Networked Federated Meta-Learning Over Extending Graphs

Muhammad Asaad Cheema, *Member, IEEE*, Vinay Chakravarthi Gogineni, *Senior Member, IEEE*, Pierluigi Salvo Rossi, *Senior Member, IEEE*, and Stefan Werner, *Fellow, IEEE*

***Abstract*—Distributed and collaborative machine learning over emerging Internet of Things (IoT) networks is complicated by resource constraints, device, and data heterogeneity, and the need for personalized models that cater to the individual needs of each network device. This complexity becomes even more pronounced when new devices are added to a system that must rapidly adapt to personalized models. Along these lines, we propose a networked federated meta-learning (NF-ML) algorithm that utilizes meta-learning and underlying shared structures across the network to enable fast and personalized model adaptation of newly added network devices. The NF-ML algorithm learns two sets of model parameters for each device in a distributed manner, with devices communicating only with their immediate neighbors. One set of parameters is personalized for the device-specific task, whereas the other is a generic parameter set learned via peer-to-peer communication. The performance of the proposed NF-ML algorithm was validated using both synthetic and real-world data, and the results show that it adapts to new tasks in just a few epochs, using as little as 10% of the available data, significantly outperforming traditional federated learning methods.**

***Index Terms*—Distributed, generic parameters, graph federated learning (GFL), meta-learning.**

I. Introduction

THE EMERGENCE of the Internet of Things (IoT), coupled with significant advancements in information and communications technology, has paved the way for the effortless collection of vast amounts of data. This surge of data has captured the interest of businesses and researchers, leading to the broader adoption of data-driven strategies for more insightful decision making. Traditionally, these strategies, including deep learning, have relied on access to large, centrally located, data sets for effective and efficient data-driven model development [1]. However, such centralized approaches are associated with high communication and storage demands owing to data transfer and aggregation from myriads of devices, particularly those generating high volumes of data, such as video cameras or lidar sensors [2]. Furthermore, transmitting a massive amount of data to a single central processing point can raise privacy and legal concerns, especially in light of stringent regulations such as the General Data Protection Regulation (GDPR) [3].

Distributed learning has emerged as an attractive alternative that offers a framework that leverages on-device processing capabilities, and facilitates efficient data analysis without the need for data transfer from devices. One prominent methodology within distributed learning is federated learning (FL), introduced in [4] as a concept that emphasizes the collaborative training of machine-learning models without centralizing client-specific data. This process involves training the model locally on each client and transmitting only model updates to a central server. The central server aggregates these updates to improve the model, which is then sent back to the devices for further learning, a procedure that continues until convergence. This approach effectively reduces the risk of data exposure and is well suited for IoT scenarios [5], [6], [7], [8]. In addition to data privacy, FL enhances the scalability of IoT networks by distributing the learning process without overwhelming the central server. Also, the absence of massive data transfers from IoT devices reduces communication costs [9], particularly beneficial in networks with limited bandwidths.

Although FL is a promising approach for IoT devices, it is not without its challenges. First, due to statistical heterogeneity, a single model may not be optimal for all clients [10]. Second, training a shared global model could still require significant communication resources [11]. Third, model aggregation cannot rely solely on a single server as blockages may occur in large systems [12], [13]. Finally, the risk posed by malicious clients, who can disrupt the learning process, raises serious security concerns [14], [15]. In this article, we focus on the first and third challenges.

Various solutions have been proposed to address the challenge of using a single model for every client, with personalized FL (PFL) emerging as a prevalent approach [16]. PFL largely follows the conventional FL process, starting with

Manuscript received 13 June 2024; accepted 9 August 2024. Date of publication 14 August 2024; date of current version 20 November 2024. This work was supported in part by the Research Council of Norway under Project ML4ITS within the IKTPLUSS Framework, and Project Resource-Aware IoT with Enhanced Intelligence and Security. *(Corresponding author: Muhammad Asaad Cheema.)*

Muhammad Asaad Cheema is with the Department of Electronic Systems, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, 7034 Trondheim, Norway (e-mail: asaad.cheema@ntnu.no).

Vinay Chakravarthi Gogineni is with the SDU Applied AI and Data Science, The Maersk Mc-Kinney Møller Institute, University of Southern Denmark, 5230 Odense, Denmark (e-mail: vigo@mmmi.sdu.dk).

Pierluigi Salvo Rossi is with the Department of Electronic Systems, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, 7034 Trondheim, Norway, and also with the Department of Gas Technology, SINTEF Energy Research, 7034 Trondheim, Norway (e-mail: salvorossi@ieee.org).

Stefan Werner is with the Department of Electronic Systems, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, 7034 Trondheim, Norway, and also with the Department of Information and Communications Engineering, Aalto University, 02150 Espoo, Finland (e-mail: stefan.werner@ntnu.no).

Digital Object Identifier 10.1109/JIOT.2024.3443467

training a single global model. Once trained, this model is locally fine-tuned at each FL client using local data. This two-step approach, which combines global training with local adaptation, is considered a promising strategy for model personalization in FL [14].

Building on the foundation of PFL, various techniques for enhancing knowledge sharing among clients have been explored. The pFedMe algorithm for PFL was introduced in [17], leveraging Moreau envelopes as client-specific regularized loss functions. This approach uniquely decouples personalization from global model learning. A method that promotes collaboration between similar clients using federated attentive message passing and enables more effective collaboration among clients was introduced in [18]. PFL was refined in [19] by deriving the generalization bounds of mixed local and global models, pinpointing the optimal mixing parameter for improved personalization. Other works have capitalized on client relationships, formulating an underlying graph structure to facilitate more effective personalization models [20], [21]. Specifically, the methodology in [20] employs prompt-based communication, thus enabling the server to determine client relationships and analyze spatiotemporal correlations while maintaining data privacy.

Despite advances in the field of PFL, it is essential to note that most existing approaches rely heavily on centralized servers for coordination and model aggregation. Some works tackling this particular issue of centralized servers are briefly discussed as follows.

For instance, a client–edge–server hierarchical learning method was proposed in [22], in which it is assumed that there is one cloud server and multiple edge servers, with each edge server connected to a set of clients. The edge servers aggregate models from the clients associated with them and then communicate their aggregation results to a central cloud server for global aggregation. However, this single cloud server approach is susceptible to bottlenecks and has a limited capacity to accommodate edge servers [23].

Graph FL (GFL) was proposed in [24] as a refined FL framework involving multiple interconnected servers, each associated with specific clients, and allowing the representation of server connections through a graph to avoid communication failures and computational overloads at the server. GFL was enhanced by using multiple servers [25] via an online graph federated multitask learning (O-GFML) algorithm within the context of kernel regression to efficiently handle online data streaming across clients. Although these techniques present a distributed solution, they often depend on servers for aggregation, limiting their full potential. Furthermore, these methods do not fully leverage the inherent relationships between devices, leaving interconnected information unexplored and underutilized. Additionally, they demonstrate a limited capacity to adapt to growing networks in which new devices/clients continuously join, hindering their scalability and flexibility.

Distributed learning was exploited for efficient use of local information and network computational resources, particularly in optimizing convex objective functions across multiagent systems [26], [27]. Adaptive cost frameworks were introduced to study the tradeoff between computational and communication costs in distributed learning environments [28], [29]. However, while these studies lay a solid foundation for distributed optimization and address challenges related to communication/computational efficiency, they focus on a fixed number of nodes and do not account for dynamic scenarios where the network topology might expand with new nodes joining. Such a dynamic scenario is particularly relevant in modern applications, where nodes (e.g., mobile devices or distributed sensors with limited data and computational resources) frequently enter the network.

This article introduces a novel networked federated meta-learning (NF-ML) framework. Unlike traditional approaches, NF-ML functions without a central server for model aggregation and seamlessly incorporates new devices into the network. By utilizing meta-learning and relying solely on connections with immediate neighbors, NF-ML develops a generic model that is quickly adaptable to newly joined devices in the network. This approach simplifies and accelerates the development of personalized models for each new device, requiring fewer training samples and learning epochs. To validate the efficiency of the proposed method, we tested it by using synthetic and real-world data. Additionally, we benchmarked NF-ML against various FL strategies, including training a test device from scratch, and traditional techniques, such as FedAvg and personalized FedAvg. These comparisons further highlight the effectiveness of NF-ML in dynamic environments. The main contributions of this manuscript are summarized as follows.

1) We introduce NF-ML, a PFL framework that leverages meta-learning and device interconnectivity to efficiently manage expanding IoT networks.
2) A key feature of NF-ML is its fully distributed training approach wherein each network device learns generic and personalized model parameters, eliminating the need for a central server and enhancing the network scalability and robustness.
3) NF-ML features fast adaptation of newly integrated devices by significantly reducing the number of epochs and data samples required for training their device-specific tasks, thus improving overall network performance.

