Towards a Self-Adaptive Architecture for Federated Learning of Industrial Automation Systems

Nicola Franco, Hoai My Van, Marc Dreiser, Gereon Weiss
Fraunhofer Institute for Cognitive Systems IKS
Munich, Germany
{name.surname}@iks.fraunhofer.de

***Abstract*—Emerging Industry 4.0 architectures deploy data-driven applications and artificial intelligence services across multiple locations under varying ownership, and require specific data protection and privacy considerations to not expose confidential data to third parties. For this reason, federated learning provides a framework for optimizing machine learning models in single manufacturing facilities without requiring access to training data. In this paper, we propose a self-adaptive architecture for federated learning of industrial automation systems. Our approach considers the involved entities on the different levels of abstraction of an industrial ecosystem. To achieve the goal of global model optimization and reduction of communication cycles, each factory internally trains the model in a self-adaptive manner and sends it to the centralized cloud server for global aggregation. We model a multi-assignment optimization problem by dividing the dataset into a number of subsets equal to the number of devices. Each device chooses the right subset to optimize the model at each local iteration. Our initial analysis shows the convergence property of the algorithm on a training dataset with different numbers of factories and devices. Moreover, these results demonstrate higher model accuracy with our self-adaptive architecture than the federated averaging approach for the same number of communication cycles.**

***Index Terms*—Self-Adaptive systems, federated learning, distributed simplex, Industry 4.0.**

I. Introduction

Distributed architectures for Industry 4.0 consist of horizontal, vertical, and end-to-end integration of the data processing chain [1]. Horizontal integration involves real-time cooperation between sensors, actuators, and controllers, implemented by distributed control systems, supervisory control and data acquisition, and programmable logic controllers. Vertical integration leads to information flow from the field to the cloud, i.e. from raw data acquisition to cloud processing, storage, and monitoring. In addition, end-to-end integration requires real-time responses for cyber-physical systems by ensuring at the same time the quality-of-service and by allocating machine learning functions close to the edge. For a successful realization, an underlying architecture should demonstrate the ability to integrate all these layers.

The nature of data captured by production systems is heterogeneous, therefore automatic pattern recognition and data mining functions suffer from the integration of different data formats and sources [2]. New procedures for analytical model management should promote decentralized and automated decision making to reduce the complexity of analyzing data from a centralized machine. To this end, data management and development, as well as the use of analytical models are inextricably linked and must be considered uniformly. For dynamic and horizontal scaling, approaches such as federated learning are currently being explored. Federated learning [3], [4], is a distributed machine learning approach, that allows analytical models to be trained and used on large amounts of data stored on different devices. For horizontal scaling of federated learning, new methods need to be developed to make the heterogeneous landscape of data sources available to the learning process and to enable the data processing chain.

An example use case is a third-party industrial partner who sells smart devices to several manufacturing facilities and wants to globally optimize a machine learning model by training it on each device without requiring confidential customer data. Therefore, the main goal of this use case is finding a good universal model that fits to different industrial data sets. We propose a self-adaptive architecture composed of entities on three different levels: centralized cloud server, smart factory, and smart industrial device. In our approach, we model the self-adaptive behaviour of the architecture through a MAPE-K control loop (Monitor, Analyze, Plan, and Execute over a shared Knowledge [5]).

The main contribution of this work is the proposal of a self-adaptive architecture that decreases the global aggregation cycles of the training procedure by using a decentralized optimization approach. A first analysis demonstrates the convergence property of the algorithm and highlights the self-adaptive behavior of the architecture. We evaluate our approach with respect to the federated averaging approach [4] on a varying number of factories and devices. The results show higher model accuracy for the same rounds of communication.

The paper is structured as follows: in Section II an overview of related works is given. In Section III our framework for the federated learning and in Section IV our self-adaptive architecture are presented. Section V presents the simulation-based evaluation of our approach. Finally, the paper is concluded and future developments are outlined.

II. Fundamentals and Related Work

Nowadays, several industries, researchers, and developers are actively working in the context of system architectures to provide solutions for flexibly integrating state-of-the-art artificial intelligent functions and components into the layered

industrial structures. Typically, industrial architectures are designed to meet the requirements of a subset of layers of the entire system hierarchy, depending on the real-time or non-real-time target. In this section, we present some of the existing industrial architectures focused on heterogeneous integration of the data processing chain, then highlight the impact of self-adaptation on the industrial domain, and conclude the section with a brief review of self-adaptive and heterogeneous federated learning techniques.

