# A Federated Learning Framework for IoT: Application to Industry 4.0

Hamza SAFRI
*Berger-levrault*
Toulouse, FRANCE
hamza.safri@berger-levrault.com

Mohamed Mehdi KANDI
*Berger-levrault*
Toulouse, FRANCE
mohamed.kandi@berger-levrault.com

Youssef MILOUDI
*CARL - Berger-levrault*
Toulouse, FRANCE
youssef.miloudi@berger-levrault.com

Christophe BORTOLASO
*Berger-levrault*
Toulouse, FRANCE
christophe.bortolaso@berger-levrault.com

Denis TRYSTRAM
*INRIA-LIG*
*University Grenoble Alpes*
Grenoble, FRANCE
denis.trystram@inria.fr

Frédéric DESPREZ
*INRIA*
Grenoble, FRANCE
frederic.desprez@inria.fr

***Abstract*—Predictive maintenance aims to anticipate industrial equipment failures in order to allow early scheduling of corrective actions. Such a maintenance approach is based on a detailed analysis that takes into account the technical and contextual characteristics of the target industrial equipment. However, this analysis requires a significant period of time to collect a representative quantity of data to learn a predictive model. Federated learning (FL in short) is a promising approach that allows several participants to build collaboratively a global predictive model. This approach has been widely explored in generic IoT applications and large scale architectures. However, the implementation of FL in actual environments requires to consider several issues to adapt to existing IoT architectures, including the management/orchestration of the federated tasks and handling the limitations of computational resources. Indeed, most of the current research focus on the aggregation of heavy deep learning algorithms. In this paper, we propose an architecture for FL in the context of IoT based on the classical 3-layer architecture standardized by ETSI [1]. We consider new features for performing federated tasks (training, aggregation and management of each participant). We also propose a stacking-based aggregation method to build the global model in a cost-efficient way. We evaluate finally the performance and effectiveness of this approach on real use-case scenarios. The comparison with other models trained in a centralized way highlights the benefit of our approach.**

***Index Terms*—Predictive maintenance, IoT, edge, Federated Learning, Stacking algorithms.**

## I. Introduction

Industrial companies adopt several maintenance strategies for equipment, depending on their budget, the available human resources, and the level of expertise. If the repair of the equipment is performed after the failure, then the state of the industrial process may degrade or stop for some time. For this reason, companies adopt nowadays maintenance strategies based on equipment metrics, such as remaining useful lifetime expectancy, or analysis on collected operation measurements, such as intensity and temperature, to anticipate failures and plan maintenance actions.

[1]https://www.etsi.org/

The fourth revolution in the industrial sector (Industry 4.0) allows suppliers and manufacturers to introduce new technologies and concepts, mainly the Internet of Things (IoT), Cloud Computing, and data analytics, to check equipment and detect potential defects. This approach is called predictive maintenance (PM). Equipment degradation often occurs gradually between normal and failed states. Predictive maintenance evaluates the real conditions and their evolution to predict when maintenance actions should be taken before anticipating the failure. Predictive maintenance is based on the deployment of sensors, connected to the equipment, that collect a set of measurements. These measurements are then used by machine learning (ML) models (we call model a learning algorithm trained on local equipment data) to predict the future state of the equipment. This approach allows detecting the cause of the potential problem and the time frame in which a failure is likely to occur. In this context, some studies [1]–[3] have shown that companies that have adopted predictive strategies can reduce maintenance costs by 30% and downtime by 70%.

Nevertheless, the design and deployment of predictive maintenance models at an industrial scale still raise many challenges related to the confidence of data, the scaling of these systems in terms of volume, the heterogeneity of the equipment, and the resource limitation. The number of models increases with the number of equipments, which makes their management and deployment very challenging. Given the computing and memory limitations of the IoT nodes, Federated Learning (FL) is a promising approach for IoT applications generally and predictive maintenance more specifically. This approach privileges local training of models to limit data transfer and encourages intelligent edge in the IoT context through the design and implementation of training strategies adapted for both edge and federated learning.

Federated learning was first proposed by Google in 2016 to predict user text input across tens of thousands of Android devices while retaining the device data [4]. In this approach, a central server distributes at time step $i$ a global model $M_i$ to a set of participants. Each participant trains a model based on its own locally available data. This local models are then sent back to the central server, where they are averaged to produce a new global model $M_{i+1}$. This updated global model now acts as the main model and is again distributed to the participants. The process is repeated until the model achieves satisfying performance that may depend on the target.

The FL approach has been extensively used and evaluated in many contexts on distributed systems. It has shown good global model performance on generic edge to cloud architectures [5]–[8]. However, it faces several issues for extreme edge and IoT architectures related to implementation in actual environments, mainly due to the adaptation of FL to common industrial IoT architectures. These architectures have an important role in the large-scale deployment of FL/IoT ecosystems due to the dependency on several important services such as edge/cloud server and edge-IoT communication protocols. Another problem in this ecosystem is the aggregation of local models. Most of the scientific proposals focus on heavy deep learning models [11]–[14]. These models require very high resources (GPU, memory), which is not always available in IoT environments. For this reason, traditional machine learning models are more suitable for these environments. However, there are only few proposals for such models aggregation.

Our paper focuses on the problem of deployment federated learning in IoT environments. The main contributions as follows.

- We propose first a generic architecture that integrates components able to manage participants and orchestrate federated tasks such as training, aggregation and evaluation of the global model.
- Then, we deal with the problem of aggregation of traditional machine learning models. We propose an aggregation method based on stacking which consists in training meta models to combine the predictions of a set of basic models.
- We implemented the architecture and the aggregation method, and we evaluated the performance and effectiveness of this approach on actual use-case data. We compare our proposals to a method in which the models are trained in a centralized way.

The remaining of the paper is organized as follows. Section II presents the existing work. The proposed architecture and its components are described in Section III. In Section IV, we present our aggregation method. The implementation of the architecture components and communication connectors is discussed in section V. Section VI describes the use case, the performance evaluation methodology, and the experimental results. Finally, we conclude in Section VII.

## II. Related work

The existing work on predictive maintenance can be classified based on some constraints, such as data transfer cost, privacy, and limitations on available resources. To consider these constraints, we propose a classification of existing work based on an architectural basis. We distinguish three types of architectures : (1) equipment-cloud architecture [5], [7], where the collected data is transferred to a centralized server, processing, and learning is done at the cloud level, (2) an edge-cloud architecture [6], [7], where data processing is performed at the edge level before the data transfer to the cloud, (3) the equipment-edge architecture [8], where the processing and learning are performed at the edge level.