The remainder of this article is organized as follows. Section II provides the fundamentals and explains the key concepts and variants of meta-learning and FL. Section III introduces and details NF-ML and highlights its unique meta-learning-based training and rapid adaptation to new devices. Section IV discusses the synthetic and real-world data used in our experiment and compares the performance of NF-ML with that of other standard methods. Finally, Section V concludes this article and outlines potential future work.

Notations: Uppercase (resp., lowercase) bold letters denote matrices (resp., column vectors). Scalars are represented using regular lowercase letters, such as a. The transpose of matrix $\mathbf{A}$ is denoted $\mathbf{A}^{\mathrm{T}}$. The gradient of a function is denoted by $\nabla a(\cdot)$. Sets are denoted by calligraphic letters (except the loss function, which is represented by $\mathcal{L}$) and $|\mathcal{C}|$ is the cardinality of set $\mathcal{C}$. The Euclidean norm of a vector is denoted as $||\cdot||_2$.

Finally, $\mathcal{U}[a, b)$ denotes a uniform distribution over interval $[a, b)$.

II. Preliminaries

In this section, we first review the concepts of FL. Next, we discuss meta-learning [30] and, finally, the fundamentals of Reptile [31], a meta-learning variant.

A. Federated Learning

In the traditional FL environment, a server is connected to a set of clients $\mathcal{S}$ to solve the following optimization problem:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \text{ where } \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \mathcal{L}_k(\boldsymbol{\theta}, \mathcal{D}_k) \tag{1}$$

where $\boldsymbol{\theta}$ represents the globally shared model parameters and $\mathcal{L}_k$ is the local objective function used in conjunction with the local data set $\mathcal{D}_k$ for the kth client [4], [16].

In each iteration of the FL algorithm, designated by n, the server initially broadcasts the global model, denoted by $\boldsymbol{\theta}_n$, to a predefined subset of clients specified by $\mathcal{S}_n$. Upon receiving the global model, each client performs local learning by executing one or more gradient updates. For instance, when employing a single gradient update, the local update can be expressed as

$$\boldsymbol{\theta}_{k,n+1} = \boldsymbol{\theta}_n - \eta \nabla \mathcal{L}_k(\boldsymbol{\theta}_n, \mathcal{D}_k) \tag{2}$$

where $\eta > 0$ is the stepsize that controls learning rate and stability. After performing local updates, the selected clients share them with the server that updates the global model as follows:

$$\boldsymbol{\theta}_{n+1} = \frac{1}{|\mathcal{S}_n|} \sum_{k \in \mathcal{S}_n} \boldsymbol{\theta}_{k,n+1}. \tag{3}$$

This update-aggregation procedure is repeated until convergence, or when a predefined performance criterion is satisfied.

B. Meta-Learning

Meta-learning aims to train a meta-function $f_{\boldsymbol{\theta}}$, a generic structure across various tasks that can quickly adapt to new tasks, even with few training examples. Different algorithms have been investigated to train the meta-function $f_{\boldsymbol{\theta}}$ [32]. Among these, the model-agnostic meta-learning (MAML) algorithm [33] is prominent due to its universal applicability and model-agnostic characteristics. Unlike previous meta-learning methods [34], [35], [36] that learn an update function or rule, MAML does not increase the number of learned parameters or impose constraints on the model architecture, such as requiring a recurrent model [37] or a Siamese network [38]. Furthermore, it can be utilized with various loss functions, including differentiable supervised losses and nondifferentiable reinforcement learning losses, to ensure efficient weight initialization. MAML involves two functions: the first one ($f_{\boldsymbol{\theta}}$) is a meta-function with parameters $\boldsymbol{\theta}$ and is trained across the set of tasks $\mathcal{T}$, and the second one ($f_{\boldsymbol{\phi}_i}$) with parameters $\boldsymbol{\phi}_i$ corresponding to a specific task $\mathcal{T}_i \in \mathcal{T}$. The generic model $\boldsymbol{\theta}$ serves as a starting point for learning a specific task. In MAML, the data set of each task is partitioned into a support set ($\mathcal{D}^{\mathrm{S}}$) and a query set ($\mathcal{D}^{\mathrm{Q}}$), which are used to iteratively update $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, respectively. At time index n, given the generic structure $\boldsymbol{\theta}_n$, the task-specific model $\boldsymbol{\phi}_{i,n}$ is updated via a gradient descent search over the task-specific loss function $\mathcal{L}_{\mathcal{T}_i}$ for $\mathcal{T}_i \in \mathcal{T}$ as follows:

$$\boldsymbol{\phi}_{i,n} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}_n} \mathcal{L}_{\mathcal{T}_i}\left(\boldsymbol{\theta}_n, \mathcal{D}_i^{\mathrm{S}}\right) \tag{4}$$

where η is the learning rate. Using the updated $\boldsymbol{\phi}_{i,n}$, MAML then calculates the loss on the query set $\mathcal{D}_i^{\mathrm{Q}}$ as

$$\mathcal{L}_{\mathcal{T}_i}\left(\boldsymbol{\phi}_{i,n}, \mathcal{D}_i^{\mathrm{Q}}\right) = \mathcal{L}_{\mathcal{T}_i}\left(\boldsymbol{\theta}_n - \alpha \nabla_{\boldsymbol{\theta}_n} \mathcal{L}_{\mathcal{T}_i}\left(\boldsymbol{\theta}_n, \mathcal{D}_i^{\mathrm{S}}\right), \mathcal{D}_i^{\mathrm{Q}}\right). \tag{5}$$

The term $\boldsymbol{\theta}_n - \alpha \nabla_{\boldsymbol{\theta}_n} \mathcal{L}_{\mathcal{T}_i}(\boldsymbol{\theta}_n, \mathcal{D}_i^{\mathrm{S}})$ in (5) represents the updated parameter $\boldsymbol{\phi}_i$. Once the loss on the query set for each task is computed using (5), the following meta-update step is executed to refine $\boldsymbol{\theta}_n$:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \beta \nabla_{\boldsymbol{\theta}_n} \sum_{\text{Task } i} \mathcal{L}_{\mathcal{T}_i}\left(\boldsymbol{\phi}_{i,n}, \mathcal{D}_i^{\mathrm{Q}}\right) \tag{6}$$

where β is the learning rate for the meta-update. Although MAML is model-agnostic and has universal model architecture adaptability, it may become computationally prohibitive in certain situations. The complexity primarily arises from the need to compute Hessian-vector products for the meta-gradient calculation in (5), with Hessian being the matrix of second-order partial derivatives. Although this product does not need to be computed directly, as efficient algorithms exist for this purpose, the computational complexity of MAML can still be prohibitive for large-scale problems or resource-constrained devices due to the linear increase in complexity with the number of inner stage gradient updates.

To sidestep the need for the Hessian, a few variants of MAML have been developed, such as FOMAML and Reptile [31]. The simplicity of the implementation and absence of a need for partitioning the data into query and support sets makes Reptile a more natural choice than FOMAML. It learns task-specific parameters $\boldsymbol{\phi}$ using generic parameters $\boldsymbol{\theta}$ and training data. Subsequently, $\boldsymbol{\theta}$ is updated using the parameters trained over the $|\mathcal{T}|$ tasks, as follows:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\epsilon}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \left(\boldsymbol{\phi}_{i,n} - \boldsymbol{\theta}_n\right) \tag{7}$$

where ϵ is a hyperparameter that controls the rate of the update. From (7), we see that Reptile updates parameters $\boldsymbol{\theta}$ in the direction of $\sum_i (\boldsymbol{\phi}_i - \boldsymbol{\theta}_n)$ without the need for Hessian-vector products.

III. Problem Formulation and Proposed Algorithm

We consider a distributed federated network of IoT devices [39] represented by an undirected graph at time t, $\mathcal{G}_t = (\mathcal{K}_t, \mathcal{E}_t)$, where $\mathcal{K}_t$ is the set of IoT devices with $K = |\mathcal{K}_t|$ and $\mathcal{E}_t \subseteq \mathcal{K}_t \times \mathcal{K}_t$ is the edge set such that $\mathcal{E}_t(k, \ell) = 1$ (resp., $\mathcal{E}_t(k, \ell) = 0$) indicates that devices k and l are (resp., are not) neighbors. A device $k \in \mathcal{K}_t$ can only communicate and share model parameters with its neighbors. The set $\mathcal{N}_k =$

Algorithm 1 NF-ML Training

At all devices $k \in \mathcal{K}$, initialize $\boldsymbol{\theta}_k$ randomly for $n = 0$, and $\mathcal{B}_k \leftarrow$ split($\mathcal{D}_k$ into batches of size B)
for $n = 1$ to N_{Global} **do**
  Local Parameter Update:
  $\mathcal{N}_k$ from $\mathcal{E}_t$ for device k
  $\boldsymbol{\theta}_{k,n} = \frac{1}{|\mathcal{N}_k|} \sum_{\ell \in \mathcal{N}_k} \boldsymbol{\theta}_{\ell,n}$
  $\tilde{\boldsymbol{\phi}}_{k,n}^{(0)} = \boldsymbol{\theta}_{k,n}$
  for $j = 1$ to $|\mathcal{B}_k|$ **do**
    $\tilde{\boldsymbol{\phi}}_{k,n}^{(j)} \leftarrow \tilde{\boldsymbol{\phi}}_{k,n}^{(j-1)} - \eta \nabla \mathcal{L}_k\left(\tilde{\boldsymbol{\phi}}_{k,n}^{(j-1)}, b_j\right)$
  end for
  $\boldsymbol{\phi}_{k,n} = \tilde{\boldsymbol{\phi}}_{k,n}^{(j)}$
  Meta Update:
  $\boldsymbol{\theta}_{k,n+1} = (1-\epsilon)\boldsymbol{\theta}_{k,n} + \epsilon \cdot \boldsymbol{\phi}_{k,n}$
  Share($\boldsymbol{\theta}_{k,n+1}$) $\rightarrow$ *Devices*$\{\ell : \ell \in \mathcal{N}_k \setminus \{k\}\}$
end for

$\{\ell \in \mathcal{K}_t : \{k, \ell\} \in \mathcal{E}_t\} \cup \{k\}$ is defined as the neighborhood set of device k, which includes not only the neighbors of device k but also the device k itself.