A. Industry 4.0

An important step towards efficient integration of industrial data analytics is the availability of machine learning and big data technologies in a dynamic distributed architecture. To support the implementation of Industry 4.0 into production environments, the key concept of an Asset Administration Shell (AAS) is predominant [6]. The AAS creates a digital representation of any industrial component (also including metadata descriptions) and identifies it with a unique address.

Recent surveys [7], [8], discuss the newest architectures and standards, such as BaSyx 4.0 [9], and the Reference Architecture Model of Industry 4.0 (RAMI 4.0) [10]. RAMI 4.0 is an abstract multi-layer standard capturing the factory hierarchy, OSI communication layers, and the product's life cycle. Based on RAMI 4.0, Basyx 4.0 implements a virtual middleware called Virtual Automation Bus (VAB) [11] that supports the AAS. The VAB connects existing protocols and network topologies into one virtual network able to handle real-time as well as non-real-time requirements. Other architectures targeting Industry 4.0 have common properties and share the idea of flexibility, such as IMPROVE [2]. The IMPROVE architecture relies on a common data model embedded inside the middleware to overcome the drawbacks of heterogeneous data handling, proprietary tools, and a unique way of data processing. According to [12], the implementation of the standard interface is made by the data management and integration broker, which transfers and routes data between the components and layers of the so-called Unified Data Transfer Architecture.

B. Self-Adaptation

In recent years, the focus of self-adaptation on industrial processes has shifted from sole development perspective to execution time aspects. The self-adaptive properties are typically implemented through feedback of software systems and achieved by MAPE-K control loops [5]. According to [13], the self-adaptive behavior of a system consists of a reconfiguration process based on changing external or internal conditions. In case of safety requirements, in [14] guarantees are provided for the self-adaptive behavior of the architecture and the running processes. In addition, machine learning approaches have been studied in [15] and [16] to manage the challenge of exploring wide spaces of possible behaviors within optimal times.

In line with this, our research targets multi-agent industrial systems, where the self-adaption of agents leads to collaboration and utilizes decentralized optimization to solve constrained decision-making. A related approach was presented in [17], where the decentralized aggregation process for shared knowledge resolves decision conflicts through subjective logic.

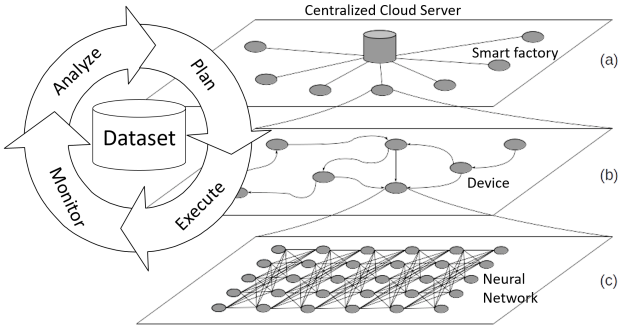


Fig. 1: Graphical representation of the framework: (a) Federated aggregation, (b) Decentralized optimization, (c) Model training.

C. Federated Learning

Federated approaches to learning have been studied extensively in various settings. Some of the early works [3], [4] present the framework as a global shared model, trained between different devices under the coordination of a central server. The distribution of computing power to edge components and preservation of data privacy are some of prominent advantages of the federated approach. Recently, heterogeneous setups have been proposed and considered from different perspectives [18]–[20]. During federated aggregation, the resource availability changes at runtime, and [20] defined it as a constrained optimization problem. In their case, the adaptive behavior makes use of a control algorithm to adapt the computing power to the availability of resources.

In our approach we assume that the smart factory dataset can be accessed from any device in this factory, which leads to a relaxation of the purely distributed framework. Our work focuses on the industrial settings of the federated aggregation and on greater accuracy achieved through the self-adaptive behavior of the architecture.

III. Framework for Federated Learning

In this section, we propose our framework, shown in Fig. 1. We conceptually divide the industrial structure into entities of three different levels: centralized cloud server, smart factory, and smart industrial device. This division is intended to reflect the existing industrial automation systems architecture as well as the training procedure of a machine learning model in this environment. For this reason, each entity level implements one of the following three stages of our training procedure:

- Federated aggregation
- Decentralized optimization
- Model training

Federated Aggregation: the centralized cloud server receives one machine learning model from each smart factory

