IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 24, 2021, accepted August 23, 2021, date of publication August 30, 2021, date of current version September 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3108839

Privacy-Aware Resource Sharing in Cross-Device Federated Model Training for Collaborative Predictive Maintenance

SOURABH BHARTI, (Member, IEEE), AND ALAN MCGIBNEY, (Member, IEEE)
Nimbus Research Center, Munster Technological University, Cork, T12 P928 Ireland

Corresponding author: Sourabh Bharti (sourabh.bharti@cit.ie)

This work was supported in part by the CONFIRM Science Foundation Ireland (SFI) Research Center for Smart Manufacturing, Ireland, in part by the European Union's Horizon 2020 Research and Innovation Program under the Marie Skłodowska-Curie Grant 847577, and in part by the Research Grant from SFI (Ireland's European Structural and Investment Funds Program and the European Regional Development Fund 2014–2020) under Grant 16/RC/3918.

ABSTRACT The proliferation of Industry 4.0 has made modern industrial assets a rich source of data that can be leveraged to optimise operations, ensure efficiency, and minimise maintenance costs. The availability of data is advantageous for asset management, however, attempts to maximise the value of this data often fall short due to additional constraints, such as privacy concerns and data stored in distributed silos that is difficult to access and share. Federated Learning (FL) has been explored to address these challenges and has been demonstrated to provide a mechanism that allows highly distributed data to be mined in a privacy-preserving manner and offering new opportunities for a collaborative approach to asset management. Despite the benefits, FL has some challenges that need to be overcome to make it fully compatible for asset management or more specifically predictive maintenance applications. FL requires a set of clients that participate in the model training process, however, orchestration, device heterogeneity and scalability can hinder the speed and accuracy in the context of collaborative predictive maintenance. To address this challenge, this work proposes a split-learning-based framework (SplitPred) that enables FL clients to maximise available resources within their local network without compromising the benefits of a FL approach (i.e., privacy and shared learning). Experiments performed on the benchmark C-MAPSS data-set demonstrate the advantage of applying SplitPred in the FL process in terms of efficient use of resources, i.e., model convergence time, accuracy, and network load.

INDEX TERMS SplitNN, federated learning, predictive maintenance, Industry 4.0.

I. INTRODUCTION

Predictive maintenance (PdM) [1] of manufacturing assets offer a number of benefits to industry, including proactive asset health management and reduced downtime [2] which are key components for cost effective operation of manufacturing systems. One application of PdM is the ability to forecast an asset's remaining useful lifetime (RUL) in terms of the number of remaining running cycles from the point of prediction beyond which the asset will no longer be able to operate. The RUL prediction is enabled by gathering asset operational data sensed by smart sensors mounted on and around these assets. The sensed data is often gathered and processed by IoT edge devices deployed at the manufacturing site or factory floor to garner various asset failure patterns. These patterns are subsequently used as input to train predictive machine learning models on these edge devices for the specific asset.

It has been shown that a predictive model trained by the failure patterns extracted from a single asset is not robust enough to handle unseen failure patterns [3], [4] and thus may result in inaccurate RUL prediction for a given asset. In other words, a predictive model trained by a richer and more diverse source of failure pattern data will provide a more robust prediction model. One can generate more failure patterns at the cost of running the asset multiple times to its failure, which is expensive and not feasible for small and medium scale organizations [3]. To this end, manufacturing organizations are starting to participate in a data-driven business ecosystem [5] where the asset failure pattern data

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

can be considered as a strategic resource to exchange with each other to create a more robust model that can benefit all participants. The use-case of predicting RUL through this collaborative ecosystem is known as collaborative PdM.

A centralized cloud infrastructure lends itself to be an ideal solution for data pooling to enable collaborative PdM, however, in the manufacturing domain, there is a reluctance to embrace this approach due to concerns relating to the protection of commercial intelligence (due to industrial competition) and the risk associated with sharing asset failure data on public cloud infrastructures. When the assets belong to different organizations, sensitive commercial information can be garnered from the shared data which is an undesired consequence the majority of participating organizations want to avoid. Even a single organization sometimes can be hesitant to centralize its own asset failure data gathered from multiple manufacturing sites due to legal constraints [3]. As a result, most of the failure pattern data-sets sit in silos and are difficult to extract, aggregate and share, which hinders the potential value that can be generated from a large pool of data.

A. FL FOR COLLABORATIVE PdM

The advent of federated learning (FL) [6] offers a potential solution that allows the manufacturing industry to bring isolated data islands together to train better models, while at the same time protecting their commercial intelligence. FL enables multiple IoT edge devices that can be deployed at manufacturing sites or even distributed across multiple organisations, to exchange failure pattern data about a particular asset without the need to share the raw data that can embed potentially sensitive commercial intelligence [3]. In a typical FL process, these edge devices participate as FL clients training the global model with their own data and exchanging global model updates with a centralized server. The server then aggregates the received model updates to create an improved global model, which can be distributed among clients for use in their PdM application. However, continuous improvement in the global model is achieved only by clients participating in multiple iterations of the training process. This iterative exchange of model updates instead of raw manufacturing data allows a manufacturer to remain in control and protect the privacy of this data at all times, while availing of the benefits of the model built through collaboration.

In traditional FL, the data across these clients can share the same feature space (horizontally partitioned) but belong to a different sample ID space [7]. In the context of collaborative PdM, the asset degradation pattern data can be horizontally partitioned across similar types of assets working under similar operating conditions across multiple manufacturing sites. On the other hand, the data across clients can also share the same sample IDs but differ on the features (vertically partitioned). This is when the same asset is being monitored using different type of sensors across manufacturing sites. For instance, the degradation of health of an industrial bearing can be monitored by capturing images in a time-series and/or by recording various other parameters such as vibration, temperature data etc.

Broadly speaking, there are two distinct application settings of a FL approach for collaborative PdM; cross-silo and cross-device [6]. While cross-silo involves geo-distributed data-centers for collaborative learning, cross-device remains the most commonly used FL setting in which clients are identified as resource-constrained IoT edge devices. A cross-device approach can facilitate timely RUL prediction as it takes full advantage of agile edge data analytics. As shown in Fig. 1a, FL clients are generally recognised as single-board edge devices capable of sensing the asset data while having limited battery and computational resources. Edge devices are less reliable (in terms of uncertain availability of computational resources), state-less and usually participate in a horizontal FL of a light-weight global predictive model. A good use-case for cross-device FL in PdM is collaborative model training on similar assets across multiple manufacturing sites, when timely RUL prediction is of paramount importance (e.g. in safety critical scenarios). This work considers a cross-device FL setting with asset failure data horizontally partitioned across multiple IoT edge devices deployed at different manufacturing sites.

B. MOTIVATION

In a cross-device FL scenario for PdM involving heterogeneous edge devices, there can be some FL clients not in possession of enough computational resources to train the global model in a timely manner. These clients can cause a delay in model aggregation and even disconnect during a training iteration. This in turn may hinder other clients from efficient learning of the failure patterns from each others' data. This situation may halt the collaborative prognosis process for a period of time and prove to be costly in scenarios such as predictive maintenance.

As depicted in Fig. 1a, a typical FL client is considered to be a standalone edge device that exchanges the global model parameter updates with the global server. However, in the context of collaborative PdM, a FL client can be part of a local network of an organisation/manufacturing site. Currently, FL provides no provision for communication resource sharing i.e., utilization of the local edge server (Fig. 1b) to interact with a global server. As a result of which, dissemination of a more evolved global model to the clients may get delayed and as a result, the system is unable to predict a failure before it occurs.