The graph $\mathcal{G}_t$ is assumed to be growing, i.e., it expands at $t+1$ as new devices join the network. The objective for these new devices is to quickly adapt to their tasks by using limited data samples without relying on the central server. In order to achieve this objective, we develop an efficient and distributed method that leverages the concept of meta-learning within a networked FL framework. The proposed approach sets up two kinds of parameters for each device: 1) localized model parameters ($\boldsymbol{\phi}_k$) for their own tasks and 2) generic parameters ($\boldsymbol{\theta}_k$) that hold information about the general structure of data available at devices participating in networked FL environment. When shared, these generic parameters ($\boldsymbol{\theta}_k$) help newly joined devices in the graph to quickly adapt to their tasks. It is worth mentioning that we will use "localized model parameters" and "personalized model parameters" interchangeably. These parameters represent the fine-tuned version of the generic parameters ($\boldsymbol{\theta}_k$) for device k, obtained through one or more gradient update steps.

The NF-ML algorithm comprises two main steps. The first step, referred to as the distributed FL step, involves several communication rounds between each device and its neighbors. At the start of each round, the local model parameters ($\boldsymbol{\phi}_k$) are initialized with generic parameters ($\boldsymbol{\theta}_k$) and then updated based on local data. A meta-update is then performed, which is shared and merged with neighboring devices for subsequent rounds. After several communication rounds, the optimized generic parameters are obtained. The second step involves incorporating new devices and adjusting their initial parameters to create a personalized model. For this purpose, the algorithm utilizes neighboring devices and their fine-tuned generic parameters for fast and efficient device integration.

A. NF-ML Distributed FL

The primary objective of the NF-ML algorithm is to learn generic model parameters $\boldsymbol{\theta}$ for each device to minimize the average loss across its neighboring devices based on its generic parameters. The objective function for the kth device can be represented as follows:

$$\min_{\boldsymbol{\theta}_k} \frac{1}{|\mathcal{N}_k|} \sum_{\ell \in \mathcal{N}_k} \mathcal{L}_\ell(U(\boldsymbol{\theta}_k), \mathcal{D}_\ell) \tag{8}$$

where $U(\boldsymbol{\theta}_k)$ represents the operator that updates $\boldsymbol{\theta}_k$ using data $\mathcal{D}_\ell$ by applying one or more gradient updates yielding $\boldsymbol{\phi}_\ell$. For instance, in the case of using one full batch gradient update

$$\boldsymbol{\phi}_\ell = U(\boldsymbol{\theta}_k) = \boldsymbol{\theta}_k - \eta \nabla \mathcal{L}_\ell(\boldsymbol{\theta}_k, \mathcal{D}_\ell) \tag{9}$$

where η is the local learning rate and $\mathcal{L}_\ell$ represents the local objective function of the ℓth device and is computed as

$$\mathcal{L}_\ell\left(\boldsymbol{\phi}_\ell; \mathcal{D}_\ell\right) = \frac{1}{m} ||\mathbf{y}_\ell - \hat{\mathbf{y}}_\ell||_2^2 = \frac{1}{m} ||\mathbf{y}_\ell - f_{\boldsymbol{\phi}_\ell}(\mathbf{X}_\ell)||_2^2 \tag{10}$$

with $f_{\boldsymbol{\phi}_\ell}$ being the personalized model, parameterized by $\boldsymbol{\phi}_\ell$.

During each communication round, both the personalized and generic model parameters are iteratively updated to effectively capture both the personalized dynamics specific to each device and the generic structure among them. In the nth communication round, NF-ML initializes the local parameters ($\boldsymbol{\phi}_{k,n}$) for the kth device with the generic parameters ($\boldsymbol{\theta}_{k,n}$) from it neighbors and then refines them using gradient updates from local data at each device as

$$\boldsymbol{\phi}_{k,n} \leftarrow \boldsymbol{\phi}_{k,n} - \eta \nabla \mathcal{L}_k\left(\boldsymbol{\phi}_{k,n}; b\right) \quad , \quad b \in \mathcal{B}_k \tag{11}$$

where b denotes a single batch, and $\mathcal{B}$ represents the set of batches obtained by splitting the data $\mathcal{D}_k$ into batches of size B. This approach can be readily adapted to batch gradient descent, where the batch size can be equal to the total available data size. Following the local update process, a meta-update step is performed based on the Reptile algorithm using local model parameters

$$\boldsymbol{\theta}_{k,n+1} = (1-\epsilon)\boldsymbol{\theta}_{k,n} + \epsilon \cdot \boldsymbol{\phi}_{k,n}. \tag{12}$$

The updated generic parameters are then shared with neighbors for parameter aggregation, to be utilized as the initialization of local parameters for the next communication round

$$\boldsymbol{\theta}_{k,n+1} = \frac{1}{|\mathcal{N}_k|} \sum_{\ell \in \mathcal{N}_k} \boldsymbol{\theta}_{\ell,n+1}. \tag{13}$$

In (13), the aggregation process involves the generic parameters of the neighboring nodes and the device itself. Alternatively, the aggregation could solely be based on the parameters of the neighboring nodes, excluding the device's parameters. The process in (11)–(13) is repeated until the convergence criterion is satisfied.

B. NF-ML Rapid Adaptation

As discussed above, when new IoT devices join the network at $(t+1)$, graph $\mathcal{G}_t$ expands and transforms into $\mathcal{G}_{t+1} = (\mathcal{K}_{t+1}, \mathcal{E}_{t+1})$ with new device indices added in $\mathcal{K}_t \subseteq \mathcal{K}_{t+1}$. The edge set $\mathcal{E}_{t+1}$, of size $|\mathcal{K}_{t+1}| \times |\mathcal{K}_{t+1}|$, is adjusted accordingly to accommodate the indices of the new devices.

We assume that new devices naturally recognize their neighboring devices based on distance or another device discovery

Algorithm 2 NF-ML Rapid Adaptation

At all test devices $k' \in \mathcal{K}_{t+1}$, $\mathcal{B}_{k'} \leftarrow$ split($\mathcal{D}_{k'}$ into batches of size B)
Get $\mathcal{N}_{k'}$ based on the $\mathcal{E}_{t+1}$
$\boldsymbol{\theta}_{k'} = \frac{1}{|\mathcal{N}_{k'}|} \sum_{\ell \in \mathcal{N}_{k'} \setminus k'} \boldsymbol{\theta}_\ell$
$\tilde{\boldsymbol{\phi}}_{k'}^{(0)} = \boldsymbol{\theta}_{k'}$
for $s = 1$ **to** N_{Local} **do**
  for $j = 1$ to $|\mathcal{B}_{k'}|$ **do**
    $\tilde{\boldsymbol{\phi}}_{k',s}^{(j)} \leftarrow \tilde{\boldsymbol{\phi}}_{k',s}^{(j-1)} - \eta \nabla \mathcal{L}_k\left(\tilde{\boldsymbol{\phi}}_{k',s}^{(j-1)}, b_j\right)$
  end for
end for
Model Evaluation: Performance is assessed by (10)

strategy [40], [41].[1] Upon entry, the new device initializes its localized model parameters by aggregating the corresponding values from its neighboring devices, which allows rapid adaptation to a personalized model with few samples. This aggregation is similar to that in (13) as mentioned below for the new device $k' \in \mathcal{K}_{t+1}$ joining the network

$$\boldsymbol{\theta}_{k'} = \frac{1}{|\mathcal{N}_{k'}|} \sum_{\ell \in \mathcal{N}_{k'} \setminus k'} \boldsymbol{\theta}_\ell. \tag{14}$$

Next, the algorithm enters a fine-tuning phase to obtain a device-specific optimal local model. An overview of this adaptation process in the context of NF-ML is presented in Algorithm 2.