Quiroz et al. [5] proposed an approach to diagnosing the failure of a rotor bar in an online-starting permanent magnet synchronous motor (LS-PMSM) using a random forest model. This work follows an equipment-cloud architecture where the data is transferred to a central server before the processing, which leads to high latency. Therefore, other works have been proposed to overcome these limitations by applying learning models at the equipment-edge level. Authors of [6] proposed to deploy a light machine learning (ML) algorithm (Light Gradient Boosting Machine- GBM) on the edge nodes and deep learning (DL) algorithm on the master nodes (edge routers). One of the advantages of the proposed approach is that the raw data is not passed to the master nodes. Instead, LightGBM learns features from the raw data and passes the learned features to the master node which further increases the accuracy with more computations (using DL). The authors assume that an edge router is powerful enough to deploy DL models.

To reduce the amount of data transmitted from the sensor nodes to the edge server, authors proposed in [7] to deploy the ML directly on the sensor node. The basic idea is to train the ML model on the computationally powerful device (in the experiments, they used edge, but argued that the cloud can also be used) and push the model to a sensor node. When a new measurement is collected, it is passed to the ML model which predicts its label.

To address the computational capacity issue, the edge-cloud architecture, has been adopted in several works. In [8], a framework called SERENA is proposed for preventive maintenance using a hybrid cloud-edge computing approach. Sensor nodes send data to an edge gateway where statistical features of the raw data, such as average, maximum, and minimum, are calculated. These characteristics are used to train a model at the cloud level before pushing it to the Edge. The model predicts new incoming data, which helps detect abnormal data and therefore potential failures.

The works we have cited above present applications of machine learning for predictive maintenance, where data is already collected. However, data collection is one of the main challenges for predictive maintenance, because we need to collect metrics on the equipment during the degradation process to have an overview of the potential failures. Data gathering has two weaknesses: on the one hand, it requires a very long period while on the other hand, the data represents the behavior of a single device. This behavior depends on technical and context aspects. In this context, a collaborative approach between equipment based on experience sharing instead of data sharing allows overcoming the data transfer cost and privacy constraints. In this perspective, federated learning is a promising approach for predictive maintenance.

Federated learning has rapidly developed in the scientific literature to address the limitations of data transfer in the network and privacy. We find several architectures [9], [10] that mainly depend on the data distribution: vertical or horizontal.

Fedavg [11] is the first aggregation algorithm proposed by google to train DL algorithms collaboratively, subsequently, several algorithms have been proposed for the same purpose to improve problems such as synchronization in the training process [12] or the management of no Independent and Identically Distributed (IID) data [13]–[15] or minimizing the number of training rounds [16].

In the literature, we have fewer works adapted to traditional ML algorithms. In this context, some approaches are proposed in the framework of vertical federated learning where an intermediate component is needed to encrypt and decrypt the data and information exchanged by participants.

The authors [17] have proposed a random forest aggregation approach. In the same vein, [17] proposed a collaborative training approach for regression logistics. However, we note that there are few works on horizontal federated learning for these algorithms.

While we have reviewed some FL-based contributions, it is interesting to note that, to our knowledge, FL has not been studied in depth for predictive maintenance in IoT. There are some exceptions. For example in [18] authors compared two algorithms: FL and no-FL. They showed that the FL algorithm can preserve data locally while achieving similar performance to the traditional no-FL approach when an ML model is trained centrally.This paper is only a validation of the applicability of federated learning for predictive maintenance because it did not take into consideration the issues of resources limitation and global model deployment and training orchestration.

These contributions have projected the federated learning process onto general architectures such as Cloud-Edge, Cloud-Fog [19]. This led us to put more effort into improving the FL architecture for a better fit to IoT environments based on standardized architectures. This projection and adaptation of FL to IoT must consider the different characteristics and constraints related to IoT, mainly the computational capacity, the amount of data exchanged, the management, and the orchestration of distributed training of participants.

## III. Proposed Architecture

### A. Existing IoT architectures

In this paper, we consider a fundamental architecture that addresses the basic idea of IoT. It is a three-layer architecture proposed in the early stages of IoT [20].The first layer of this architecture named *Devices layer* is in charge of identifying connected objects and collecting data like location, intensity, motion, vibration, etc. The second one is the $Network\ layer$, whose role is to transfer the collected data. This data is used by IoT applications deployed at the application layer. This architecture can be extended to a four–layer [21] or five-layer [23]architectures to consider other important aspects such as security, data processing and business model.

### B. Federated learning architecture for IoT

Federated learning is a paradigm that handles the problem of data transfer limitation in the network. Following this idea, we propose an architecture based on the three-layer architecture presented in the section III-A. This architecture contains components that manage the participants (i.e. devices, equipments, etc.), orchestrate the training, monitor the performance of the models and the hosts resources.

*1) Architecture components:* FL is based on a local training as close as possible to the devices and then an aggregation in the server. To achieve this goal, we deploy a training orchestrator for the basic models at both the device level and at the application layer for the aggregation. We propose the learning architecture presented in Fig 1 and 2. The architecture is composed of the three following sub-layers:

- **Data Generators (DG)** : It corresponds to a sub-layer of the *devices layer*, which contains the connected equipments that collect the data.
- **Local Orchestrator (LO)**: is the second sub-layer of the *devices layer* that contains the local orchestrators (LO) deployed on the IoT nodes (IoT server or gateway)
- **Global Orchestrator (GO)**: is the main component of the architecture. On the first hand, it manages the configuration and the dynamic deployment of the local orchestrators. On the other hand, it synchronizes the training and the aggregation of the outputs of each local orchestrator.

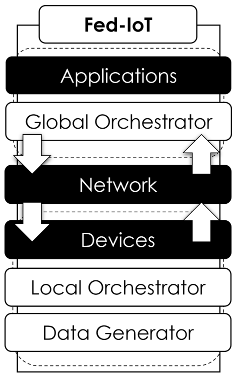

Fig. 1: layered Federated IoT architecture.

The architecture is composed of two networks as illustrated in Fig. 2, a sensor network links the data generating sub-layer and the local orchestrator sub-layer. Its role is the local transfer of data. A second unmeasurable network (WIFI, etc) links the local and global orchestrators. This network allows sending of the necessary parameters for the local training process and to distribute the generalized models to the different local orchestrators. In the following, we present in more detail the global orchestrators and the local orchestrators components.