TABLE I: Main Notations

N	Number of smart factories
N_i	Number of smart industrial devices for factory i
T	Number of global rounds
L	Number of local rounds
E	Number of epochs
$\mathcal{I}$	Set of smart factories
$\mathcal{N}_i$	Set of smart industrial devices for factory i
$\mathcal{G}_i^a$	Assignment graph for factory i
$\mathcal{G}_i^c$	Communication graph for factory i
$\boldsymbol{x}_k$	Measurements vector
$\boldsymbol{y}_k$	Labels vector
$\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$	Subset of the dataset
$\mathcal{D}_i$	Dataset for smart factory i
$\boldsymbol{w}^0$	Initial global model
$\boldsymbol{w}^t$	Global model at time t
$\boldsymbol{w}_j^t$	Local model for device/factory j at time t
$F_i(\boldsymbol{w})$	Global loss for factory i
$\bar{\ell}_j$	Average loss for model j on dataset $\mathcal{D}_i$
ℓ_{jk}	Local loss for model j on subset $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$
$\mathcal{L}$	Vector of losses
$\mathcal{L}_j$	Vector of losses for device j
$\boldsymbol{A}$	Matrix of constraints

for the global aggregation round. Afterwards, following the *Federated Averaging* algorithm [4], all models are averaged together. At the end of this stage, the server sends back the aggregated model to the factories.

Decentralized Optimization: each smart factory sends the aggregated model to its own devices and splits the dataset into a number of subsets equal to the number of devices. At each local round, all devices calculate the loss for each subset. To solve the multi-assignment problem, the devices run the *Distributed Simplex* algorithm [21] and agree on a common solution that minimizes the sum of the loss functions and assigns the correct subset. In the final step, the factory chooses the best model among the devices for the federated aggregation.

Model Training: each smart industrial device trains the machine learning model on a subset previously selected by the decentralized optimization step. The model is updated according to an optimization method.

A. Problem formulation

In order to describe the application scenario in the federated learning framework, we introduce main notations which are summarized in Table I.

We consider a set $\mathcal{I} = \{1, \dots, N\}$ of smart factories (Fig. 1(a)), where each factory i has a loss function $F_i(\boldsymbol{w})$ and a model vector $\boldsymbol{w} \in \mathbb{R}^d$, with equal dimension d between all factories. For the centralized cloud server, the optimization problem is considered as follows,

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad \frac{1}{N} \sum_{i \in \mathcal{I}} F_i(\boldsymbol{w}), \tag{1}$$

where N is the total number of smart factories. To avoid large amounts of data transferring and to reduce the number of communication rounds, each factory sends one model to the central server for the federated aggregation. In this context, this model is the one that achieved the best accuracy on the entire factory dataset.

From Fig. 1(b), we see that each factory has its own set of devices $\mathcal{N}_i$, $\forall i \in \mathcal{I}$ with $|\mathcal{N}_i| = N_i$, where $|\cdot|$ express the cardinality of the set. We consider a unique dataset $\mathcal{D}_i = \{\boldsymbol{x}, \boldsymbol{y}\}_i$ for factory i accessible to each device of that factory, with measurements $\boldsymbol{x}$ and labels $\boldsymbol{y}$.

Assumption 1 (Shared Knowledge)**.** *Each device of factory i, has access to the entire dataset $\mathcal{D}_i = \{\boldsymbol{x}, \boldsymbol{y}\}_i$ of that factory.*

To deal with a decentralized framework, the dataset is partitioned into N_i subsets, $\mathcal{D}_i = \{\{\boldsymbol{x}_1, \boldsymbol{y}_1\}, \dots, \{\boldsymbol{x}_{N_i}, \boldsymbol{y}_{N_i}\}\}$, where $\{\boldsymbol{x}_r, \boldsymbol{y}_r\} \cap \{\boldsymbol{x}_k, \boldsymbol{y}_k\} = \emptyset$, $\forall r, k \in \mathcal{N}_i$ with $r \neq k$. Each device receives a machine learning model $\boldsymbol{w}$ from the factory, Fig. 1(c), and updates it according to an optimization method, i.e. the Stochastic Gradient Descent (SGD) method, with the same learning rate and number of epochs used on each device. We name $\boldsymbol{w}_j$ the updated version of the model $\boldsymbol{w}$ in device j. Furthermore, we define $\ell_{jk} \triangleq \ell(\boldsymbol{w}_j; \boldsymbol{x}_k, \boldsymbol{y}_k)$ as the loss of model j on subset k, and $\bar{\ell}_j = \frac{1}{N_i} \sum_{k \in \mathcal{D}_i} \ell_{jk}$ as the average loss of model j over the entire dataset $\mathcal{D}_i$, where N_i is the number of devices. Without loss of generality, we assume having the same number of data points in each subset.