IV. Experiments

In this section, we demonstrate the effectiveness of the proposed NF-ML algorithm through a series of experiments using both synthetic and real-world data sets.

A. Experiment With Synthetic Data

To evaluate the proposed methodology using synthetic data, we utilized a graph based on the caveman graph structure [42]. This graph consists of $K = 24$ devices divided into three clusters ($Q \in \{1, 2, 3\}$) each made of eight devices, as shown in Fig. 1. Connections within each cluster are dense, whereas sparse connections bridge different clusters, reflecting weaker associations between the devices of separate clusters.[2] Additionally, the graph expands as new devices join. In Fig. 1, the blue devices represent the initial devices of the graph at a specific time, whereas the red devices indicate those that joined the network at a later stage.

The kth devices in the graph is associated with $m = 1000$ samples characterized by $d = 64$ features, resulting in an input matrix $\mathbf{X}_k \in \mathbb{R}^{d \times m}$, drawn from a uniform distribution $\mathcal{U}[0, 1)$, while the corresponding output is given by

$$\boldsymbol{y}_k = \mathbf{X}_k^{\mathrm{T}} \boldsymbol{w}_k \tag{15}$$

[1]The design of an efficient and trustworthy method to connect these devices falls outside the scope of this article.

[2]This structure mimics the relationships observed in different friend groups within social networks.

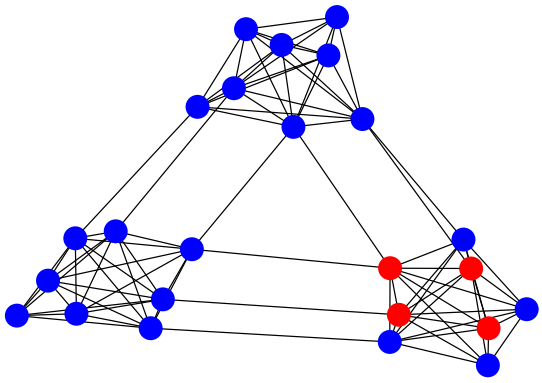

Fig. 1. Synthetic network (red nodes represent the newly joined devices).

where $\boldsymbol{w}_k \in \mathbb{R}^d$ is the weight vector. More specifically, the weight vector combines two sets of weights: one specific to each device [denoted $\boldsymbol{w'}_k$ and sampled from $\mathcal{U}[0, 1)$], and one common to all devices in the same cluster [denoted $\boldsymbol{w}^Q$ and sampled from $\mathcal{U}[0, 1)$]. The two sets of weights are combined according to

$$\boldsymbol{w}_k = \boldsymbol{w}^Q + 0.1\boldsymbol{w'}_k. \tag{16}$$

After the output is computed for each device as per (15), it undergoes the following nonlinear transformation:

$$\boldsymbol{y}_k = \boldsymbol{y}_k^{1.5} + 3\boldsymbol{y}_k. \tag{17}$$

The primary objective of this experiment is to estimate the device-specific weights to achieve the NF-ML objective in (8) and exploit the input–output pairs available at each device. To facilitate parameter sharing and aggregation among neighboring nodes, each device estimates its specific weights using a simple neural network.[3] In the context of NF-ML training, our experimental setup utilizes 20 devices for the training process while keeping four devices and their edges for testing purposes. The local update and meta-update learning rates are set to $\eta = 0.001$ and $\epsilon = 0.9$, respectively. The training process is conducted over 30 communication rounds, employing the mean-squared error (MSE) criterion as a performance indicator. A mini-batch strategy is incorporated for parameter updates, executed with a batch size of 4. Upon completion of the NF-ML training, test devices are subsequently integrated into the graph and fine-tuned over 50 epochs with a learning rate set to 0.001 and utilizing a small number of training data samples.[4]

A series of experiments was designed to evaluate the proposed algorithm. To underscore the effectiveness of meta-learning, we initially assessed the performance of test devices integrated into the network without prior knowledge. This scenario is henceforth referred to as "learning from scratch,"

[3]This neural network has two linear layers. The first takes a 64-size input and outputs 128 neurons. The second layer reduces it to a single output. A ReLU activation is used between the layers.

[4]It is important to note that each scenario hyperparameters are chosen based on the grid search. While no specific optimization strategy has been employed for this process, adopting one could enhance the NF-ML model's performance.

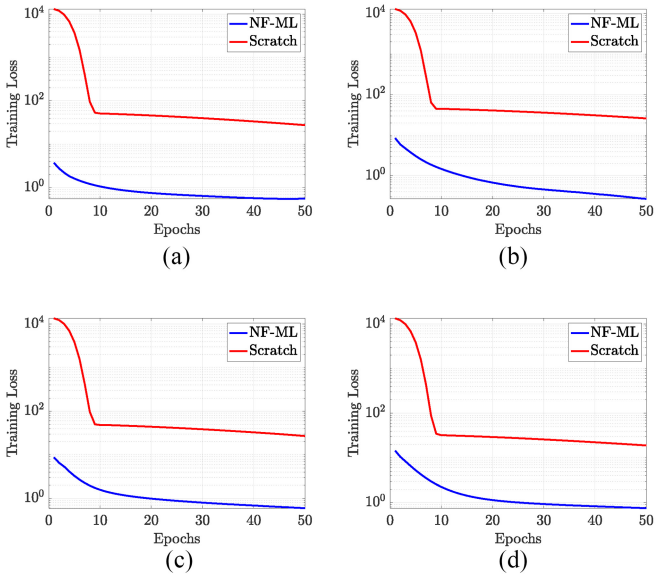

Fig. 2. Training loss on test devices utilizing synthetic data. (a) Test Device 1. (b) Test Device 2. (c) Test Device 3. (d) Test Device 4.

implying that the local model parameters are either randomly initialized or set to zero. Subsequently, comparisons were conducted with established methods that leverage a central server, namely, FedAvg and Personalized FedAvg. In the conventional FedAvg setup, devices learn and transmit their models to a server that aggregates them and disseminates the updated model back to the devices for further refinement, as per (2) and (3). Upon the integration of test devices into the network, they utilize the pre-existing global server model for task execution. In contrast, the Personalized FedAvg approach necessitates an additional step of locally fine-tuning the server model for each test device through an extra 50 epochs.

Fig. 2 illustrates the training loss as a function of the number of epochs for both the proposed NF-ML algorithm and the scenario in which learning was initiated from scratch. In both cases, the model was trained using only 10% of the data from each test device. It is evident that NF-ML, by utilizing parameters from neighboring devices and employing meta-learning, achieves faster convergence. This led to a reduction in the number of required epochs and a substantial decrease in the learning time and energy consumption of newly integrated devices. This significant performance advantage was consistently maintained across all the test devices for up to 50 epochs and extended beyond this range.

Building upon the previous discussion, Fig. 3 presents a direct comparison between the two device integration approaches, focusing specifically on their predictive performance. The results clearly show that the model trained from scratch struggles to accurately predict the test data, particularly during peaks or sudden fluctuations. In contrast, the NF-ML model demonstrates superior prediction performance, highlighting the agility of NF-ML in adapting to new tasks even under the constraints of limited data availability.

In the subsequent analysis, the focus was on investigating the influence of the training data set size on the MSE of the predictions. To this end, the models were trained on the test

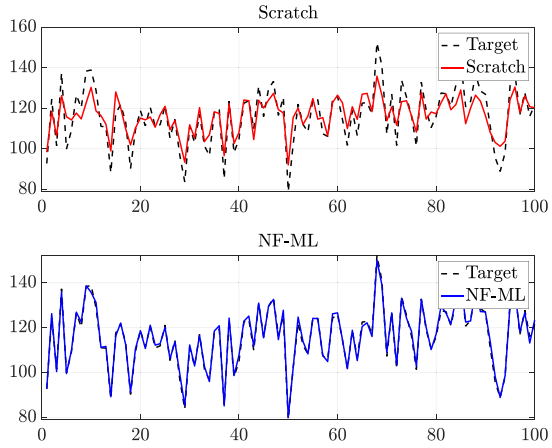

Fig. 3. Prediction for Test Device 3 using synthetic data.