C. CONTRIBUTIONS

1) This work proposes a framework (SplitPred) for collaborative PdM which provides a conventional cross-device horizontal FL to support reliable model training at FL clients. This is realised by enabling an IoT edge device (FL client) to off-load part of its model training task to other edge resources that reside on the same network (Fig. 1b).
2) SplitPred utilises split-learning deployment strategies, whose secure data-sharing nature enables edge devices

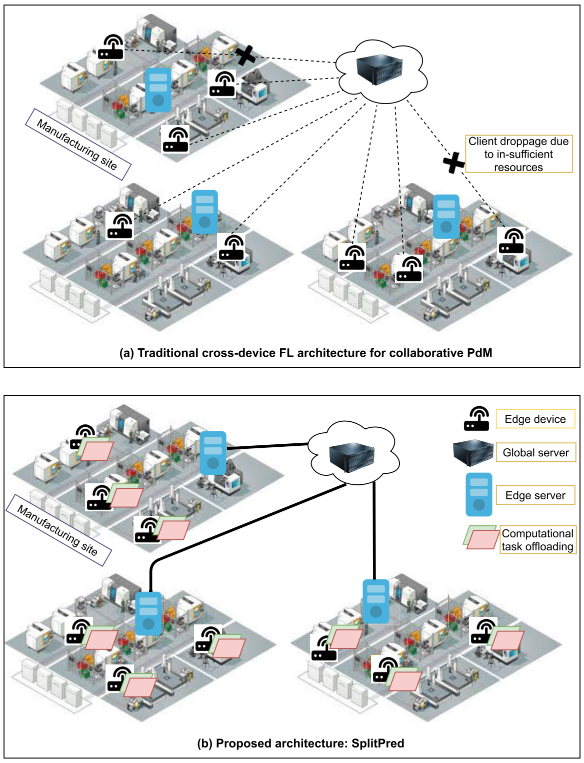
Manufacturing site
Client droppage due to in-sufficient resources
(a) Traditional cross-device FL architecture for collaborative PdM
Edge device
Global server
Edge server
Computational task offloading
Manufacturing site
(b) Proposed architecture: SplitPred

FIGURE 1. Comparison between conventional FL and proposed framework for collaborative PdM.

at FL client's local network to collaboratively train the global model while maintaining data-privacy.

3) SplitPred also ensures that edge devices exchange model updates with the local edge server (Fig. 1b), which subsequently passes them on to the global server for aggregation. This helps expedite the local model convergence time as the edge devices sometimes are not able to have real-time access to the global server [8] which results in delayed model update exchanges.
4) SplitPred is evaluated against a case study of remaining useful lifetime (RUL) prediction for aircraft components in a simulated collaborative PdM set-up using the benchmark C-MAPSS data-set [9]. The results compare SplitPred's performance with traditional FL and centralised approaches using the following metrics: model convergence time, accuracy and network resource consumption.

The remaining of the paper is organised as follows. Section II discusses the related research, section III gives the preliminaries while section IV explains SplitPred's logical architecture and interaction among system components in detail, section V gives the performance evaluation of SplitPred, section VI concludes the paper and finally section VII provides future research scope.

II. RELATED RESEARCH

A. EDGE/FOG BASED AND HYBRID PdM

In [10], an edge based PdM architecture is proposed that utilizes the edge computing capabilities of an IoT based manufacturing system. A cooperation mechanism between edge and cloud computing is also discussed to classify the functionality of edge devices and cloud based resources. It is proposed that edge computing is more suited for real-time processing of manufacturing data while the cloud should be utilised to learn more complex data patterns [11]. The fault detection system proposed in [12] uses a two tier architecture with a real-time fault detector implemented on single-board computers and a fault detector based on Long Term Short Memory (LSTM) executing at the back-end. Other similar proposals [13]–[15] advocate for the utilization of edge/fog computing for IoT based manufacturing. However, none of the above techniques support deployment of the solution fully on edge devices (e.g. single-board computers). In addition, there is no provision for resource sharing among edge devices to support collaborative failure pattern learning. Existing solutions rather limit edge computing interaction to failure detectors that run pre-trained models. Contributions such as [16] propose resource sharing based distributed learning of machine failure pattern by off-loading the computational tasks from a central server to fog nodes (i.e. network edge device deployed close to the edge device) to minimize the system response time for delay-sensitive applications. The model is divided into layers that are allocated to different devices. The framework is not fully edge based and there is no provision of resource sharing among edge devices. Moreover, the privacy preserving aspect of the raw data is not taken into consideration.

B. PRIVACY PRESERVING COLLABORATIVE PdM

Federated learning (FL) [17] has come through as a popular privacy preserving distributed learning technique in recent years. For PdM, it enables manufacturing assets to learn failure patterns from other similar assets without the need reveal their own local data. This creates a virtual network of similar assets distributed across multiple manufacturing sites, this approach is even more beneficial when the assets belong to different organisations. The assets in these virtual networks can be represented by digital agents often referred to as 'digital twins' [18].

One of the primary requirements of data-driven prognosis is for data-sets generated by the participating digital twins to be statistically homogeneous [19]. In the context of PdM, the data-sets generated by two digital twins are statistically homogeneous if they contain similar failure patterns of the corresponding assets working under the same operational conditions. While existing approaches are few and limited to the application of FL [19]–[22] without considering the computational challenges in manufacturing environment, it has been observed that these solutions are able to tackle the data privacy issue in collaborative prognosis. The collaborative prognosis mechanism proposed in [19] deploys a digital twin based FL solution that predicts the RUL of an aircraft engine. A LSTM based predictive model is used for local training and traditional federated averaging technique is used for model aggregation. Results demonstrate the mechanism's ability to match the prediction accuracy of centralised learning while maintaining data privacy. A similar mechanism proposed in [20] predicts the RUL of the aircraft engine with clients using a simple multi-layer perceptron (MLP) model with Kalman filter for training. Results showed that even a simple MLP model can result in good accuracy while maintaining data privacy. Both of these mechanisms use benchmark C-MAPSS data-set for training and testing [9].

The above discussed approaches do not address the challenges of applying conventional FL within a cross-device setting for collaborative PdM. The approach proposed in [22] is designed for cross-silo FL setting. Whereas the one proposed in [19] utilises LSTM as the global learning model which is not suitable for cross-device FL setting. [20] utilises a lightweight MLP based global model (suitable for edge devices) but assumes that the edge devices are equipped with constant availability of computational and communication resources. [21] does not discuss about the application setting of FL for collaborative PdM. Thus, the existing FL approaches for PdM lack provision for computational resource sharing and distributed model deployment among edge devices installed within a FL client's local network. In addition, the existing approaches do not have provision for communication resource sharing which results in a higher number of message exchanges throughout the global model training process.

III. PRELIMINARIES

A. FEDERATED LEARNING

A traditional FL setup (Fig. 2) involves multiple edge devices working as standalone clients with the single global server. Usually, a fixed number of FL clients (D) are selected randomly from a pool of edge devices which have shown their interest in participating in the learning process. For an iteration of training at time t, each participating client (d) downloads the global learning model (θ_t) from the global server and start training the model with it's own local data. If m_d represents the local training data set samples for client d, then $\sum_{d=1}^{D} m_d = M$, where M is the total size of data samples from D clients. The FL attempts to optimize $f(\theta)$.

$$\begin{aligned} &\min_{\theta} f(\theta) \\ &f(\theta) = \sum_{d=1}^{D} \frac{m_d}{M} F_d(\theta) \\ &F_d(\theta) = \frac{1}{m_d} \sum_{i \in m_d} f^i(\theta) \end{aligned} \tag{1}$$

where $f^i(\theta)$ is the loss function associated with sample i in the data-set of client d.

During the local training, client d updates the global learning model using an optimization algorithm such as stochastic gradient descent (SGD) and Adam to minimize their loss functions. At the end of local model training, each client shares the updated global model (θ_{t+1}^d) with the global

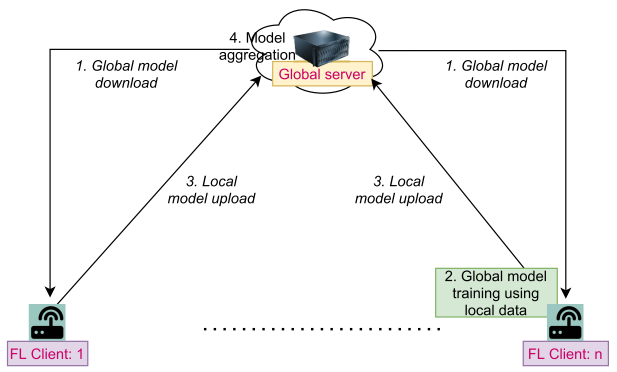

FIGURE 2. Traditional Federated Learning.

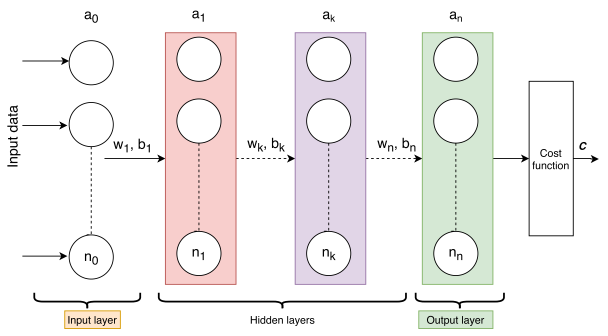

FIGURE 3. A simple neural network design.

server:

$$\theta_{t+1}^d = \theta_t - \alpha_d \lambda_d \tag{2}$$

where α_d is the learning rate and λ_d is the gradient computed at client d on its local data-set with θ_t. The global server aggregates the received local models and computes the improved global model (θ_{t+1}) as follows.

$$\theta_{t+1} = \sum_{d=1}^{D} \frac{m_d}{M} \theta_{t+1}^d \tag{3}$$

The improved global model is subsequently downloaded by FL clients for the next round of training. This process continues until the global model converges.

