IEEE ComSoc IEEE Communications Society | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS Connecting the Mobile World
IEEE Transactions on Machine Learning in Communications and Networking

Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/TMLCN.2022.1234567

DFL: Dynamic Federated Split Learning in Heterogeneous IoT

Eric Samikwa, Antonio Di Maio, Torsten Braun

[1]Institute of Computer Science, University of Bern, Switzerland
Email: eric.samikwa@unibe.ch, antonio.dimaio@unibe.ch, torsten.braun@unibe.ch

Corresponding author: Eric Samikwa (email: eric.samikwa@unibe.ch).

ABSTRACT Federated Learning (FL) in edge Internet of Things (IoT) environments is challenging due to the heterogeneous nature of the learning environment, mainly embodied in two aspects. Firstly, the statistically heterogeneous data, usually non-independent identically distributed (non-IID), from geographically distributed clients can deteriorate the FL training accuracy. Secondly, the heterogeneous computing and communication resources in IoT devices often result in unstable training processes that slow down the training of a global model and affect energy consumption. Most existing solutions address only the unilateral side of the heterogeneity issue but neglect the joint problem of resources and data heterogeneity for the resource-constrained IoT. In this article, we propose Dynamic Federated split Learning (DFL) to address the joint problem of data and resource heterogeneity for distributed training in IoT. DFL enhances training efficiency in heterogeneous dynamic IoT through resource-aware split computing of deep neural networks and dynamic clustering of training participants based on the similarity of their sub-model layers. We evaluate DFL on a real testbed comprising heterogeneous IoT devices using two widely-adopted datasets, in various non-IID settings. Results show that DFL improves training performance in terms of training time by up to 48%, accuracy by up to 32%, and energy consumption by up to 62.8% compared to classic FL and Federated Split Learning in scenarios with both data and resource heterogeneity.

INDEX TERMS Federated Learning, Split Learning, Internet of Things, Resource and Data Heterogeneity.

I. INTRODUCTION

With the rapid development of the Internet of Things (IoT) technology, an enormous amount of data is being generated by various edge devices [1]. These devices can range from simple sensors to complex devices with computing capabilities [2]. The generated data can be utilized for training Machine Learning (ML) models to provide insights and make predictions, leading to better decision-making processes and intelligent applications [3]. Recently, we have witnessed the rise of new distributed learning paradigms, such as Federated Learning (FL), which protects user privacy by allowing remote devices to train the model collaboratively without sharing privacy-sensitive data over the network [4].

FL enables the training of ML models by aggregating local updates from edge devices instead of centralizing the data [5]. It also helps to save communication bandwidth by eliminating the need to transmit large raw data samples. Each device trains only using its local data to overcome the limitations of the computing power and hardware storage of a single device, allowing training tasks to be executed in parallel. However, despite recent breakthroughs, the deployment of FL still faces many challenges due to the heterogeneous learning environments that significantly limit its performance and hinder its real-world applications [6].

The heterogeneous learning environment is mainly embodied in two aspects. The first one is statistical heterogeneity, usually non-Independent Identically Distributed (non-IID) training data dispersed across participant devices [7], [8]. Studies have shown that learning from a balanced dataset, where the samples are uniformly distributed across classes, is crucial in boosting the model performance [9], [10]. The characteristics of the data in distributed IoT devices are often highly skewed in a non-IID manner, which can be attributed to the distinct geographical locations of the devices, diverse application preferences, and other environmental influences [11]. The data on the devices cannot be audited by the centralized server due to privacy constraints, as doing so would necessitate transmitting sensitive information over the network. Thus, skewed data distribution inherent in IoT scenarios can significantly deteriorate the distributed training

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

performance, leading to suboptimal model accuracy and learning instability [12].

The second critical aspect contributing to the heterogeneous learning environment in IoT systems is the diversity in computing and communication resources across IoT devices. This heterogeneity manifests in various forms, from differences in processing power and memory capacity to variability in network connectivity and bandwidth. The wide spectrum of computational capabilities leads to significant challenges for training a global model. Devices with limited processing power or memory, often referred to as *stragglers*, can significantly slow down the overall training process [13]. Furthermore, devices with poor network connectivity or limited bandwidth face challenges in transmitting data or model updates efficiently. In typical IoT systems, especially those comprising resource-constrained devices, these disparities give rise to hybrid heterogeneous computing clusters [14]. Addressing this heterogeneity is crucial for optimizing both the training process and the energy consumption, ensuring that the distributed learning framework is scalable, efficient, and effective across the diverse landscape of IoT devices.

Previous studies have explored various grouping and clustering strategies to address the challenges of the heterogeneous learning environments [15], [16], [17], [18]. These aimed at creating multiple disjoint clusters based on some pre-defined clustering criteria to group devices with similar or complementary training data or computing resources. Other approaches aim at creating personalized models for various devices to reduce the impact of resource heterogeneity in edge devices [19], [20], [21]. However, most existing studies address only one aspect of system heterogeneity: either the statistical, i.e., non-IID data distribution, or the heterogeneity of computing resources. It is worth noting that the resource heterogeneity among various devices does not necessarily correlate with the distribution of their training data. Furthermore, previous studies have paid less attention to the heterogeneous learning environment in the dynamic distributed resource-constrained IoT [3], [22].

In this paper, we propose Dynamic Federated split Learning (DFL) to address the joint problem of data and resource heterogeneity for distributed training in heterogeneous dynamic IoT. DFL enhances training efficiency in heterogeneous dynamic IoT through resource-aware split computing of deep neural networks and dynamic clustering of training participants based on the similarity of their sub-model layers. Through resource-aware split learning, the allocation of the training tasks to resource-constrained participants is adjusted to align with the varied computational capabilities and the dynamic nature of communication resources of the participating devices. To navigate the complexities of training data heterogeneity without compromising data privacy or necessitating the transmission of raw data over networks, DFL utilizes the layerwise similarity of neural network representations.This approach effectively assesses the similarity between neural network layers, enabling a more tailored and efficient training approach. We envision the adoption of DFL for efficient distributed training in dynamic edge IoT environments with heterogeneous training data and resources. In summary, this paper makes the following contributions:

- We introduce *dynamic federated split learning* for efficient distributed training of deep neural networks in heterogeneous dynamic edge IoT environments.
- We analyze the heterogeneous learning environment in decentralized resource-constrained IoT and model the joint problem for minimizing the effects of heterogeneous training data and both computing and communication resources on distributed model training.
- We propose an iterative solution that enhances the training efficiency through resource-aware split computing of deep neural networks and dynamic clustering of training participants based on the similarity of their sub-model layers without directly accessing data.
- We consider the tradeoffs in minimizing the training time, model loss, and energy consumption on the IoT devices by introducing time-energy-accuracy sensitivity parameters for the optimization that account for the application's constraints in cases of tradeoffs.
- We implement and evaluate the DFL prototype on a real testbed comprising heterogeneous IoT devices and non-IID training data. The evaluation results demonstrate the effectiveness of the proposed DFL scheme.

The rest of the paper is organized as follows. Section II discusses the background and related work in distributed ML techniques, including FL for edge IoT environments. Section III presents the DFL scheme and its components in detail. Section IV evaluates the performance of DFL through experiments, and Section V concludes the paper with a summary of the contributions.

II. BACKGROUND AND RELATED WORK

A. FEDERATED LEARNING

In the ML algorithm [5], model aggregation, Federated Averaging (FedAvg), is carried out on the server where a global model is generated by averaging local model updates. Figure 1 shows an illustration of FL process with two clients, Device A and Device B, involved in the training of a global model. Each device carries out local training to update the weight of its local model, e.g., ω_A for device A. The local training is carried out using the devices' local dataset, e.g., $d_A \in D$ for device A, where D is the overall dataset available on all devices. A global model ω initialized on a server is shared with client devices (1a/b) for Device A and Device B, respectively. Over the entire training of FL, only the model parameters are communicated between clients and the server for global aggregation (2a/b). Therefore, the clients are not required to share their raw training data with the server or with other clients. The local data remains local and confidential, which makes FL a privacy-preserving ML technique. However, in the case of resource-constrained devices, training an entire model is often too computationally

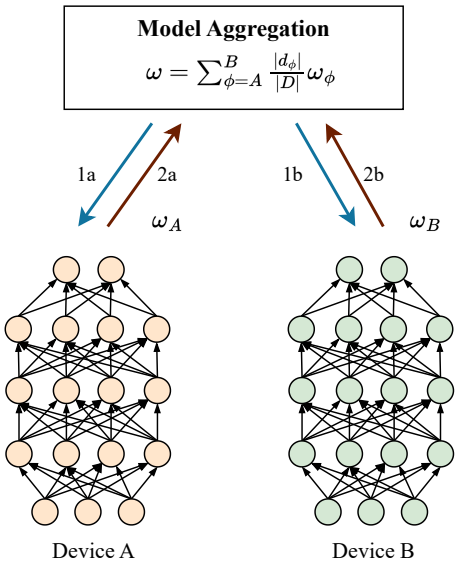

FIGURE 1. **Federated learning representation with two devices.**

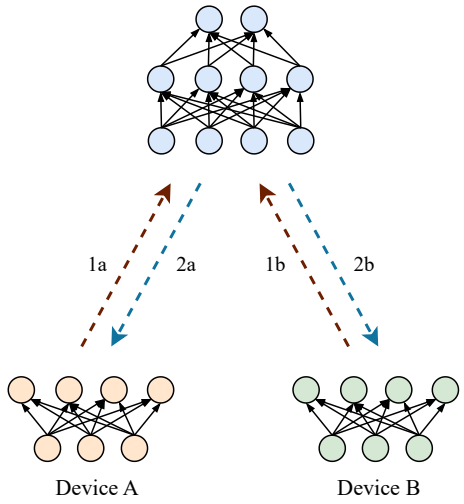

FIGURE 2. **Split learning representation with two client devices.**

demanding and can lead to prolonged training times [19]. FL is known to be less efficient for heterogeneous devices that have different computational capabilities, which is common in IoT systems. Thus, the model computation on resource-constrained devices is a significant bottleneck for the FL training process [23], [22].

Konecny et al. [24] proposed a synchronous FedAvg approach for aggregating the model updates from multiple devices. However, this algorithm assumes that the devices have the same computational capabilities, which is impractical for IoT environments. To address this issue, Gui et al., [16] proposed an approach that partitions the training process among multiple devices based on their computing resources. The approach combines the clustering of training participants with similar training speeds with a resource allocation algorithm to optimize the computational resources. However, their method does not account for the variations in the distribution of the non-IID training data among devices. Furthermore, the classic FL technique does not account for the resource-constrained nature of devices in IoT [22].

On the other hand, for the heterogeneity of the learning environment, the non-IID data distribution in IoT environments is also a significant challenge for FL algorithms [9]. Tu et al. [25] proposed FedDL, a clustering-based approach for grouping devices with similar data distributions during FL training. FedDL utilizes a reference distribution in the form of a common dataset at the server to measure the model affinities of different clients using Kullback–Leibler divergence. Furthermore, Wang et al., [26] proposed a meta-learning-based approach for addressing the non-IID data distribution issue. The approach uses a meta-learner to adapt the model parameters based on the data distribution of each device, enabling the model to generalize better on new devices. While the previous studies have made significant contributions to addressing the challenges of heterogeneous training data, they still face challenges in solving the joint problem of resource heterogeneity and non-IID data distribution since they pay less attention to the heterogeneity of computing and communication resources in the devices.

B. SPLIT LEARNING

The Split Learning (SL), training technique was first presented by Gupta et al. [27]. Figure 2 shows a representation of the SL with two client devices, Device A and Device B, in which the server does not directly access raw data on client devices. From one specific layer, called *split layer* or cut layer, the neural network is split into two sub-networks. On the device side, the model is trained up to the split layer. Then, the activation feature maps of the split layer (*intermediate activations*) are sent to the server, i.e., process indicated by (1a/b) for Device A and Device B, respectively. The server continues training until the last layer of the model. After the training loss is calculated and the gradient is updated, the respective *intermediate gradients* of the split layer is sent to the device so that the gradients are updated on the device (2a/b). An iteration of the above process over the entire client dataset is often referred to as *training round*. The training rounds are repeatedly carried out until the model converges. SL is also referred to as *Split Computing* as a generalization of both model training and inference with model partitioning.

An advantage of SL compared to FL is that each client trains only a portion of the whole neural network, made of just a few layers, which considerably reduces the computational burden on clients [28]. This is essential for the advancement of artificial intelligence-enabled IoT, where ML methods enhance both applications and management of IoT networks [29]. However, classic SL does not consider the computational heterogeneity of devices in IoT systems and the variable network channel conditions that affect data

transmission throughput and computing resources, which, in turn, impact the total training time and energy consumption.

Deep Neural Networks (DNN)s consists of layers with different computational requirements and output data volumes [30]. Splitting the model enables the distribution of computation across multiple devices during model training. To ensure efficient distributed execution, it is crucial to identify the optimal split layer for performance. The choice of the split layer can impact the time needed to complete a training task and the energy consumption of the system involved [31]. As the set of possible model split points is limited, there is a model splitting configuration that minimizes training time, energy consumption, or both. However, it should be noted that the optimal model splitting for training time and energy may not always align due to variations in training time and energy requirements across different devices.