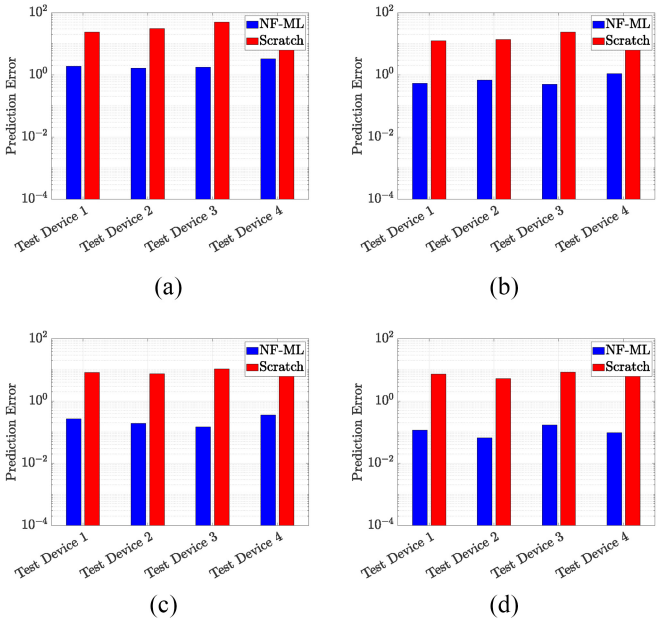

Fig. 4. Prediction loss with varying percentage of training data on test devices using synthetic data. (a) 10% training data. (b) 20% training data. (c) 30% training data. (d) 40% training data.

devices using training data sets of varying sizes. Fig. 4 shows the prediction losses for each test device as the size of the training data increased from 10% to 40%. As expected, larger training data sets are associated with a decrease in prediction losses. This effect is particularly noticeable in the NF-ML models, where the prediction loss significantly decreases from approximately 10^0 to 10^{-1} when the training size increases from 10% to 40%. Conversely, models trained from scratch exhibit only a marginal improvement, with prediction losses generally remaining at approximately 10^1.

Fig. 5 compares the proposed NF-ML algorithm with the approach when new devices are trained from scratch, FedAvg, and Personalized FedAvg. In this comparison, NF-ML outperforms the others across all newly integrated test devices in the graph, as evidenced by its lower prediction loss on the test data. This test set comprises 100 samples, and for

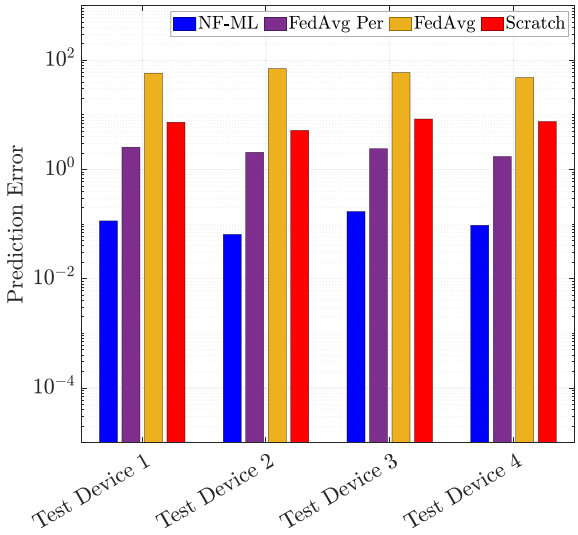

Fig. 5. Comparative analysis of prediction loss on test devices using synthetic data.

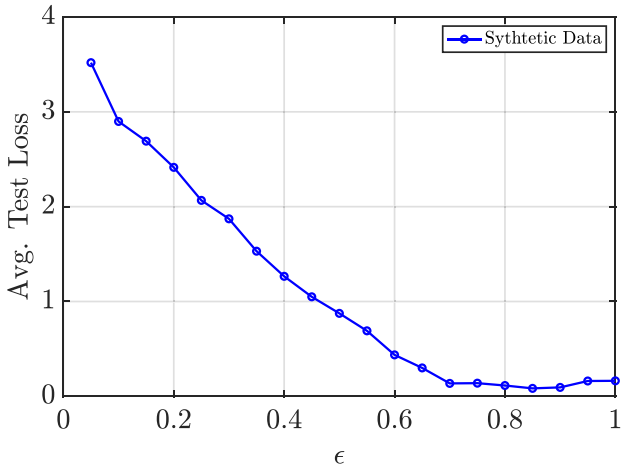

Fig. 6. Impact of the meta-update step size (ϵ) in the scenario with synthetic data.

fine-tuning, each model, including NF-ML and Personalized FedAvg, utilized 400 samples, the same number used for training from scratch. Although Personalized FedAvg shows improvements over alternative state-of-the-art models, it still does not achieve the efficiency of the proposed NF-ML. In addition to its superior performance, a key feature of NF-ML is its decentralized functionality, which obviates the need for a centralized server.

Fig. 6 shows the impact of the meta-update step size (ϵ) on test performance using synthetic data by fine-tuning the test devices over 30 epochs after the NF-ML training is complete. The average loss for the test devices decreases with ϵ, reaching optimal performance when $\epsilon \in (0.8, 0.9)$.

B. Experiments With Real-World Data (Temperature)

To assess the effectiveness of our proposed algorithm using real-world data, we utilized hourly temperature readings from 55 land stations in the Large Molène area for January 2014, sourced from the National Meteorological and Climatic Service database [43]. We constructed an undirected graph linking each sensor to its four nearest neighbors and a visual representation of this graph is shown in Fig. 7. To simulate a growing graph, the sensors were divided into two sets. Initially, the graph $\mathcal{G}_t$ at time t consists of sensors marked in blue, and then, at time $t+1$, new sensors marked in red are integrated to form $\mathcal{G}_{t+1}$. Of these, 37 sensors that provided complete records for the period are included in our study.[5] In this configuration, each sensor functions as a device within the FL framework.

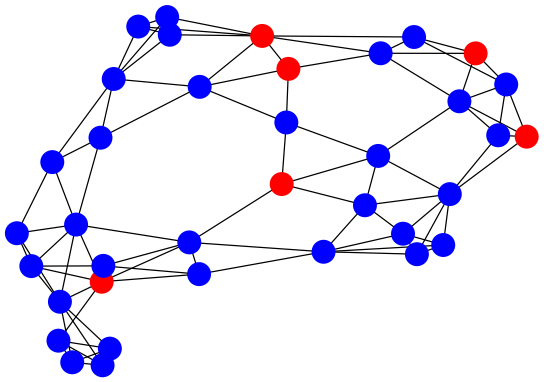

Fig. 7. Temperature-sensor network (red nodes represent the newly joined devices).

TABLE I
MODEL FOR THE EXPERIMENT WITH REAL-WORLD DATA

Layer Type	Configuration
Input	10
Fully Connected (Linear)	16 neurons, followed by ReLU activation
Output (Linear)	1 neuron

In this experiment, each device was tasked with predicting the temperature at the next timestamp using data from the previous ten timestamps. To facilitate this task, each device is equipped with its own neural network model, as neural networks can effectively extract the relations in an ordered set of continuous points [44]. The detailed specifications of which are listed in Table I. The architecture of the network includes an input layer, a hidden layer, and an output layer, with nonlinearity introduced via the ReLU activation function. For data preprocessing, we employ min–max normalization on the data at each test device using the scikit-learn library.[6] The normalization transformation, fitted to the training data, was subsequently applied to the test data to ensure consistency in the data processing.

We trained our algorithm on real-world measurements using 31 devices, while six devices and their associated edges were reserved for testing. We set the local learning and meta-learning rates to $\eta = 0.001$ and $\epsilon = 0.9$, respectively, and used MSE as the loss metric. We employed the Adam optimizer and conducted 400 communication rounds to determine the parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\phi}_k$ for each participating device.

[5]Data Source: Molene-Dataset available at https://github.com/bgirault-usc/Molene-Dataset.

[6]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing

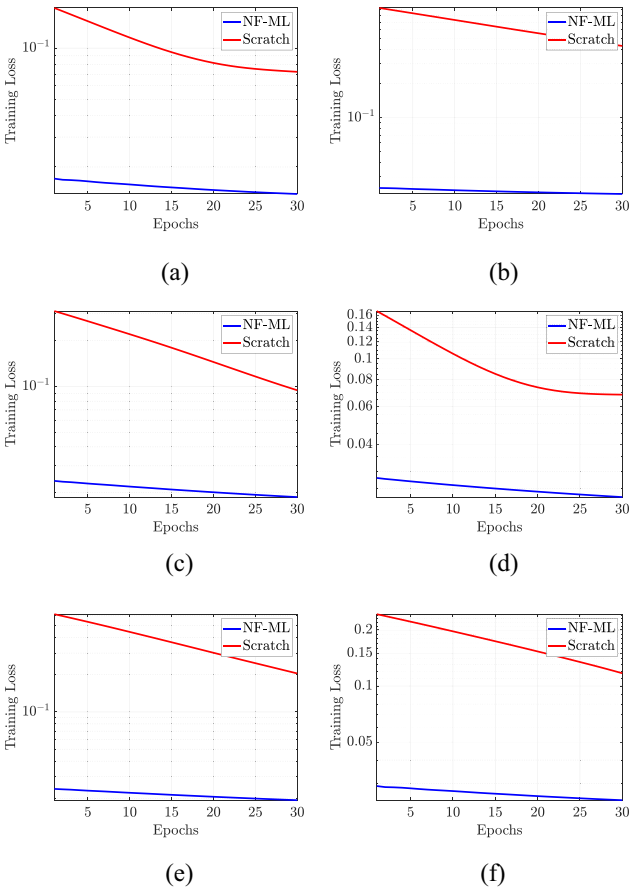

Fig. 8. Training loss on test devices utilizing real-world (temperature) data. (a) Test Device 1. (b) Test Device 2. (c) Test Device 3. (d) Test Device 4. (e) Test Device 5. (f) Test Device 6.