B. SPLIT LEARNING

Most of the learning models such as neural networks attempt to minimize $F(w, b)$ (Eq. 1) with w and b being the model parameters to be optimised. Fig. 3 shows a simple neural network design with one input layer, k hidden layers and one output layer [23]. Each layer has a number of neurons where n_k is the number of neurons, a_k is the vector of neuron activation, w_k is the weight matrix, b_k is the bias vector at layer k. The training process involves (1) Forward-propagation and (2) Back-propagation.

Forward-propagation: The training data is fed to the input layer which is transformed into intermediate features ($a_1, a_2, \ldots, a_k$) with the help of weights and bias involved between layers. Each layer outputs a weighted vector of neurons (z) which is fed into an activation function $g(z)$ to produce the activation for the next layer. The output from the 'output layer' is fed to a cost function which gives the loss value. There are number of activation and cost functions proposed in the literature that can be used as per the suitability for a specific application.

Back-propagation: The final step of the training process is to back-propagate the loss to update the weights and bias for each layer. This is realised by computing the partial derivatives of the cost function with respect to the weights (w) and bias (b) in the network and updating their values. This process continues until the difference between the current value and previous value is negligible. A simple Gradient Descent [24] update rules are as follows (Eq. 4).

$$w_k = w_k - \alpha \frac{\partial C}{\partial w_k}, \quad b_k = w_k - \alpha \frac{\partial C}{\partial b_k} \tag{4}$$

where α is the learning rate for the algorithm and C is the cost function.

Split learning (details in section IV.C) works on the principle of splitting the neural network and lets different devices process different layers of the model. In other words, the learning model is distributed among multiple devices that consent to aggregate the model at the end of training. A simple split neural network (SplitNN) architecture [25] can be realised using two devices d_1 and d_2 where d_1 has all input (X) values and d_2 has their corresponding *labels*. d_1 trains the X values on the bottom half of neural network and shares the activations and gradients from the split layer with d_2 which, in-turn calculates the loss using *labels* and back-propagates it to d_1. The process continues until the model converges. Here, d_1 and d_2 are learning the same model in a collaborative manner without revealing the training data.

IV. SplitPred: PROPOSED ARCHITECTURE

Fig. 4 represents the logical view of SplitPred in which edge devices deployed on multiple manufacturing sites participate in the collaborative PdM process. The contribution of SplitPred is its ability to enable edge devices in the local network to share their computational and communication resources while training the global model. In addition SplitPred enables edge devices to utilise the already allocated bandwidth to the edge server in order to interact with the global server to reduce the number of message exchanges and the model convergence time. As the collaborative model training at the FL client's local network involves multiple edge devices sharing their resources, any provision to enable this should also make sure that the local data being used for training is well protected. To this end, SplitPred utilises the split neural network (SplitNN) architectures [25] deployed on the local network edge devices to enable privacy preserving resource sharing while training the global model. More details about the SplitNN deployment strategies are given in section IV.C.

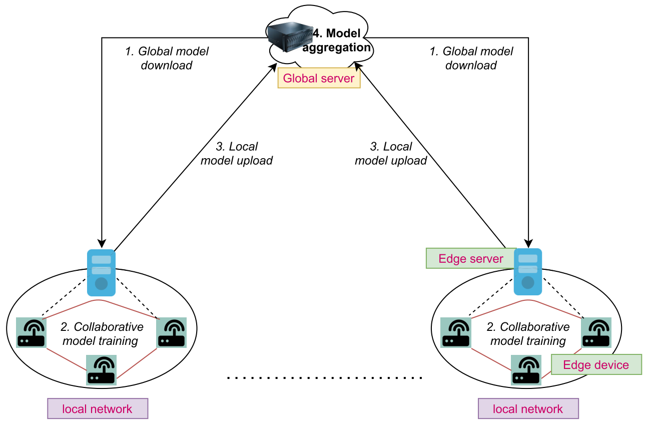

FIGURE 4. SplitPred logical view.

A. SYSTEM COMPONENTS AND ASSUMPTIONS

As the SplitPred is deployed on the local network of a manufacturing site, it utilizes a three-layer local network architecture as follows:

- Field devices: This layer is composed of sensors, actuators etc, responsible to sense and reliably extract parameters from the manufacturing site and receive actuation instructions based on processed manufacturing data. Sensors transmit the raw sensed observations to the preferred edge device (can be via wired or wireless communications protocols e.g. IEEE 802.15.4) to feed the input to the learning models.
- Edge devices: The edge device layer is composed of IoT edge devices that receive raw data from field devices and process it for inference. Edge devices are single-board computers (e.g. Raspberry Pi), equipped with limited computational and communication capabilities (eg. WiFi).
- Edge server: This layer consists of an on-premise application edge server responsible for furnishing the final results to the global server [26]. A SplitPred edge server receives the global model and assigns a suitable edge device for training. It is a computationally rich device to which edge devices can also off-load their neural network segment, although it should be noted that the edge server can also be a single-board computer with similar capabilities as edge device.

It is assumed that the global model is downloaded beforehand by the manufacturing site's local network edge server and the appropriate edge device(s) are designated as FL clients to gather the asset relevant data and utilise it for global model training. Manufacturing assets (machines/equipment) can be represented as a set $A = \{A_1, A_2, \ldots, A_k\}$. The asset parameters, operational condition along with ambient parameters are captured by shop floor sensors, represented as a set $S = \{s_1, s_2, \ldots, s_n\}$. Each sensor is considered as a sensing and communicating device with limited computational capabilities to process the sensed data. Thus, sensors transmit the sensed data to edge devices denoted by $\mathrm{ED} = \{\mathrm{ED}_1, \mathrm{ED}_2, \ldots, \mathrm{ED}_m\}$. Sensors can communicate with edge device within their local network range. In addition, there is one edge server (ES) for the shop floor which interacts directly with the user and/or global server.

The data transmission between sensors and edge device is completed in one-hop (Eq. 5). Two neighbouring edge devices can also communicate with each other in one-hop. Apart from this, all edge devices can communicate with the edge server in one hop (Eq. 6).

$$\forall s_i \in S, \quad \underset{\mathrm{ED}_j \in \mathrm{ED}}{mindist}(s_i, \mathrm{ED}_j) \leq R_{\mathrm{ED}} \tag{5}$$

$$\forall \mathrm{ED}_j \in \mathrm{ED}, \quad mindist(\mathrm{ED}_j, \mathrm{ES}) \leq R_{\mathrm{ES}} \tag{6}$$

Here, R_{ED} and R_{ES} represents the coverage of edge device and edge server, respectively. The distance ($dist$) is Euclidean and can be calculated using the latitude and longitude of the devices.

B. DATA PROCESSING

Edge devices can independently choose their computing strategy in terms of local processing and/or off-loading it to another edge device or edge server. Since the overall objective of SplitPred is to enable agile edge analytics, both local and collaborative data processing are modelled around their respective execution time [27].

1) LOCAL PROCESSING

To fully enable edge analytics, each edge device can participate in training different NN models or segments of different NN models, in a sequential manner. Thus, at time t, an edge device can work on a set of processing tasks $T(t) = \{T_1, T_2, \ldots, T_z\}$. The total amount of processing required for a task T_i can be estimated in terms of the number of layers (l) in NN model (N_i^l) for T_i and the amount of data (d_i) to be used to execute on the layers. Thus the execution time ($ET_i(t)$) for T_i executing on an edge device (ED_1) can be estimated as

$$ET_i(t) = ET_i^q(t) + \frac{C(N_i^l, d_i)}{P(\mathrm{ED}_1)} \tag{7}$$

where ET_i^q and $C(N_i^l, d_i)$ are queuing time, total processing for T_i, $P(\mathrm{ED}_1)$ is the processing capability of ED_1 at time t.

2) OFF-LOADING

An edge device can off-load the part of T_i to another neighbouring edge device or edge server. In other words, an edge device (ED_1) can off-load a segment of the NN to be executed on another device (for eg. ED_2). In this case, the total execution time ($ET_i^o(t)$) of a task T_i can be estimated as a sum of local execution time on ED_1, data transmission time and local execution time on ED_2.

$$ET_i^o(t) = ET_i^{\mathrm{ED}_1}(t) + \frac{d(T_i)}{\lambda} + ET_i^{\mathrm{ED}_2}(t) \tag{8}$$

where $d(T_i)$ is the amount of data transmitted and λ is the data transmission rate which can be modelled using Shannon Capacity Theoerm [28].