The computing and network resources in IoT systems, which are usually subject to change, can impact both the training time and energy consumption [31]. IoT devices may require less time to transmit data from an intermediate layer in scenarios with high network throughput. However, in low network throughput scenarios, the transmission time can be significantly longer. Moreover, the available computing resources on the IoT devices and servers can change over time as a result of third-party processes.

C. FEDERATED SPLIT LEARNING

FL and SL have recently been blended to take advantage of both, namely Federated Split Learning or SplitFed Learning (SFL) [32], [33]. Figure 3 shows a representation of the SFL with two clients, Device A and Device B. In this approach, all clients compute in parallel with the server, e.g., 1a (intermediate activations) and 2a (intermediate gradients) for Device A, and similarly, 1b and 2b for Device B. The client-side sub-model synchronization can be carried out by aggregating all client-side local networks in a separate parameter server. Initial or aggregated sub-models are shared from the server to the devices (3a/b) for Device A and Device B, respectively. Trained models are then sent from individual devices to the server for aggregation (4a/b).

Thapa et al. [33] introduced Split Federated Learning as a method to achieve parallel training and accelerate device training. Similarly, FedSL [34] combines parallel training and acceleration of device training for sequentially partitioned data, specifically in health applications. Turina et al. [32] proposed a new hybrid Federated Split Learning architecture that combines the benefits of FL and SL in terms of efficiency and privacy. However, these approaches do not consider the computational heterogeneity of target devices and disregard energy consumption in their optimization process. In contrast, Wu et al. [13] proposed FedAdapt, a mechanism for adaptive offloading in federated learning. However, FedAdapt does not account for differences in energy consumption between on-device processing and data transmission. Moreover, these approaches disregard the

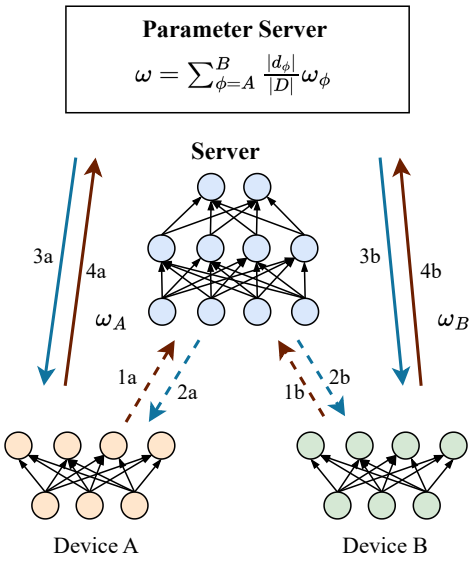

FIGURE 3. **Federated split learning representation with two client devices.**

heterogeneity (non-IID) of the training data on participant devices that affect the model's accuracy.

Mu et al. [35] proposed communication and storage efficient federated split learning (CSE-FSL) that utilizes a single server-side model regardless of the number of clients. CSE-FSL introduces a single-model training strategy that performs model updates using intermediate activations from many clients sequentially, mimicking multi-epoch training. However, CSE-FSL does not consider resource heterogeneity on client devices, which impacts the stability of the training due to stragglers and leads to uneven model updates.

Gui et al., [16] proposed Group Synchronous Parallel (GSP), which uses a density-based algorithm to group edge nodes with similar training speeds together. Their method is also responsible for coordinating the communication of nodes in the group to eliminate stragglers during the training process. However, their proposed solution does not account for the heterogeneous training data available in the distributed IoT devices. Moreover, the similarity of computing resources among different sets of devices may not coincide with the similarity of their training data distributions.

FedMask [19] allows computation and communication efficient federated learning on heterogeneous devices through personalization of the models. However, FedMask does not account for the varying network and computation conditions that affect the training and resource-constrained nature of IoT devices. Furthermore, the process of creating neural network masks can have an effect on the training accuracy.

In our previous work, we introduced *Adaptive REsource-aware Split-learning (ARES)* for efficiency distributed learning in IoT systems [31]. ARES jointly accelerate model training time and minimizes energy consumption in resource-constrained IoT devices through device-targeted split points while accounting for time-varying network throughput and dynamic computing resources. ARES considers the time-

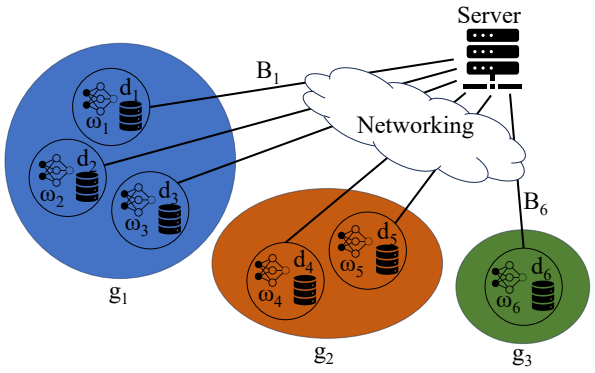

FIGURE 4. **System Model with partition set** $G = \{\{1, 2, 3\}, \{4, 5\}, \{6\}\}$.

varying network conditions and available computing resources to determine the optimal distribution of DNN models during the training. However, ARES focuses only on the unilateral side of the heterogeneity issue (i.e., computing and communication resources) and the dynamic IoT networks.

Our work addresses the challenges of resource heterogeneity and heterogeneous training data that comprise the edge IoT environment. DFL dynamically clusters devices based on the similarities in their sub-model layers, employing resource-aware split computing to mitigate the effects of diverse training resources. To determine the heterogeneity of the training data without transmitting raw data, DFL utilizes the similarity of neural network layers. Our approach can seamlessly integrate client selection strategies to improve efficiency across large-scale device deployments. We envision the adoption of DFL for efficient distributed model training in heterogeneous IoT, including industrial IoT, healthcare IoT, multimedia IoT, and environmental IoT.

III. DYNAMIC FEDERATED LEARNING IN HETEROGENOUS ENVIRONMENTS

A. SYSTEM MODEL

We consider a scenario containing a set Φ of IoT devices with heterogeneous computation and communication resources, which can communicate with a logically centralized *server* through wireless links and wired backbone connectivity (Figure 4). The devices cooperate with the parameter server to perform distributed training of a set of DNN models shared within groups of devices with "similar" data. In particular, the training can be carried out by clusters (groups) of devices to minimize the impact of the heterogeneous data and resource distribution over the system. Clustering is a process of dividing a given set of objects into multiple disjoint clusters based on some pre-defined clustering criteria such that the objects in the same cluster are more similar to each other than those in different clusters. Various studies have shown that grouping devices with similar training data distributions (IID) improves the learning stability and increases the model accuracy [36], [37], [11]. Thus, clustering can potentially minimize the effects of the heterogeneous learning environment in FL with the non-IID training data. On the other hand, clustering devices based on their computing and communication resources have shown to be effective in improving training time and minimizing the effects of stragglers on training [16]. Moreover, with regards to the SL approach, grouping devices based on the distribution of their resources impacts the model split points, which in turn affects both the training time and energy consumption.

We assume that each IoT device $\phi \in \Phi$ hosts a local dataset d_ϕ, with size $|d_\phi| > 0$, that does not change during the training period. Let us define the ϕ *device loss* $F_\phi(\omega)$

TABLE 1. **Symbols Table**

Symbol	Description
Φ	Index set of devices in the distributed learning task
G	Index set of groups in the training task
R	Total number of training rounds in the task
d_ϕ	Dataset on device ϕ
D_g	Combined dataset on a set of devices g
ξ	Minibatch size for the training
N	Total number of layers of the model
L_i	i-th layer of the model
V_s	Total volume of intermediate activations from layer L_s of the model for the current minibatch of size ξ
V'_s	Total volume of intermediate gradients from layer L_{s+1} during backpropagation for the current minibatch of size ξ
k	Training round k
$\gamma_i^{(k)}(\phi)$	Forward propagation time for layer L_i on device ϕ at round k
$\gamma'^{(k)}_i(\phi)$	Backward propagation time for layer L_i on device ϕ at round k
$\gamma_i^{(k)}(C)$	Forward propagation time for layer L_i on the server at round k
$\gamma'^{(k)}_i(C)$	Backward propagation time for layer L_i on the server at round k
ω_i	Weights of layer L_i of a DNN model
ω_g	Aggregated weights for federated group g
$\delta^{(k)}_{s,T}(\phi)$	Training for the k-th training round on the IoT device ϕ when the model is split at layer L_s
$\delta^{(k)}_{s,\omega}(\phi)$	Model placement time for the k-th training round on the IoT device ϕ when the model is split at layer L_s
$B_\phi^{(k)}$	Network throughput (transmitting) at round k for device ϕ
$B'^{(k)}_\phi$	Network throughput (receiving) at round k for device ϕ
$s_\phi^{(k)}$	Split point for device ϕ at round k
s_k	Round k split vector of device-specific split points
$\Delta_s^{(k)}(\phi)$	Training time at round k for device ϕ
$\Delta_k(s_k)$	System-wide training time at round k
$P_{\mathrm{c}}(\phi)$	Device ϕ power consumption during computation
$P_{\mathrm{t}}(\phi)$	Device ϕ power consumption during data transmission
$P_{\mathrm{r}}(\phi)$	Device ϕ power consumption during receiving
$E_s^{(k)}(\phi)$	Total energy consumption for device ϕ for round k
β	Application time-energy-accuracy sensitivity coefficient
α	Non-IID degree of the training data determined by Dir(α)

given the model parameters ω as in Equation 1, where $f(\omega, x, y)$ is the *local loss* on the sample $(x, y) \in d_\phi$. The particular definition of the local loss function (e.g., sum of squares or cross entropy [38]) depends on the specific ML task (e.g., classification, regression).

$$F_\phi(\omega) = \frac{1}{|d_\phi|} \sum_{(x,y)\in d_\phi} f(\omega, x, y) \tag{1}$$

We denote $g \subseteq \Phi$ as a subset of devices, and define the *group dataset* D_g of size $|D_g|$ as the union of all local datasets d_ϕ owned by the members of group g, i.e., $D_g = \bigcup_{\phi\in g} d_\phi$. Let us define the device ϕ's *local model* ω_ϕ as the model parameters associated with device ϕ, and assume that all devices $\phi \in g$ share a common model ω_g, i.e., $\forall\phi : \omega_g = \omega_\phi$. We can now define the *group loss* for group g as

$$\mathcal{L}_g(\omega_g) = \sum_{\phi\in g} \frac{|d_\phi|}{|D_g|} F_\phi(\omega_g) \tag{2}$$

Let us define the *partition set* $G = \{g | g \subseteq \Phi \wedge \bigcup_{g\in G} g = \Phi \wedge \forall g_i, g_j \in G : g_i \neq g_j \implies g_i \cap g_j = \emptyset\}$ as a possible set of all disjoint groups of devices covering the whole set Φ, which satisfy specific clustering criteria, i.e., hosting datasets with similar data distributions (i.e., IID datasets). We denote the set of all possible partition sets on Φ as $\mathcal{P}(\Phi)$. Our proposed scheme's goal is to compute, for each group $g \in G$ in the partition set, an optimal model ω_g^* common for all devices member of the same partition $g \in G$, which minimizes the group loss (Equation 3).

$$\omega_g^* = \arg\min_{\omega_g} \mathcal{L}_g(\omega_g), \quad \forall g \in G \tag{3}$$

Finally, we define the *optimal global group loss* as the sum of all group losses for all groups $g \in G$ in the partition set G computed for the optimal model ω_g for all devices in group g (Equation 4).

$$\mathcal{L}(G) = \sum_{g\in G} \mathcal{L}_g(\omega_g^*) \tag{4}$$

We consider a scenario where the optimization (*training*) is performed using *minibatches* [39]. By allowing more frequent updates, minibatch training can lead to quicker convergence and better accuracy than batch training, due to added stochasticity aiding in avoiding local minima. Additionally, compared to single-sample training, it utilizes the computational efficiency of simultaneous processing, enhancing speed and hardware utilization. We assume that the dataset on each IoT device ϕ is partitioned in equal-sized subsets called *minibatches*, containing ξ entries each. The size ξ of each minibatch is defined as a training hyper-parameter and remains constant during training. An *epoch* is the process of training the DNN over all minibatches (i.e., the whole dataset) one time, and we indicate each of them with k. Similarly, we also refer to k as a *training round* during the distributed training process.