We further address problem (1) by rewriting the factory i's loss $F_i(\boldsymbol{w})$ as sum of individual losses $\bar{\ell}_j$, as follows,

$$F_i = \frac{1}{N_i} \sum_{j \in \mathcal{N}_i} \bar{\ell}_j = \frac{1}{N_i^2} \sum_{j \in \mathcal{N}_i} \sum_{k \in \mathcal{D}_i} \ell_{jk}. \tag{2}$$

As previously mentioned, our approach focuses on the local optimization of (1). Therefore, we describe our problem in terms of minimizing each loss F_i individually, which leads to the following formulation,

$$\min_{\boldsymbol{w}_1, \dots, \boldsymbol{w}_{N_i}} \quad \frac{1}{N_i^2} \sum_{j \in \mathcal{N}_i} \sum_{k \in \mathcal{D}_i} \ell_{jk}, \tag{3}$$

where the minimization is made with respect to the models $\boldsymbol{w}_1, \dots, \boldsymbol{w}_{N_i}$. We shift our attention from minimizing the sum of losses with respect to the models, and instead we focus on the subsets. Therefore, we propose to handle this problem as *primal assignment problem*, following the standard from [22], [23].

Let's consider a bipartite graph $\mathcal{G}_i^a = \{\mathcal{N}_i, \mathcal{D}_i; E_i^a\}$, as in Fig. 2, where E_i^a is the set of edges and the edge $(j,k) \in E_i^a$ exists if and only if the device j can be assigned to the subset k. For each edge (j,k), we introduce the binary decision variable $p_{jk} \in \{0,1\}$, which associates the subset k to the device j. In other words, if $p_{jk} = 1$ then subset k is assigned to device j, 0 otherwise. Problem (3) is now modelled as multi-assignment problem as follows,

$$\begin{aligned} \min_{\boldsymbol{p} \geq 0} \quad & \frac{1}{N_i^2} \sum_{(j,k) \in E_i^a} \ell_{jk} p_{jk} \\ \text{subject to} \quad & \sum_{\{k | (j,k) \in E_i^a\}} p_{jk} = 1, \quad \forall j \in \{1, \dots, N_i\}, \\ & \sum_{\{j | (j,k) \in E_i^a\}} p_{jk} = 1, \quad \forall k \in \{1, \dots, N_i\}, \end{aligned} \tag{4}$$

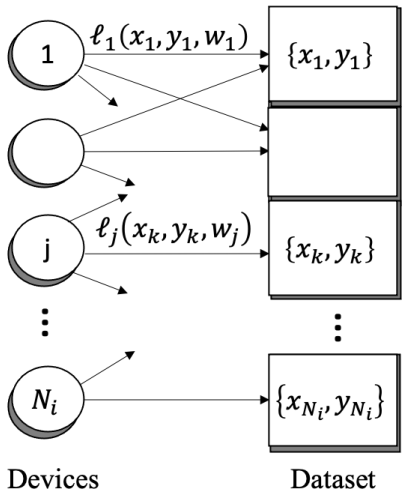


Fig. 2: Dataset assignment graph $\mathcal{G}_i^a$ of smart factory i.

where $\boldsymbol{p} = [p_{11}, p_{12}, \ldots, p_{N_i,N_i}]^\intercal$ represents the vector of decision variables. Both the first and second constraints show that each device must be assigned to a subset and each subset must be assigned to a device, respectively. Without loss of generality, we assume that $\mathcal{G}_i^a$ is *complete*[1]. Note that (4) has $N_i^2 = |E_i^a|$ decision variables, and $d_i = 2N_i$ constraints.

B. Decentralized optimization

We write problem (4) in matrix form, following the standard linear programming set-up:

$$\begin{aligned} \min_{\boldsymbol{p} \geq 0} \quad & \mathcal{L}^\intercal \boldsymbol{p} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}, \end{aligned} \tag{5}$$

where $\mathcal{L} \in \mathbb{R}^{N_i^2}$ represents the vector of losses $\mathcal{L} = [\ell_{11}, \ell_{12}, \ldots, \ell_{N_i N_i}]^\intercal$, and $\boldsymbol{p} \in \mathbb{R}^{N_i^2}$ the vector of decision variables. $\boldsymbol{A} \in \mathbb{R}^{d_i \times N_i^2}$ is the matrix of constraints and $\boldsymbol{b} \in \mathbb{R}^{d_i}$ is a vector of ones. We make use of the *distributed simplex algorithm* [21] to solve problem (5). Since our setup is not completely distributed and each device is connected to the factory that has access to the industrial data set, we adjust some of the original assumptions.

We model the network between the N_i devices of factory i by a directed graph (digraph) $\mathcal{G}_i^c = (\mathcal{N}_i, E_i^c)$. The device set is represented by $\mathcal{N}_i$ and the edge set by $E_i^c \subseteq \mathcal{N}_i \times \mathcal{N}_i$, where $(j,k) \in E_i^a$ if there is an edge which goes from device j to device k.