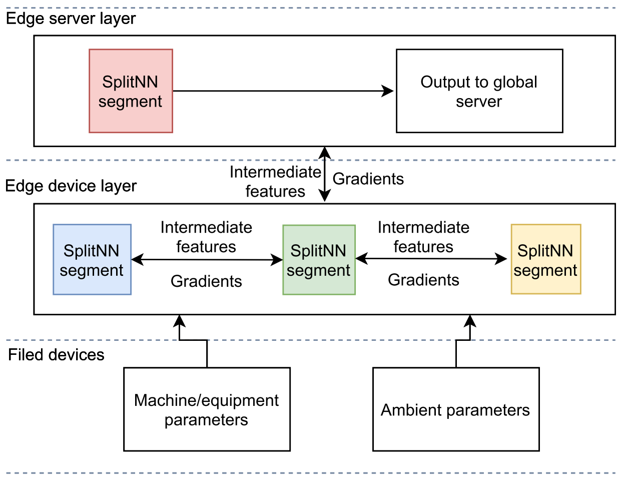

FIGURE 5. Functional view for SplitNN deployment.

3) PROBLEM DEFINITION

With the objective of minimizing the execution time for all tasks, the problem addressed by SplitPred can be defined as:

$$min \sum_{i=1}^{z} \left[ET_i(t) + \sum_{j=1}^{m=|\mathrm{ED}|} ET_i^o(t) \Pr[a_i^o(t) = j] \right] \tag{9}$$

$$s.t. \; \forall T_i \in T, a_i^o(t) \in [0, m] \tag{10}$$

$$\forall T_i \in T, ET_i(t) \geq 0, ET_i^o(t) \geq 0 \tag{11}$$

where $a_i^o(t) \in [0, m]$ is the off-loading indicator taking the value j if the task if off-loaded to ED_j. $\Pr[a_i^o(t) = j]$ is the probability whose value becomes 1 for the ED on which the task is off-loaded to. The constraint in Eq. 10 satisfies the condition of local processing when $a_i^o(t) = 0$. On the other hand, Eq. 11 makes sure that the execution time never becomes negative.

The following subsection discusses the realisation of local processing and off-loading in the context of the SplitPred architecture.

C. PRIVACY PRESERVING RESOURCE SHARING VIA SplitNN DEPLOYMENT STRATEGIES

Central to SplitPred is the distribution of neural network layers across multiple edge devices (Fig. 5), as such there can be a number of SplitNN deployment strategies [25] based on the different combination of neural layer distribution. SplitPred leverages these strategies to enable privacy-preserving resource sharing among edge devices on the local network. Although there can be a number of combinations, this section discusses three deployment strategies (SplitPred1, SplitPred2 & SplitPred3) that can be used as foundational building blocks for any combination required to enable privacy-preserving resource sharing.

1) SplitPred1

Fig. 6 presents the workflow of a simple scenario which involves an edge device (ED) & edge server (ES) in direct

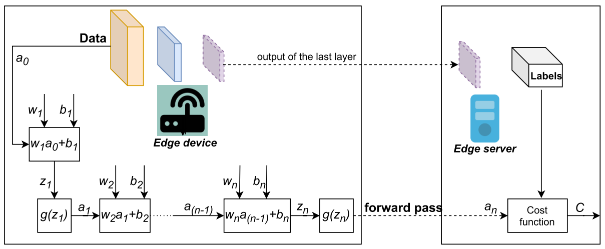

FIGURE 6. SplitNN with two entities.

communication with each other. The SplitNN model is distributed between ED and ES in such a manner that all top layers are kept at ED and the bottom most layer along with labels are kept at ES. ED feeds input data to the sequential executions of $layer_1, layer_2, \ldots, layer_n$ and shares the $layer_n$ output ($output_n$) with ES which feeds the $output_n$ and labels to a cost function $C(output_n$, Labels). As Fig. 6 suggests, this strategy does not involve a split layer and the functionality of ES is limited to cost function implementation and initialization of back-propagation. To initiate the back-propagation, σ (Eq. 12) is calculated at ES and shared with ED where remaining partial derivatives (Eq. 13) are calculated. This process continues until the model converges. In real time processing, this scenario can be used to initialize the split learning process and gradually move towards SplitPred2 deployment strategy.

$$\sigma = \frac{\partial C}{\partial a_n} \tag{12}$$

$$\begin{aligned} \frac{\partial C}{\partial a_{n-1}} &= \sigma . \frac{\partial a_n}{\partial z_n} . \frac{\partial z_n}{\partial a_{n-1}} \\ &\ldots \\ \frac{\partial C}{\partial w_1} &= \frac{\partial C}{\partial a_1} . \frac{\partial a_1}{\partial z_1} . \frac{\partial z_1}{\partial w_1} \end{aligned} \tag{13}$$

2) SplitPred2

In scenarios with more raw data and fast depleting computational and energy resources, ED may opt to off-load the partial execution of neural network to the ES. As per the workflow presented in Fig. 7, SplitPred2 involves ED executing the front end ($layer_1, layer_2, \ldots, layer_k$) of the neural network and shares the activations and gradients corresponding to $layer_k$ with ES while the ES executes the back end ($layer_{k+1}, layer_{k+2}, \ldots, layer_n$) of the neural network. SplitPred2 provides the flexibility for ED to retain the labels as well which further strengthens the raw data privacy aspect. To this end, the output of the last layer ($layer_n$) executed at ES; $output_n$, is sent back to ED and the cost function $C(output_n$, labels) is executed. Unlike SplitPred1, $layer_k$ is considered as the split-layer in SplitPred2 which also enables ES to equally participate in the training process and hence the functionality of ES is not limited to cost function implementation. Back-propagation is initiated by computing the σ (Eq. 12) at ED and sending it to ES where following partial derivatives are calculated.

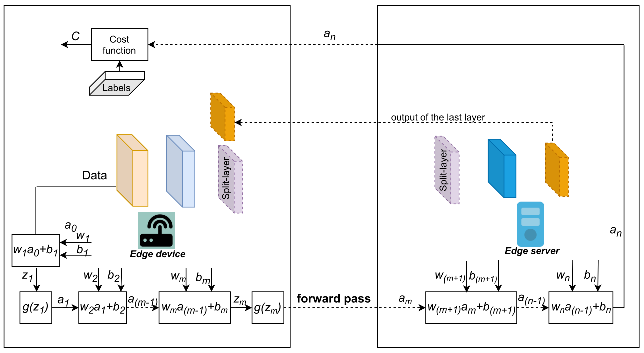

FIGURE 7. SplitNN with edge device retaining labels.

$$\begin{aligned} \frac{\partial C}{\partial a_{n-1}} &= \sigma . \frac{\partial a_n}{\partial z_n} . \frac{\partial z_n}{\partial a_{n-1}} \\ &\ldots \\ \frac{\partial C}{\partial a_k} &= \frac{\partial C}{\partial a_{k+1}} . \frac{\partial a_{k+1}}{\partial z_{k+1}} . \frac{\partial z_{k+1}}{\partial a_k} \end{aligned} \tag{14}$$

ES shares $\frac{\partial C}{\partial a_k}$ with ED in order to complete the back-propagation. ED calculates the remaining partial derivatives as shown in Eq. 15. This process continues until the model converges.

$$\begin{aligned} \frac{\partial C}{\partial a_{k-1}} &= \frac{\partial C}{\partial a_k} . \frac{\partial a_k}{\partial z_k} . \frac{\partial z_k}{\partial a_{k-1}} \\ &\ldots \\ \frac{\partial C}{\partial w_1} &= \frac{\partial C}{\partial a_1} . \frac{\partial a_1}{\partial z_1} . \frac{\partial z_1}{\partial w_1} \end{aligned} \tag{15}$$

As off-loading the back end of the NN to a distant edge server may consume the limited resources of an ED, it may also opt for off-loading the part of its NN to a neighbouring edge device (ED2) using SplitPred3.