*2) Global Orchestrator (GO):* The global orchestrator is the cornerstone of architecture. This component provides several functionalities mainly:

- Aggregation management according to the configured method.

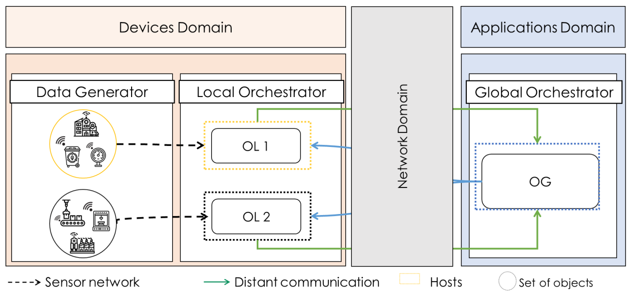

Fig. 2: Overview of interaction between different components of proposed architecture

- Training orchestration of global models.
- On-demand deployment of local orchestrators at the IoT node .
- Monitoring of execution environments and model performance.

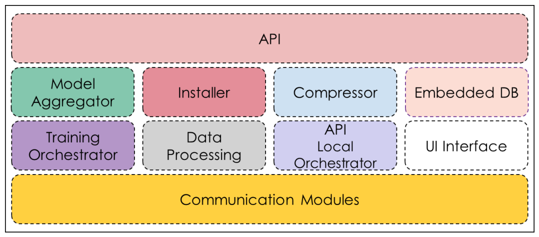

Fig. 3: Global Orchestrator components

These functionalities are provided by several blocks as shown in Fig 3, mainly:

- Aggregation models: in this block we can implement an aggregation logic that will be used by GO to generalize the different local models.
- Training Orchestrator: this block is responsible for triggering training in LOs and ensuring the synchronization of the participants' responses.
- Local Orchestrator API: this block is deployed at the IoT nodes to provide local training management and orchestration.
- UI interface: a management and monitoring interface, which allows to view the status of LOs and information about data generators, configure LOs, call local models for predictions, and display model performance according to LOs and DGs.
- Installer: is the component responsible for the deployment of LOs at the IoT nodes starting from the installation of the dependencies to the component startup.
- Compressor: is a dedicated block for the compression of files and voluminous data.
- Communication module: a software connector that initiates communication between the global orchestrator and the IoT node.

*3) Local Orchestrator (LO):* The local orchestrator is one of the components deployed by the global orchestrator at the IoT node. This component provides:

- The data preparation of each data generator(participant).
- Local training orchestration of the data generators participating in the global model training.
- Monitoring of hosts.

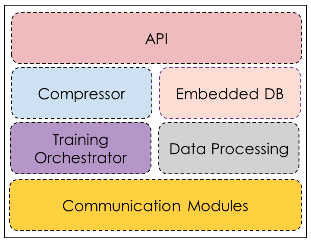

Fig. 4: Local Orchestrator components

*4) Communication Protocol:* Figure 5 details the different interactions between the components. The participants are IoT nodes that manage the data of several connected objects. These participants must initiate a communication with the global orchestrator to announce that the node is ready to execute a federated task (1). In this perspective, the global orchestrator ensures the deployment of a local orchestrator at node IoT to guarantee an orchestration of the federated tasks, mainly the model training. This is the LO Configuration step (2). It is important to note that each GO manages several LOs. Once the LOs are well configured, a second registration is necessary but this time the DGs concerned by the federated tasks inform the LOs about the location of the collected data. This operation can also be performed via a REST client (4).This registration can be rejected if it is not correctly described. Each local orchestrator manages several DGs, which means that the LOs orchestrate the training of the local models of each DG. The two previous steps represent the backbone of our architecture. The deployment of these components allows us to launch a training based on the coordination between LOs and GOs which mainly use the API to exchange the necessary configurations for models training and aggregation (5). These configurations are implemented at the internal component *Training Orchestrator* for the training and the validation and the component *Data Processing* for data preparation of each DG. At the end, the IoT nodes send the output of the models to the server to aggregate (6) and integrate the aggregated model (7). In this context, we propose in this paper section IV an aggregation method based on the stacking.

## IV. Federated stacking: FedStack

### A. Stacking

Stacking or stacked generalization is an ensemble machine learning algorithm [26]. It uses a meta-learning algorithm to learn the best combination of the predictions of two or more basic machine learning models. As shown in Fig 6, stacking leverages the capacity of a set of models to make predictions that outperform any model in the set. The overall model produced after training consists of two main levels:

- Base models: in this level the models use the same data for training to generate a set of predictions.

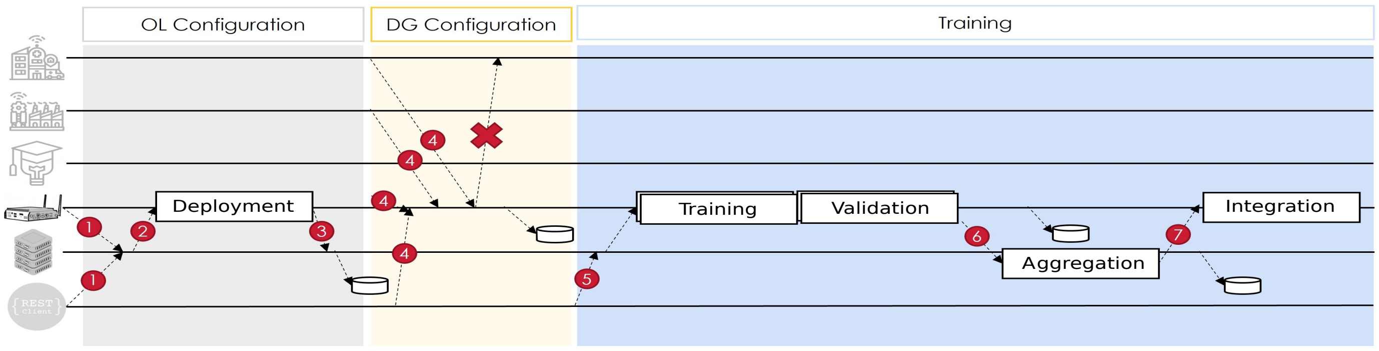

Fig. 5: Communication protocol : Description of different phases of architecture components configuration and training orchestration

- Meta-model(s): the meta-model aggregates the predictions of the base models to produce a final prediction. This meta-model can be constituted of several levels with several models except the last level where there is a single model

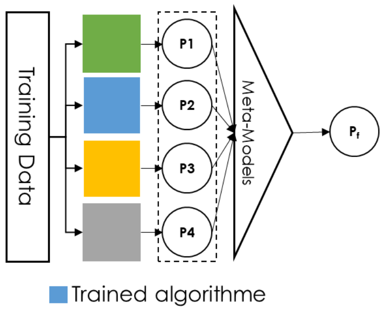

Fig. 6: Stacking: building a meta-model from several base models.