We consider FL training, where tasks on resource-constrained participants are adjusted to match their heterogeneous computing capabilities using resource-aware SL [40]. On the other hand, resource-capable participants carry out the normal FL training. Let us denote $\omega_g^{(k)}$ as the group g's common model at the beginning of the k-th epoch. During each epoch k, each device ϕ performs a local training on its local dataset d_ϕ, then sends the updated model to the server, which returns the aggregated model of all group members. In particular, model aggregation for Federated and Split Learning is carried out using weighted averaging, whereas local training is performed using gradient descent with learning rate η (Equation 5).

$$\omega_g^{(k+1)} = \sum_{\phi\in g} \frac{|d_\phi|}{|D_g|} \left[\omega_g^{(k)} - \eta \nabla F_\phi(\omega_g^{(k)})\right] \tag{5}$$

1) Latency-Aware Model Splitting

We assume that every model $\omega_g, \forall g \in G$ shares the same architecture (but not the weights) composed of N consecutive layers, each denoted by L_i, with $i \in \{1, \ldots, N\}$. To facilitate distributed learning between IoT devices and the edge server, the model is partitioned into two separate *sub-models* for each device engaged in the training process. For an IoT device ϕ during any given round k, the first $s_\phi^{(k)}$ layers $\{L_i | 1 \leq i \leq s_\phi^{(k)}, i \in \mathbb{N}\}$ are executed on the IoT device ϕ, while the last $N - s_\phi^{(k)}$ layers $\{L_i | s_\phi^{(k)} + 1 \leq i \leq N, i \in \mathbb{N}\}$ are executed on the edge server. We call $s_\phi^{(k)}$ the model *split point* for the IoT device ϕ at training round k. We assume that during any given training round k the IoT device ϕ can transfer data to the edge server with an average throughput $B_\phi^{(k)}$ and receive data from the edge server with an average throughput $B'^{(k)}_\phi$.

During the training process, for every minibatch with size ξ, the intermediate activations produced by the layer $L_{s_\phi^{(k)}}$ at the split point are transferred to the layer $L_{s_\phi^{(k)}+1}$ on the edge server via wireless communications with throughput $B_\phi^{(k)}$. Conversely, during the backpropagation phase, the intermediate gradients produced by layer $L_{s_\phi^{(k)}+1}$ are sent back to layer $L_{s_\phi^{(k)}}$ on the IoT device through the network with throughput $B'^{(k)}_\phi$. We model the transmission time for intermediate activations from the IoT device ϕ to the edge server for each training round k. Given the volume of activations V_s and the wireless channel throughput $B_\phi^{(k)}$, the transmission time is $\frac{V_s}{B_\phi^{(k)}}$. Similarly, the transmission time for intermediate gradients of volume V'_s for the current minibatch from the edge server to the IoT device ϕ over a wireless channel with bandwidth $B'^{(k)}_\phi$, during training round k is $V'_s / B'^{(k)}_\phi$. Once the server has completed forward propagation for all data entries in the minibatch during training round k, it computes a loss function over all outputs, which takes a time $\Theta^{(k)}$ for each minibatch.

We account for the variability in computing capabilities of the IoT devices and the edge server, which may fluctuate over time due to different computation loads from third-party processes. We define $\gamma_i^{(k)}(\phi)$ and $\gamma'^{(k)}_i(\phi)$ as the time taken by the IoT device ϕ to execute forward and backward propagation, respectively, at layer L_i of the model during training round k for a training sample. We define $\gamma_s^{(k)}(\phi)$ as the time taken by the IoT device ϕ to execute forward and backward propagation on a sub-model with split at layer $L_{s_\phi^{(k)}}$ (i.e., the first s layers of the model) during training round k. For a minibatch with size ξ during training round k on the IoT device ϕ, the device processing time is the sum of the time required to perform forward and backward propagation on all layers between L_1 and L_s deployed on the IoT device ϕ, multiplied by the minibatch size, i.e., $\gamma_s^{(k)}(\phi) = \xi \sum_{i=1}^{s} \left(\gamma_i^{(k)}(\phi) + \gamma'^{(k)}_i(\phi)\right)$. The methodology for determining forward and backward propagation times on the devices is further elaborated in Section III C.

We now define $\gamma_i^{(k)}(C)$ and $\gamma'^{(k)}_i(C)$ as the time taken for forward and backward propagation, respectively, of layer L_i of the model on the edge server during training round k for a single entry of the dataset. Let's denote $\gamma_s^{(k)}(C)$ as the time taken to execute forward and backward propagation of the last $N-s$ layers of the model on the server during training round k. If the last $N-s$ layers of the model run on the edge server, the total time to perform the training task for all entries of a minibatch of size ξ during training round k on the edge server is the sum of the time required to perform forward and backward propagation on all layers between L_{s+1} and L_N deployed on the edge server, times the minibatch size, i.e., $\gamma_s^{(k)}(C) = \xi \sum_{i=s+1}^{N} \left(\gamma_i^{(k)}(C) + \gamma'^{(k)}_i(C)\right)$.

The process of forward propagation, intermediate data transmission, and backward propagation is sequential. Therefore, we can define the time taken to complete the k-th training round on the IoT device ϕ when the model is split at layer L_s, denoted as $\delta_{s,T}^{(k)}(\phi)$.

$$\delta_{s,T}^{(k)}(\phi) = \frac{d_\phi}{\xi}\left(\gamma_s^{(k)}(\phi) + \gamma_s^{(k)}(C) + \frac{V_s}{B_\phi^{(k)}} + \frac{V'_s}{B_\phi^{\prime(k)}} + \Theta^{(k)}\right) \tag{6}$$

This time is the sum of the minibatch-wise quantities defined so far (training times on the IoT device, training time on the edge server, transmission time from IoT device to edge server and back, and loss function computation on the edge server) divided by the minibatch size ξ (to obtain the average training time for each data entry), multiplied by the number d_ϕ of data entries in the dataset on ϕ (Equation 6).

The variables upon which $\delta_{s,T}^{(k)}(\phi)$ depends are either constant or variable depending on the computing and networking resources. The formulation in Equation 6 also describes the scenario where the training is carried out using classical FL, where the round training time is determined by the value of the total local forward and backpropagation time $\gamma_s^{(k)}(\phi)$. In such cases, the forward and backward propagation time on the server $\gamma_s^{(k)}(C) = 0$, the transmission time for intermediate activations from IoT device to edge server $\frac{V_s}{B_\phi^{(k)}} = 0$, and he transmission time for intermediate gradients from the edge server to device $\frac{V'_s}{B_\phi^{\prime(k)}} = 0$.

After each epoch, the model or submodel weights are collected from the IoT devices to the server for aggregation. Then, the aggregated model is transmitted back to the clients for further training rounds. We denote ω_i as the weights of layer L_i in the DNN model. As such, the concatenation of all the layer weights represents the model weights, i.e., $\omega_\phi = (L_i, \cdots, L_N)$, where $|L_i|$ represents the size of the model weights for layer L_i. We define $\delta_{s,\omega}^{(k)}(\phi)$ as the total communication time needed to transmit the weights of the current model layers from the IoT device to the server with throughput $B_\phi^{(k)}$ and receive an updated submodel from the server with throughput $B_\phi^{\prime(k)}$ after aggregation (Equation 7).

$$\delta_{s,\omega}^{(k)}(\phi) = \sum_{i=1}^{s} \frac{|L_i|}{B_\phi^{(k)}} + \sum_{i=1}^{s} \frac{|L_i|}{B_\phi^{\prime(k)}} \tag{7}$$

Thus, $\delta_{s,\omega}^{(k)}(\phi)$ represents the dynamic model placement time cost. Note that the number of layers whose weights are updated back to the IoT device after aggregation can be different than the number of layers whose weights are transmitted to the parameter server for aggregation, as the server may compute a different optimal split point during each epoch's optimization. We can therefore define the total *training time* for a round $\Delta_s^{(k)}(\phi)$ on a device ϕ as the time needed to carry out the classic FL training or the SL training, plus the time needed to exchange model or submodel weights between the server and the IoT device (Equation 8).

$$\Delta_s^{(k)}(\phi) = \delta_{s,T}^{(k)}(\phi) + \delta_{s,\omega}^{(k)}(\phi) \tag{8}$$

2) Energy-Aware Model Splitting

We define the computing power consumption of the IoT device ϕ as $P_{\mathrm{c}}(\phi)$, the power required by the IoT device ϕ to transmit data to the edge server over the wireless channel as $P_{\mathrm{t}}(\phi)$, and the power required to receive data from the edge server as $P_{\mathrm{r}}(\phi)$. We assume that $P_{\mathrm{c}}(\phi)$, $P_{\mathrm{t}}(\phi)$, and $P_{\mathrm{r}}(\phi)$ are device-specific characteristics that do not change over time and that are known, for instance, through a benchmark to be performed offline. In line with previous studies, we assume that the edge server is not constrained by energy considerations, as usually it is powered by a stable and cost-effective power grid. Consequently, we postulate that the energy required to complete the distributed training of a model between IoT devices and the edge server is equivalent to the energy consumed by the IoT devices, resulting in system-wide energy conservation.

It is important to note that the energy required by device ϕ to perform forward and backpropagation for a minibatch at training round k is the product between its computing power consumption $P_{\mathrm{c}}(\phi)$ and the time $\delta_s^{(k)}(\phi)$ needed to

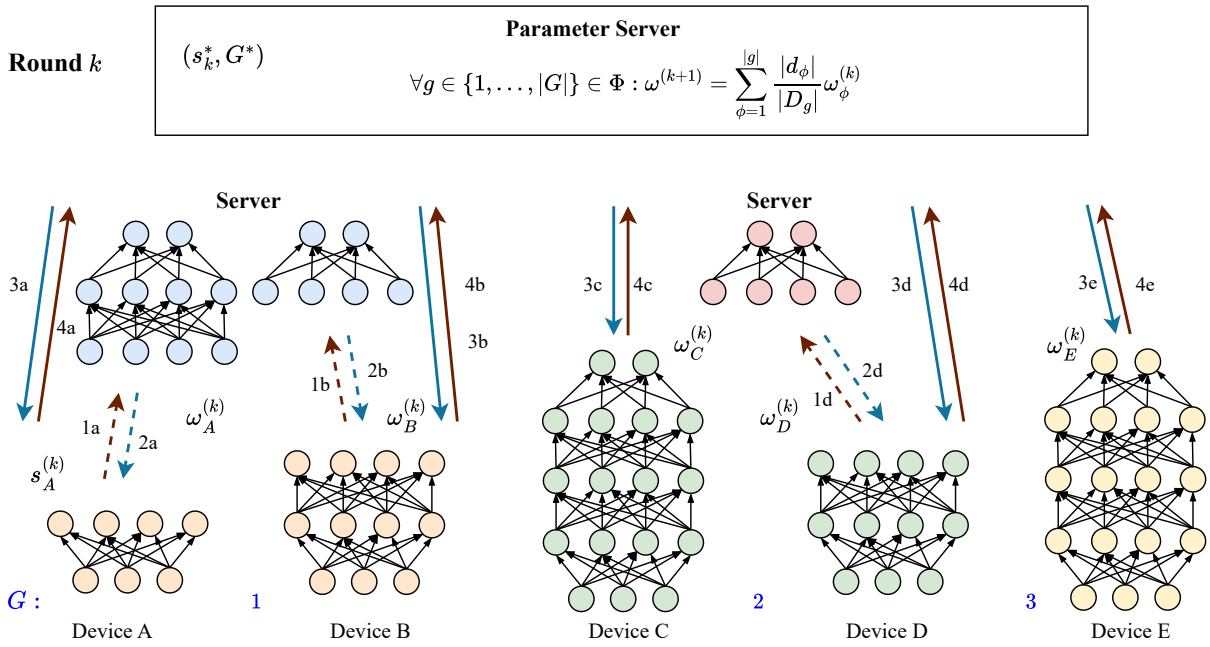

FIGURE 5. **Dynamic federated learning via resource-aware split computing and similarity-based layerwise model aggregation for five client devices.**

compute the activations and updated weights for the first s layers $\{L_i | 1 \leq i \leq s, i \in \mathbb{N}\}$ of the model for all the minibatch of size ξ. The energy for transmitting intermediate activations generated by layer L_s on the IoT device ϕ for all entries of a minibatch of size ξ to the edge server via a wireless channel with upload throughput of $B_\phi^{(k)}$ bit/s is determined as the product of the wireless upload power consumption $P_{\mathrm{t}}(\phi)$ and the time needed to upload a total volume V_s of data to the edge server ($V_s/B_\phi^{(k)}$). Likewise, we determine the energy consumed by IoT device ϕ to receive the intermediate gradients generated by layer L_{s+1} for a minibatch of size ξ, a total volume V'_s, from the edge server through the wireless channel with download throughput of $B'^{(k)}_\phi$ bit/s as the product between the wireless download power consumption $P_{\mathrm{r}}(\phi)$ and the time needed to download a volume V'_s of data to the edge server ($V'_s/B'^{(k)}_\phi$).

We can now define the training energy consumption for a round $E_s^{(k)}(\phi) \in \mathbb{R}$ as the energy consumed by the IoT device ϕ during the whole k-th training round of a model split at layer L_s. The quantity $E_s^{(k)}(\phi)$ is the sum of the energy consumed by the IoT device ϕ during forward and backpropagation for all entries of a minibatch of size ξ during training round k for the first s layers, plus the energy needed to exchange the intermediate activations and gradients for all entries of the minibatch with the edge server, all multiplied by a normalization factor d_ϕ/ξ (Equation 11).

$$E_{s,T}^{(k)}(\phi) = \frac{d_\phi}{\xi}\left(P_{\mathrm{c}}(\phi)\delta_s^{(k)}(\phi) + P_{\mathrm{t}}(\phi)\frac{V_s}{B_\phi^{(k)}} + P_{\mathrm{r}}(\phi)\frac{V'_s}{B_\phi^{'(k)}}\right) \tag{9}$$

The parameters $D_s^{(k)}(\phi), B_\phi^{(k)}, B_\phi^{'(k)}$ can vary according to the available computing and networking resources.