3) SplitPred3

SplitPred3 enables the resource sharing among multiple edge devices. As per the workflow presented in Fig. 8, SplitPred3 involves more than one EDs sharing different segments of the SplitNN model while labels are placed with ES. The execution of forward pass and back-propagating the gradients remains similar to the architecture shown in Fig. 6. The first edge device (ED_1) executes the first segment ($layer_1, layer_2, \ldots, layer_k$) of the neural network and shares activations and gradients corresponding to $layer_k$ with another edge device (ED_2) that executes the second segment ($layer_{k+1}, layer_{k+2}, \ldots, layer_j$). The activations and gradients corresponding to $layer_j$ are shared with ES where the last segment ($layer_{j+1}, layer_{j+2}, \ldots, layer_n$) is executed and $C(output_n$, labels) is executed. Here, $layer_k$ and $layer_j$ are two split layers. For back-propagation, ES calculates the following partial derivatives (Eq. 16) and shares $\frac{\partial C}{\partial a_j}$ with ED_2.

$$\begin{aligned} \frac{\partial C}{\partial a_{n-1}} &= \sigma . \frac{\partial a_n}{\partial z_n} . \frac{\partial z_n}{\partial a_{n-1}} \\ &\ldots \\ \frac{\partial C}{\partial a_j} &= \frac{\partial C}{\partial a_{j+1}} . \frac{\partial a_{j+1}}{\partial z_{j+1}} . \frac{\partial z_{j+1}}{\partial a_j} \end{aligned} \tag{16}$$

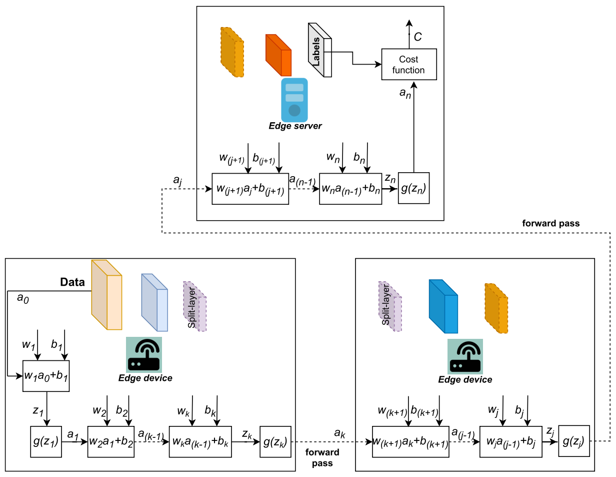

FIGURE 8. SplitNN model segmentation on edge devices.

ED$_2$ calculates the following partial derivatives (Eq. 17) and shares $\frac{\partial C}{\partial a_k}$ with ED$_1$ where the remaining partial derivatives are calculated as per Eq. 15. This process continues till the model converges.

$$\frac{\partial C}{\partial a_{j-1}} = \frac{\partial C}{\partial a_j}\cdot\frac{\partial a_j}{\partial z_j}\cdot\frac{\partial z_j}{\partial a_{j-1}}$$
$$\cdots$$
$$\frac{\partial C}{\partial a_k} = \frac{\partial C}{\partial a_{k+1}}\cdot\frac{\partial a_{k+1}}{\partial z_{k+1}}\cdot\frac{\partial z_{k+1}}{\partial a_k} \quad (17)$$

Each of the deployment strategies outlined limits the sharing of raw manufacturing data among on-premise devices (edge devices and edge server). The participatory edge devices share only the output of their last layer of neural network with each other. This not only ensures data privacy but also reduces the number of message exchanges in the on-premise network. The deployment strategy is determined by the ED specific local parameters such as residual energy, workload etc. There are various offloading decision mechanisms proposed in the literature such as [29], however, this discussion falls outside the scope of the work presented in this paper.

V. PERFORMANCE EVALUATION

This section investigates the efficiency of SplitPred by simulating a collaborative prognosis scenario using publicly available benchmark C-MAPSS data-set [9]. We use a multi-agent system architecture as described in [19] since the architecture has been shown to be well suited for real world industries. Here, the notion of 'agent' refers to a digital agent representing a manufacturing asset and is equipped with storing historical failure data and processing capabilities. From a collaborative prognosis perspective, agents representing similar types of manufacturing assets working under similar operating conditions as other participatory network agents. Hence, the collaborative prognosis is executed among an established group of agents containing independent and identically distributed (IID) data.

The performance of SplitPred is evaluated in terms of its prediction accuracy, model convergence time and the number of messages exchanged between devices. A comparative analysis with traditional FL based mechanisms without the provision of resource sharing is also discussed.

A. DATA-SET MAPPING FOR COLLABORATIVE PdM

The problem observes an aircraft engine that starts with initial operating values captured from various sensors. As the engine operating time (in terms of operation cycles) increases, it begins to degrade. This degradation is captured by the sensor values used to train a predictive model. The lifetime of an engine is estimated in terms of the number of operation cycles before the engine runs to failure. The engine run-to-failure time-series data is available for training and testing. The goal is to predict the RUL based on the time-series data which ends some cycles prior to the last operation cycle (i.e. failure).

The benchmark data-set [9] is composed of simulated engine degradation time-series observations. Each time-series (FD001, FD002, FD003, FD004) contains training data, testing data and RUL values (ground truth). Each tuple of the training/testing time-series consists of engine failure trajectory-id, cycle count, three operating setting and twenty one sensor observations. An engine failure trajectory is composed of various operation cycles. The sensor values at the beginning of each operation cycle in a trajectory are similar. The time-series FD001 & FD003 contain data of turbofans operating under the same condition throughout [19], it fits well to a cross-device FL setting with asset failure data horizontally partitioned across multiple IoT edge devices. Thus similar to existing FL based solutions such as [19], time-series FD001 is used throughout the experiments. In practice, time-series FD001 data is distributed equally among intelligent agents hosted by IoT edge devices constituting a collaborative PdM scenario. Each agent can train the global model using it's local time-series data of failure trajectories. The experiments performed in this section pertain to the applicability of SplitPred as a framework to enable participatory agents off-load a part of their global model training task to the nearest agent on its local network.

B. PRE-PROCESSING AND GLOBAL MODEL ARCHITECTURE

A linear degradation approach [20] is used for RUL estimation. Although, it is acknowledged that a piece-wise degradation approach results in better accuracy, it requires a number of experiments prior to determine the optimal initial RUL value. This is not feasible in real-time edge based collaborative learning such as SplitPred and thus more suited for centralised learning. From exploratory data analysis, measurements from sensor 1, 5, 6, 10, 16, 18 and 19 have been observed to be constant throughout the engine operation time and thus viewed as less useful in providing insight regarding their effect on engine lifetime. Hence, these features are not

TABLE 1. Predictive model architecture.

Layer	InN	OuN	Activation
1	18	16	ReLU
2	16	16	ReLU
3	16	1	Sigmoid

considered for the training and testing. To avoid bias in the learning process, MinMax scaler is used to normalise the sensor values (features) [19]. As a result of this, all features are transformed into values between 0 and 1.

A simple multi-layer NN model (Table 1) is used for training and prediction. The rationale behind choosing a simple NN model is the limited computational resources available with edge gateways. Moreover, optimal learning model selection is not in the scope of this proposal as it attempts to investigate the applicability of SplitNN on edge. NN is also the benchmark predictive model used to implement FL based solutions. The model used in this work has 2 hidden layers, 1 input and 1 output layer. InN and OuN (Table 1) are the number of input and output neurons respectively. Root Mean Square Error (RMSE) loss function (Eq. 18) is used to train the NN and Rectified Linear Unit (ReLU) is the activation function used for initial layers. Whereas, the output layer uses sigmoid activation function. RMSE is particularly used against Mean Absolute Error (MAE) loss function since it helps magnify the large errors which are important in time-critical scenarios. Moreover, it has been reported in the literature [30] that model convergence is smoother with RMSE when compared with MAE.

$$\begin{aligned} \text{RMSE} &= \sqrt{\frac{1}{m}\sum_{i=1}^{m} l_i^2} \\ \text{ReLU}(z) &= max(0, z) \\ \text{Sigmoid}(z) &= \frac{1}{1+e^{-z}} \end{aligned} \tag{18}$$

where m is the number of examples in the data, l_i is the loss in ith example and computed as $PredictedRUL_i - ActualRUL_i$. The same predictive model is used to implement both centralised and FL based solutions also for comparative analysis.

C. EXPERIMENT DESIGN

The experiments are simulated using Pysyft [31] where digital agents work as Pysyft virtual workers. In a local network, *field device layer* of the experimental set-up consists of twenty one sensors capturing the **machine parameters** and {trajectory-id, cycle count, three operating settings} capturing the **ambient & operational parameters**. These values are fed to the *edge device layer* that employs two edge gateways (ED$_1$ and ED$_2$) hosting virtual workers communicating with one server (ES) employed at the *Edge server layer*.

