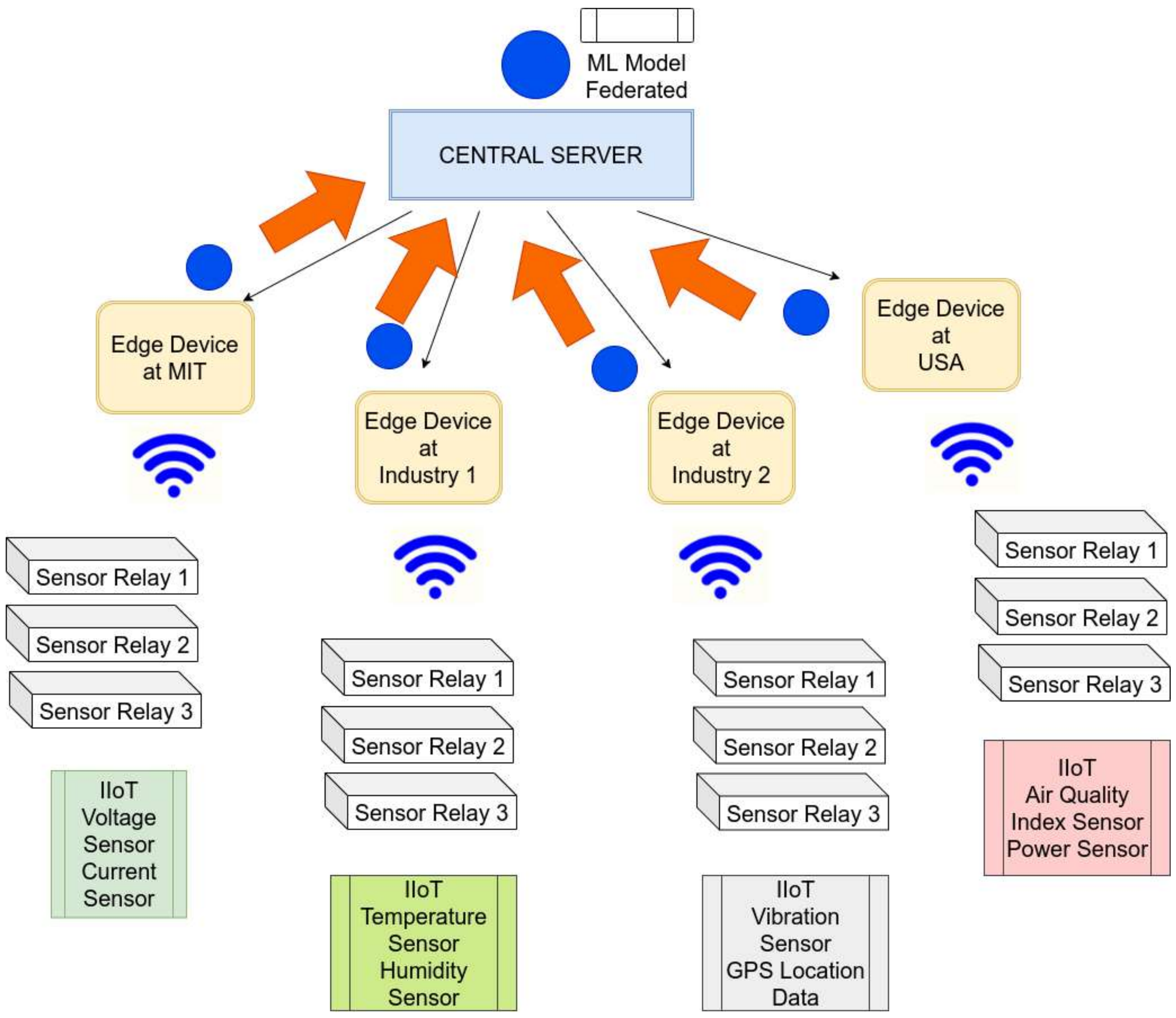


Fig 13. Federated Learning models sent back to the central server

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.

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

Research Date	Study Ref.	IIoT at the Edge	ML at the Edge	Power Efficiency at the Edge
2020	[7]	YES	NO	NO
2021	[8]	YES	YES	NO
2022	[1]	YES	YES	NO

2023	Our work	YES	YES	YES

9. Limitations and future scope

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.

10. Conclusion

The Federated Learning concept (Fig.13) 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.

Compliance with Ethical Standards:

* Disclosure of potential conflicts of interest – All authors declare that there are no potentila conflicts of interest.

* Research involving human participants and/or animals – No human and/or animal participants.

* Informed consent – All are agree to the consent.