Assumption 2 (Strong Connectivity)**.** *The graph $\mathcal{G}_i^c$ is strongly connected, therefore, for every pair of nodes (i,j), there exists a path of directed edges that goes from i to j.*

The distributed simplex algorithm relies on a distributed information structure, where each device initially knows a subset of the problem columns. A column of problem (5) is $\boldsymbol{h}_j \in \mathbb{R}^{1+d_i}$ and is defined as $\boldsymbol{h}_j \triangleq \left[\ell_{jk},\ \boldsymbol{A}_{:j}^\intercal\right]^\intercal$, where $\ell_{jk} \in \mathbb{R}$ is the loss of model $\boldsymbol{w}_j$ on subset k, and $\boldsymbol{A}_{:j} \in \mathbb{R}^{d_i \times 1}$ is the j-th column of the matrix $\boldsymbol{A}$. To fully represent (5), we use the notation $(\mathbb{H}, \boldsymbol{b})$, where $\mathbb{H} = \{\boldsymbol{h}_j\}_{j \in \mathcal{N}_i}$ is the set of all columns. We assume there exists a partitioning $\mathcal{P} = \left\{\mathbb{P}^{[1]}, \ldots, \mathbb{P}^{[N_i]}\right\}$ of the problem columns with $\mathbb{H} = \cup_{j \in \mathcal{N}_i} \mathbb{P}^{[j]}$ and $\mathbb{P}^{[j]} \cap \mathbb{P}^{[k]} = \emptyset$. The distributed linear program is a tuple $(\mathcal{G}_i^a, (\mathbb{H}, \boldsymbol{b}), \mathbb{P})$ which consists of the communication graph $\mathcal{G}_i^a$, the linear program $(\mathbb{H}, \boldsymbol{b})$ and the problem partition $\mathbb{P}$. Considering the distributed linear program $(\mathcal{G}_i^a, (\mathbb{H}, \boldsymbol{b}), \mathbb{P})$, there always exists a solution of $(\mathbb{H}, \boldsymbol{b})$ as proven in [21].

IV. Self-Adaptive Architecture for Federated Learning

In this section, we describe our main contribution. The approach consists of managing the stages of the training procedure following the MAPE-K control loop. As mention in Section III, each entity implements different stages of the training procedure and has different roles. In the context of self-adaptation, the roles are monitor, analyze, plan and execute. The smart industrial devices are responsible for the monitoring and executing phase. The smart factory analyzes the model and plans the next local round. In the following, we introduce each role individually.

Algorithm 1: Centralized Cloud Server

Input: $\mathcal{I} = \{1, \ldots, N\}$, T
Output: $\boldsymbol{w}$
SERVER:
  initialize: $\boldsymbol{w}^0$
  for each global round $t = 0, 1, 2, ..., T-1$ **do**
    for each factory $i \in \mathcal{I}$ **do**
      $\boldsymbol{w}_i^{t+1} \leftarrow \textsc{Factory}_i(\boldsymbol{w}^t)$
    end
    $\boldsymbol{w}^{t+1} \leftarrow \frac{1}{N} \sum_{i \in \mathcal{I}} \boldsymbol{w}_i^t$
  end
return $\boldsymbol{w}^T$

Analysis. Starting from the centralized cloud server in Algorithm 1, the model vector $\boldsymbol{w}^0$ is randomly initialized and sent to each factory. T is the number of global aggregation rounds. When the factory receives the model, it starts the local training procedure, as shown in Algorithm 2. Once the local training is complete, the factory selects the model with the minimal loss and sends it to the central server for the global aggregation phase. The analysis takes place in this phase, where each model received from the smart devices is evaluated and only the one that had the best performance on the whole dataset, is sent to the central server.

Plan. In Algorithm 2, each smart factory plans the local training procedure by initializing the available devices and partitioning the dataset. L is the number of local training rounds, $\mathcal{N}_i$ is the set of devices and $\mathcal{D}_i$ is the factory dataset. At global round t, factory i receives the global model $\boldsymbol{w}^t$ from the centralized server and initializes the local model $\boldsymbol{w}_j^0$ of

[1] i.e. there exists an edge between every device $j \in \mathcal{N}_i$ and every subset $k \in \mathcal{D}_i$.