We, therefore, define $E_{s,\omega}^{(k)}(\phi)$ as the total communication energy needed to transmit the weights of the current model layers from the IoT to the server with throughput $B_\phi^{(k)}$ and receive back from the server with throughput $B_\phi^{'(k)}$ after model aggregation (Equation 10).

$$E_{s,\omega}^{(k)}(\phi) = P_{\mathrm{t}}(\phi)\sum_{i=1}^{s}\frac{|L_i|}{B_\phi^{(k)}} + P_{\mathrm{r}}(\phi)\sum_{i=1}^{s}\frac{|L_i|}{B_\phi^{(k)}} \tag{10}$$

Note that the model weights can be of different numbers of layers between uploaded and received after aggregation. We can there define the total *total energy consumption* for a round $E_s^{(k)}(\phi)$ for a device ϕ as the energy needed to carry out the classic FL training or the SL training plus the time needed to exchange model weights between the server and the IoT device (Equation 11).

$$E_s^{(k)}(\phi) = E_{s,T}^{(k)}(\phi) + E_{s,\omega}^{(k)}(\phi) \tag{11}$$

In our scenario, we assume that each of the Φ IoT devices has its own model split point $s_\phi^{(k)} \in \{1,\dots,N\}$, which can be different at each training round k according to the variable system context (e.g., available wireless throughput, computational load on IoT devices and edge server, etc.). We define the system *split vector* s_k as the vector of device-specific split points $s_k = \left(s_1^{(k)},\dots,s_\phi^{(k)}\right)^{\mathsf{T}} \in S = \{1,\dots,N\}^\Phi$ for each IoT device ϕ during training round k. The symbol S identifies the space of all possible split vectors.

This formulation is a generalization that includes the classic FL. For resource-capable participant devices it may not be necessary to split the computation of the models during the training process. Thus, for such a device ϕ, the split point is generally the same, and at the end of the model,

i.e., $s_\phi^{(1)} = s_\phi^{(2)} \cdots s_\phi^{(k)} = s_\phi^{(k+1)} \cdots = s_\phi^{(R)} = N$ This means no intermediate activations or gradients are transferred during the training round (i.e., $V_s = V_s' = 0$). Furthermore, the server does not carry out any processing during forward and backward propagation (i.e., $\gamma_s^{(k)}(C) = 0$). We can, therefore, express the total training time for a classic FL round through Equations 6 to 8.

We can now define the *system-wide training time* $\Delta_k(s_k)$ for training round k and split vector s_k as the sum, for all groups $g \in G$ of devices in the system, of the longest training time of any IoT device ϕ in each group (Equation 12). It is worth noting that reducing the training time of the slowest device within each group (*straggler*) reduces the total training time for a training round, as the model aggregation on the parameter server can be performed only when all devices in each cluster g have completed one training epoch and transmitted their local model update to the server.

$$\Delta_k(s_k, G) = \frac{1}{|G|} \sum_{g \in G} \max_{\phi \in g} \Delta_{s_\phi^{(k)}}^{(k)}(\phi) \tag{12}$$

Similarly, we define the *system-wide energy consumption* $E^{(k)}$ as the sum of the energy consumed by all IoT devices for all the groups in the system during training round k (Equation 13).

$$E_k(s_k) = \sum_{g \in \Phi} E_{s_\phi^{(k)}}^{(k)}(\phi) \tag{13}$$

Figure 5 shows a representation for DFL for five client devices with three groups in the partition set G. The parameters $s_A^{(k)}$ and $\omega_A^{(k)}$ represents the split point and the submodel weights for device A, respectively, at round k.

B. PROBLEM FORMULATION

The main goal of DFL is to determine the split vector s_k, for each training round k, and a clustering vector a that jointly optimizes the objectives: minimizing the global model's training time, minimizing the energy consumption on the devices and minimizing the training loss. Minimizing the training loss $\mathcal{L}(G)$, system-wide training time $\Delta_k(s_k, G)$, and minimizing the IoT devices' total energy consumption $E_k(s_k)$ are potentially conflicting objectives. For an IoT device ϕ, a split point that minimizes energy does not necessarily coincide with the split point that minimizes model training time [31]. Besides the allocation of training devices in clusters of similar data distributions affect the split vector. Therefore, the clustering and the split vector that jointly optimizes model training time and energy depend on the considered application requirements. In particular, some applications may be energy-sensitive, meaning that extending the devices' lifetime is more important than reducing the global model training time. While some other applications consider accuracy more important. Some other applications may be more time-sensitive, meaning that minimizing training time is more important than extending device's lifetime.

We define the parameter β as a coefficient that represents the application's preferences of the optimization toward minimizing the training loss, time, or energy. We define β_i such that the closer β_1 is to one ($\beta_1 \to 1$) the lower the training time. The closer β_2 is to one ($\beta_2 \to 1$) the low the system's energy consumption. While the closer β_3 is to one ($\beta_3 \to 1$) the application is more biased towards minimizing the training loss. We can express the application's training time-energy-accuracy sensitivity coefficient $\beta = (\beta_1, \beta_2, \beta_3) = (\beta_i)_{i \in \{1,2,3\}} \in [0,1]^{1 \times 3}$. The values of β_1, β_2, β_3 are selected to guarantee that $\sum_{i=1}^{3} |\beta_i| = 1$. In this work, we assume that the particular value of β does not change over time and is initially fixed by a policy maker that interprets the application's requirements.

This definition of β allows us to convert a multi-objective optimization problem that aims at minimizing training time, energy, and prediction loss separately in a single-objective optimization problem. In particular, we design a *round cost function* $U_k(s_k, G) \in \mathbb{R}$, which depends on the split vector s_k at training round k and the partition set G, as a linear combination of the training loss, training time and energy consumption, mediated by the coefficient β i.e., $\beta_1 \cdot \Delta_k(s_k, G) + \beta_2 \cdot E_k(s_k) + \beta_3 \cdot \mathcal{L}(G)$. For generalization, we define a *performance vector* $X(s_k, G) = (\Delta_k, E_k, \mathcal{L})$ for the training that indicates the training time, energy consumption, and model loss, respectively. We can, therefore, define a compact form of the utility function as (Equation 14), where β^T indicates the vectorized training optimization coefficients.

$$U_k(s_k, G) = \beta^T X(s_k, G) = \beta^T (\Delta_k, E_k, \mathcal{L}) \tag{14}$$

It is worth noting that when $\beta_i = 1$, the round cost function will consider only the i-th objective, while when $\beta_i = 0$, the round cost function will disregard the i-th objective. For a training round k, the optimal split vector s_k^* and optimal partition set G^*

$$(s_k^*, G^*) = \underset{(s_k, G) \in S \times \mathcal{P}(\Phi)}{\arg\min} \quad U_k(s_k, G) \tag{15}$$

minimize the round cost function U_k to achieve the best tradeoff between global model training time, system energy consumption, and training loss. Let us now assume that the model needs R training rounds for its generalization error to be minimized (*model convergence*). The ultimate goal of our proposed approach is to compute a split vector s_k for each of the R training rounds and a partition set G so that the training time, prediction accuracy, and system energy consumption until convergence is minimized. We assume that the partition set G does not change over the subsequent training rounds once established since the training data D_ϕ on each device remains the same during a training session. This is because the federated clustering mechanism is based on the heterogeneity of the training data on the individual devices. For the split vector, we introduce the *split matrix* $\sigma \in S^R = \{1, \ldots, N\}^{\Phi \times R}$ as a matrix with Φ rows and R columns, in which the k-th column contains the elements of the split vector s_k. We, therefore, define a general optimization for the R rounds cost functions for

every training round k until convergence in Equation 16.

$$(\sigma^*, G^*) = \underset{(\sigma,G)\in S^R\times\mathcal{P}(\Phi)}{\arg\min} \sum_{k=1}^{R} U_k(s_k, G) \tag{16}$$

The optimal split matrix σ^* and the partition set G^* minimize the general cost function and achieve the best tradeoff between model training time, prediction accuracy, and system energy consumption.

C. DFL ARCHITECTURE AND OPERATION

To enhance the efficiency and effectiveness of the distributed model training in the heterogenous learning environment, DFL utilizes a strategy to group devices based on the similarity of their local training data distributions without the need to directly access or share this data. This is achieved by examining the representations of specific layers within the DNN submodels trained on the devices. Analyzing these weights provides insights into the data distributions and learning patterns of each participant without compromising the privacy of their local data while enabling efficient and effective collaborative learning across the network. The core idea is to use the weights of these submodel layers as a basis for comparison. The similarity between devices is quantified using Centered Kernel Alignment (CKA) [41], a measure that compares kernal matrices of corresponding layers from different devices' models.

For a comparison of two layers from different submodels, we initially determine the normalized weights W_A and W_B from the submodels from devices A and B respectively. This is done to ensure that the scale of weights does not affect the comparison. Following the normalization, the next step involves the computation of the kernel matrices K_A and K_B. This is achieved by applying a suitable kernel function, such as a linear or Gaussian kernel, to the normalized weight matrices. Previous studies have indicated that the linear kernel is more efficient [42]. Using the linear kernel, we have $K_A = W_A W_A^T$ and $K_B = W_B W_B^T$, which are the kernel matrices for the respective weight sets. The CKA between the two kernel matrices is computed as follows:

$$\text{CKA}(K_A, K_B) = \frac{\text{HSIC}(K_A, K_B)}{\sqrt{\text{HSIC}(K_A, K_A)\cdot\text{HSIC}(K_B, K_B)}} \tag{17}$$

In this equation, HSIC stands for Hilbert-Schmidt Independence Criterion [43], which measures the dependence between the two sets of variables. The HSIC is computed as follows:

$$\text{HSIC}(K_A, K_B) = \frac{1}{(\zeta-1)^2}\text{tr}(K_A K_B) \tag{18}$$

In this formula, ζ is the number of dimensions in the weight matrices and $\text{tr}(\cdot)$ indicates the trace of a matrix.

DFL utilizes a clustering approach to group the IoT devices based on the average similarity index of their submodel layers. DFL leverages the layerwise CKA values of each set of participating devices to determine a pairwise similarity matrix. Let us denote $K_{A,i}$ and $K_{B,i}$ as the kernel matrices of layer L_i from device $A \in \Phi$ and $B \in \Phi$ respectively. We define $H = \{H_{A,B}\}_{A,B\in\Phi}$ as the *pairwise similarity matrix* of any all IoT devices in the system, where its elements are the average similarity scores of the first $N' < N$ model layers from the participant pair (Equation 19).

$$H_{A,B} = \frac{1}{N'}\sum_{i=1}^{N'} \text{CKA}(K_{A,i}, K_{B,i}) \tag{19}$$

The pairwise similarity matrix is utilized to determine a *dissimilarity matrix* $H' = \{1 - H_{A,B}\}_{A,B\in\Phi}$. It is worth noting that the layerwise similarity scores can be determined for a set of participants with any given number of submodel layers for each participant. However, a considerable number of layers is required to achieve significant similarity scores. Furthermore, an offline bench-marking process has shown that the layerwise similarity results in better similarity/dissimilarity scores.

DFL operates through the synergy of various modules: the estimation module, the network module, the neural network clustering module, and the optimization module. The estimation module's role is to periodically estimate the time each layer of the model $\{L_i | 1 \le i \le N, i \in \mathbb{N}\}$ requires for forward and backpropagation on the servers and across the Φ participating IoT devices. During every training round k, an estimation module on IoT device ϕ assesses $\gamma_s^{(k)}(\phi), \forall s \in \{1, \ldots, N\}$, representing the time for executing forward and backward propagation on a sub-model split at layer $L_{s_\phi^{(k)}}$. Likewise, an estimation module on the edge server calculates $\gamma_s^{(k)}(C), \forall i \in \{1, \ldots, N\}$, indicating the time to process the last $N - s$ layers during training round k. This estimation is based on benchmarking the forward and backpropagation time for each layer. Typically, it's not required to update these estimates every training round. We assume that benchmarks for the IoT devices and the edge server are performed every M_D and M_C training rounds, respectively. The estimations for all rounds between two benchmarks are set to identical values, where M_D and M_C denote the intervals for device and server computation benchmarking, depending on the variability of resources.

The network module is designed to periodically calculate the values of $B_\phi^{(k)}$ and $B'^{(k)}_\phi$ for every ϕ in the set $\{1, \ldots, \Phi\}$. These values represent the average throughput of the wireless channel from the IoT device ϕ to the edge server and vice versa, at each training round k. The module operates on the edge server. During training round k, for each minibatch processed by the model split at layer L_s, a volume of intermediate activations V_s is sent from the IoT device to the edge server, and a volume of intermediate gradients V_s' is sent back from the edge server to the IoT device. For each minibatch, the network module calculates the transmission time for the volume V_s of intermediate activations to the edge server and then computes the average wireless channel throughput as the ratio of V_s to the

transmission time. Similarly, the module assesses the time required to transmit the volume V_s' of intermediate gradients back to the IoT device and calculates the average wireless channel throughput as the quotient of V_s' to its transmission time. Additionally, the module estimates the time needed to transmit the data volume $\sum_{i=1}^{s}|L_i|$ of model layers to the parameter server for aggregation and the initial volume $\sum_{i=1}^{s}|L_i|$ during model placement. At the end of a training round k, the network module estimates the overall average throughput. for the entire round.