We employ approximately 10% of the total sensor data for NF-ML rapid adaptation, equating to roughly three days of hourly temperature readings, using a mini-batch size of 64 and a learning rate of 0.001.

In Fig. 8, we compare our NF-ML-based methodology with model training from scratch on newly integrated devices. The NF-ML model demonstrates a significantly faster convergence rate, highlighting the enhanced performance of our algorithm on the test devices. Throughout the 30-epoch training period, the approach with new models trained from scratch consistently showed higher training losses than those trained using the NF-ML methodology. The proposed methodology requires fewer epochs to converge, particularly relevant with limited training data samples, such as 10%. This implies savings in terms of computational resources, convergence time, and amount of training data.

Fig. 9 compares our proposed model and models trained from scratch, emphasizing the prediction error in the test devices. Our proposed methodology can accurately forecast most testing samples, except for a few dips in the test data. This limitation may arise from the model being fine-tuned on only 10% of the training data, where the min–max scaling, learned from this limited subset, might not fully represent the entire data range. Despite this, our proposed NF-ML rapid

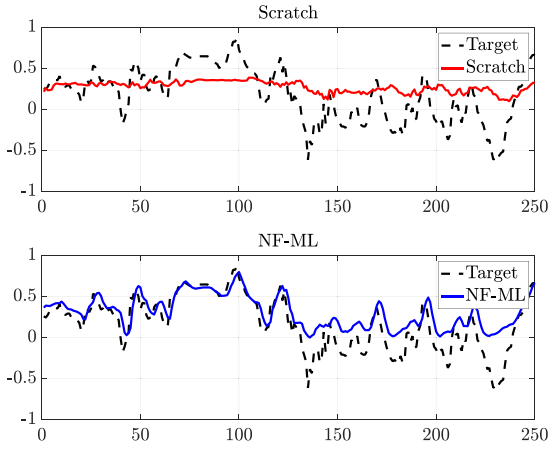

Fig. 9. Prediction for Test Device 2 using real-world (temperature) data.

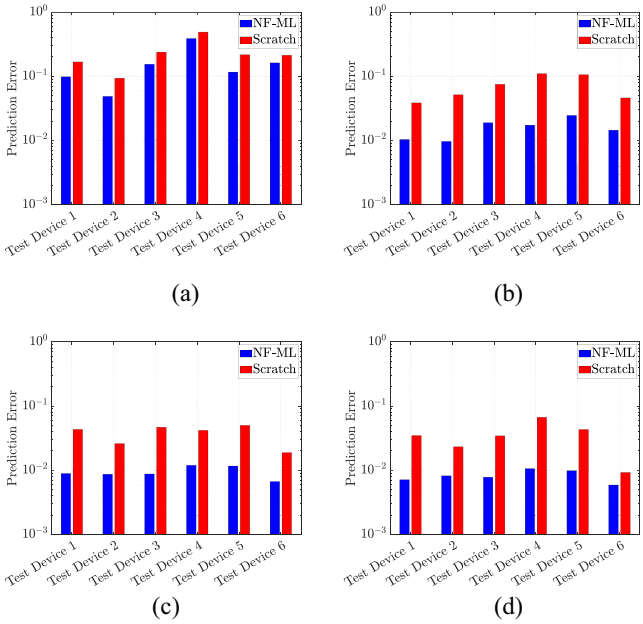

Fig. 10. Prediction loss with varying percentage of training data on test devices using real-world (temperature) data. (a) 10% training data. (b) 20% training data. (c) 30% training data. (d) 40% training data.

adaptation achieves good results using only 10% of the total training data compared to the case when new models are trained from scratch, which significantly struggles to capture the core data trends across all testing samples.

Fig. 10 demonstrates the improvement in the predictive accuracy with the size of the training data (increasing from 10% to 40%). The NF-ML algorithm consistently outperforms models trained from scratch across these varying training data sizes. For instance, with Test Device 2 and Test Device 4, we observe a rapid reduction in loss when the training data expands from 10% to 20%. Similarly, Test Device 3 and Test Device 5 show a marked decline in loss when the training data are increased from 20% to 30% with NF-ML.

Fig. 11 compares our NF-ML method, FedAvg, and its personalized variant in real-world scenarios, focusing on the

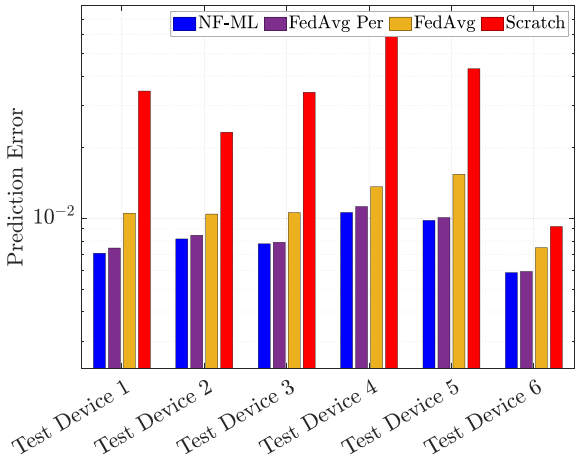

Fig. 11. Comparative analysis of prediction loss on test devices using real-world (temperature) data.

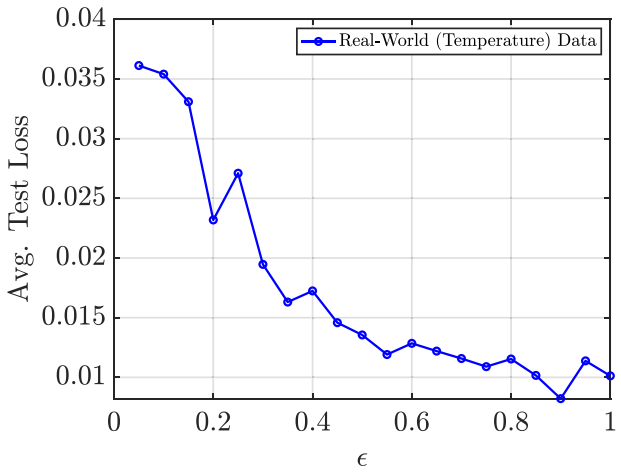

Fig. 12. Impact of the meta-update step size (ϵ) in the scenario with real-world (temperature) data.

integration of new devices into the network. This analysis used 13 days of data to fine-tune the NF-ML and PFL and training models from scratch. Additionally, ten days of data were used for testing. NF-ML stands out in this scenario, consistently achieving the lowest prediction loss across all test devices, demonstrating its superior performance. Although the Personalized FedAvg approach demonstrated improved performance on certain test devices, such as Test Device 3 and Test Device 6, NF-ML maintained its superiority and consistently outperformed the other methods due to its distributed nature.

Fig. 12 shows the impact of the meta-update step size (ϵ) on test performance using temperature data by fine-tuning the test devices over 50 epochs. The average loss for the test devices decreases with ϵ, reaching optimal performance when $\epsilon \in (0.85, 0.95)$.

C. Experiments With Real-World Data (Images)

To assess the efficacy of our proposed algorithm in diverse scenarios, we considered the publicly available MNIST digit

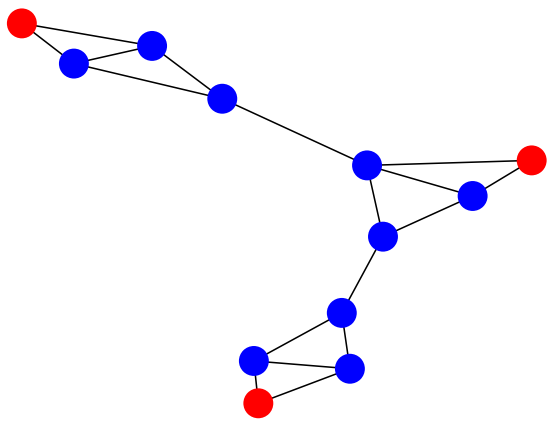

Fig. 13. Classification-device network (red nodes represent the newly joined devices).

classification data set. The MNIST data set, comprising images of handwritten digits, was divided among 12 nodes representing the devices in our extended model. Each node was designated specific label ranges to mimic real-world distributed data scenarios: the first four nodes handled digits labeled from 0 to 4, the subsequent four nodes managed labels 4–7, and the final set of four nodes processed labels 7–9. This distribution ensures that each cluster of nodes specializes in a segment of the overall data set. The underlying graph topology, shown in Fig. 13, is based on the caveman graph structure, akin to groups of people, where persons within a group are mostly connected but have few connections to other groups.