References

[1] Christopher Briggs, Zhong Fan, Senior Member, IEEE, and Peter Andras, Senior Member, IEEE, Federated Learning for Short-term Residential Load Forecasting DOI 10.1109/OAJPE.2022.3206220

[2] Mojtaba Vaezi , Senior Member, IEEE, Amin Azari , Member, IEEE, Saeed R. Khosravirad , Member, IEEE, Mahyar Shirvanimoghaddam , Senior Member, IEEE, M. Mahdi Azari , Member, IEEE, Danai Chasaki , Senior Member, IEEE, and Petar Popovski , Fellow, IEEE, Cellular, Wide-Area, and Non-Terrestrial IoT: A Survey on 5G Advances and the Road Toward 6G, IEEE Communications Surveys & Tutorials, Vol. 24, No. 2, Second Quarter 2022

[3] Yi Liu, Neeraj Kumar, Zehui Xiong, Wei Yang Bryan Lim, Jiawen Kang, Dusit Niyato, Communication-Efficient Federated Learning for Anomaly Detection in Industrial Internet of Things DOI: 10.1109/GLOBECOM42002.2020.9348249

[4] Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda, Leandros Maglaras, Helge Janicke, Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning DOI 10.1109/ACCESS.2022.3165809

[5] Taimur Hafeez, Lina Xu, Gavin Mcardle, "Edge Intelligence for Data Handling and Predictive Maintenance in IIOT," 10.1109/ACCESS.2021.3069137, 2021.

[6] Abdul Rehman, Imran Razzak, Guandong Xu, Federated Learning for Privacy Preservation of Healthcare Data from Smartphone-based Side-Channel Attacks, Journal Of Biomedical And Health Informatics, DOI 10.1109/JBHI.2022.3171852

[7] Mahmoud Parto, Christopher Saldana, Thomas Kurfess, “A Novel Three-Layer IoT Architecture for Shared, Private, Scalable, and Real-time Machine Learning from Ubiquitous Cyber-Physical Systems,” ScienceDirect, Procedia Manufacturing 48 (2020) 959-967, 2020.

[8] Mingqian Liu, Ke Yang, Nan Zhao, Yunfei Chen, Hao Song, Fengkui Gong, Intelligent Signal Classification in Industrial Distributed Wireless Sensor Networks Based Industrial Internet of Things, IEEE Transactions on Industrial Informatics, vol. 17, no. 7, July 2021

[9] Yung-Lin Hsu, Hung-Yu Wei, Optimized Data Sampling and Energy Consumption in IIoT: A Federated Learning Approach, IEEE Transactions On Communications, Vol. 70, No. 12, December 2022

[10] Xianyi Cheng, Zhenzhong Jia, Ankit Bhatia, Reuben M. Aronson, Matthew T. Mason, Sensor Selection and Stage & Result Classifications for Automated Miniature Screwdriving, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Madrid, Spain, October 1-5, 2018

[11] Mi Wen , Member, IEEE, Rong Xie , Kejie Lu , Senior Member, IEEE,
Liangliang Wang , and Kai Zhang, FedDetect: A Novel Privacy-Preserving Federated Learning Framework for Energy Theft Detection in Smart Grid, IEEE Internet of Things Journal, vol. 9, no. 8, April 15, 2022

[12] Stefano Savazzi Member, Vittorio Rampa Member, Sanaz Kianoush Member, Mehdi Bennis Fellow, An Energy and Carbon Footprint Analysis of Distributed and Federated Learning, DOI 10.1109/TGCN.2022.3186439

[13] Satya Prakash Yadav, Bhoopesh Singh Bhati, Dharmendra Prasad Mahato, Sachin Kumar, Federated Learning for IoT Applications, ISSN 2522-8595 ISSN 2522-8609 (electronic) EAI/Springer Innovations in Communication and Computing ISBN 978-3-030-85558-1 ISBN 978-3-030-85559-8 (eBook) https://doi.org/10.1007/978-3-030-85559-8, Springer

[14] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Dusit Niyato, Chunyan Miao, Federated Learning Over Wireless Edge Networks, ISSN 2366-1186 ISSN 2366-1445 (electronic), Wireless

Networks ISBN 978-3-031-07837-8 ISBN 978-3-031-07838-5 (eBook) https://doi.org/10.1007/978-3-031-07838-5, Springer

[15] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, Han Yu, Federated Learning, ISBN: 9781687336976 paperback, ISBN: 9781687336983 ebook, ISBN: 9781687336990 hardcover, DOI 10.2200/S00960ED2V01Y201910AIM043, Morgan & Claypool

[16] Wes McKinney, Python for Data Analysis, ISBN: 978-1-449-31979-3, O'Reilly Media, Inc.

[17] MQTT protocol : www.mqtt.org

[18] Tensorflow : www.tensorflow.org

[19] Sudipan Saha and Tahir Ahmad, Federated Transfer Learning: Concept and Applications, arXiv:2010.15561v3 [cs.LG] 6 Mar 2021

[20] A. P. Singh and S. Chaudhari, "Embedded machine learning-based data reduction in application-specific constrained IoT networks," in Proc. 35th Annu. ACM Symp. Appl. Comput., Mar. 2020.

[21] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, "A systematic survey of industrial Internet of Things security: Requirements and fog computing opportunities," IEEE Commun. Surveys Tuts., vol. 22, no. 4, 4th Quart., 2020.

[22] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays Sean Augenstein, Hubert Eichner, Chloé Kiddon, Daniel Ramage, "Federated Learning For Mobile Keyboard Prediction" 1811.03604

[23] Karim Gamal, Ahmed Gaber, Hossam Amer Queen's University, Microsoft, "Federated Learning Based Multilingual Emoji Prediction In Clean And Attack Scenarios", 2304.01005

[24] Faris F. Gulamali Girish N. Nadkarni, "Federated Learning In Risk Prediction: A Primer And Application To Covid-19- Associated Acute Kidney Injury"