Let us define the *clustering vector* $a = (a_1, \cdots, a_{|\Phi|}) \in G^{|\Phi|}$ as a vector whose ϕ-th component $a_\phi \in G$ indicates the group to which device ϕ belongs. It is worth noting that any partition set $G \in \mathcal{P}(\Phi)$ in the set $\mathcal{P}(\Phi)$ of all possible partition sets is fully described by a unique clustering vector $a \in G^{|\Phi|}$, so that G is a function $G(a)$ of a. The neural network clustering module's task is to determine the optimal allocation vector that minimizes the group loss sum. To determine the groups of devices, the network clustering module utilizes the similarity matrix H determined from the average layerwise CKA values of all submodel pairs. This similarity matrix is crucial as it captures the nuanced relationships between devices' data based on their learned representations. The use of CKA as a metric ensures that the similarity matrix reflects the true alignment in the feature space of the neural network models, making it a reliable basis for clustering. DFL utilizes the density-based clustering approach, DBSCAN, to determine the clusters of participant devices using the dissimilarity matrix. The transformation from a similarity matrix to a dissimilarity matrix H' is a key step, allowing DBSCAN to effectively identify clusters based on density, which is essential for handling varying cluster shapes and sizes in the device network. By clustering devices based on weight similarities, DFL ensures that devices with similar data distributions are grouped together. This targeted grouping based on similarity not only enhances the robustness of the clustering process but also ensures a more homogeneous data distribution within each cluster. This approach can lead to more efficient training and potentially enhanced overall model performance. The devices in each cluster collaboratively train sub-models.

The optimization module utilizes the estimates from the estimation module, the network module, and the allocation vector from the neural network similarity module solves the optimization problem described in Equation 16 by exploring the space $S \times G^{|\Phi|}$ to find the optimal split vector and device group allocation vector (s_k^*, a^*) that minimizes the cost function U_k according to the current estimated system context for every training round k. The optimization module iteratively explores the sub-spaces of S and the corresponding allocation vectors by utilizing the output of the neural network module and determining the model distribution within device groups. The optimization module is deployed and executed on the server. After the optimal split vectors $s_{(k)}^*$ for all groups $g \in G$ are computed, the edge server communicates the new split points to each of the Φ IoT devices in the system via the downlink wireless channel.

Algorithm 1: Layerwise Neural Network Similarity-based Participant Clustering

Input:
$F(\omega) \in [0,1]$: local loss function $l(x_j, y_j, \omega)$ on the samples $\forall(x_j, y_j) \in d_\phi$ given the model parameters ω
$\lambda \in (0, +\infty) \subset \mathbb{R}$: training loss updating threshold for participant clustering
$\vartheta \in \mathbb{N}$: training loss tracking window size
Output:
$a = (a_1, \cdots, a_{|\Phi|}) \subseteq \Phi^{|G|}$: clustering vector

/* Initialization: loss vectors */
1 $\psi^{(0)} \leftarrow 0$
/* Initialization: cluster vector */
2 $a_1 = \ldots = a_{|\Phi|} = 0$
/* Loops for each round k */
3 **for** $k \in \{1, \ldots, R\}$ **do**
  /* Compute cumulative loss for all devices */
4 **for** $\phi \in \Phi$ **do**
5 $\omega_\phi^{(k)} \leftarrow \omega_\phi^{(k-1)} - \eta \nabla F_\phi(\omega_\phi^{(k-1)})$
  $F_\phi(\omega^{(k)}) \leftarrow \frac{1}{|d_\phi|} \sum_{(x,y)\in d_\phi} f(\omega^{(k)}, x, y)$
  /* Rolling window loss tracking */
6 $j \leftarrow \max\{1, k - \vartheta + 1\}$
7 $\psi_\phi^{(k)} \leftarrow \frac{1}{k-j+1} \sum_{i=j}^{k} F_\phi(\omega^{(i)})$
8 $\psi^{(k)} \leftarrow \frac{1}{|\Phi|} \sum_{\phi\in\Phi} \psi_\phi^{(k)}$
9 **if** $|\psi^{(k-1)} - \psi^{(k)}| > \lambda$ **then**
10 $\psi^{(k-1)} \leftarrow \psi^{(0)}$
11 **else**
  /* Updating group allocation vector via neural networks layerwise similarity */
12 **for** $A, B \in \Phi$ **do**
13 $H_{A,B} \leftarrow \frac{1}{N'} \sum_{i=1}^{N'} \text{CKA}(K_{A,i}, K_{B,i})$
14 $(a_1, \cdots, a_{|\Phi|}) \leftarrow \text{DBSCAN}(1 - \{H_{A,B}\})$
15 **return** $(a_1, \cdots, a_{|\Phi|})$

The comprehensive workflow and sequential interactions of the different components within DFL are delineated in Algorithms 1 and 2. DFL tracks the training loss for the IoT devices to determine the effective clusters of the devices based on the training progress as delineated in Algorithm 1. This process is facilitated through the implementation of rolling window averages for loss monitoring during the distributed training on IoT devices (refer to lines 3 to 9 in the algorithm). The employment of a rolling window for tracking loss is instrumental in reducing the variance associated with loss measurements, thereby ensuring a more

Algorithm 2: Dynamic Resource-Aware Distributed Neural Network Training

Input:
$\tau_i^{(k)}$: returns the average time needed to transmit activations from the i-th layer during round k
$f_{\text{device}}^{(k)}(L_i)$, $f_{\text{edge}}^{(k)}(L_i)$: return the average computation time on the IoT device and server for split point i
$\Omega : \mathbb{R} \to \mathbb{N}, \Omega(x) = \lfloor x/Q \rfloor$: discretizes throughput in equal intervals of Q bit/s
$\Gamma \in \mathbb{N}$: throughput measurement window size
$M_{\text{D}}, M_{\text{C}}$: device and server computation benchmark intervals
$(\beta_i)_{i\in\{1,2,3\}} \in [0,1]^{1\times 3}$: training time-energy-accuracy sensitivity coefficient

Output:
$(s_k, a) \in S \times G^{|\Phi|}$: split vector for each round and clustering vector

/* Initialization */
1 $b^{(1)} \leftarrow Q$
2 $|G| \leftarrow 1$
/* Loops for each round k */
3 **for** $k \in R$ **do**
/* Device computation benchmark */
4 **if** $k \mod M_D = 1$ **then**
5 **for** $\phi \in \Phi$ **do**
6 **for** $s, i = 1, \cdots, N$ **do**
7 $\gamma_s^{(k)}(\phi) \leftarrow f_{\text{device}}^{(k)}(L_i)$
/* Server computation benchmark */
8 **if** $k \mod M_C = 1$ **then**
9 **for** $s, i = 1, \cdots, N$ **do**
10 $\gamma_s^{(k)}(C) \leftarrow f_{\text{edge}}^{(k)}(L_i)$
11 **for** $g \in G$ **do**
12 **if** $k > 1$ **then**
13 $b^{(k)} \leftarrow V_{s^{(k-1)}} / \tau_{s^{(k-1)}}^{(k-1)}$
14 $b'^{(k)} \leftarrow V'_{s^{(k-1)}} / \tau'^{(k-1)}_{s^{(k-1)}}$
/* Rolling window throughput estimations */
15 $j \leftarrow \max\{1, k - \Gamma + 1\}$
16 $B^{(k)} \leftarrow \frac{1}{k-j+1} \sum_{i=j}^{k} b^{(i)}$
17 $B'^{(k)} \leftarrow \frac{1}{k-j+1} \sum_{i=j}^{k} b'^{(i)}$
/* Changed resources? */
18 **if** $\Omega(B^{(k-1)}) = \Omega(B^{(k)})$ ***and*** $\Omega(B'^{(k-1)}) = \Omega(B'^{(k)})$ ***and***
19 $k \mod M_D \neq 1$ ***and*** $k \mod M_C \neq 1$ **then**
20 $s_g^{(k)} \leftarrow s_g^{(k-1)}$
21 **else**
22 **for** $\phi \in g$ **do**
23 Compute $\{\Delta_1^{(k)}(\phi), \ldots, \Delta_N^{(k)}(\phi)\}$
24 Compute $\{E_1^{(k)}(\phi), \ldots, E_N^{(k)}(\phi)\}$
25 $(s_k, a) \leftarrow \arg\min \beta^T (\Delta_k, E_k, \mathcal{L})$

stable and accurate representation of the training progress. The size of the rolling window $\vartheta \in \mathbb{N}$ is tailored to align with the dynamics of the training process. It should be large enough to smooth out transient fluctuations in loss but small enough to reflect meaningful accuracy changes in the training progression and can be set as a parameter for the process.

Upon achieving a predefined threshold in loss change, DFL proceeds to construct the pairwise dissimilarity matrix. This matrix is derived using layerwise neural network similarity, specifically the CKA metrics, which provide a nuanced understanding of the similarity in feature representations learned by different devices. Subsequently, a density-based clustering approach is applied to categorize the devices into groups based on this dissimilarity matrix (lines 11 to 14).

At every training round k, DFL initially determines if the estimation module needs to perform benchmarking, as outlined in Algorithm 2. The estimation module can then determines the values $\gamma_s^{(k)}(\phi)$ and $\gamma'^{(k)}_s(\phi), \forall s \in \{1, \ldots, N\}$, and $\gamma_s^{(k)}(C)$ and $\gamma'^{(k)}_s(C), \forall i \in \{1, \ldots, N\}$. These values represent the forward and backpropagation times (refer to lines 4 to 10 in the algorithm). The time intervals M_{D} and M_{C}, designated for these measurements, are adaptable to reflect the fluctuating computational capabilities of both the edge server and IoT devices. Then, DFL estimates the average uplink and downlink throughput values $B_\phi^{(k)}$ and $B'^{(k)}_\phi, \forall \phi \in \{1, \ldots, \Phi\}$ at every training round k between each IoT device ϕ. This estimation is accomplished by employing rolling-window averages (lines 12 to 17), utilizing recently measured throughput values b. The rolling-window averages for throughput estimation is beneficial as it mitigates the impact of transient, erratic fluctuations in throughput, thereby yielding more precise channel estimations.

For each group $g \in G$ encompassing all devices, DFL evaluates whether any benchmarking has been conducted during the current training round or if there has been a change in the system context. If either condition is met, the optimal split vector for the group is recalculated to adapt to the new circumstances. In the absence of such changes, the previously computed split vector is retained (Algorithm 2 lines 18 to 25). This approach ensures that the device clusters and their respective training strategies remain aligned with the most current performance, optimizing the efficiency and effectiveness of the distributed training in heterogeneous dynamic environments. The convergence analysis for the training is detailed in Appendix A.

IV. EVALUATION

In this section, we assess the performance of the DFL. We start by describing the hardware and setup of our testbed. We highlight the methods and share the results from testing the DFL in different scenarios and compare its performance to SFL and FL. SFL combines the training parallelization in FL and model splitting in SL for efficient distributed training in resource-contrained devices. FL presents a benchmark of current widely adopted distributed ML method. We

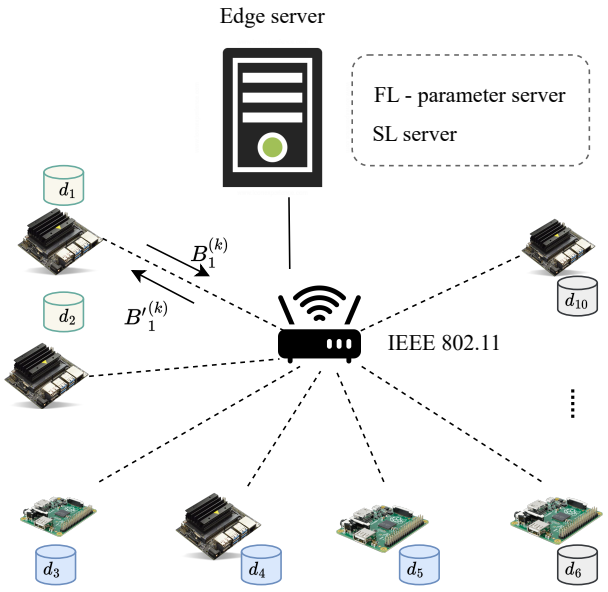

FIGURE 6. Overview of the testbed used for the evaluation.

consider heterogeneous and dynamic IoT resources and non-identically distributed (non-IID) training data. We implemented DFL[1] in Python, and the deep learning framework in DFL is PyTorch[2]. The key metrics for the evaluation are training time, energy use, model convergence and accuracy.