TABLE I: Notation

| ***Notation*** | ***Description*** |
|---|---|
| $l$ | model levels (for $1 \leq l$) |
| $K_l$ | set of models at level $l$ |
| $P$ | cardinality of a set of participants |
| $N_p$ | cardinality of a set of instance of participant $p$ |
| $N_p^{(val)}$ | cardinality of validation set of participant $p$ |
| $N_l^{(val)}$ | cardinality of validation set of level $l$ |
| $D_p$ | data related to participant $p$ |
| $D_{l,p}$ | data related to participant $p$ at level $l$ |
| $D_p^{(train)}$ | training set of participant $p$ |
| $D_p^{(val)}$ | validation set of participant $p$ |
| $D_p^{(test)}$ | testing set of participant $p$ |
| $D_l$ | stacked dataset of level $l$ |
| $M_{k,p}$ | k-th model of participant $p$ |

## *B. Federated stacking (Fedstack)*

This algorithm is an adaptation of the classical stacking presented in [26] for federated learning. As presented in Fig 7, FedStack consists of training locally machine learning or deep learning models adapted to different participants (Fig 7a). This approach allows each participant to train models that perform well on their own data and models that are adapted to other participants. Once the local models are trained, the predictions will be grouped by local models before being sent to the server for aggregation (Fig 7b). Aggregation is achieved by training a schema of models from the predictions of all participants' base models. Once the models are well trained, they will be used locally by each participant in conjunction with their local models to minimize the influence of data heterogeneity on model performance (Fig 7d).

Given $P$ participants. Let $D_p$ be the set of data of participant $p$ with $C$ possible classes:

$$D_p = \{(x_{n,p}, y_{n,p}), n = 1..N_p\}, p = 1, ..., P \qquad (1)$$

Where $y_{n,p}$ is the class value and $x_{n,p}$ is a vector representing the attribute values of the $p$-th participant. We split the set $D_p$ into $D_p^{(train)}$, $D_p^{(val)}$, $D_p^{(test)}$ respectively training and validation and test set of participant $p$.

We have $K_1$ local learning algorithms in each participant (where index 1 indicates level 1).

The execution of the $k$-th algorithm on training set $D_p^{(train)}$ induces a model $M_{k,p}$ (for $k = 1, ..., K_1$) in order to create the *level-1 (local) models* of the $p$-th participant. According to the output of the first level, we distinguish two cases:

- The output of the level 1 models is a single class among the C classes
- The output of the level 1 models is the probability of each class

*1) The prediction of single class:* For each instance $(x_{n,p}, y_{n,p})$ in $D_p^{(val)}$. Let $v_{k,p}$ denote the prediction function of the model $M_{k,p}$. Let :

$$z_{k,n,p} = v_{k,p}(x_{n,p}) \qquad (2)$$

At the end of the validation process of each participant, the data set assembled from the outputs of the local $K_1$ models

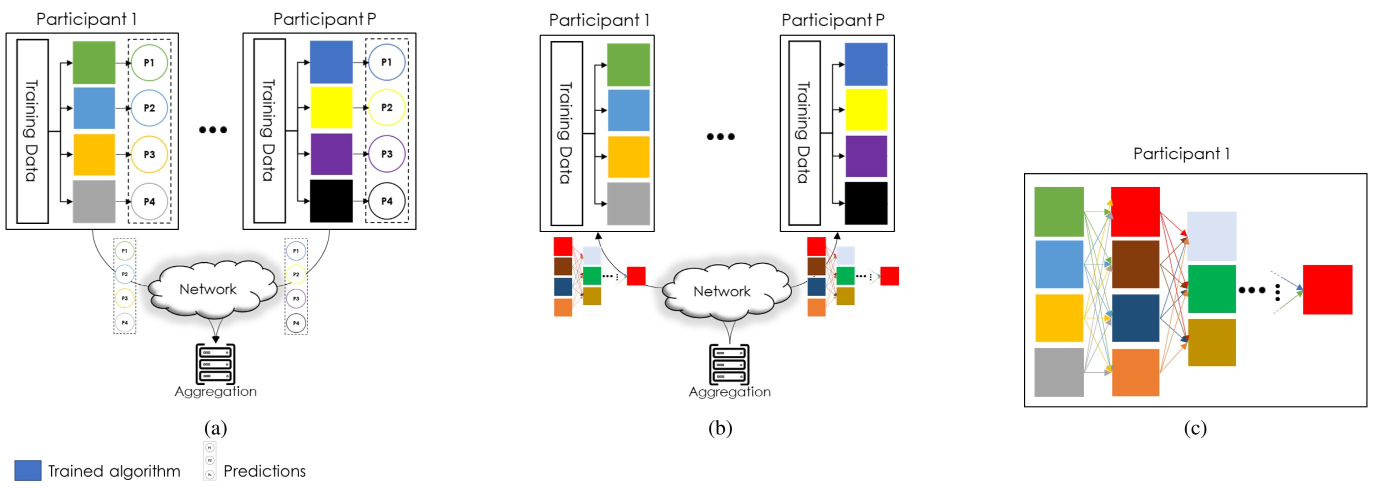

Fig. 7: The FedStack training process, Fig (a) Sending the predictions of the local models, Fig (b) the predictions aggregation and the sending back the meta-models, Fig (c) Integration of the meta-models

of the participant $p$ is:

$$D_{1,p} = \{((z_{1,n,p}, z_{2,n,p}, ...z_{k,n,p}, ..., z_{K_1,n,p}), y_{n,p}) \\ n = 1, ..., N_p^{(val)}, p = 1, ..., P\} \quad (3)$$

The predictions on each participant's validation set are sent to the server to be assembled in order to build a stacked dataset that can be defined as follows:

$$D_1 = \cup_{p=1}^{P} D_{1,p} \quad (4)$$

$D_1$ is the data used by the server to train the first level of the meta-model.