Algorithm 2: Smart Factory

Input: $\boldsymbol{w}^t$, $\mathcal{N}_i = \{1, \ldots, N_i\}$, $\mathcal{D}_i = \{\boldsymbol{x}, \boldsymbol{y}\}_i$, L
Output: $\boldsymbol{w}_i^{t+1}$
FACTORY$_i$ $(\boldsymbol{w}^t)$**:**
  initialize: $\boldsymbol{w}_j^0 \leftarrow \boldsymbol{w}^t \quad \forall j \in \mathcal{N}_i$
  Split the dataset $\mathcal{D}_i$ into N_i subsets
  for each local round $l = 0, 1, 2, ..., L-1$ **do**
    for each device $j \in \mathcal{N}_i$ **do**
      $\boldsymbol{w}_j^{l+1}, \bar{\ell}_j \leftarrow$ **DEVICE**$_j(\boldsymbol{w}_j^l, \mathcal{D}_i)$
    end
  end
  /* Find the minimum at last step */
  $\boldsymbol{w}_i^{t+1} \leftarrow \boldsymbol{w}_{min}^L : \{min \in \mathcal{N}_i \mid \bar{\ell}_{min} < \bar{\ell}_j \ \forall j \in \mathcal{N}_i\}$
return $\boldsymbol{w}_i^{t+1}$

each device j with it. Before the first local round, the factory sends the model to each device and gives access to the entire partitioned dataset $\mathcal{D}_i$. Afterwards, the factory receives the updated model $\boldsymbol{w}_j^{l+1}$ and the corresponding average loss $\bar{\ell}_j$. Finally, after the last local round, each factory sends the model with the minimum loss back to the server.

Algorithm 3: Smart Industrial Device

Input: $\boldsymbol{w}_j^l$, $\mathcal{D}_i = \{\{\boldsymbol{x}_1, \boldsymbol{y}_1\}, \ldots, \{\boldsymbol{x}_{N_i}, \boldsymbol{y}_{N_i}\}\}$, E
Output: $\boldsymbol{w}_j^{l+1}$
DEVICE$_j$ $(\boldsymbol{w}_j^l, \mathcal{D}_i)$**:**
  /* Initialize α at first step */
  if l **is equal to** 0 **then** $\alpha \leftarrow j$
  $\boldsymbol{w}_j^{l+1} \leftarrow$ *SGD*$(\boldsymbol{w}_j^l; \boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha)$ for E epochs
  for each $\{\boldsymbol{x}_k, \boldsymbol{y}_k\} \in \mathcal{D}_i$ **do**
    $\ell_{jk} \leftarrow \ell(\boldsymbol{w}_j^l; \boldsymbol{x}_k, \boldsymbol{y}_k)$
  end
  $\bar{\ell}_j \leftarrow \frac{1}{N_i} \sum_{k \in \mathcal{D}_i} \ell_{jk}$
  /* Penalize previous subset */
  $\ell_{j\alpha} \leftarrow 1$
  $\mathcal{L}_j \leftarrow [\ell_{j1}, \ldots, \ell_{jk}, \ldots, \ell_{jN_i}]^\intercal$
  $\alpha \leftarrow$ *Decentralized Optimization*$(\mathcal{L}_j, N_i)$ ref. [21]
return $\boldsymbol{w}_j^{l+1}, \bar{\ell}_j$

Execute. In Algorithm 3, each device creates its vector $\mathcal{L}_j$ of losses, where the entries correspond to the computed losses of model $\boldsymbol{w}_j^l$ on each subset. $\mathcal{L}_j$ and N_i are used to initialize the *Decentralized Optimization* function. The function executes the *Distributed Simplex* algorithm [21] and solves the multi-assignment problem. α is a scalar and represents the solution of the assignment problem, in other words, it makes the choice on which subset the model will be trained in the next local round. In the first local round, α is initialized with value j, corresponding to the device number, so each device initially trains the model on the subset j. In this algorithm, the updated model $\boldsymbol{w}_j^{l+1} = \boldsymbol{w}_j^l - \eta \nabla \ell(\boldsymbol{w}_j^l; \boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha)$ is computed according to the *Stochastic Gradient Descent* [24], on the subset chosen by α. E corresponds to the number of epochs and η is the *learning rate*. Before the next local round, we penalize the decision to train the model in the previously chosen subset, therefore we assign the maximum value 1 to the loss $\ell_{j\alpha}$, which corresponds to the previous choice.

Monitor. During the training phase, in Algorithm 3, the device computes the loss on each subset. Afterwards, it calculates the average $\bar{\ell}_j$ of the previously computed losses and send it to the smart factory. As mentioned above, the average is used to analyze which device performed best.