A. EXPERIMENTAL SETUP

1) Testbed Setup

In our experiments, we deployed a real testbed made of 10 heterogeneous IoT devices. The specifications of these devices are detailed in Table 2. Each Jetson Nano in our setup has onboard power sensors situated at its power input. These sensors can be accessed either automatically using the `tegrastats` tool [3] or manually through the Sysfs pseudo-file system [4] on Linux. For power measurements, we tapped into the Sysfs pseudo-file, particularly from the I2C folder of the Jetson Nano. We designed a multithreaded power consumption benchmarking class to observe power variations in the I2C during different training tasks on the Jetson Nano. For the Raspberry Pi devices, we utilized a plugin power monitor to track power usage. This approach enables accurate estimations of the distinct computation and communication energy consumption, as well as training and communication time.

All Raspberry Pis in our setup ran the Raspbian GNU/Linux 10 (Buster) OS, with Python 3.7 and PyTorch 1.4.0 installed. The Jetson Nanos are operated with the NVIDIA JetPack SDK, which contains the Jetson Linux Driver package (L4T) responsible for managing hardware resources and power. Both the Jetson Nanos and our server used the same Python and PyTorch versions. All devices are connected to the server in a network using a router. The IoT devices and the edge server can directly communicate via an IEEE 802.11 wireless link, whose available throughput can be controlled by a traffic shaping tool (Linux `tc` [5]).

We select the Visual Geometry Group (VGG) DNN model [44] as the global model deployed on the edge server and IoT devices.The VGG model is well-suited to assess a split learning scenario, as the compared algorithms can select a split point among 10 layers (VGG-8). We selected VGG as the evaluation model because it is widely adopted in IoT applications based on edge intelligence. These include image classification in industrial IoT [45], smart security [2], smart vehicles [46], healthcare IoT [47]. Furthermore, DFL models a DNN as a sequential connection of independently executable layers and not as an indivisible executable unit. Therefore, our approach is generalizable and can be applied to any other sequential DNNs composed of any different arrangement of independently executable layers. This evaluation approach is commonly used in the related literature [13], [22].

TABLE 2. Testbed Specifications

Raspberry Pi 3B	
CPU	1.2 GHz core ARM Cortex-A53
RAM	900 MHz 1 GB LPDDR2
Operating System	Raspbian GNU/Linux 10 (Buster)
NVIDIA Jetson Nano	
CPU	Quad-core ARM Cortex-A57 MPCore
RAM	900 MHz 2 GB LPDDR4
Nano GPU	NVIDIA Maxwell arch. CUDA core
Operating System	NVIDIA JetPack SDK
	Jetson Linux Driver package (L4T)
Edge Server	
GPU	NVIDIA GeForce RTX 2060
CPU	Intel Core i9-10900 10th Gen
RAM	2933 MHz 32 GB DDR4
Operating System	Ubuntu 20.04.2 LTS

TABLE 3. Jetson Nano Power Modes

Modes	HIGH	MEDIUM	LOW
CPUS ONLINE	4	2	1
CPU MAX FREQ	1200 MHz	900 MHz	900 MHz
GPU MAX FREQ	900 MHz	600 MHz	600 MHz
MEM MAX FREQ	900 MHz	600 MHz	600 MHz

[1] `Repository link to be added on final version`
[2] https://pytorch.org/
[3] https://github.com/topics/tegrastats
[4] https://man7.org/linux/man-pages/man5/sysfs.5.html
[5] https://man7.org/linux/man-pages/man8/tc.8.html

2) Datasets

We utilize two widely adopted datasets, CIFAR-10 [48] and CINIC-10 [49], for our training and testing processes. The CIFAR-10 dataset consists of 60k 32x32x3 color images in 10 classes, with 6k images per class. The dataset contains a large set of 50k training and 10k testing labeled images. The CINIC-10 dataset was developed to bridge the gap between standard academic benchmarks and real-world data. CINIC-10 is compiled by merging CIFAR-10 with images from the ImageNet database, providing a larger and more diverse set of images. It consists of 32x32x3 color images and is organized into 10 classes. The dataset comprises a total of 270,000 images with 90,000 images in each of the training, validation, and testing splits. Each class in every split contains 9,000 images, ensuring a balanced representation of categories. This dataset, due to its larger size and diversity, presents a more challenging benchmark and is suitable for evaluating the robustness capabilities of ML models and methods.

We split each dataset into 10 subsets according to the Dirichlet distribution Dir(α) with $\alpha \in \{0.1, 0.5, 1\}$ to simulate scenarios with non-IID data [50]. The non-IID degree is determined by α. When α is very small, e.g., $\alpha = 0.1$, the data non-IID degree is more significant, implying that the data owned by a particular client cannot cover all classes, i.e., $|Y_i| \leq |Y|$, where Y_i is the label space of data distributed on the i-th client. As α increases, the distribution of the data among the clients becomes less heterogeneous, i.e., the non-IID degree reduces. The non-IID degree is more significant when α is smaller. Additionally, we introduce variations in the number of images per class within each device, further enhancing the non-IID nature of the dataset. By manipulating the allocation of subsets across client devices and introducing imbalances, we generated a non-IID version of the CIFAR-10 and CINIC-10 datasets, enabling us to evaluate the distributed learning schemes under realistic and challenging conditions.

B. VARIABLE COMPUTING RESOURCES AND POWER MONITORING

We employed NVIDIA drivers on the Jetson Nano to manage hardware resources and oversee power control. The available computational resources on the Jetson Nano can be adjusted by altering its power modes. NVIDIA's JetPark software provides two standard power modes, with an additional option to customize power modes via its drivers. To categorize the power consumption, we established three distinct configurations for CPU, GPU, and Memory clock speeds. These configurations represent three tiers of power consumption: HIGH, MEDIUM, and LOW. Table 3 displays the specific hardware resource configurations for each tier. We tracked parameters P_{c}, P_{t}, and P_{r} by separately executing computational processes (both forward and backpropagation). During these processes, we captured power readings from the I2C sensors. Figure 7 presents a visualization of the

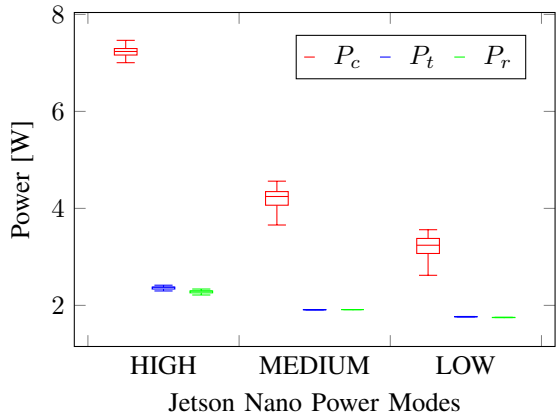

FIGURE 7. Computing, network transmission, and receiving power for the three Jetson Nano computing modes (power modes).

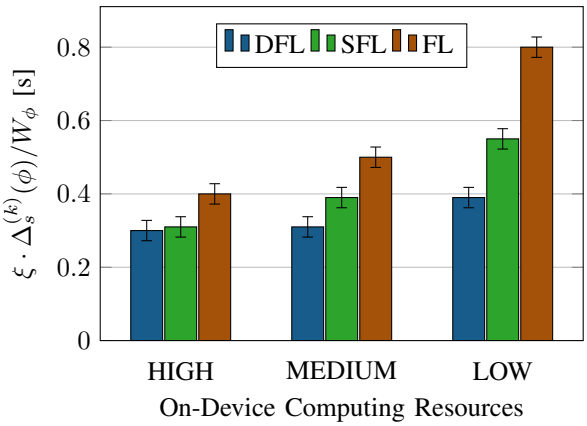

FIGURE 8. Average training time for Jetson Nano with various computing resources for minibatch with size $\xi = 100$ for CIFAR-10 dataset.

Jetson Nano's power consumption during computation, data transmission, and reception across the three predefined power modes: HIGH, MEDIUM, and LOW.

We evaluate the performance of DFL in the contexts with varying computational resources. Specifically, we assess DFL's behavior across the three distinct power modes of the Jetson Nano, each providing a different computational resource availability. For comparison, we examined the performance of DFL against FL, where the model training is entirely completed on the device, and the SFL approach in which the model computation is split during the training, but the split points remain consistent regardless of available resources. In SFL, the model split layers are set offline to adhere to time or energy constraints. Our metric of interest is the average training time for a mini-batch, represented by $\xi \cdot \Delta^{(k)}/d_\phi$ for a mini-batch size of ξ. In our experiments, we standardized the mini-batch size ξ to 100 across all evaluation scenarios. Figure 8 illustrates the comparative average training times for a mini-batch under DFL, SFL, and FL in varying computational resource contexts provided by the Jetson Nano. We have marked the confidence intervals

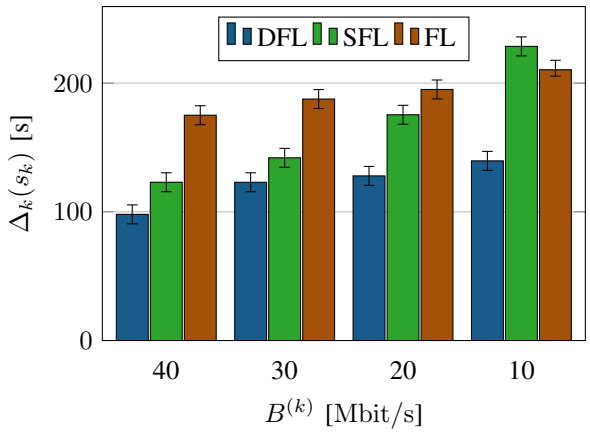

FIGURE 9. Average training time per round for the system for CIFAR-10 dataset on the heterogeneous IoT devices.

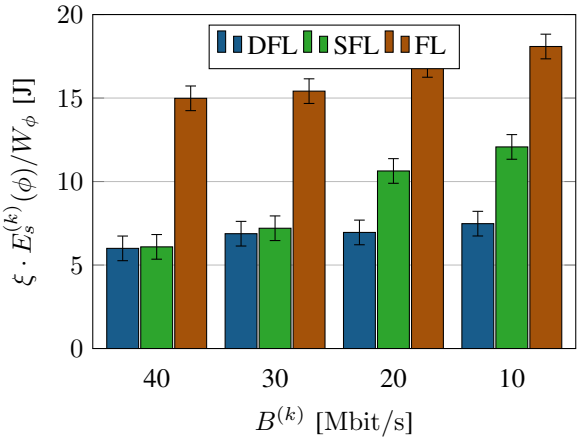

FIGURE 10. Average energy consumption for training minibatch with size $\xi = 100$ on Raspberry Pi for CIFAR-10 dataset.

at a 95% significance level. Notably, DFL consistently outperforms both FL and SFL in all three power configurations, achieving the shortest average mini-batch training time. The most pronounced efficiency of DFL is evident in the LOW power mode, where it realizes a 51% reduction in the training time compared to FL. This is attributed to the ability of DFL to adapt the training tasks based on the available training resources on the devices and their heterogeneity.

C. TRAINING TIME AND ENERGY OPTIMIZATION ON HETEROGENOUS IOT DEVICES

We examine the performance of DFL, focusing on optimizing training time and energy consumption across a variety of heterogeneous devices under different network throughput conditions. Initially, we varied the network throughput between the IoT devices and the server from 10 $\mathrm{Mbit/s}$ to 40 $\mathrm{Mbit/s}$. During this phase, we closely monitored how DFL optimized training time per round. The findings, illustrated in Figure 9, reveal that DFL consistently outperforms both FL and SFL in average training time across all examined throughput levels, achieving reductions in training time by as much as 48% and 37.5% compared to FL and SFL, respectively. This enhancement in training time is largely due to DFL's capability to dynamically account for the diverse range of resources available within the IoT systems. Specifically, DFL efficiently allocates a fewer number of layers to devices with limited resources during training, thereby effectively minimizing the impact of strugglers. Additionally, DFL takes into consideration the available network throughput for the transmission of intermediate activations or model updates between training rounds, further optimizing the training process.

Subsequently, we assessed the energy consumption required to train a minibatch of size ξ on a Raspberry Pi across varying network throughput (from 10 $\mathrm{Mbit/s}$ to 40 $\mathrm{Mbit/s}$). The results, depicted in Figure 10, demonstrate the average energy consumption $\xi \cdot \Delta^{(k)}/d_\phi$ by the Raspberry Pi for completing a minibatch training session. DFL significantly lowers average energy consumption compared to both FL and SFL across all network throughput levels, achieving reductions up to 62.8% and 49.8% respectively. These reductions in energy consumption are attributable to DFL's adaptive approach in model training. It strategically minimizes energy use on the devices by considering both computing and communication power requirements for the effective distribution of the training tasks. This adaptability ensures that DFL not only enhances training efficiency but also promotes energy conservation in IoT devices, which is vital for sustainable and long-term IoT deployments.

D. CONVERGENCE AND ACCURACY PERFORMANCE: IMPACT OF NON-IID DEGREE AND LOCAL DATA SIZE

In this evaluation, we delve into the influence of differing levels of non-IIDness and the volume of training samples available on each device on model convergence and accuracy. Specifically, we evaluate the training test accuracy over a set number of rounds for the schemes DFL, SFL, and FL using the two datasets: CIFAR-10 and CINIC-10. Notably, since SFL and FL employ a logically congruent mechanism for model parameter aggregation through FedAvg, we present a singular result set (SFL) to depict their convergence and accuracy trends. To induce variability in training data distributions, we manipulated the non-IID degree (heterogeneous level of data distributions) by modulating the α value within the range $\{0.1, 0.5, 1\}$. For every degree of non-IIDness, the performance of DFL and SFL was evaluated contingent on the number of training samples, denoted as $n = |D_\phi|$.