Let $L$ be the number of the meta-model levels. $D_l$ is the dataset used to train and validate the $K_l$ learning algorithms of level $l$. Let $V_{k,l}$ denote the prediction function of the model $M_{k,l}$, for $k = 1, ..., K_l$.

$$\begin{cases} Z_{2,k,n} = V_{k,2}((z_{p,1,n}, z_{p,2,n}, ..., z_{p,K_1,n})) \\ \text{p=,...,P, n=1,...,N}_p^{(val)} \\ Z_{l,k,n} = V_{k,l}((Z_{l-1,n,1}, Z_{l-1,n,2}, ..., Z_{l-1,n,K_{l-2}})) \\ l = 3, ...L, n = 1, ..., N_{l-1}^{(val)} \end{cases} \quad (5)$$

And the stacked data for the next level can be written as following:

$$D_{l+1} = \{((Z_{l,n,1}, Z_{l,n,2}, ..., Z_{l,n,K_{l-1}}), y_n), n = 1, ..., N_l^{(val)}\} \quad (6)$$

At the end of the training, the models of the different levels will be locally evaluated by each participant using the $D_p^{(test)}$ set in conjunction with the local models.

Given a new instance $x_m$ coming for the participant p ($x_{m,p}$). The final prediction $\hat{Y}_m$ for this instance can be represented as follows

$$\hat{Y}_m = Z_{L,1,m} \quad (7)$$

*2) Probability of class:* Let considerate the case where the output from the level-0 models is C class probabilities rather than a single class prediction. Let $q_{k,c,p}$(x) denote the probability of the $c$-th class from the local model $k$ of participant $p$ :

$$z_{k,c,n,p} = q_{k,c,p}(x_{n,p}) \quad (8)$$

the $\mathcal{P}_{k,n,p}$ gives the model's class probabilities for the n-th instance from the local model $k$ of participant $p$.

$$\mathcal{P}_{k,n,p} = (z_{k,1,n,p}, z_{k,2,n,p}, ..., z_{k,C,n,p}) \quad (9)$$

At the end of the prediction of each participant, the dataset assembled from the outputs of the local $K_1$ models is :

$$D_{1,p} = \{((z_{1,1,n,p}, z_{1,2,n,p}, ..., z_{1,C,n,p}, ..., z_{k,1,n,p}, z_{k,2,n,p} \\ , ..., z_{k,C,n,p}, ..., z_{K_1,1,n,p}, z_{K_1,2,n,p}, ..., z_{K_1,C,n,p}), y_{n,p}) \\ n = 1, ..., N_p^{(val)}\} \quad (10)$$

The predictions of all participants are assembled together to build a stacked dataset as in expression (4) and the prediction function is defined as (5). The stacked data for the next level can be written as following:

$$D_{l+1} = \{((z_{l,1,1,n}, z_{l,1,2,n}, ..., z_{l,1,C,n}, ..., z_{l,k,1,n}, z_{l,k,2,n}, ..., \\ z_{l,k,C,n}, ..., z_{l,K,1,n}, z_{l,K,2,n}, ..., z_{l,K,C,n}), y_{n,p}) \\ , n = 1, ..., N_l^{(val)}\} \quad (11)$$

## C. Algorithm

The FedStack algorithm is shown below in algorithm 1. It starts by initiating at the server the number of participants for training and the number of stacking levels according to the use case (lines 2 and 3). The server transfers a configuration to the participants. The configuration describes the local model to be trained and the model parameters (lines 4,5,6). Once the participants receive the configuration, they train the base level models using the local data, then they perform the predictions of the validation set (lines 15 to 24). Each client then sends its predictions to the server (line 25), which concatenates them to train the level 1 models of global model(lines 97). For $levels > 1$, a concatenation of the previous level predictions is performed (lines 8 to 13). Once the models of $levels > 1$ are trained, they will be sent to the participant to be included in the prediction pipeline with the local models.

**Algorithm 1:** FedStack

```
1  Server execute :
2  $S_t \leftarrow$ set of m registred edge
3  $levels \leftarrow$ set of levels
4  for each client $k \in S_t$ in parallel do
5      $predictions_k \leftarrow ClientTrainingl(k, level_0\_Models)$
6  end
7  $predictions = \bigcup_{k=1}^{m} predictions_k$
8  for each level $l \geq 1 \in levels$ do
9      $M_l \leftarrow$ cardinality of level l models
10     for each model $j \in$ level $l$ do
11         $predictions_{lj} \leftarrow$
             $TrainModel(Model\_params, predictions)$
12     end
13     $predictions = \bigcup_{k=1}^{M_l} predictions_{lk}$
14 end

15 Client execute :
16 ClientTraining($k$,$level_0\_Models$) :
17     $local\_participants \leftarrow$
         set of n registred participants in edge
18     for each participant $p \in local\_participants$ do
19         $D_p \leftarrow$ data of the participant p
20         $M_p \leftarrow$ cardinality of local models
21         for each model $j \in level_0\_Models$ do
22             $predictions_{pj} \leftarrow$
                 $TrainModel(Model\_params, D_p)$
23         end
24         $predictions_p = \bigcup_{s=1}^{M_p} predictions_{ps}$
25     end
26 return $\bigcup_{p=1}^{n} predictions_p$
27
```

## V. Proof of Concept

The aggregation method represents a dynamic element for each component of the architecture described in Section III-B. We implement FedStack as an aggregation algorithm. This choice allows us to define the necessary configurations to exchange between the different components of the architecture.

*1) Communication interfaces:* We define two interaction interfaces between GO and LO:

- **SSH connector**: in the implementation, we use the SSH connector to initiate the communication between GO and the host of LO.
- **RestAPI**: this interface allows GO to manage and orchestrate the LOs.