V. EVALUATION

A. Setup

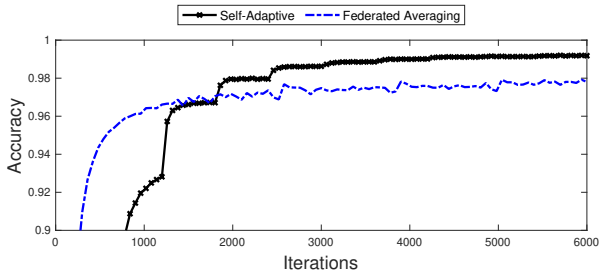
We evaluate the performance of our algorithm by conducting simulation experiments with i.i.d. data. The environment we modeled with the deep learning toolbox in MATLAB[2] consists of a varying number of factories, between 3 to 5, and devices, between 10 to 15. We choose a convolutional neural network (CNN) model and train it on the *Digits*[3] dataset, which consists of 10,000 synthetic grayscale images of handwritten digits and (1-9) labels. The neural network has a 28×28 input layer, 3 convolution bi-dimensional layers with pooling layers in between and a fully connected layer with the softmax function for classification at the end.

First, we divide the dataset with equal sizes into factories and split it up further into the number of devices. Each factory has a different portion of the dataset, but with the same number of images and labels. We initialize each factory and device with an equal NN model. We compare our approach with the *federated averaging* (FA) [4] for the same number of devices and iterations, i.e. the number of rounds used for FA is equal to $T \cdot L$. The evaluation consists on averaging the resulting loss of each device $(\bar{\ell}_j)$ of all factories. The average accuracy is shown in figure 3 for both self-adaptive and federated averaging approaches. Each iteration corresponds to training the model on a batch. We performed our simulations with a number of epochs $E = 30$, learning rate $\eta = 0.001$ and batch size equal to 128.

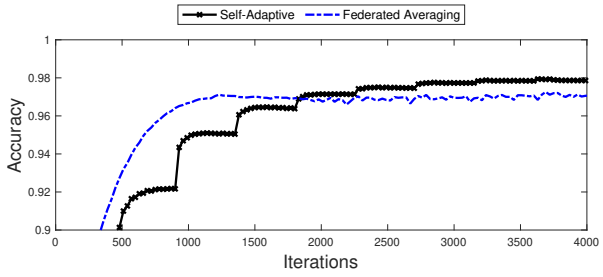
B. Results and Discussions

In Figure 3, the average accuracy is displayed for both the federated averaging and self-adaptive approaches. We notice that after an initial phase, the self-adaptation algorithm achieves better model accuracy than federated averaging. The self-adaptive behaviour creates small adjustments during the transition between local decentralized optimization and global aggregation. During local rounds, the accuracy asymptotically reaches a stable value and after a global round it jumps to a higher value. The number of global aggregations is L times lower than FA, because the model is internally trained for L local rounds and later sent to the server. In FA each device sends the model to the central server at each global round, therefore, high accuracy is reached faster than the self-adaptive but requires more communication between the central server

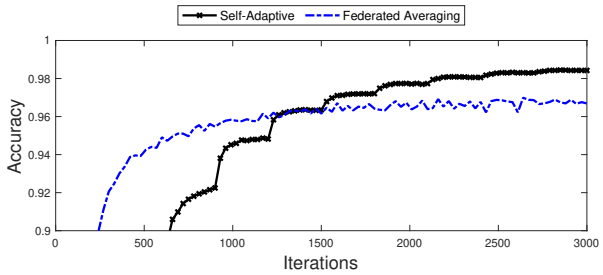
[2] www.mathworks.com/products/deep-learning.html
[3] www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html



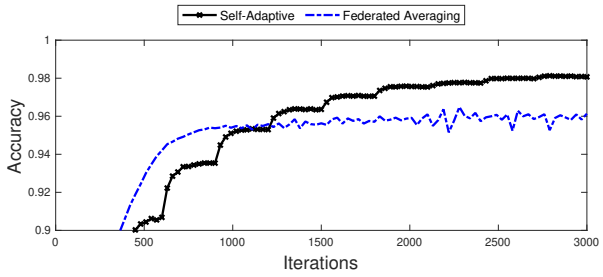
(a) Average accuracy of 3 smart factories with 10 devices each trained for 10 global and local rounds.



(b) Average accuracy of 3 smart factories with 15 devices each trained for 15 global and local rounds.



(c) Average accuracy of 4 smart factories with 10 devices each trained for 10 global and local rounds.



(d) Average accuracy of 5 smart factories with 10 devices each trained for 10 global and local rounds.

Fig. 3: Comparison of results between self-adaptive and federated averaging approaches.

and the device. In our approach, the model is trained internally for a number of local cycles and then sent to the central server. The slow behavior of the self-adaptive approach is due to the internal training constraint and generates more internal cycles of communication between devices than with the central server.