In this experiment, we partitioned devices into training and testing groups, with nine devices as training units shown in blue and three as testing units depicted in red, which joined the network later with limited resources. Each device operates a small convolutional neural network (CNN) to perform the classification task. The CNN architecture includes: a first convolutional layer with 32 filters of size 3×3, using a stride of 1 and padding of 1 followed by a max-pooling layer with a size of 2×2 and a stride of 2; a second convolutional layer with 64 filters of the same size, stride, and padding, also followed by a max-pooling layer with the same pool size and stride; a fully connected layer to flatten the pooled output and link it to 128 neurons; and finally, an output layer of neurons corresponding to the MNIST data set's ten-digit classes. For the NF-ML training, we set the learning rate $\eta = 0.00005$, the meta-update rate $\epsilon = 0.9$, and the batch size of 256, utilizing the cross-entropy criterion and the Adam optimizer. This setup was implemented over 50 communication rounds to refine the generic $\boldsymbol{\theta}_k$ and local $\boldsymbol{\phi}_k$ for each participating device in the training. Also in this case, we conducted a series of simulations to evaluate the efficiency of our proposed algorithm under varying conditions, specifically differing data sizes and numbers of epochs.

In Fig. 14, we present a comparison between our NF-ML-based approach and the method of training models from scratch on newly integrated test devices. The NF-ML model demonstrates significantly faster convergence, achieving substantial training accuracy after a single epoch, even with only

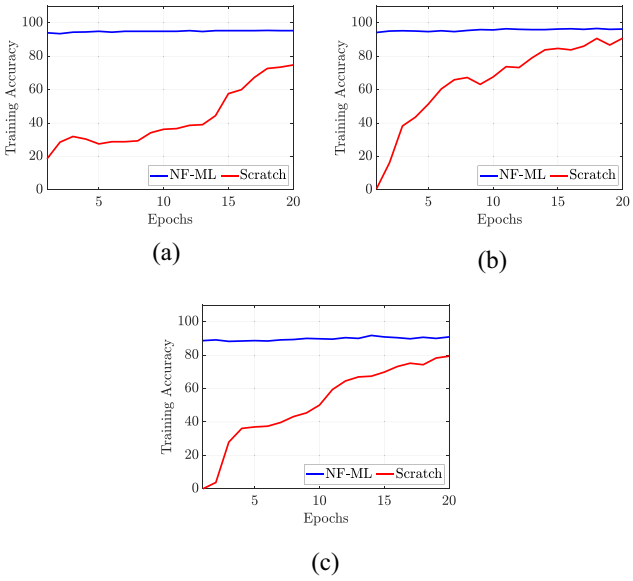

Fig. 14. Training loss on test devices utilizing real-world (images) data. (a) Test Device 1. (b) Test Device 2. (c) Test Device 3.

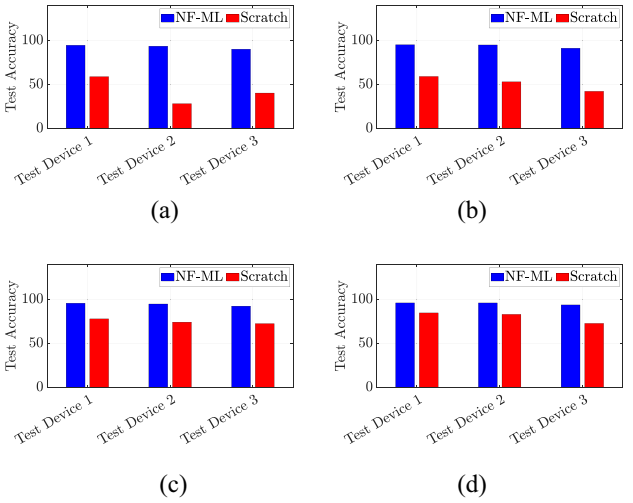

Fig. 15. Test accuracy with varying percentage of training data on test devices using real-world (images) data. (a) 10% training data. (b) 20% training data. (c) 30% training data. (d) 40% training data.

10% data utilization. In stark contrast, the approach of training from scratch struggles to reach similar performance levels even after 20 epochs, highlighting the enhanced efficiency of our algorithm on the test devices.

Fig. 15 illustrates the comparative performance of our proposed NF-ML algorithm against the traditional approach of training models from scratch across varying data set sizes ranging from 10% to 40% of the available data to the device with increments of 10%. This comparison is conducted over a short span of only 5 epochs to emphasize the rapid learning capabilities of newly integrated devices. The results show that our algorithm consistently maintains high accuracy levels (approximately 90% across all data set sizes), significantly outperforming the traditional training approach. Notably, with just 10% of the data, our proposed algorithm achieves an

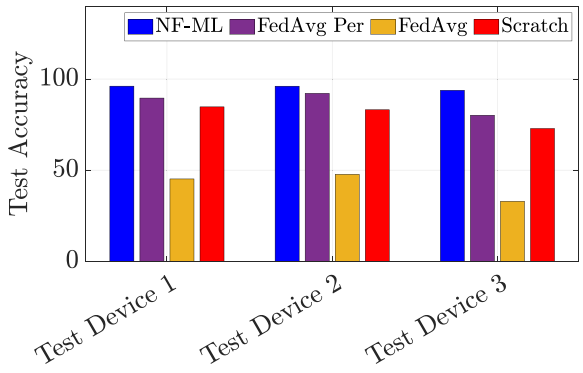

Fig. 16. Comparative analysis of prediction loss on test devices using real-world (images) data.

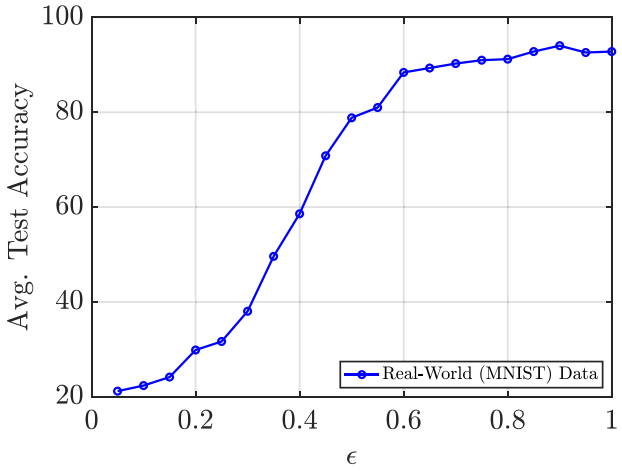

Fig. 17. Impact of the meta-update step size (ϵ) in the scenario with real-world (images) data.

accuracy that the model trained from scratch fails to reach even when provided with 40% of the total available data. For testing purposes, we use 10% of the data available to the device.

Fig. 16 compares the performance of our proposed NF-ML algorithm against established methods, such as FedAvg and its personalized variant, specifically for the MNIST classification task. Newly integrated devices were allowed to use 40% of the available data, with the training restricted to just 5 epochs. This setup was used to fine-tune the NF-ML and PFL algorithms and train a model from scratch. We reserved 10% of the device data for testing. The results show that our proposed NF-ML algorithm significantly outperforms the other algorithms in terms of testing accuracy across all test devices.

Fig. 17 shows the impact of the meta-update step size (ϵ) on test performance using MNIST data by fine-tuning the test devices over 3 epochs only. The average accuracy for the test devices increases with ϵ, reaching optimal performance when $\epsilon \in (0.8, 1)$.

V. Conclusion

This article introduces a fully distributed NF-ML algorithm for expanding networks in a federated environment. With

this approach, new devices can join the network and rapidly adapt to the underlying task by leveraging parameters from neighboring devices and fine-tuning them with minimal data over a few epochs, thus ensuring efficiency and reduced computational costs. The effectiveness of the proposed algorithm is demonstrated using synthetic data from a connected caveman graph, where each device performs regression; real-world data from temperature sensors for predicting future values; and real-world handwritten digit for classification tasks. Future works will focus on adversarial settings (incorporating various attack scenarios) and examining related robustness of the proposed algorithm.

REFERENCES

[1] Z. Du, C. Wu, T. Yoshinaga, K.-L. A. Yau, Y. Ji, and J. Li, "Federated learning for vehicular Internet of Things: Recent advances and open issues," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 45–61, May 2020.

[2] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, "Federated learning for the Internet of Things: Applications, challenges, and opportunities," *IEEE Internet Things Mag.*, vol. 5, no. 1, pp. 24–29, Mar. 2022.

[3] *General Data Protection Regulation (GDPR)*, Intersoft Consult, Inc., Hamburg, Germany, 2018.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[5] 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.

[6] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*.

[7] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2020.

[8] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5476–5497, Apr. 2020.

[9] A. M. Elbir and S. Coleri, "Federated learning for channel estimation in conventional and RIS-assisted massive MIMO," *IEEE Trans. Wireless Commun.*, vol. 21, no. 6, pp. 4255–4268, Jun. 2022.