*2) Configuration files:* To ensure a dynamic and on-demand configuration and deployment of local orchestrators and the triggering of training or metrics recovery, the global orchestrator uses a set of configuration templates:

- **Deploying a LO**: To deploy a LO, the GO receives a JSON configuration file containing information about the host machine (Ip, username, ssh key or password, and ssh port) and information about the application deployment, such as the port and a unique name for management. This operation allows the GO to connect to the host, install the dependencies and start the application.
- **Adding a DG**: each DG must be registered at LO by sending a configuration file that contains the data location and unique identifier.
- **Triggering a training**: this operation is based on a configuration file sent to the LO via API. This file describes first the dependencies to instantiate the models and then the set of models used by level, second the parameters of each model, and finally the percentage of data used for the test. Once the file has been sent, the GO orchestrates the training between the different LO. The LO, on the other hand, orchestrate the local training of the different learning algorithms using the data of each DG. We use a sequential training to minimize the consumption of resources. Once the model is well trained, we can access the evaluation metrics via the API of each LO.

## VI. Experiments

### A. Use case

Airport baggage handling systems are one of the activities where an unforeseen production failure can have serious business consequences: aircraft delays, baggage reshipment, and compensation. These systems are complex, automated, often made up of dozens of different assets, each with its own specificities. They must be able to handle irregular flows of baggages of high disparity, over relatively long distances and in a very short time span. If failures are rather rare because of drastic preventive maintenance plans, they are nonetheless constrained by major hazards, unpredictable behaviors and the slightest failure of a single component can eventually lead to a chaotic situation. Maintaining a system in good working condition at the lowest cost is, therefore, a major challenge for suppliers of these systems and their customers. For this purpose, companies are moving towards a more proactive maintenance strategy, based on health monitoring of their equipment and more intelligent monitoring and decision support systems. In this context, we propose to apply the FedStack algorithm to data from a set of sensors deployed in conveyors located at airports.The main objective of the model is to identify periods when conveyors are operating loaded (class 1) and unloaded (class 2). The participating conveyors can be grouped according to their characteristics (length, speed, intensity...) and usage, as shown in the table II,which leads to both data and behavior heterogeneity.

### B. Data collection

The data comes from several automated handling systems (baggage sorting, postal sorting...). Several baggage conveyors (straight and curved), each with six measured variables (electric motor current, electric motor temperature, oil-gearbox

TABLE II: Conveyor characteristics

| Groups | Description |
|---|---|
| G1 | Medium conveyor (4.8m) with slope (about 3°) then short conveyor (1.4m) 90° injector.Speed about 1m/s, Power: 1,1 kW and 0,55 kW |
| G2 | Long conveyor (12.4m) with a declining slope (-11,2°). Speed 0.75m/s, Power : 2,2 kW |
| G3 | Short conveyor (2m) linked to 2 long ones (14m), the first one with a small positive slope. Speed about 1m/s, Power : 0,75 kW and 2,2 kW |
| G4 | Long conveyors (respective sizes: 14.6m - 8.8m - 9m) and slopes (+2.5° then -1.46° then 0.63°). Speeds about 0.5 m/s then 1m/s, Power: 1.1 kW and 0.55 kW |

temperature, linear speed of the conveyor belt, belt centering), are instrumented with objects communicating on an LPWan network. The measurements are sampled every second and transmitted in compressed frames periodically (1 to 4) minutes. The data is transmitted and processed in real-time. The data is a history of about two years of laboratory experiments and 12 months of operation on conveyors in an airport(see III). These measurements will be adapted to supervised learning models by calculating the intensity gradients before creating a sliding window to reconstruct the data for the learning models based on the conveying time. This time differs according to the type of conveyors.

TABLE III: Collected Data

| Data | Date-Time | Speed | Intensity | Baggage | Marche | Onload |
|---|---|---|---|---|---|---|
| De-scrip-tion | The date and time of the measure-ment | Speed mea-surements at each time point (each second). | Intensity measure-ments at each time point (each second) | 1 if there is a baggage that passes at time t, 0 otherwise. | 1 if the conveyor is running, 0 otherwise | 1 if the conveyor is loaded, 0 other-wise |
| Type | Datetime | float | float | int | int | int |

## C. Setup

To validate our Framework, we set up the architecture described in Fig 8 where we have deployed a global orchestrator at the server. From this, we deploy local orchestrators in three hosts, namely a Raspberry Pi and two Docker containers which allow us to study the deployment and the functioning of the solution in physical and virtualized environments. These hosts are connected in different networks. Each LO orchestrates the local learning for a set of conveyors. The characteristics of these hosts are described in table IV. This setup will allow us to validate :

- The ability of LOs deployed in an IoT node (in this case the Raspberry Pi or container docker) with limited resources to train models of each registered participant.
- The efficiency of the Fedstack aggregation approach with heterogeneous participants.

## D. analysis of experiments

In this section, we compare three FedStack configurations with a local approach in which each conveyor trains a random forest with default parameters. We also compare these approaches with a data generalization approach where we combine the collected data before training a single model of the same type as the local random forest models. FedStack is a configurable algorithm that depends mainly on the used data . In our work, we choose the configurations described in Table V for FedStack with default parameters.

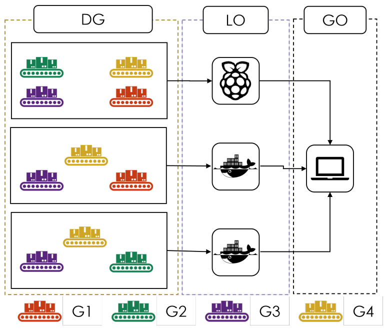

Fig. 8: Experimental setup with 4 groups of conveyors (represented by different colors according to table III), the OLs are deplyed on a Rasberry Pi and 2 Docker containers, and the OG is hosted by a server

TABLE IV: Hosts Configurations

| hosts | Configuration |
|---|---|
| Server | CPU: 1.90GHz  2.11 GHz |
|  | RAM: 16,0 Go |
| Raspberry Pi | CPU: 0,6 GHz – 1,2 Ghz |
|  | RAM: 1,0 Go |
| Container | CPU: 1.90 GHz – 2.11 GHz |
|  | RAM: 1,0 Go |

TABLE V: Fedstack configurations

| Configuration | level 1 models (local models) | level 2 models | level 3 models |
|---|---|---|---|
| 1 | Random Forest (RF) | Logistic Regression (LR) | ▨ |
|  | XGBoost (XG) |  |  |
| 2 | Random Forest (RF) | Naive Bayes (NB) | ▨ |
|  | XGBoost (XG) |  |  |
| 3 | Random Forest (RF) | Naive Bayes (NB) | Naive Bayes (NB) |
|  | XGBoost (XG) | Logistic Regression (LR) |  |

We evaluate the approaches using two metrics:

$$recall = \frac{TP}{TP + FN}$$

$$Overestimation = \frac{(TP + FP)}{(TP + FN)} - 1$$

where: TP: True positives FP: False positives FN: False negatives

The first metric is recall, in this context, is used to identify the periods predicted by the model as a load period. The second metric is used to measure the overestimation of load periods by the model. These two ratios are used together to evaluate the performance of trained models. The more the recall ratio is close to 100% and the overestimation tends towards 0, the better the model.