VI. Conclusions and Outlook

In this paper, we propose a self-adaptive architecture for federated learning of industrial automation systems. Our approach consists on defining the industrial setting into entities of three different levels: centralized cloud server, smart factory, and smart industrial device. We model the self-adaptive behaviour of the architecture through MAPE-K control loops. Based on this, we define the training procedure as a multi-assignment problem and use decentralized optimization. The results of our solution show a greater model accuracy than the *federated averaging* approach [4], for the same rounds of communication.

Despite these recent findings, the proposed architecture can be evaluated further. For instance, different neural networks and datasets will be tested and communication errors taken into account. Furthermore, the adoption of the architecture in a real industrial scenario may lead to unanticipated challenges. For this reason, the integration with current Industry 4.0 solutions is one of our targeted next steps.

Acknowledgment

This work was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy as part of a project to support the thematic development of the Institute for Cognitive Systems.

References

[1] Keliang Zhou, Taigang Liu, and Lifeng Zhou. Industry 4.0: Towards future industrial opportunities and challenges. In *2015 12th International conference on fuzzy systems and knowledge discovery (FSKD)*, pages 2147–2152. IEEE, 2015.

[2] Emanuel Trunzer, Iris Kirchen, Jens Folmer, Gennadiy Koltun, and Birgit Vogel-Heuser. A flexible architecture for data mining from heterogeneous data sources in automated production systems. In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 1106–1111. IEEE, 2017.

[3] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[5] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[6] Constantin Wagner, Julian Grothoff, Ulrich Epple, Rainer Drath, Somayeh Malakuti, Sten Grüner, Michael Hoffmeister, and Patrick Zimermann. The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2017.

[7] Emanuel Trunzer, Ambra Calà, Paulo Leitão, Michael Gepp, Jakob Kinghorst, Arndt Lüder, Hubertus Schauerte, Markus Reifferscheid, and Birgit Vogel-Heuser. System architectures for industrie 4.0 applications. *Production Engineering*, 13(3-4):247–257, 2019.

[8] Matti Yli-Ojanperä, Seppo Sierla, Nikolaos Papakonstantinou, and Valeriy Vyatkin. Adapting an agile manufacturing concept to the reference architecture model industry 4.0: A survey and case study. *Journal of Industrial Information Integration*, 15:147–160, 2019.

[9] Tarik Terzimehic, Monika Wenger, Alois Zoitl, Andreas Bayha, Klaus Becker, Thorsten Müller, and Hubertus Schauerte. Towards an industry 4.0 compliant control software architecture using iec 61499 & opc ua. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2017.

[10] Karsten Schweichhart. Reference architectural model industrie 4.0 (rami 4.0). *An Introduction. Available online: https://www. plattform-i40. de I*, 40, 2016.

[11] Thomas Kuhn, Pablo Oliveira Antonino, and Frank Schnicke. Industrie 4.0 virtual automation bus architecture. In *European Conference on Software Architecture*, pages 477–489. Springer, 2020.

[12] Emanuel Trunzer, Simon Lötzerich, and Birgit Vogel-Heuser. Concept and implementation of a software architecture for unifying data transfer in automated production systems. In *IMPROVE-Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency*, pages 1–17. Springer Vieweg, Berlin, Heidelberg, 2018.

[13] Danny Weyns. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering*, 2017.

[14] Danny Weyns, Nelly Bencomo, Radu Calinescu, Javier Camara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, et al. Perpetual assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 31–63. Springer, 2017.

[15] Federico Quin, Thomas Bamelis, Singh Buttar Sarpreet, and Sam Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–12. IEEE, 2019.

[16] Jeroen Van Der Donckt, Danny Weyns, Federico Quin, Jonas Van Der Donckt, and Sam Michiels. Applying deep learning to reduce large adaptation spaces of self-adaptive systems with multiple types of goals. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 20–30, 2020.

[17] Ana Petrovska, Sergio Quijano, Ilias Gerostathopoulos, and Alexander Pretschner. Knowledge aggregation with subjective logic in multi-agent self-adaptive cyber-physical systems. *SEAMS '20*, 2020.

[18] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019.

[19] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[20] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[21] Mathias Bürger, Giuseppe Notarstefano, Francesco Bullo, and Frank Allgöwer. A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. *Automatica*, 48(9):2298–2304, 2012.

[22] Dimitri P Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123, 1988.

[23] Dimitri P Bertsekas. *Network optimization: continuous and discrete models*. Athena Scientific Belmont, MA, 1998.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. The MIT Press, 2016.