1) MODEL TRAINING SCHEDULE

Initial experiments are performed based on SplitPred1 with ED$_1$ executing all three layers of the NN model (section VI.B) whereas the labels (RUL values) are kept at ES. To enable SplitPred2, ED$_1$ offloads the last layer (16, 1) of the NN model to the ES and keeps the labels to strengthen the data privacy aspect. In SplitPred3, ED$_1$ offloads the second layer (16, 16) to ED$_2$ whereas the labels and the last layer (16, 1) are kept at ES. Global model training is performed on all three scenarios to test their ability to capture the failure pattern, observe convergence time and the number of message exchanges in the training process.

D. RESULTS AND COMPARATIVE ANALYSIS

This section investigates the performance of FL client for the model convergence time, test data accuracy, number of data-points exchanges in one round of federated training and memory consumption. The discussion on results pertain to the performance of FL client with the following implementation scenarios: (1) conventional FL; when a FL client is considered as a standalone edge device without the provision of resource sharing and its direct interaction with global server is in place. (2) SplitPred; when a FL client utilises SplitPred(1, 2 &3) to off-load part of its global model training to another edge device/server in the local network and utilises allocated bandwidth to the edge server to interact with the global server.

1) HYPER-PARAMETER TUNING

Fig. 9 and 10 depict the effect of learning rate (lr) and the model optimizer (*Adam* and *SGD*) on the convergence time and accuracy of SplitPred(1, 2 & 3). The convergence time of a predictive model represents the number of iterations required for it to converge to an optimal loss value. This parameter becomes crucial in edge based collaborative learning as each iteration of the predictive model not only consumes computational resources of edge devices, but also involves a number of message exchanges between them. The number of iterations to reach an optimal loss value is a valid indicator for the convergence time. Since RUL prediction is a regression problem, RMSE was chosen as the measurement of prediction accuracy. A low RMSE value represents better accuracy. As evident from Fig. 9, SplitPred2 exhibits the lowest RMSE value and fastest convergence with hyper-parameters: lr =.005 and *Adam* optimizer. Similar optimal hyper-parameters can be observed for SplitPred1 and SplitPred3 also. The subsequent results used for comparative analysis correspond to the optimal hyper-parameter values for SplitPred1, SplitPred2 and SplitPred3. The results shown for conventional FL client also correspond to the optimal hyper-parameters; lr =.005 & optimizer = *Adam*. The federation rounds are kept as 10 while the local iterations are varied between 10-40. The accuracy and model convergence time is shown in Fig. 11 and 12, in addition a centralised approach is implemented using the same global model parameters

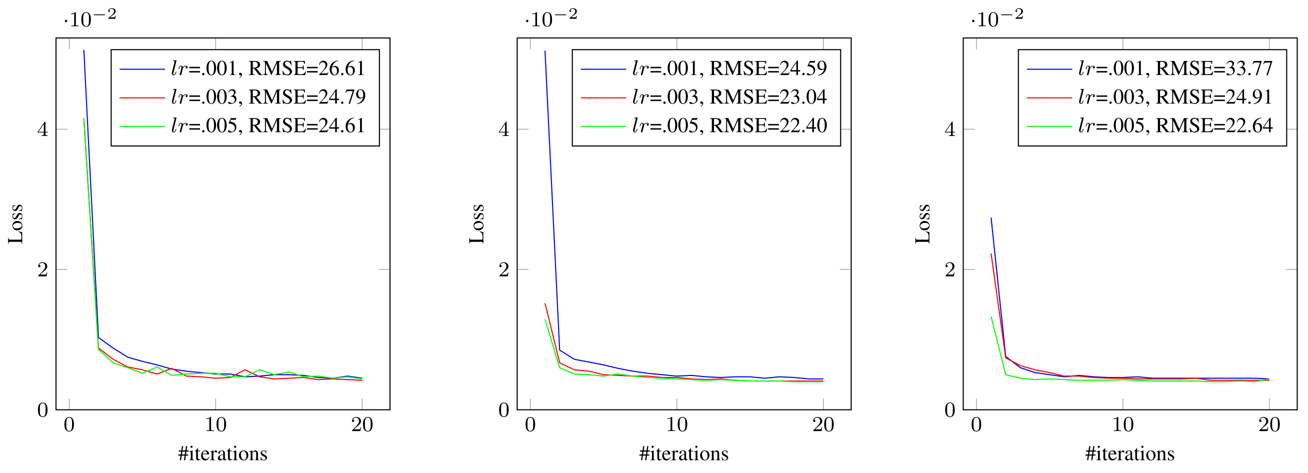

FIGURE 9. Convergence time and RMSE values of SplitPred1, SplitPred2 & SplitPred3 with different learning rates for *Adam* Optimizer.

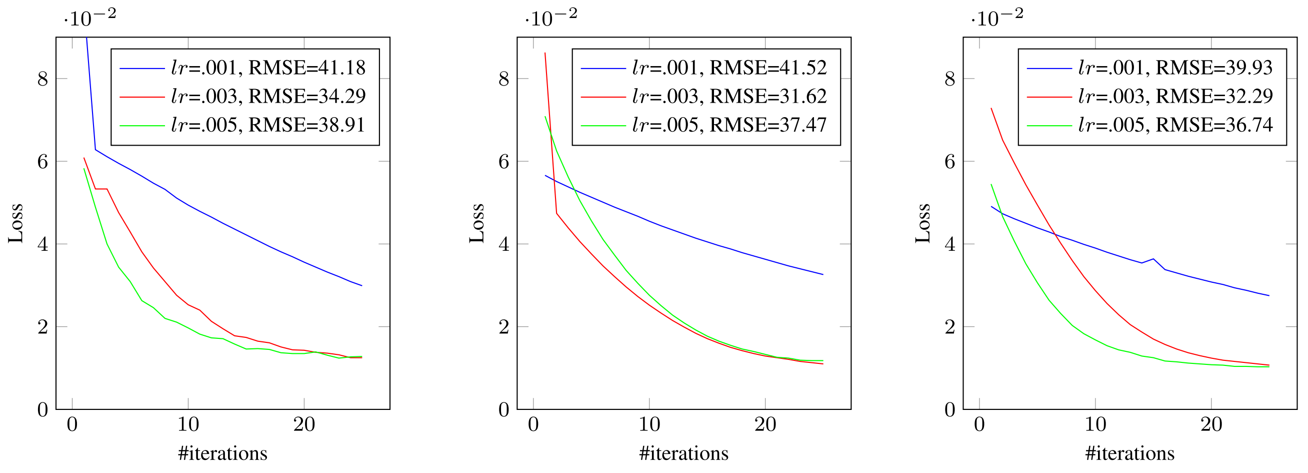

FIGURE 10. Convergence time and RMSE values of SplitPred1, SplitPred2 & SplitPred3 with different learning rates for *SGD* Optimizer.

TABLE 2. Effect of activation functions on model (Table 1) training time.

Layer 1	Layer 2	Layer 3	Training time
Sigmoid	Sigmoid	Sigmoid	308s
ReLU	**ReLU**	**Sigmoid**	**303s**
ReLU	ReLU	ReLU	304s

presented in Table 1. This allows a comparison against a centralised approach with the added benefit of data-privacy. However, to ensure the fairness in comparative analysis, the centralised approach results are not compared in terms of the number of message exchanges.

The parameters in Table 1 are also varied to investigate their effect on model training time and accuracy. The rationale behind using ReLU activation function is to minimize the training time as ReLU is computationally efficient when compared with Sigmoid. Table 2 shows the effect of using different activation functions at different layers of the predictive model whereas Table 3 shows the effect of varying number of neurons in the hidden layers. It was observed that choice of activation function has no effect on

TABLE 3. Effect of number of neurons in hidden layers.

# neurons in hidden layers	Training time	RMSE	Activation functions
8, 8	327s	31.65	ReLU, Sigmoid
16, 16	**303s**	**22.40**	**ReLU, Sigmoid**
32, 32	313s	22.54	ReLU, Sigmoid
64, 64	308s	22.23	ReLU, Sigmoid

RMSE value. On the other hand, the number of neurons in each layer affected RMSE values. The results shown corresponds to SplitPred2 with *Adam* optimizer, learning rate of.005, 25 iterations.