The figures 9 and 10 represent the performance of the models for each conveyor. The x axis represents the different conveyors used in learning process. While the y axis represents the evaluation metric. We notice that the models are very close regarding the recall, but we note a large variation in overestimation. We observe that the combined model obtained a recall higher than 75% in 60% of the conveyors but it overestimated the load periods which means that the model has many false positives. These results are due to the heterogeneity of the conveyors data caused by the characteristics of the conveyors of each group such as speed and length and also the difference in the characteristics of the conveyors of the same group such as the conveying time.The Fedstack configurations performed well on the different conveyors. The Fedstack model with a LR metamodel obtained results very close to those of the local models. However, a Fedstack configuration based on NB minimized the overestimation which tends to 0 while the recall exceeds 70% in all conveyors. Moreover this configuration has improved remarkably the performance of the G3-1 conveyor where we have increased the recall from 62% to 70% and with a small overestimation of 0.07 which is acceptable. The Fedstack 1 and 2 configurations have the same performance which shows that the NB algorithm in this use case was able to generalize the predictions perfectly. Contrary to the data combination, the Fedstack was less influenced by the heterogeneity of the data. This is due to the fact that the local models are used and adapted to the behavior of the participants' data. In addition, the Meta-models generalize the predictions (which represent a participant's experience) of the local models instead of the local models parameters which minimizes both the use and the influence of the raw data.

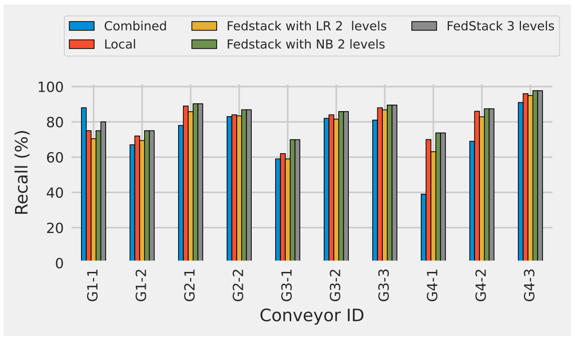

Fig. 9: Recall:ration of the predicted load periods by the final models of each conveyor

The figures 12 and 11 represent the CPU and RAM consumption of the local orchestrator on the Raspberry Pi. The figures describe two phases of resource consumption, the first (red phases) before training is triggered where the figure shows that the baseline consumption of a local orchestrator does not exceed 10% of the CPU and 25% of the RAM required for API. Once the training is triggered (green phases), we

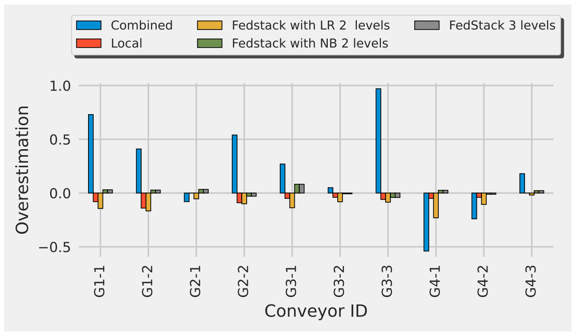

Fig. 10: Overestimation : The overestimated load periods by the final models of each conveyor

see that the orchestrator used 400% of the CPU, i.e., all 4 cores (0.6-1.2 GHz) and the RAM consumption has not exceeded 80% of 1 GB total. This consumption includes the installation of dependencies such as the libraries needed to run the models and local API server. In the second training phase the RAM consumption does not exceed 60% because of the dependencies pre-installed in the first training. This consumption corresponds to the training of two models (RF,XG) per conveyor which means 8 models in each step. The use of a Raspberry pi with very limited resources to perform the training orchestration task demonstrates the applicability of the approach in resource-limited environments such as IoT Gateways

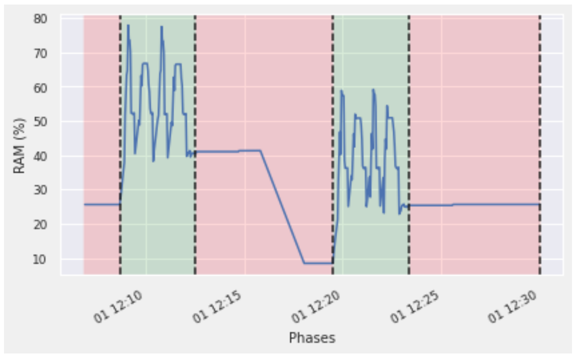

Fig. 11: Ram consumption in two phases : the first one before training (red phase) and the second one during training (green phase)

## VII. Conclusion and future work

In this paper, we presented a federated learning framework for IoT environments. For this purpose, we proposed an extension of the basic IoT architecture with two levels of orchestration. The first level is an orchestration for the management of participants and the local models training. The second level is an orchestration for a global management of the training and the aggregation of local ML models. We also presented an aggregation method based on stacking and we showed that it outperforms the traditional centralized method. Moreover, we noticed that under very limited resources, we

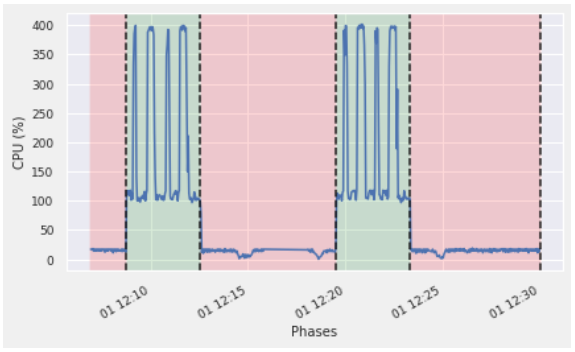

Fig. 12: CPU consumption in two phases : the first one before training (red phase) and the second one during training (green phase)

are able to train multiple ensemble models using a sequential training.