For CIFAR-10, at each chosen α, the number of local samples on devices was methodically drawn from subsets $n \in \{50, 100, 150\}$, $n \in \{300, 600, 900\}$, and $n \in \{1000, 3000, 5000\}$. Meanwhile, for the CINIC-10 dataset, owing to its more extensive sample repository, the number of local samples was chosen from the sets: $n \in \{50, 100, 150\}$, $n \in \{300, 600, 900\}$, and $n \in \{5000, 7000, 9000\}$. This deliberate stratification not only facilitated the examination of model behavior across varying data volumes but also

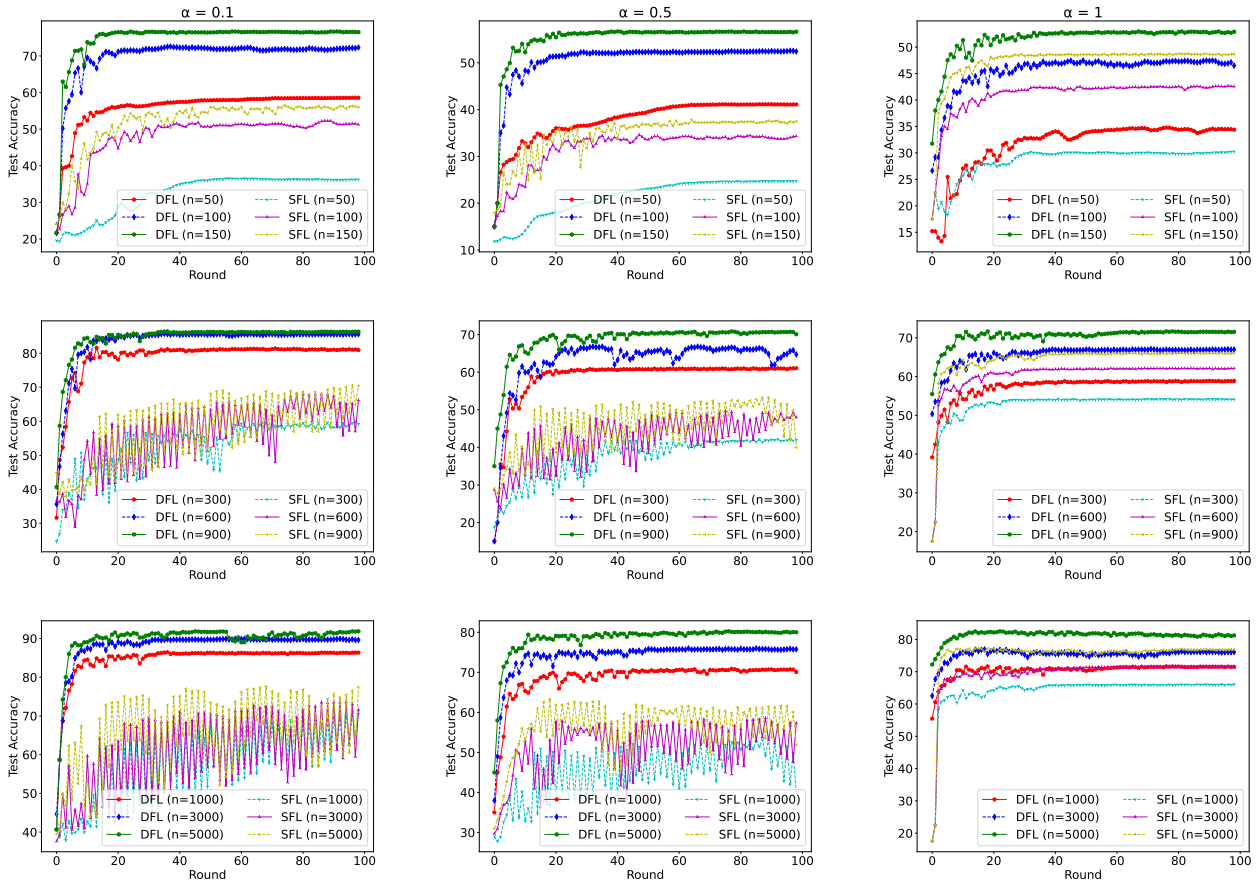

FIGURE 11. Accuracy comparison for DFL and SFL for CIFAR-10 dataset for various values of α and local data sizes for 100 training rounds.

probed the model's response to incremental shifts in data availability for different values of α. As the data repository expands, the model's sensitivity to these incremental sample size augmentations decreases since each additional data point (or small set of data points) contributes less to the overall diversity or information content of the dataset. Thus, more substantial jumps in data volume become necessary to discern tangible variations in training dynamics and model outcomes. Furthermore, since CINIC-10 dataset amalgamates CIFAR-10 with elements from the ImageNet repository, it naturally contains a more generous cache of training samples than its CIFAR-10 counterpart.

Figure 11 shows the average test accuracy performance for DFL and SFL, given the various degrees of non-IIDness α and local data size n for the CIFAR-10 dataset. Similarly, Figure 12 shows the average test accuracy performance for DFL and SFL for various degrees of non-IIDness α and local data size n for the CINIC-10 dataset. Notably, DFL consistently outperforms SFL in terms of test accuracy and model convergence for both datasets. DFL improves the accuracy by up to 29% for CIFAR-10 and up to 32.7% for CINIC-10 datasets in extreme non-IID scenarios.

A pivotal reason for the superior performance of DFL lies in its underlying methodology. DFL leverages user clustering, where participant devices with similar data distributions are grouped together. This strategy capitalizes on the inherent data similarities, enabling more efficient and focused learning within clusters. Such clustering mitigates the challenges presented by extreme non-IIDness. When users with alike data patterns are clustered, the local updates within that cluster tend to be more consistent, leading to improved aggregation results. In contrast, without such clustering, in SFL, the model has to reconcile vastly different local updates, which can impede convergence and potentially compromise model accuracy. The adaptability and precision of DFL in handling diverse data distributions via user clustering, therefore, provide it with a distinct edge in performance. This is essential to enable multitask learning in real-world dynamic IoT where a single aggregated model does not efficiently capture the heterogeneous learning landscape as a result of distinct geographical locations of devices, diverse application preferences, and other environmental influences.

In the context of both CIFAR-10 and CINIC-10 datasets, it is evident that the volume of samples available on a device plays a crucial role in determining the test accuracy and convergence speed of the models. When dealing with varying degrees of non-IIDness, represented by the α value in the set $\{0.1, 0.5, 1\}$, it's observed that the performance of both DFL

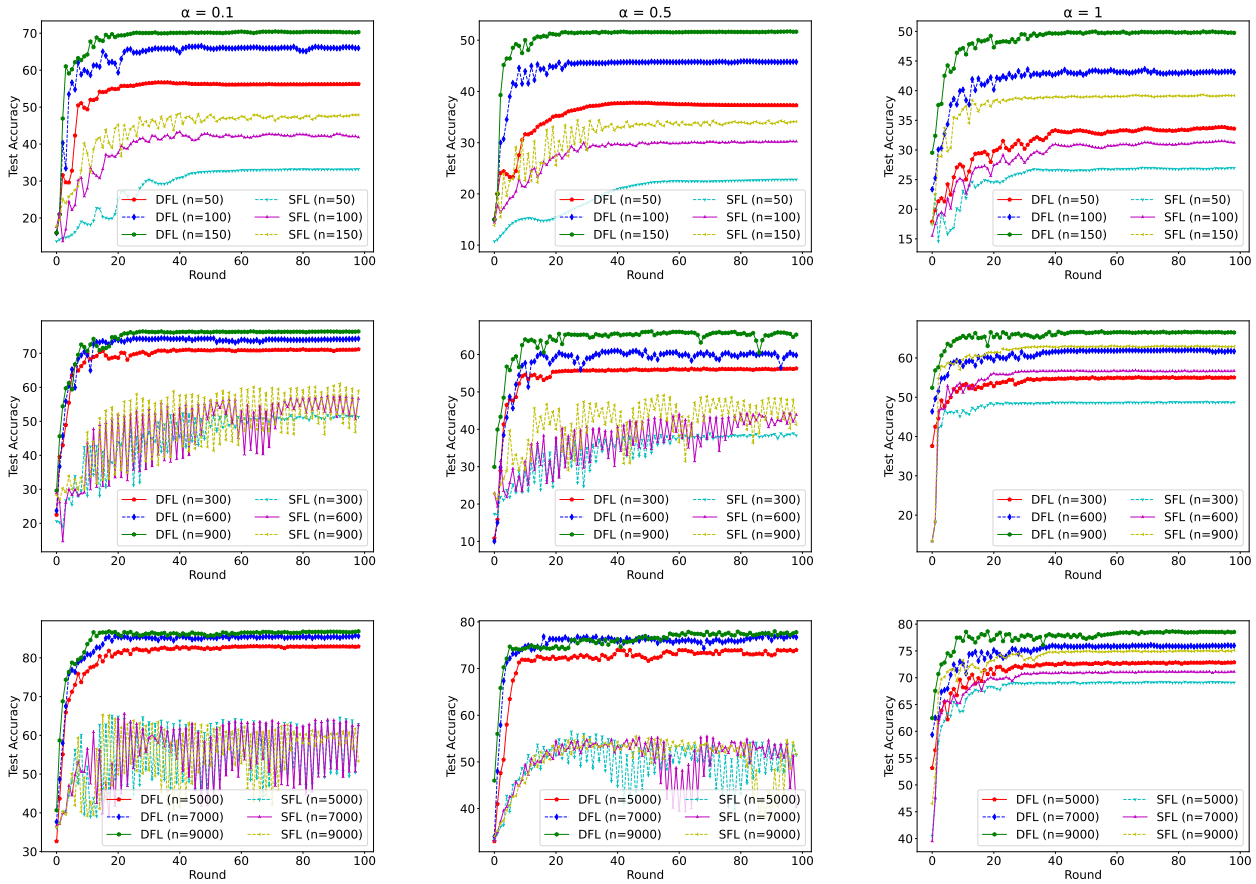

FIGURE 12. **Accuracy comparison for DFL and SFL for CINIC-10 dataset for various values of α and local data sizes for 100 training rounds.**

and SFL diminishes as the number of local data samples n decreases. This decline in performance is attributed to the models' limited exposure to data, hindering their ability to effectively generalize across the diverse underlying data distributions. The impact of this limitation is particularly prominent in SFL, which aggregates models from heterogeneously distributed clients. Due to the divergent nature of data across these clients, especially under high non-IID conditions, the SFL model experiences fluctuations in accuracy with high variance. This variability is a direct consequence of the challenges posed by aggregating models trained on datasets that significantly differ from one another, leading to inconsistencies and instability in the learned model.

DFL demonstrates a more robust performance relative to SFL, especially in scenarios with limited data availability. This superior performance is attributed to DFL's proficiency in leveraging inherent data similarities among devices, fostering a more focused and efficient learning process. As the local training data volume on the devices increases, DFL shows a consistent improvement in accuracy. This improvement is facilitated by the increased data availability, which provides a more comprehensive representation of the underlying distributions and is especially beneficial in clustered groups of devices with similar data characteristics.

Overall, while both DFL and SFL are influenced by the quantity of local data and the non-IID nature of that data, DFL's approach to harnessing data similarities and effectively managing the challenges of non-IID data results in steadier gains in model accuracy, particularly as the local data volume increases.

E. ADAPTABILITY TO CHANGING NETWORK CONDITIONS

We examine the training time and energy consumption optimization of DFL, FL, and SFL, through different available network throughputs. We further examine the *adaptiveness* of the three schemes, i.e., how the performance of each solution changes according to variations in link throughput over time. Therefore, we utilize the traffic shaper to set the network throughput, for every round k, to a random value extracted by a uniform distribution from 5 $\mathrm{Mbit/s}$ to 40 $\mathrm{Mbit/s}$ during the training. We utilize the CINIC-10 dataset for the training using the three schemes, DFL, FL, and SFL, with the number of local samples per device $n = 5000$ and the non-IID degree $\alpha = 0.5$ The DFL training optimization coefficient is set at $\beta_i = 0.3$.

Figure 13 illustrates the effect of varying system context on the training time and energy consumption of IoT devices

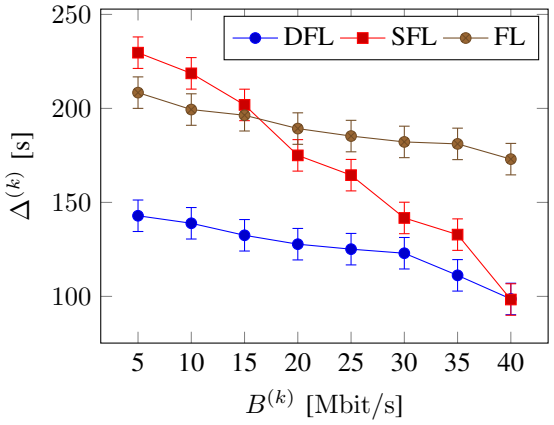

FIGURE 13. **Average training time per round under variable network throughput context for CINIC-10 dataset.**

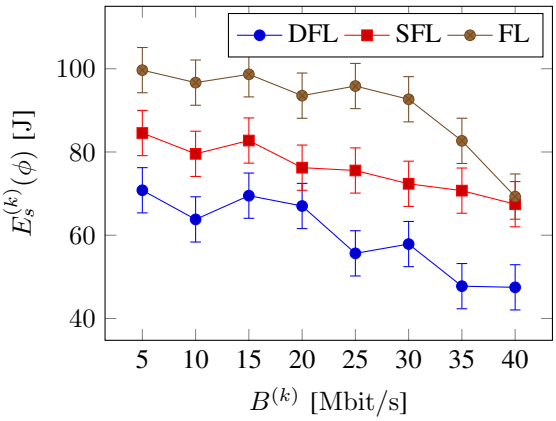

FIGURE 14. **Energy consumption on Jetson Nano under variable network throughput context for CINIC-10 dataset.**

during each training round k. The link throughput $B^{(k)}$ fluctuates at each round k, following a uniform distribution from 5 $\mathrm{Mbit/s}$ to 40 $\mathrm{Mbit/s}$. For clarity, the results are presented in a sorted manner. In all scenarios of network throughput, DFL consistently outperforms both FL and SFL in terms of average training time per round. DFL's reduces training time by up to 37.9% compared to SFL and 46.2% compared to FL. This efficiency stems from DFL's dynamic adaptability in allocating neural network training tasks based on the current communication resources. Particularly in situations of low network throughput, DFL prioritizes local training over transmission of intermediate results and model placements. Conversely, at higher throughput levels, DFL leverages increased transmission to optimize training time.

Conversely, SFL lacks this adaptability to the available resources in a heterogeneous learning environment. Consequently, the training time under the SFL approach shows significant variability in response to changes in network throughput. In contrast, FL primarily involves transmission during model aggregation, leading to comparatively less variability in training time. At some point, the training times of SFL and FL converge or even where SFL surpasses FL in terms of longer training durations. This is primarily due to the increased overhead of transmitting intermediate activations and gradients, particularly in scenarios characterized by lower network throughput.

Figure 14 illustrates the impact of different system contexts on the energy consumption of IoT devices during their training sessions for the Jetson Nano. In all scenarios of network throughput, DFL consistently outperforms both FL and SFL in terms of average energy consumption per round. The efficiency of DFL is highlighted by its capacity to reduce training time significantly by up to 32% when compared to SFL and 42.7% in relation to FL. This notable enhancement is the result of DFL's deliberate integration of energy consumption considerations into its training process. By doing so, DFL not only speeds up the training but also ensures it is executed with optimal energy efficiency.

Conversely, traditional FL approaches exhibit the highest energy usage among the devices. This is largely attributable to the substantial computational demands of resource-intensive models in FL. The burden of processing these complex models on the devices leads to heightened energy requirements, underscoring the advantage of DFL's more resource-aware approach in IoT training environments.

V. CONCLUSION

In this article, we introduced DFL, a novel solution designed to tackle the challenges arising from heterogeneous data and resource landscapes in edge IoT environments. DFL effectively counters the dual challenges of heterogenous, usually non-IID, data and the diversity in computing and communication resources, which are major obstacles in traditional FL methodologies. DFL offers resource-aware split computing tailored for deep neural networks coupled with the dynamic clustering of training participants. The latter is achieved through layerwise neural network similarity, facilitated by CKA, allowing for efficient training processes without sacrificing data privacy. By clustering IoT devices based on the similarity of their sub-model layers, DFL adeptly addresses data heterogeneity and enhances the use of available computational resources across IoT devices.

Empirical evaluation of DFL conducted on a real testbed of IoT devices with non-IID datasets has shown significant improvements in training time, accuracy, and energy usage, particularly when benchmarked against classic FL and SFL approaches. These results underscore the efficacy of DFL in enhancing the performance of distributed learning IoT environments. By enabling more efficient and effective distributed model training, DFL has the potential to accelerate the adoption of FL in real-world IoT systems, paving the way for more intelligent, responsive, and efficient IoT solutions.

REFERENCES

[1] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[2] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[3] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained iot devices: Panoramas and state-of-the-art," *arXiv preprint arXiv:2002.10610*, 2020.

[4] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, pp. 1273–1282, PMLR, 2017.

[6] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[7] Y. Li, D. Yuan, A. S. Sani, and W. Bao, "Enhancing federated learning robustness in adversarial environment through clustering non-iid features," *Computers & Security*, p. 103319, 2023.

[8] B. Li, M. N. Schmidt, T. S. Alstrøm, and S. U. Stich, "On the effectiveness of partial variance reduction in federated learning with heterogeneous data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3964–3973, 2023.

[9] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[10] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-iid data quagmire of decentralized machine learning," in *International Conference on Machine Learning*, pp. 4387–4398, PMLR, 2020.

[11] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.

[12] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[13] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "Fedadapt: Adaptive offloading for iot devices in federated learning," *arXiv preprint arXiv:2107.04271*, 2021.

[14] E. Samikwa, A. Di Maio, and T. Braun, "Disnet: Distributed micro-split deep learning in heterogeneous dynamic iot," *IEEE internet of things journal*, 2023.

[15] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–9, IEEE, 2020.

[16] Z. Gui, X. Yang, H. Yang, W. Li, L. Zhang, Q. Qi, J. Wang, H. Sun, and J. Liao, "Grouping synchronous to eliminate stragglers with edge computing in distributed deep learning," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 429–436, IEEE, 2021.

[17] C. Li, G. Li, and P. K. Varshney, "Federated learning with soft clustering," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7773–7782, 2021.

[18] T. Liu, J. Ding, T. Wang, M. Pan, and M. Chen, "Towards fast and accurate federated learning with non-iid data for cloud-based iot applications," *arXiv preprint arXiv:2201.12515*, 2022.

[19] A. Li, J. Sun, X. Zeng, M. Zhang, H. Li, and Y. Chen, "Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 42–55, 2021.

[20] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, "Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training," *arXiv preprint arXiv:2206.00187*, 2022.

[21] Z. Tang, S. Shi, B. Li, and X. Chu, "Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 909–922, 2022.

[22] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," *arXiv preprint arXiv:2003.13376*, 2020.

[23] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, 2020.

[24] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[25] L. Tu, X. Ouyang, J. Zhou, Y. He, and G. Xing, "Feddl: Federated learning via dynamic layer sharing for human activity recognition," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 15–28, 2021.

[26] L. Yang, J. Huang, W. Lin, and J. Cao, "Personalized federated learning on non-iid data via group-based meta-learning," *ACM Transactions on Knowledge Discovery from Data*, vol. 17, no. 4, pp. 1–20, 2023.

[27] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[28] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *arXiv preprint arXiv:2103.04505*, 2021.

[29] H. Song, J. Bai, Y. Yi, J. Wu, and L. Liu, "Artificial intelligence enabled internet of things: Network architecture and spectrum access," *IEEE Computational Intelligence Magazine*, vol. 15, no. 1, pp. 44–51, 2020.

[30] L. Lockhart, P. Harvey, P. Imai, P. Willis, and B. Varghese, "Scission: Context-aware and performance-driven edge-based distributed deep neural networks," *arXiv e-prints*, pp. arXiv–2008, 2020.

[31] E. Samikwa, A. Di Maio, and T. Braun, "Adaptive early exit of computation for energy-efficient and low-latency machine learning over iot networks," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 200–206, IEEE, 2022.

[32] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pp. 250–260, IEEE, 2021.

[33] C. Thapa, M. A. P. Chamikara, and S. Camtepe, "Splitfed: When federated learning meets split learning," *arXiv preprint arXiv:2004.12088*, 2020.

[34] A. Abedi and S. S. Khan, "Fedsl: Federated split learning on distributed sequential data in recurrent neural networks," *arXiv preprint arXiv:2011.03180*, 2020.

[35] Y. Mu and C. Shen, "Communication and storage efficient federated split learning," in *ICC 2023-IEEE International Conference on Communications*, pp. 2976–2981, IEEE, 2023.

[36] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19586–19597, 2020.

[37] Y. Kim, E. Al Hakim, J. Haraldson, H. Eriksson, J. M. B. da Silva, and C. Fischione, "Dynamic clustering in federated learning," in *ICC 2021-IEEE International Conference on Communications*, pp. 1–6, IEEE, 2021.

[38] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[39] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670, 2014.

[40] E. Samikwa, A. Di Maio, and T. Braun, "Ares: Adaptive resource-aware split learning for internet of things," *Computer Networks*, vol. 218, p. 109380, 2022.

[41] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *International Conference on Machine Learning*, pp. 3519–3529, PMLR, 2019.

[42] H. M. Son, M. H. Kim, and T.-M. Chung, "Compare where it matters: Using layer-wise regularization to improve federated learning on heterogeneous data," *arXiv preprint arXiv:2112.00407*, 2021.

[43] C. Cortes, M. Mohri, and A. Rostamizadeh, "Algorithms for learning kernels based on centered alignment," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 795–828, 2012.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv e-prints arXiv:1409.1556*, 2014.

[45] X. Liu, W. Yu, F. Liang, D. Griffith, and N. Golmie, "Toward deep transfer learning in industrial internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12163–12175, 2021.
[46] Z. Bi, L. Yu, H. Gao, P. Zhou, and H. Yao, "Improved vgg model-based efficient traffic sign recognition for safe driving in 5g scenarios," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 11, pp. 3069–3080, 2021.
[47] Y. Gao, M. Kim, C. Thapa, S. Abuadbba, Z. Zhang, S. A. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for internet of things," *arXiv preprint arXiv:2103.02762*, 2021.
[48] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.
[49] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.
[50] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

Appendix A
CONVERGENCE ANALYSIS

An important aspect of this analysis is that the standard FSL approach primarily affects the distribution of the training tasks rather than the model convergence itself [33]. The SL component allows for efficient distribution of model computation, but the fundamental convergence properties stem from the FL framework and the conditions assumed [16]. The cluster-level training in DFL capitalizes on the inherent data similarities, enabling more efficient and focused learning.

The key assumptions stem from FL: We assume that the gradients of each local loss function are ζ-smooth:

$$F_\phi(\omega^{(k+1)}) \leq F_\phi(\omega^{(k)}) + \nabla F_\phi(\omega^{(k)})^T (\omega^{(k+1)} - \omega^{(k)}) + \frac{\zeta}{2}\|\omega^{(k+1)} - \omega^{(k)}\|^2 \quad (20)$$

The variance of the stochastic gradients is bounded, maintaining control over the randomness in local updates:

$$\mathbb{E}\|\nabla F_\phi(\omega) - \nabla F(\omega)\|^2 \leq \sigma_g^2 \quad (21)$$

We start by examining the local updates within each cluster. For participant ϕ in cluster g, the update is:

$$\omega_{g,\phi}^{(k+1)} = \omega_{g,\phi}^{(k)} - \eta \nabla F_\phi(\omega_{g,\phi}^{(k)}) \quad (22)$$

By leveraging the smoothness assumption, the expected improvement in the local loss function is bounded. Specifically, we have:

$$\mathbb{E}[F_\phi(\omega_{g,\phi}^{(k+1)})] \leq F_\phi(\omega_{g,\phi}^{(k)}) - \left(\eta - \frac{\eta^2\zeta}{2}\right) \mathbb{E}\|\nabla F_\phi(\omega_{g,\phi}^{(k)})\|^2 + \frac{\eta^2\sigma_k^2}{2} \quad (23)$$

Within each cluster, the client-side updates are aggregated to form the updated group model following Equation 5. Using this aggregated model, the expected loss function for the cluster $\mathcal{L}_g(\omega_g)$ decreases over time. The convergence rate can be expressed as:

$$\mathbb{E}[\mathcal{L}_g(\omega_g^{(k+1)})] - \mathcal{L}_g^* \leq \left(1 - \frac{\eta\mu_g}{|g|}\right)(\mathbb{E}[\mathcal{L}_g(\omega_g^{(k)})] - \mathcal{L}_g^*) + \frac{\eta^2\sigma_g^2}{2|g|} \quad (24)$$

where μ_g is the strong convexity parameter and the decay rate of the error term $(1 - \frac{\eta\mu_k}{|g|})$ depends on the learning rate.

Since clusters remain separate and train independently, we here consider the global performance as a weighted sum of the individual cluster performances. The global objective function $\mathcal{L}(G)$ converges as:

$$\mathbb{E}[\mathcal{L}(G)] - \mathcal{L}^* \leq \sum_{g\in G}\left(1 - \frac{\eta\mu_g}{|g|}\right)^R (\mathbb{E}[\mathcal{L}_g(\omega_g^0)] - \mathcal{L}_g^*) + \frac{\eta\sigma^2}{2\mu} \quad (25)$$

where each cluster's error contribution decays exponentially with the rate $(1 - \frac{\eta\mu_g}{|g|})$ over R training rounds.

Thus, the model training in DFL converges under standard assumptions of smoothness and bounded variance. Clustering participants using CKA on model parameters early in the training process enhances convergence by reducing the variance of updates within clusters.