2) CONVERGENCE TIME

The results shown in Fig. 11 depicts the number of iterations took by an FL client to converge to an optimal loss value with SplitPred. As evident from Fig. 11, the convergence time for SplitPred1, SplitPred2 & SplitPred3 overlap with each other and is very close to the centralised approach whereas

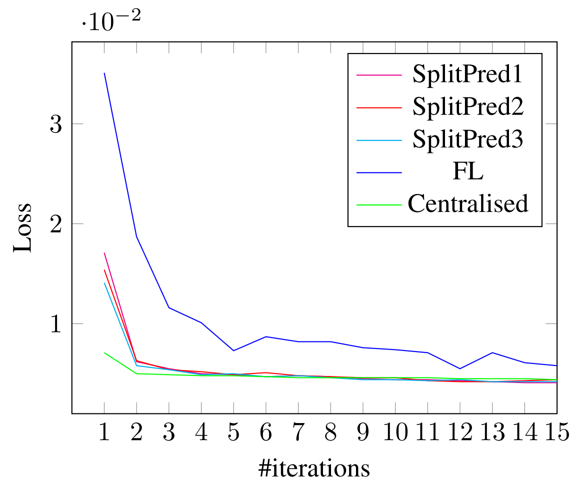

FIGURE 11. Convergence time comparison with conventional FL.

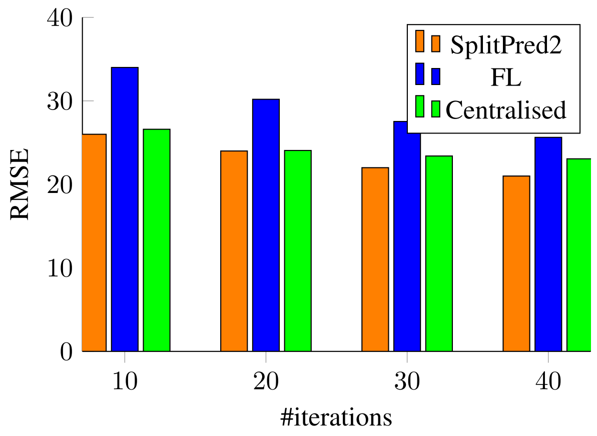

FIGURE 12. RMSE value comparison with conventional FL.

conventional FL client takes more time to converge to an optimal loss value. The explanation for slower convergence of conventional FL client is the lack of sufficient computational and communication resources to train the global model whereas SplitPred deals with this issue leveraging computational off-loading and minimizing the number of messages in global model update exchange. From application point of view, faster convergence in SplitPred can also reduce RUL estimation response time which can prove to be crucial for time-critical manufacturing applications.

3) ACCURACY

The RMSE values shown in Fig. 12 correspond to the local model accuracy achieved by the FL client. For representation purposes, SplitPred2 is used for this comparison as it exhibits minimum RMSE values (Fig. 9, 10). The RMSE values achieved with SplitPred2 are closer to the centralised processing with the same global model (shown in Table 1). On the other hand, due to the increased model convergence time, conventional FL client takes more iterations to reach to an optimal RMSE values when compared with SplitPred2. In order to further compare SplitPred's performance in terms of accuracy, other sophisticated centralised machine

TABLE 4. RMSE value comparison with other centralised ML approaches.

SplitPred2	Centralised			
	MLP [33]	SVR	CNN [33]	Deep LSTM [32]
22.40	37.56	23.96	18.45	**16.14**

learning algorithms such as Support Vector Machine (SVM), Convolutional Neural Network (CNN) and Deep LSTM [32] were executed. Table 4 suggests that SplitPred outperforms centralised MLP and support vector regression (SVR) based techniques whereas Deep LSTM results in the best accuracy among all. It is to be noted that there are no efficient means yet to implement split-learning using LSTM or RNN [34]. Moreover, if the computational complexity is to be represented as O(W), then $W = 4IH+4H^2+3H+HK$ for Deep LSTM [35]. On the other hand, for SplitPred, $W = IH+HK$. Where I, H and K represent the number of input units, hidden units and output units, respectively. Using more complex measures such as CNN, LSTM with split-learning or increasing the number of layers come with additional computational burden which is not suitable for resource-constrained edge devices and diminishes the purpose of utilising split-learning [34] for training global model at FL client. The overall objective of cross-device FL mechanisms is to support agile data analytics which is usually enabled by executing lightweight learning models on the edge.

On the other hand, SplitPred uses a simple NN with only two hidden layers to make it suitable for resource-constrained IoT devices. A simpler NN model is used for this case study since the model prediction accuracy is not the prime concern of this proposal which attempts to introduce privacy preserving resource sharing for FL when applied for collaborative PdM. Centralised techniques also have their limitations in terms of data-privacy and huge number of message exchanges. Thus, it can be concluded from Table 4 that SplitPred not only ensures data privacy but also maintains a reasonable level of accuracy while being computationally lightweight as compared with CNN and Deep LSTM.

4) NUMBER OF MESSAGE EXCHANGES AND MEMORY OVERHEAD

Fig. 13 shows the number of data-points exchanged among network devices against the number of examples (samples) in the data-set for one iteration of the training process. It shows that when compared with conventional FL, SplitPred exhibits more than a 10 fold decrease in the amount of data exchanged. This is because the edge devices in SplitPred only interact with the local edge server which is not the case in FL. Moreover, in SplitPred, edge devices share only the output of the last layer executed on them while collaboratively training the global model in the local network which minimizes the number of data-points to be shared. This result also reveals that although different deployment scenarios result in the

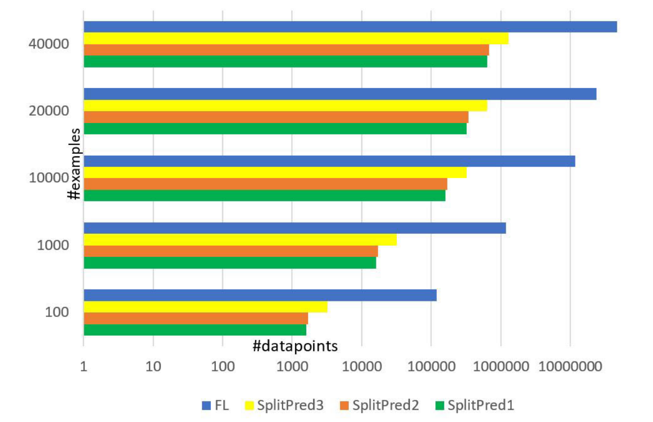

FIGURE 13. Comparison of number of data-points shared in one iteration.

TABLE 5. Memory overhead comparison.

SplitPred1	SplitPred2	SplitPred3	FL
41 MB	55 MB	75 MB	**11 MB**

same accuracy values, they may affect the number of data-points to be shared among edge devices. The increased number of data-points in SplitPred3 is the direct outcome of message exchanges while offloading the number of layers from ED_1 to ED_2.

Table 5 shows the comparison among the SplitPred scenarios and FL in terms of their collective (for all devices involved) memory requirement. It is evident from the results that FL consumes much less memory when compared with SplitPred. The reason for this is that in one iteration of FL each participating edge device shares its model updates with the global server which aggregates them and shares the update with each edge device. Neither edge device nor server has to remember/store the model parameters learnt in the previous iteration. Whereas in SplitPred, since the model is distributed among multiple edge gateways, all of these devices have to store the model parameters until the back-propagation is complete. This storage of the model parameters at each device contributes to the collective memory requirement. As evident from Table 5, the memory requirement for SplitPred grows with the number of devices involved in the training process. However, while the collective memory requirement is greater than FL, it is viewed as acceptable and managable within modern IoT edge devices. Moreover, the ability to off-load allows for the provision to share this memory burden among edge devices involved.

VI. CONCLUSION

In this article, a resource sharing framework known as SplitPred was proposed to enable local edge devices within a FL client's network, to collaboratively train a global model applied in the context of collaborative PdM application. Various split-learning based distributed model deployment strategies were also applied and tested for their convergence time, accuracy, number of message exchanges and memory requirements. Comparison with state-of-the-art FL based mechanism revealed that the utilisation of SplitPred results in improved convergence time, accuracy along with reduced number of message exchanges during the training process. However, FL outperforms SplitPred in terms of memory requirement to train the global model.

VII. FUTURE SCOPE

Finally we present the following open issues for the future research:

Trade-off between global model accuracy and computational overhead: This work employs a simple NN model with two hidden layers as the global learning model for collaborative PdM. The simplicity of NN attempts to maintain the trade-off between accuracy and the computational overhead on resource-constrained edge devices. However, other centralised models such as LSTM and CNN result in better accuracy but are computationally intensive. This makes these models not suitable for resource-constrained edge devices and can prove to be counter-productive in scenarios which attempt to minimize network and edge resource consumption. Thus, the trade-off between accuracy and computational overhead is still an open issue. Although split-learning cannot be used along with LSTM, lightweight CNN models (both 1D & 2D) can be explored to further improve the accuracy.

Privacy leakage at the local network and model exchange: The possible privacy leakage [34] at the split-layer is often handled with (1) increasing the number of hidden layers at the off-loading device and (2) use of differential privacy. However, both of these techniques and especially differential privacy can suffer from reduced global model accuracy. Therefore, reducing the risk of privacy leak while maintaining the global model accuracy needs to be addressed. Technologies such as Blockchain can be utilised to enable secure and reliable model exchanges as the consensus mechanism involved can help identify the potential malicious FL client(s).

Memory overhead for local edge devices: Global model training using Split-learning involves storage of gradients by each device involved in the training process which in turn results in increased memory overhead on edge devices on the local network. Data compression techniques can be explored to reduce the memory requirement while making sure that the compression/decompression process does not contribute to the model convergence time.

REFERENCES

[1] J. J. M. Jimenez, S. Schwartz, R. Vingerhoeds, B. Grabot, and M. Salaün, "Towards multi-model approaches to predictive maintenance: A systematic literature survey on diagnostics and prognostics," *J. Manuf. Syst.*, vol. 56, pp. 539–557, Jul. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612520301187

[2] M. Sharp, R. Ak, and T. Hedberg, "A survey of the advancing use and development of machine learning in smart manufacturing," *J. Manuf. Syst.*, vol. 48, pp. 170–179, Jul. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612518300153

[3] M. Mohr, C. Becker, R. Möller, and M. Richter, "Towards collaborative predictive maintenance leveraging private cross-company data," in *Proc. GI-Jahrestagung*, 2020, pp. 1–6.

[4] Z. Balogh, E. Gatial, J. Barbosa, P. Leitao, and T. Matejka, "Reference architecture for a collaborative predictive platform for smart maintenance in manufacturing," in *Proc. IEEE 22nd Int. Conf. Intell. Eng. Syst. (INES)*, Jun. 2018, pp. 299–304.

[5] *International Data-Space Association*. Accessed: Mar. 2021. [Online]. Available: https://www.fraunhofer.de/content/dam/zv/en/fields-of-research/industrial-data-space/IDS-Reference-Architecture-Model.pdf

[6] E. B. P. Kairouz and H. B. Mcmahan, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, no. 1, pp. 1–210, 2021, doi: 10.1561/2200000083.

[7] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, Feb. 2019, doi: 10.1145/3298981.

[8] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. H. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Commun. Mag.*, vol. 58, no. 10, pp. 88–93, Oct. 2020.

[9] *PHM08 Challenge Data Set*. Accessed: Mar. 2021. [Online]. Available: http://ti.arc.nasa.gov/project/prognostic-data-repository

[10] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge computing in IoT-based manufacturing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 103–109, Sep. 2018.

[11] P. Bellavista, R. D. Penna, L. Foschini, and D. Scotece, "Machine learning for predictive diagnostics at the edge: An IIoT practical example," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[12] D. Park, S. Kim, Y. An, and J.-Y. Jung, "LiReD: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks," *Sensors*, vol. 18, no. 7, p. 2110, Jun. 2018, doi: 10.3390/s18072110.

[13] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4665–4673, Oct. 2018.

[14] Z. Zhou, J. Hu, Q. Liu, P. Lou, J. Yan, and W. Li, "Fog computing-based cyber-physical machine tool system," *IEEE Access*, vol. 6, pp. 44580–44590, 2018.

[15] T. M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, and M. Vilar-Montesinos, "A fog computing and cloudlet based augmented reality system for the industry 4.0 shipyard," *Sensors*, vol. 18, no. 6, p. 1798, 2018.

[16] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4225–4234, Jul. 2019.

[17] *Federated Learning: Collaborative Machine Learning Without Centralized Training Data*. Accessed: Jan. 2021. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[18] M. Liu, S. Fang, H. Dong, and C. Xu, "Review of digital twin about concepts, technologies, and industrial applications," *J. Manuf. Syst.*, vol. 58, pp. 346–361, Jan. 2021. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612520301072

[19] M. Dhada, A. K. Jain, M. Herrera, M. P. Hernandez, and A. K. Parlikad, "Secure and communications-efficient collaborative prognosis," *IET Collaborative Intell. Manuf.*, vol. 2, no. 4, pp. 164–173, Dec. 2020.

[20] R. H. L. Rosero, C. Silva, and B. Ribeiro, "Remaining useful life estimation in aircraft components with federated learning," in *Proc. PHM Soc. Eur. Conf.*, vol. 5, no. 1, 2020, p. 9, doi: 10.36001/phme.2020.v5i1.1228.

[21] N. Aussel, S. Chabridon, and Y. Petetin, "Combining federated and active learning for communication-efficient distributed failure prediction in aeronautics," 2020, *arXiv:2001.07504*. [Online]. Available: http://arxiv.org/abs/2001.07504

[22] R. Kanagavelu, Z. Li, J. Samsudin, Y. Yang, F. Yang, R. S. M. Goh, M. Cheah, P. Wiwatphonthana, K. Akkarajitsakul, and S. Wangz, "Two-phase multi-party computation enabled privacy-preserving federated learning," 2020, *arXiv:2005.11901*. [Online]. Available: http://arxiv.org/abs/2005.11901

[23] *Machine Learning for Beginners: An Introduction to Neural Networks*. Accessed: May 2021. [Online]. Available: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9

[24] *Gradient Descent for Machine Learning*. Accessed: Apr. 2021. [Online]. Available: https://machinelearningmastery.com/gradient-descent-for-machine-learning/

[25] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804518301590

[26] L. Kong, X.-Y. Liu, H. Sheng, P. Zeng, and G. Chen, "Federated tensor mining for secure industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 2144–2153, Mar. 2020.

[27] X. Xu, B. Shen, S. Ding, G. Srivastava, M. Bilal, M. R. Khosravi, V. G. Menon, M. A. Jan, and W. Maoli, "Service offloading with deep Q-network for digital twinning empowered internet of vehicles in edge computing," *IEEE Trans. Ind. Informat.*, early access, Nov. 24, 2020, doi: 10.1109/TII.2020.3040180.

[28] *The Shannon-Hartley Theorem*. Accessed: Apr. 2021. [Online]. Available: https://www.ingenu.com/2016/07/back-to-basics-the-shannon-hartley-theorem/

[29] L. Dong, M. N. Satpute, J. Shan, B. Liu, Y. Yu, and T. Yan, "Computation offloading for mobile-edge computing with multi-user," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 841–850.

[30] M. Dhada, A. Parlikad, and A. S. Palau, "Federated learning for collaborative prognosis," in *Proc. Int. Conf. Precis., Meso, Micro, Nano Eng. (COPEN)*, 2019, pp. 1–6.

[31] *Encrypted Training With Pytorch + Pysyft*. Accessed: Mar. 2021. [Online]. Available: https://blog.openmined.org/encrypted-training-on-mnist/

[32] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *Proc. IEEE Int. Conf. Prognostics Health Manage. (ICPHM)*, Jun. 2017, pp. 88–95.

[33] G. S. Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2016, pp. 214–228.

[34] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "Can we use split learning on 1D CNN models for privacy preserving training?" in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 305–318, doi: 10.1145/3320269.3384740.

[35] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," 2014, *arXiv:1402.1128*. [Online]. Available: http://arxiv.org/abs/1402.1128

SOURABH BHARTI (Member, IEEE) received the Ph.D. degree in information technology from the Indian Institute of Information Technology and Management, Gwalior, India. He is currently a Marie Skłodowska-Curie Research Fellow with the CONFIRM Centre for Smart Manufacturing, Nimbus Research Centre, Munster Technological University, Cork, Ireland. He is also an Associate Investigator within the nationally funded CONNECT Centre, Ireland. His research interests include predictive maintenance, wireless networked systems, and the Internet of Things (IoT).

ALAN MCGIBNEY (Member, IEEE) received the Ph.D. degree in electronic engineering from Cork Institute of Technology, Ireland. He is currently a Group Lead for IoT Systems and User Interaction with the Nimbus Research Centre, Munster Technological University, Cork, Ireland. His research interests include the Internet of Things (IoT), cyber physical systems, and distributed ledger technology. He also leads a number of EU and nationally funded projects with a particular focus in the areas of distributed optimization, energy efficiency, distributed software architectures, and IoT platforms. He is also a Funded Investigator within the nationally funded CONFIRM Centre for Smart Manufacturing contributing the hub research in the areas of networked systems and the IoT.

• • •