Despite the positive results presented in this paper, the proposed architecture does not completely cover all the challenges of federated learning and IoT ecosystems. We give below some possible further extensions:

- Participant selection methods to minimize training time and resource consumption. This selection must take into account the technical diversity of the devices (intensity, length, etc.), the contextual diversity (different airports) and the statistical heterogeneity of the data.
- Deployment and configuration approaches based on the "publish-subscribe" pattern. This pattern allows, on the first hand, a spatial decoupling where subscribers are not required to establish a direct connection with publishers, nor to modify the behavior of the publisher at the time of subscription. On the other hand, a temporal decoupling, subscribers and publishers should not be online at the same time. This aspect could minimize the number of messages sent between the components of the architecture and the complexity of managing participants.

## Acknowledgment

This work was supported by the research program on Edge Intelligence of the Multi-disciplinary Institute on Artificial Intelligence MIAI at Grenoble Alpes (ANR-19-P3IA-0003)

## References

[1] R. Berger "Predictive maintenance- Is the timing right for predictive maintenance in the manufacturing sector?," Rol. Berger, no. november, 2015. Available: https://www.rolandberger.com//publications//publication_pdf/roland_berger_predictive_maintenance_20141215.pdf.
[2] Deloitte, "Advancing asset management," 2021. Available: www.deloitte.com/us/about.
[3] G. Sullivan, R. Pugh, A. P. Melendez, and W. Hunt, Operations maintenance best practices-a guide to achieving operational efficiency (release 3) , Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2010.
[4] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, 'Learning Differentially Private Recurrent Language Models', arXiv:1710.06963 [cs], Feb. 2018. Available: http://arxiv.org/abs/1710.06963.
[5] Quiroz, Juan C., Norman Bin Mariun, Mohammad Rezazadeh Mehrjou, Mahdi Izadi, Norhisam Bin Misron and Mohd Amran Mohd Radzi. "Fault Detection of Broken Rotor Bar in LS-PMSM Using Random Forests." ArXiv abs/1711.02510 (2017)
[6] H. Yao, P. Gao, P. Zhang, J. Wang, C. Jiang and L. Lu, "Hybrid Intrusion Detection System for Edge-Based IIoT Relying on Machine-Learning-Aided Detection," in IEEE Network, vol. 33, no. 5, pp. 75-81, Sept.-Oct. 2019, doi: 10.1109/MNET.001.1800479.
[7] A. P. Singh and S. Chaudhari, 'Embedded machine learning-based data reduction in application-specific constrained IoT networks', in Proceedings of the 35th Annual ACM Symposium on Applied Computing, New York, NY, USA, Mar. 2020, pp. 747–753. doi: 10.1145/3341105.3373967.
[8] D. Bowden, A. Marguglio, L. Morabito, and et al, 'A cloud-to-edge architecture for predictive analytics', in Workshops of the EDBT/ICDT 2019 Joint Conference. Proceedings. Online resource, 2019, p. 7.
[9] M. Aledhari, R. Razzak, R. M. Parizi and F. Saeed, "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications," in IEEE Access, vol. 8, pp. 140699-140725, 2020, doi: 10.1109/ACCESS.2020.3013541.
[10] Qiang Yang; Yang Liu; Yong Cheng; Yan Kang; Tianjian Chen; Han Yu, Federated Learning , Morgan Claypool, 2019.
[11] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, 'Communication-Efficient Learning of Deep Networks from Decentralized Data', arXiv:1602.05629 [cs], Feb. 2017. Available: http://arxiv.org/abs/1602.05629
[12] Y. Wang, 'CO-OP: Cooperative Machine Learning from Mobile Devices', ERA, Fall 2017. https://era.library.ualberta.ca/items/7d680f04-7987-45c5-b9cd-4fe43c87606f
[13] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," arXiv. 2018.
[14] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, 'Hybrid-FL for Wireless Networks: Cooperative Learning Mechanism Using Non-IID Data', arXiv:1905.07210 [cs, stat], Mar. 2020. Available: http://arxiv.org/abs/1905.07210
[15] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, 'Client-Edge-Cloud Hierarchical Federated Learning', arXiv:1905.06641 [cs], Oct. 2019. Available: http://arxiv.org/abs/1905.06641
[16] N. Guha, A. Talwalkar, and V. Smith, 'One-Shot Federated Learning', arXiv:1902.11175 [cs, stat], Mar. 2019. Available: http://arxiv.org/abs/1902.11175
[17] Y. Liu, Y. Liu, Z. Liu, J. Zhang, C. Meng, and Y. Zheng, 'Federated Forest', IEEE Trans. Big Data, pp. 1–1, 2020, doi: 10.1109/TBDATA.2020.2992755.
[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, 'Federated Multi-Task Learning', arXiv:1705.10467 [cs, stat], Feb. 2018. Available: http://arxiv.org/abs/1705.10467
[19] Q. Wu, K. He and X. Chen, "Personalized Federated Learning for Intelligent IoT Applications: A Cloud-Edge Based Framework" in IEEE Open Journal of the Computer Society, vol. 1, pp. 35-44, 2020, doi: 10.1109/OJCS.2020.2993259.
[20] O. Said and M. Masud, 'Towards Internet of Things: Survey and Future Vision', International Journal of Computer Networks, vol. 5, pp. 1–17, Feb. 2013.
[21] DARWISH, Dina Gamal. Improved Layered Architecture for Internet of Things. International Journal of Computing, 2015, vol. 4, no 4, p. 214-223.
[22] P. Sethi and S. R. Sarangi, 'Internet of Things: Architectures, Protocols, and Applications', Journal of Electrical and Computer Engineering, vol. 2017, p. 9324035, Jan. 2017, doi: 10.1155/2017/9324035.
[23] S. M. A. Group et al., 'Internet of Things (IoT): A Literature Review', Journal of Computer and Communications, vol. 03, no. 05, Art. no. 05, 2015, doi: 10.4236/jcc.2015.35021.
[24] "Machine-to-Machine communications (M2M); Functional architecture Technical Specification," 2013. Accessed: Oct. 12, 2021. [Online]. Available: http://portal.etsi.org/chaircor/ETSI_support.asp.
[25] "TR 102 935 - V2.1.1 - Machine-to-Machine communications (M2M); Applicability of M2M architecture to Smart Grid Networks; Impact of Smart Grids on M2M platform," 2012. Available: http://portal.etsi.org/chaircor/ETSI_support.asp.
[26] D. H. Wolpert, 'Stacked generalization', Neural Networks, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.