[10] S. K. Pye and H. Yu, "Personalised federated learning: A combinational approach," 2021, *arXiv:2108.09618*.

[11] V. C. Gogineni, S. Werner, Y.-F. Huang, and A. Kuh, "Communication-efficient online federated learning strategies for kernel regression," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4531–4544, Mar. 2023.

[12] A. Khan et al., "A distributed and elastic aggregation service for scalable federated learning systems," 2022, *arXiv:2204.07767*.

[13] V. C. Gogineni, S. Werner, F. Gauthier, Y.-F. Huang, and A. Kuh, "Personalized online federated learning for IoT/CPS: Challenges and future directions," *IEEE Internet Things Mag.*, vol. 5, no. 4, pp. 78–84, Dec. 2022.

[14] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[15] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM. Comput. Surv.*, vol. 54, no. 6, pp. 1–36, 2021.

[16] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural. Netw. Learn. Syst.*, vol. 34, no. 12, pp. 9587–9603, Dec. 2023.

[17] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with Moreau envelopes," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 21394–21405.

[18] Y. Huang et al., "Personalized cross-silo federated learning on non-IID data," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 7865–7873.

[19] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive Personalized federated learning," 2020, *arXiv:2003.13461*.

[20] S. Chen, G. Long, T. Shen, T. Zhou, and J. Jiang, "Spatial-temporal prompt learning for federated weather forecasting," 2023, *arXiv:2305.14244*.

[21] F. Chen, G. Long, Z. Wu, T. Zhou, and J. Jiang, "Personalized federated learning with a graph," in *Proc. Int. Joint Conf. Artif. Intell.*, 2022, pp. 2575–2582.

[22] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.

[23] F. Gauthier, V. C. Gogineni, S. Werner, Y.-F. Huang, and A. Kuh, "Personalized graph federated learning with differential privacy," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 9, pp. 736–749, Oct. 2023.

[24] E. Rizk and A. H. Sayed, "A graph federated architecture with privacy preserving learning," in *Proc. IEEE Int. Workshop Signal Process. Adv. Wireless Commun.*, 2021, pp. 131–135.

[25] V. C. Gogineni, S. Werner, Y.-F. Huang, and A. Kuh, "Decentralized graph federated multitask learning for streaming data," in *Proc. 56th Annu. Conf. Inf. Sci. Syst.*, 2022, pp. 101–106.

[26] L. Yuan, Z. Wang, L. Sun, S. Y. Philip, and C. G. Brinton, "Decentralized federated learning: A survey and perspective," *IEEE Internet Things J.*, early access, May 30, 2024, doi: 10.1109/JIOT.2024.3407584.

[27] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[28] A. S. Berahas, R. Bollapragada, N. S. Keskar, and E. Wei, "Balancing communication and computation in distributed optimization," *IEEE Trans. Autom. Control*, vol. 64, no. 8, pp. 3141–3155, Aug. 2019.

[29] A. S. Berahas, R. Bollapragada, and E. Wei, "On the convergence of nested decentralized gradient methods with multiple consensus and gradient steps," *IEEE Trans. Signal Process.*, vol. 69, pp. 4192–4203, Jul. 2021.

[30] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[31] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*.

[32] J. Vanschoren, "Meta-learning: A survey," 2018, *arXiv:1810.03548*.

[33] S. Zhang, F. Ye, B. Wang, and T. G. Habetler, "Few-shot bearing fault diagnosis based on model-agnostic meta-learning," *IEEE Trans. Ind. Appl.*, vol. 57, no. 5, pp. 4754–4764, Sep./Oct. 2021.

[34] Y. Bengio, S. Bengio, and J. Cloutier, "Learning a synaptic learning rule," in *Proc. Int. Joint Conf. Neural. Netw.*, vol. 2, 1991, p. 969.

[35] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in *Proc. 30th Adv. Neural. Inf. Process. Syst.*, vol. 29, 2016, pp. 3988–3996.

[36] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–11.

[37] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1842–1850.

[38] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. ICML Deep Learn. Workshop*, vol. 2, 2015, pp. 1–8.

[39] E. T. M. Beltrán et al., "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 4, pp. 2983–3013, 4th Quart., 2023.

[40] L. Dai and X. Wei, "Distributed machine learning based downlink channel estimation for RIS assisted wireless communications," *IEEE Trans. Commun.*, vol. 70, no. 7, pp. 4900–4909, Jul. 2022.

[41] P. C. Ccori, L. C. C. De Biase, M. K. Zuffo, and F. S. C. da Silva, "Device discovery strategies for the IoT," in *Proc. IEEE Int. Symp. Consum. Electron.*, 2016, pp. 97–98.

[42] U. Kang and C. Faloutsos, "Beyond 'caveman communities': Hubs and spokes for graph compression and mining," in *Proc. IEEE Int. Conf. Data Min.*, 2011, pp. 300–309.

[43] B. Girault, "Stationary graph signals using an isometric graph translation," in *Proc. Eur. Signal Process. Conf.*, 2015, pp. 1516–1520.

[44] A. Borovykh, C. W. Oosterlee, and S. M. Bohté, "Generalization in fully-connected neural networks for time series forecasting," *J. Comput. Sci.*, vol. 36, Sep. 2019, Art. no. 101020.

Muhammad Asaad Cheema (Member, IEEE) received the master's degree in electrical engineering from the National University of Sciences and Technology (NUST), Islamabad, Pakistan, in 2020. He is currently pursuing the Ph.D. degree with the SPIN Group, Norwegian University of Science and Technology, Trondheim, Norway.

During his M.S. at NUST, he had the opportunity to spend one semester on the ERASMUS+ mobility program at Frederick University, Nicosia, Cyprus, from October 2019 to January 2020. His research interests include graph signal processing, machine learning, and Internet of Things.

Vinay Chakravarthi Gogineni (Senior Member, IEEE) received the bachelor's degree in electronics and communication engineering from Jawaharlal Nehru Technological University, Anantapur, India, in 2005, the master's degree in communication engineering from Vellore Institute of Technology, Vellore, India, in 2008, and the Ph.D. degree in electronics and electrical communication engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 2019.

He is currently an Assistant Professor with the SDU Applied AI and Data Science, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Odense, Denmark. Prior to this, he worked as a Postdoctoral Research Fellow with the Norwegian University of Science and Technology, Trondheim, Norway, and Simula, Oslo, Norway. From 2008 to 2011, he was with a couple of MNCs in India. His research interests include machine learning, distributed machine learning, geometric deep learning, machine unlearning, and their application in healthcare, Industrial Internet of Things, and fusion energy.

Dr. Gogineni was a recipient of the ERCIM Alain Bensoussan Fellowship in 2019 and the Best Paper Award at APSIPA ASC-2021, Tokyo, Japan. He is a member of the editorial board for the IEEE SENSORS JOURNAL.

Pierluigi Salvo Rossi (Senior Member, IEEE) was born in Naples, Italy, in 1977. He received the Dr.Eng. degree (summa cum laude) in telecommunications engineering and the Ph.D. degree in computer engineering from the University of Naples "Federico II," Naples, in 2002 and 2005, respectively.

He is currently a Full Professor and the Deputy Head of the Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), Trondheim, Norway. He is also a part-time Senior Research Scientist with the Department of Gas Technology, SINTEF Energy Research, Trondheim. Previously, he worked with the University of Naples "Federico II"; the Second University of Naples, Caserta, Italy; NTNU; and Kongsberg Digital AS, Asker, Norway. He held visiting appointments with Drexel University, Philadelphia, PA, USA; Lund University, Lund, Sweden; NTNU; and Uppsala University, Uppsala, Sweden. His research interests fall within the areas of communication theory, data fusion, machine learning, and signal processing.

Prof. Salvo Rossi was awarded as an Exemplary Senior Editor of the IEEE Communications Letters in 2018. He is (or has been) on the editorial board of the IEEE SENSORS JOURNAL, IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY, the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS, the IEEE COMMUNICATIONS LETTERS, and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS.

Stefan Werner (Fellow, IEEE) received the M.Sc. degree in electrical engineering from the Royal Institute of Technology, Stockholm, Sweden, in 1998, and the D.Sc. degree (Hons.) in electrical engineering from the Signal Processing Laboratory, Helsinki University of Technology, Espoo, Finland, in 2002.

He is a Professor with the Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, the Director of IoT@NTNU, and an Adjunct Professor with Aalto University, Espoo. He was a Visiting Melchor Professor with the University of Notre Dame, Notre Dame, IN, USA, in Summer 2019 and an Adjunct Senior Research Fellow with the Institute for Telecommunications Research, University of South Australia, Adelaide, SA, Australia, from 2014 to 2020. He held an Academy Research Fellowship, funded by the Academy of Finland, from 2009 to 2014. His research interests include adaptive and statistical signal processing, wireless communications, and security and privacy in cyber–physical systems.

Prof. Werner is a member of the editorial board for the *EURASIP Journal on Advances in Signal Processing* and the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS.