Dynamic Optimization of Edge Aggregation Structures and Update Frequencies for Efficient Distributed Hierarchical Model Training

Xiaolong Xu, *Senior Member, IEEE*, Jiayang Sun, Guangming Cui, Lianyong Qi, *Senior Member, IEEE*, Muhammad Bilal *Senior Member, IEEE*, Wanchun Dou, Zhipeng Cai, *Fellow, IEEE* and Jon Crowcroft, *Fellow, IEEE*

***Abstract*—Edge computing enables distributed machine learning models to be deployed and trained near the user space. However, the intricate nature of edge computing raises several challenges to distributed machine learning frameworks: 1) inferior convergence arising from non-independent and identically distributed (non-IID) edge data; 2) inefficient structural adaptation, where device dynamism complicates the adjustment of aggregation structure; and 3) reduced training efficiency, as resource heterogeneity and fluctuations create systemic stragglers. To address these issues, a distributed hierarchical model training framework has been proposed by considering the dynamic aggregation structure and frequency in this paper. This framework designs an Edge Aggregation Structure and Frequency method, namely EASF, for distributed model training in heterogeneous edge computing environments. First, a dynamic distributed aggregation structure method is formulated to consider various data distribution patterns. This method constructs and modifies the aggregation structure in a distributed manner to adapt to variations in working edge devices. Second, a self-adapted aggregation frequency method and a timeout abandonment mechanism are proposed to allow each node to update its aggregation frequency adaptively. Lastly, a theoretical analysis demonstrates the convergence property of the EASF method in dynamic environments. Extensive experiments have been conducted on a set of open testbeds. Results show that the EASF significantly improves the efficiency and accuracy of hierarchical model training in heterogeneous edge computing.**

***Index Terms*—Edge computing, distributed machine learning, hierarchical model training**

X. Xu, J. Sun, and G. Cui are with the School of Software, Nanjing University of Information Science & Technology, China. X. Xu and G. Cui are also with the Jiangsu Province Engineering Research Center of Advanced Computing and Intelligent Services.
E-mail: xlxu@ieee.org; {202412211536, gcui}@nuist.edu.cn.

L. Qi is with the College of Computer Science and Technology, China University of Petroleum (East China), Qingdao, China. E-mail: lianyongqi@gmail.com.

Muhammad Bilal is with the School of Computing and Communications, Lancaster University, Bailrigg, Lancaster LA1 4WA, United Kingdom. E-mail: m.bilal@ieee.org.

W. Dou is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: douwc@nju.edu.cn.

Z. Cai is with the Department of Computer Science, Georgia State University, USA. Email: zcai@gsu.edu

Jon Crowcroft is with the Department of Computer Science and Technology, University of Cambridge, United Kingdom. E-mail: jon.crowcrf@cl.cam.ac.uk.

Manuscript received xxx; revised xxx.

This work was supported in part by the National Natural Science Foundation of China under Grant 92267104 and Grant 62372242, and in part by the Jiangsu Provincial Major Project on Basic Research of Cutting-edge and Leading Technologies, under Grant BK20232032.

I. Introduction

EDGE computing, an emerging computing paradigm [1], decentralizes computational tasks from a remote cloud to edge servers [2]. This shift towards localized data processing facilitates services with low latency, high reliability, and high privacy [3]. Furthermore, edge computing empowers artificial intelligence with the capability to process data at the network's edge and train models distributedly [4]–[7].

The evolution of edge computing has highlighted the potential and value of distributed machine learning [8], [9]. This approach delegates the execution of training tasks to edge devices, which retrieve model data from edge servers and use local data for training [10]. Leveraging the computing and storage capabilities of edge devices, distributed machine learning methods can enhance service quality and efficiency [11], [12]. They provide intelligent support for various application fields, such as intelligent perception, prediction, and optimization, security, and privacy [13].

Federated Learning (FL) [14] is a representative distributed machine learning framework applied to edge computing. It employs a centralized structure where workers are trained using local data, and model parameters are aggregated on a central server [15]. When the number of training devices is large, a substantial volume of model parameters needs to be transmitted across the network in multiple rounds, potentially leading to network congestion and communication failures. The central server faces significant communication pressure and is the single point of failure. Additionally, FL is susceptible to aggregation delays caused by stragglers, resulting in decreased overall training efficiency [16]. To address these issues, several decentralized structures have been proposed, such as Gossip Learning (GL) [17], [18]. Training devices are permitted to aggregate model parameters from other devices, effectively mitigating single points of failure. However, when the data is non-Independent and Identically Distributed (non-IID) [19], GL, which lacks a model synchronization mechanism, tends to yield inferior convergence [20].

Hierarchical model training frameworks, such as Hierarchical Federation Learning (HFL) [21] and E-Tree learning [16], introduce intermediate levels to perform partial aggregation

0000–0000/00$00.00

operations, which effectively alleviates the communication load on the central node and mitigates the impact of single-point failures [22]. However, to achieve optimal performance in realistic edge environments, several critical and interconnected challenges must be addressed. First, data is typically non-IID and varies significantly in quality across different devices. Designing an appropriate aggregation structure that reduces both communication costs and the model deviation caused by this data disparity remains a considerable challenge. Second, this challenge is amplified by device dynamism, as nodes frequently join and leave. Existing adaptation methods often rely on a central controller, which requires extra deployment overhead, may be slow to respond to such changes, and introduces a fatal single point of failure. Third, beyond structural concerns, the combination of resource heterogeneity and dynamic fluctuations in device capabilities, such as available computational power and bandwidth, makes setting an optimal aggregation frequency a major hurdle. Any fixed frequency inevitably leads to systemic inefficiencies, forcing faster devices to idle while waiting for stragglers and ultimately degrading overall training performance. These issues highlight a pressing need for a framework without centralized control, capable of supporting both on-the-fly structural management and locally adaptive update scheduling.

To address these challenges in edge computing environments, we propose a distributed hierarchical model training framework featuring the Edge Aggregation Structure and Frequency (EASF) method. EASF employs a Distributed aggregation Structure Formation (DSF) method to manage the aggregation structure and an Adaptive Adjustment of aggregation Frequency (AAF) method, complemented by a timeout mechanism, to orchestrate the training process. The main contributions of this paper are as follows:

- We propose the DSF method, where edge devices construct and adjust the aggregation structure in a distributed manner. This approach is specifically designed to handle the challenges of non-IID data and device dynamism, while eliminating the deployment overhead and single-point-of-failure risks of a central controller.
- We present the AAF method, complemented by a timeout abandonment mechanism. It enables each node to adaptively adjust its aggregation frequency through real-time, collaborative decision-making between parent and child nodes, thereby maximizing resource utilization and improving training efficiency.
- We conducted a theoretical analysis of the EASF, showing its convergence in dynamic edge computing environments.
- We conduct extensive experiments in simulated dynamic edge computing environments. The results show that our method achieves higher model accuracy and faster convergence speed compared to existing methods.

II. Related Work

The heterogeneity of data and resources in edge computing environments presents significant challenges to the performance of distributed machine learning. To address these issues, researchers have explored solutions from various perspectives. One line of research focuses on mitigating heterogeneity within traditional centralized frameworks through methods such as client selection and automatic grouping [23]–[26]. Another promising approach involves leveraging hierarchical architectures to enhance scalability and reduce communication bottlenecks [20], [27], [28]. A third, complementary direction concentrates on optimizing a critical training parameter—the aggregation frequency—to dynamically balance local computation with global synchronization in response to environmental changes [16], [29]–[32].

A. Addressing Heterogeneity in Distributed Machine Learning

Data heterogeneity primarily manifests in distribution, scale, and quality, while resource heterogeneity is reflected in computational power, storage capacity, and communication bandwidth. The development of suitable distributed machine learning methods to alleviate the impact of heterogeneity on model training has become a pivotal focus of numerous research endeavors.

For instance, Ma et al. [23] proposed an iterative algorithm to optimize the joint utilization of storage and computation capacities, which in turn reduces service response time and outsourcing traffic. Fraboni et al. [24] introduced a client selection scheme that groups clients into different sampling distributions, enhancing the convergence stability and quality of FL. Liu et al. [25] developed an innovative communication method named FedCPF for Vehicle Edge Computing. This method formulates local training strategies and participation rules for clients, resulting in efficient communication and quicker convergence. He et al. [26] proposed an algorithm named Auto-Group, which automatically groups users based on data distribution, thereby speeding up the training process.

The aforementioned methods are primarily applied to distributed model training with a centralized structure. However, this structure faces significant challenges in balancing resource utilization and communication pressure, especially as the number of devices scales.

B. Leveraging Hierarchies to Mitigate Heterogeneity

The hierarchical model training framework possesses the ability to classify clients based on distance or resource. It excels in diminishing communication overhead and accommodating non-IID data. During the design phase of a hierarchical structure, several considerations are paramount, including the criteria for grouping and the strategy for model aggregation.

For example, Hu et al. [20] introduced a hierarchical architecture called Spread, which distributes the load of model aggregation, ensuring high-quality model training. Lim et al. [27] proposed a dynamic resource allocation framework for HFL. This framework optimizes resource distribution and incentivizes participation, ultimately enhancing the performance of edge intelligence systems. You et al. [28] proposed a hierarchical personalized federated learning algorithm named HPFL. By jointly optimizing the scheduling strategy and bandwidth allocation of edge servers, HPFL strikes a balance between training loss and round delay.

The aforementioned works effectively distribute the aggregation load and enable dynamic resource allocation, showing clear advantages in scalability and efficiency. Therefore, acknowledging these benefits, the hierarchical structure is employed for model training in this paper.

C. Adaptive Frequency in Heterogeneous Environments

The aggregation frequency is a critical parameter in distributed model training, directly influencing the trade-off between communication overhead and model convergence speed. To improve training efficiency, especially in heterogeneous edge computing environments, several studies have focused on optimizing this frequency.

In the context of FL for edge computing, Wang et al. [29] introduced an approach that adjusts a single, identical frequency for all clients to balance computation and communication. Xu et al. [30] proposed to jointly optimize a uniform local update frequency and model compression ratio for all clients based on a theoretical linkage. Yan et al. [31] adapted the local update frequency to balance the training of different model parts created by neural composition, a technique used to serve clients with varying model sizes. For the client-edge-cloud hierarchical network, Luo et al. [32] proposed to adaptively compute different frequencies for nodes to equalize their round completion times based on a benchmark calculated by the cloud server. Yang et al. [16] proposed the KMA and RAF algorithm to construct a hierarchical structure and determine a fixed aggregation frequency based on a pre-training phase.

However, these methods often rely on a centralized controller to determine the aggregation structure and frequency, or to force all clients to align with a uniform or benchmark-derived frequency, which can lead to resource underutilization. RAF is primarily designed for static edge computing environments and lacks adaptability to dynamic changes in device status and available resources. In contrast, our proposed method in this paper addresses these limitations by introducing a framework that employs distributed control, allowing each node to collaboratively adjust its frequency in real-time with its parent, ensuring that model training maintains high efficiency and accuracy in dynamic edge environments.

III. Theoretical Foundations and Problem Formulation

This section formulates the distributed hierarchical model training framework in an edge computing environment from the edge devices' perspective. The ideas employed in this paper can be applied to other hierarchical frameworks easily. Now, we delve into the theoretical foundations of hierarchical model training and outline the problem formulation.

A. Theoretical Foundations

Fig. 1 depicts the structure of hierarchical model training. This structure follows a tree-based approach, where leaf nodes correspond to training nodes, and non-leaf nodes function as aggregation nodes.

To illustrate a conventional approach to hierarchical structure construction, we take the K-Means and average accuracy

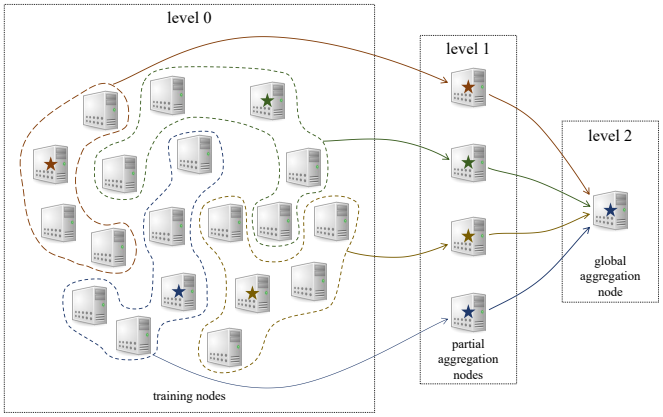


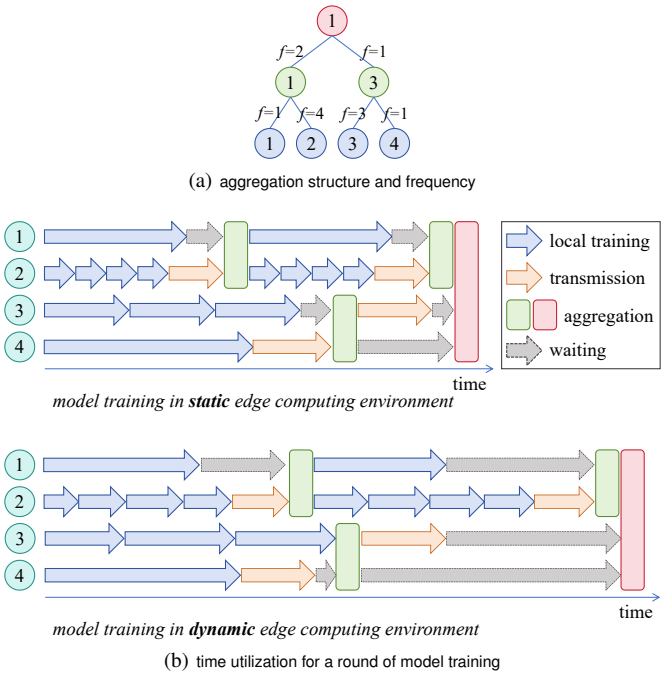
Fig. 1. An example of hierarchical model training structure.

(KMA) algorithm [16] as a representative example. This algorithm enables an edge device to serve as both a training node and multiple aggregation nodes within a hierarchical structure. Initially, edge devices are divided into several clusters based on the network topology and data distribution. The clustering principle considers communication delay and data distribution, where the difference in data distribution is quantified by the pre-training accuracy of each device. Subsequently, a central device is selected as the aggregation node for each cluster, with the criterion being to minimize the overall communication delays between the central device and other devices within the cluster, i.e.,

$$c_l^h = \underset{n_i^h \in C_l^h}{\operatorname{argmin}} \sum_{n_j^h \in C_l^h, i \neq j} d_{i,j}, \tag{1}$$

where C_l^h denotes cluster l at level h, c_l^h is the central node of C_l^h, n_i^h is a node of device v_i at level h, and $d_{i,j}$ is the communication delay between devices v_i and v_j. Then, employing a bottom-up approach, the central device is recursively selected as the aggregation node for each cluster until the root node is chosen. A key characteristic of this approach is its reliance on a centralized controller for construction, resulting in a static structure that is less responsive to the frequent topological changes in dynamic edge environments.

Upon the construction of a tree-based aggregation structure, the hierarchical model training can be initiated. The training process follows a bottom-up approach. Initially, each leaf node utilizes its dataset for training, updates the model parameters locally, and transmits the model parameters to the parent node. After receiving the model parameters sent by each child node, the non-leaf node averages them to aggregate the model. If the preset number of updates is not reached, the updated model is dispatched to each child node to continue training. When the preset number of updates is achieved, the model parameters are transmitted to the parent node. By analogy, when the root node receives the model parameters from each child node and accomplishes the global model aggregation, a complete round of training concludes. After that, the root node disseminates the global model to each node via the aggregation structure for the next round of training.



(a) aggregation structure and frequency

(b) time utilization for a round of model training

Fig. 2. Time utilization of static aggregation frequency in hierarchical model training: Impact of static vs. dynamic edge environments on stragglers and idle time.

Regarding the preset of the number of updates mentioned above, several weak synchronization methods have been proposed, such as the Resource-based Aggregation Frequency controlling (RAF) algorithm [16]. During the pre-training phase, each aggregation node collects the calculation and transmission time of all its child nodes. The aggregation node determines the time required for this aggregation by considering the total time of the child node with the longest duration. Subsequently, it calculates the number of updates that can be performed for each child node, which corresponds to the aggregation frequency. Crucially, this frequency is determined based on a one-time pre-training phase, rendering it a fixed value that cannot adapt to real-time fluctuations in device resources.

The aggregation frequency set by the RAF algorithm may be as shown in Fig. 2a. Given a static edge computing environment, where the computational and communication resources of the devices remain constant, the time utilization of hierarchical model training, which employs the aggregation structure and frequency in Fig. 2a, is presented in the top figure in Fig. 2b. Among them, nodes of different colors in Fig. 2a perform different tasks, and the colors of the processes in Fig. 2b correspond to the tasks of the same colors in Fig. 2a. Each device performs maximum updates within a restricted timeframe, thereby accelerating the convergence speed of the global model.

However, the actual edge computing environment is inherently dynamic, with potential variations in numerous factors, including the working status of devices, computational resources, and channel bandwidth. In such environments, structure formation methods that rely on a central controller can be slow to adapt to device changes and may introduce a fatal single point of failure. Furthermore, adhering to a preset aggregation frequency, calculated based on a prior state, can prove suboptimal. This may lead to the scenario depicted in the bottom figure in Fig. 2b, where resource allocation is inefficient, causing some devices to idle for extended periods and thereby diminishing overall training efficiency. These limitations motivate our work to design a framework that overcomes these challenges, the formal problem of which is defined next.

B. Problem Formulation

We characterize a dynamical edge computing environment with N edge devices as $E_t = \{G_{E_t}, P_{E_t}, O_{E_t}\}$, which encompasses:

- **Network Topology Graph:** Denoted by $G_{E_t} = \{V_{E_t}, B_{E_t}\}$, where $V_{E_t} = \{v_i\}_{i=1}^{N}$ represents the set of edge devices, and $B_{E_t} = \{b_{i,j}\}(i,j \in [1,N])$ signifies the network bandwidths between these devices. Here, $b_{i,j}$ indicates the bandwidth value between devices v_i and v_j. Although G_{E_t} is a connected graph, it may not be fully connected, as devices may establish connections within the network through multi-hop communication.
- **Computational Power:** Denoted by $P_{E_t} = \{p_i\}_{i=1}^{N}$, where p_i is the available computational power of device v_i, which may fluctuate due to other tasks.
- **Working Status:** Denoted by $O_{E_t} = \{o_i\}_{i=1}^{N}$, where o_i is the working status of device v_i.

The aggregation structure $S_t = \{C_l^h, c_l^h\}(h \in [0, H-1])$ delineates the hierarchical arrangement, comprising H levels. In this context, $C_l^h = \{n_i^h\}(i \in [1,N])$ constitutes the l-th cluster at level h. Each node in this cluster shares a common parent node, denoted as c_l^h. An edge device v_i may be depicted as a node within S_t multiple times, albeit only once per level.

The hierarchical model training follows a weak synchronization protocol. Here, f_i^h signifies the aggregation frequency, indicating the number of updates that n_i^h executes prior to transmitting data to its parent node. For leaf nodes, f_i^h corresponds to local training iterations; for non-leaf nodes, it relates to model aggregations.

Initially, each device commences with identical model parameters $\boldsymbol{\omega}$, and the loss function is denoted by $F(\boldsymbol{\omega})$. During the r-th training round, operations adhere to the aggregation structure S_t. For the node n_i^0 in cluster C_l^0 at level 0, undertakes training using its dataset D_i, updating its model parameters as follows:

$$\boldsymbol{\omega}_i^m = \boldsymbol{\omega}_i^{m-1} - \alpha_i \nabla F(\boldsymbol{\omega}_i^{m-1}), \tag{2}$$

where $\boldsymbol{\omega}_i^m$ is the model parameters after the m-th update, and α_i represents the learning rate. After f_i^0 local updates, the updated model parameters $\boldsymbol{\omega}_i^{f_i^0}$ are transmitted to the parent node c_l^0.

For the cluster C_l^h at the h-th level, assuming that the device corresponding to the central node c_l^h is v_i, then the aggregation node of C_l^h is n_i^{h+1}. It gathers model parameters from all nodes in C_l^h and aggregates them by:

$$|D_{i,h+1}| = \textstyle\sum_{n_j^h \in C_l^h} |D_{j,h}|, \tag{3}$$

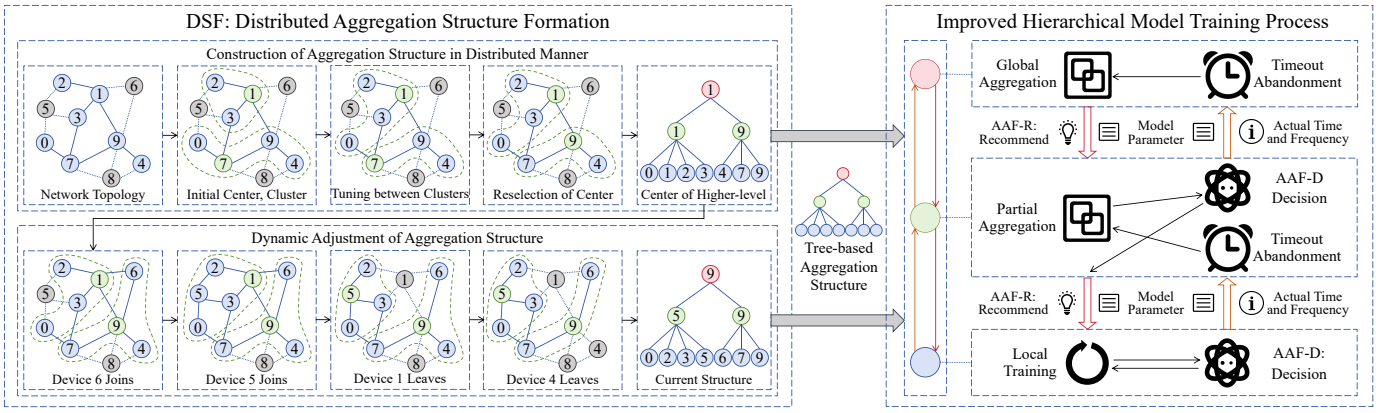


Fig. 3. The overall structure of the EASF method.

$$\boldsymbol{\omega_i^m} = \frac{\sum_{n_j^h \in C_l^h} |D_{j,h}| \, \boldsymbol{\omega}_j^{f_j^0}}{|D_{i,h+1}|}, \tag{4}$$

where n_j^h is a node in cluster C_l^h, $|D_{j,h}|$ is the aggregate of sample counts from all child nodes of device v_j at level h. Specifically, when $h = 0$, $|D_{j,0}|$ signifies the number of samples of D_j. After performing the aggregation, device v_i disseminates the model parameters to its descendant nodes along S_t. Subsequent to f_i^h iterations of aggregation, v_i uploads the model parameters to its parent node.

By analogy, when the root aggregation node finalizes the aggregation, this round of training concludes. The global model parameters are signified as $\boldsymbol{\omega}_G^r$. The time spent on this training round, represented as T_r, refers to the total duration from the start of model parameter dissemination by the root aggregation node to the completion of global aggregation. For the subsequent training round, the root node distributes the model parameters to all descendant nodes via S_t.

After R training rounds, the training duration T_{total} and the final global model parameters $\boldsymbol{\omega}_G^{total}$ are defined as follows:

$$T_{total} \triangleq \sum_{r=1}^{R} T_r, \tag{5}$$

$$\boldsymbol{\omega}_G^{total} \triangleq \underset{\boldsymbol{\omega} \in \{\boldsymbol{\omega}_G^r, r=1,2,\ldots,R\}}{\operatorname{argmin}} F(\boldsymbol{\omega}). \tag{6}$$

The optimization challenge is to construct and adjust the optimal aggregation structure S_t and aggregation frequency f_i^h for each device. The goal is to minimize the global loss function of the model within the given time T:

$$\min_{S_t, f_i^h} F\left(\boldsymbol{\omega}_{total}\right), \quad s.t. \quad T_{\text{total}} \leq T. \tag{7}$$

IV. Edge Aggregation Structure and Frequency

In this section, we propose a novel method, EASF, designed for the construction and adjustment of the edge aggregation structure and frequency within a dynamic edge computing environment. EASF is applicable to most scenarios involving hierarchical model training, regardless of whether privacy protection is required. This is because it maintains privacy by transmitting only model parameters and other necessary content between nodes, rather than local data. The overall structure is shown in Fig. 3.

The left half of Fig. 3 presents the Distributed aggregation Structure Formation (DSF) method. The upper portion outlines the structure construction process, where devices in the network iteratively determine clusters and central nodes in a distributed manner, resulting in a tree-based aggregation structure. The lower portion describes the dynamic adjustment of the aggregation structure, accommodating changes in the edge environment when devices join or leave. Detailed explanations of DSF are provided in Subsection IV-A.

The right half of Fig. 3 illustrates the improved hierarchical model training process, which incorporates the Adaptive Adjustment of aggregation Frequency (AAF) method and a timeout abandonment mechanism into the basic training process. The AAF method consists of two components: AAF-R (where the parent node recommends the aggregation frequency for child nodes) and AAF-D (where the child node decides the aggregation frequency). The timeout abandonment mechanism allows the parent node to discard model data from laggards. Subsection IV-B provides a comprehensive discussion of these methods.

A. Dynamic Distributed Aggregation Structure

In this subsection, we introduce a distributed method for dynamically constructing and adjusting the tree-based aggregation structure. This method aims to simultaneously achieve the following two objectives:

- Minimize communication consumption by ensuring low delays and hops between devices within each cluster.
- Enhance the efficiency of model training by maintaining the data distribution of each cluster close to the global.

Aggregation Structure Constructing. Before constructing the aggregation structure, each edge device is required to undergo the same number of epochs of pre-training to determine its respective pre-training accuracy. The pre-training aims to measure sample distribution via pre-training accuracy to enable effective clustering. The number of epochs is predetermined based on the selected model to ensure sufficient accuracy differences among devices are revealed with low computational cost. Each device only needs to undergo one

round of pre-training, which can be completed at any time before participating in the formal training. Additionally, it needs to engage in communication with proximate nodes to obtain both its own degree and the degrees of neighboring nodes within the network topology. Following a period of information exchange, each node can accumulate a substantial amount of pre-training accuracy data from other nodes, thereby enabling the calculation of the global average pre-training accuracy, denoted as ACC.

We strive for each cluster's data distribution to resemble the global distribution. However, due to privacy constraints, direct sample data transmission between devices is often prohibited. To assess sample distribution, we rely on pre-trained accuracy. Clusters with an average pre-trained accuracy close to the global average ACC tend to exhibit more uniform data distribution and better training performance [16]. To quantify the impact of device variations within a cluster on the average pre-trained accuracy, we design the following formula:

$$\delta = |Acc_{old} - ACC| - |Acc_{new} - ACC|, \tag{8}$$

where Acc_{old} represents the average pre-trained accuracy before the cluster change, and Acc_{new} represents the accuracy after the change. A positive δ indicates that the average pre-trained accuracy after the cluster change is closer to ACC, while a negative δ indicates the opposite. Larger δ values suggest greater improvement in the alignment between cluster average pre-trained accuracy and ACC after the change.

The detailed process of constructing the aggregated structure is as follows:

Central Node Selecting: The selection of the central node is predicated on the two-hop probability, which is similar to the method in paper [33]. Initially, each device computes its two-hop return probability as follows:

$$THP(v_i) = \sum_{v_j \in Nbr(v_i)} \left(\frac{1}{Deg\,(V_i) \times Deg\,(v_j)} \right), \tag{9}$$

where $Nbr(v_i)$ represents the set of neighboring nodes of device v_i, and $Deg\,(v_i)$ denotes its degree. Devices with higher THP values are more likely to return to themselves within two hops when transmitting messages, indicating that they have more neighbors in the network topology. Compared to devices with lower THP values, selecting such devices as initial central nodes will reduce communication overhead and improve communication efficiency during the clustering process. Consequently, devices with THP values exceeding those of all their neighboring devices are designated as central nodes.

Initial Clustering Building: Along the network topology diagram, the central node dispatches messages to other nodes, prompting them to select a cluster. The non-central node waits for a certain duration to receive these messages and then selects the cluster associated with the central node that offers the least communication delay, provided it is within a limited hop count. If no central node falls within the hop count, the non-central node opts for the central node with the fewest hops. Next, the non-central node sends a message to the selected node expressing the intent to join its cluster. After dispatching the information messages, the central node waits for a certain period before incorporating all nodes that have expressed interest in joining its cluster.

Algorithm 1 Generation of Transferable Nodes List

Input: Pre-training accuracy $\{acc_i\}$ of all nodes in the cluster C_l^0, the global average pre-training accuracy ACC, the rate $\rho_e(\rho_e > 1.0)$ of excess size
Output: A list TN_l of transferable nodes

1: Initialize TN_l as an empty list
2: $Acc_l = \sum_{n_i^0 \in C_l^0} acc_i / \text{Size}(C_l^0)$
3: **for** node n_j^0 in C_l^0 **and** n_j^0 is not c_l^0 **do**
4: $\quad Acc_{l\_j} = \sum_{n_i^0 \in C_l^0, i \neq j} acc_i / (\text{Size}(C_l^0) - 1)$
5: $\quad$ calculate δ_j by Equation (8)
6: $\quad$ **if** $\delta_j > 0$ **then**
7: $\quad\quad$ Append n_i^0 to TN_l
8: $\quad$ **end if**
9: **end for**
10: **if** $\text{Size}(C_l^0)$ exceeds ρ_e times the average cluster size of nearby central nodes **then**
11: $\quad$ **while** $\text{Size}(TN_l)$ is less than half of $\text{Size}(C_l^0)$ **do**
12: $\quad\quad$ Identify the node n_k^0 with the maximum hops from the central node in C_l^0 and not in TN_l
13: $\quad\quad$ Append n_k^0 to TN_l
14: $\quad$ **end while**
15: **end if**

Adjusting between Clusters: The central node obtains the cluster sizes (i.e., the number of nodes in the cluster) of nearby central nodes through communication within a certain hop limit. If the size of its own cluster surpasses the average cluster size, it should generate a list of transferable nodes by Algorithm 1. The central node evaluates whether the removal of any non-self node will bring the cluster's average pre-training accuracy Acc_l closer to the global average ACC. If so, it adds the node to the list (lines 3-9). However, when Acc_l is already very close to ACC, removing any node may cause a deviation in the average value. If the cluster size is large, but the central node is reluctant to transfer nodes, smaller clusters nearby may struggle to acquire their source nodes, making it difficult to balance the data distribution of each cluster. Therefore, we mandate that the central node, whose cluster size exceeds a certain ratio of the average of the nearby central nodes', selects nodes distant from itself as the transferable node (lines 10-15).

Upon receiving a list of transferable nodes from another central node, the central node calculates the impact value δ of each node if it were to join the cluster, based on Equation (8). Considering hop count and communication delay constraints, the node with the highest positive δ value is selected. If the cluster size is significantly smaller than the nearby average cluster size and none of the transferable nodes are suitable for the cluster, the central node is compelled to select a node with the lowest hop count. After selecting a node, the central node notifies the source cluster center and refrains form selecting any other transferable node until it receives a response from the source central node.

The central node that sends the list of transferable nodes may receive responses from multiple central nodes desiring to incorporate certain nodes into their clusters. Provided that the cluster size does not fall below a specific ratio of the average cluster size of the nearby central nodes, it is sequentially determined whether the removal of the node can align the cluster's average pre-training accuracy more closely with the global average by Equation (8). The central node then issues a response indicating agreement or disagreement. If agreement is reached, these two central nodes update the cluster information and notify their neighboring central nodes.

In the event of a change in the cluster of the central node, or if several changes in nearby central nodes' clusters are detected, the central node attempts to regenerate and resend a list of transferable nodes, followed by subsequent operations. This process continues until there is no change in the cluster or after several iterations.

Central Node Reselecting: The central node of a cluster will be tasked with gathering information from all nodes within the cluster. To minimize communication consumption, it is crucial to select an appropriate central node. Following the initial clustering and adjustment phases, the current central node may not be the most suitable. Consequently, we allow the current central node to coordinate all nodes within the cluster to reselect the central node. Through a series of inter-cluster communications, each node obtains the delay associated with other nodes in the cluster and conveys this information to the central node. The central node then selects the new central node in accordance with Equation (1).

Since edge devices are permitted to serve as aggregation nodes, in practice, some edge devices may lack sufficient bandwidth, power, computational capability, storage capacity, or other resources to fulfill the role effectively. If such a device is selected as the cluster central node by Equation (1), it will be necessary to reselect a central node from the remaining devices within the cluster. Similarly, this principle applies to subsequent dynamic adjustments.

The new central nodes communicate with each other to obtain the communication delay between them. This allows the determination of the global central node according to Equation (1), completing the construction of the 3-level aggregation structure. If the total number of devices is too large, resulting in a significant number of central nodes (clusters), the 3-level aggregation structure may still impose substantial communication pressure on the global central node. In this case, the clustering method similar to the one previously described is employed to construct additional levels of clusters for the central nodes until the global central node is determined. During this process, the data distribution does not need to be considered.

Dynamic Adjustment of Aggregation Structure. In a dynamic edge computing environment, the working status of each device changes over time. Consequently, it becomes necessary to adapt the aggregation structure to ensure compatibility with hierarchical model training. In addition to the two objectives introduced at the beginning of this subsection, the adjustment also strives to maintain the parent-child relationships within the nodes of the tree-based aggregation structure. The rationale behind this will be further elaborated upon in subsequent content.

Departure Nodes Removing: When a device exits the training process, it may proactively inform other devices. However, there may be instances where it is unable to do so due to issues such as system failure. In such cases, the departure needs to be detected by neighboring nodes, which then take on the responsibility of notifying the relevant devices. If the departing device v_i is located solely at level 0 of the aggregation structure, it functions only as a training node n_i^0. Upon its cluster's central node c_l^0 in cluster C_l^0 becoming aware of its departure, c_l^0 can remove n_i^0 from C_l^0 and relay this information to other nodes.

If the departing device v_i appears at its maximum level $h_{max}(h_{max} \geq 1)$, it recurs across all levels from 0 to h_{max}. Assume that v_i serves as the central nodes of clusters from $C_{l\_0}^0$ to $C_{l\_h_{max}}^{h_{max}}$. The leaf node n_i^0 in $C_{l\_0}^0$ can still be directly removed from the cluster. However, the central node $c_{l\_0}^0$ necessitates reselection. If v_i is capable of sending messages upon departure, it should notify the node in $C_{l\_0}^0$ that has the least delay from it. This chosen node then acts as a temporary central node and organizes the cluster to reselect the central node $c_{l\_0}^0$ by Equation (1). If v_i is unable to send departure messages, the first node in its cluster that detects its departure assumes the role of a temporary central node to perform the same operation. Suppose the new central node $c_{l\_h}^h$ of the cluster $C_{l\_h}^h$ at level h, where v_i was originally located, is acted on by device v_j. Then v_j needs to communicate with other nodes at level $h+1$ and replace n_i^{h+1} with n_j^{h+1} in the corresponding cluster $C_{l\_h+1}^{h+1}$. If $h+1 < h_{max}$, it implies that v_i initially served as a cluster center at level $h+1$. In this scenario, the process of selecting a new central node and node replacement must be performed recursively until the h_{max} level is reached.

New Devices Integrating: The departure of devices may result in an increased disparity in data distribution across clusters, thereby slowing down the convergence speed of the model. Consequently, when a new device joins the training, it should be integrated into an appropriate cluster.

When device v_i wants to participate, it needs to communicate with central nodes of clusters at level 0 within a certain communication delay and hop count to ascertain their cluster size and average pre-training accuracy. Following this, v_i executes Algorithm 2 to select an appropriate cluster. It joins the cluster where its inclusion would result in the highest improvement of data distribution (lines 1-8). If there are no candidate clusters that satisfy the conditions of communication delay and hop count, v_i joins the cluster whose central node has the fewest hops from v_i (lines 9-12).

It is worth noting that the burden on coordinating nodes throughout the structure's construction and adjustment is minimal, owing to simple computations and lightweight information exchange. This inherently low overhead is complemented by our dynamic reselection mechanism, which ensures the central node role is not permanently fixed, preventing any single device from being disproportionately burdened over time. These design principles collectively enhance network fairness and stability, making the framework suitable for

Algorithm 2 Appropriate Cluster Selection

Input: The average pre-training accuracy $\{Acc_l\}$ of clusters $\{C_l^0\}$, the global average pre-training accuracy ACC, the pre-training accuracy acc_i of device v_i
Output: The cluster $C_{l^*}^0$ that v_i chooses to join
1: Initialize $C_{l^*}^0 =$ None, $max\_\delta = -100.0$
2: **for** cluster C_l^0 in $\{C_l^0\}$ **do**
3: $Acc_{l\_i} = \frac{Acc_l \times \text{Size}(C_l^0) + acc_i}{\text{Size}(C_l^0)+1}$
4: calculate δ_l by Equation (8)
5: **if** $\delta_l > max\_\delta$ **then**
6: $C_{l^*}^0 = C_l^0$, $max\_\delta = \delta_l$
7: **end if**
8: **end for**
9: **if** $C_{l^*}^0$ is None **then**
10: Identify the central node c_q^0 with the least number of hops from v_i through communication
11: $C_{l^*}^0 = C_q^0$
12: **end if**

Algorithm 3 Recommendation of Aggregation Frequency and Computation Time for Child Nodes

Input: Computation time t_{pred}^m and t_{real}^m, slow growth rate $\rho_g(\rho > 1.0)$, predicted time $t_{predavgcomp}^i$ and $t_{predtran}^i$ for child nodes
Output: Predicted computation time t_{pred}^{m+1}, recommended aggregation frequency rf_i^h and recommended computation time rt_{comp}^i
1: **if** t_{pred}^m is None **or** $t_{real}^m < t_{pred}^m$ **then**
2: $t_{pred}^{m+1} = t_{pred}^m$
3: **else**
4: $t_{pred}^{m+1} = \min(t_{real}^m, t_{pred}^m \times \rho_g)$
5: **end if**
6: **for** each child node n_i^h of the aggregation node **do**
7: $rt_{comp}^i = t_{pred}^{m+1} - t_{predtran}^i$
8: $rf_i^h = \left\lfloor \frac{rt_{comp}^i}{t_{predavgcomp}^i} \right\rfloor$
9: **end for**

resource-constrained and heterogeneous edge environments.

B. Hierarchical Model Training

In this subsection, we improve the basic hierarchical model training process. Our objective is to maximize device resource utilization and achieve superior results within the given time constraints. Subsequent content will detail each of the additional steps integrated into the training process.

Recommendation for Child Nodes (AAF-R). To dynamically adjust the aggregation frequency, we propose a method for calculating the recommended aggregation frequency and computation time for child nodes. This calculation is performed by the aggregation nodes. The procedure mainly consists of the following two steps:

Step 1: During the m-th aggregation, the aggregation node n_j^{h+1} within the cluster C_l^h collects the model parameters from each child node n_i^h. It also gathers their respective computation times t_{comp}^i, model transmission times $t_{download}^i$, t_{upload}^i, and the actual aggregation frequency f_i^h. The average computation time for each training or aggregation of each child node in this aggregation is then computed as:

$$t_{avgcomp}^i = \frac{t_{comp}^i}{f_i^h}. \tag{10}$$

The actual computation time expended by the aggregation node n_j^h in this aggregation is given by

$$t_{real}^m = \max_{n_i^h \in C_l^h} (t_{comp}^i + t_{download}^i + t_{upload}^i). \tag{11}$$

Considering that the time cost of model parameter aggregation is relatively small [16] - and such minor costs are often disregarded in time calculations - and the overhead of our AAF method is even smaller, we focus exclusively on the dominant costs of training and transmission. Therefore, the aggregation node predicts the average computation time $t_{predavgcomp}^i$, the model transmission time $t_{preddownload}^i$, and the model upload transmission time $t_{predupload}^i$ of each node in the $(m+1)$-th aggregation. The prediction model at this stage is designed to be modular and can be replaced by other forecasting algorithms. For this study, we employ exponential smoothing as a representative method due to its low computational overhead and effectiveness, making it suitable for resource-constrained edge environments. The formula is given as:

$$t_{m+1} = \beta t_{m+1} + (1-\beta) t_m. \tag{12}$$

The total transmission time for each child node n_i^h in the $(m+1)$-th aggregation is computed as:

$$t_{predtran}^i = t_{preddownload}^i + t_{predupload}^i. \tag{13}$$

Step 2: According to Algorithm 3, the aggregation node n_j^{h+1} recommends aggregation frequency and computation time for each child node. Initially, n_j^{h+1} predicts the computation time t_{pred}^{m+1} for the $(m+1)$-th aggregation (lines 1-5). This strategy ensures a gradual increase in computation time for aggregation. It extends the trainable time for each training node and reduces the probability of instances where the training time for a node, which only trains once, would exceed the recommended computation time. Subsequently, n_j^{h+1} calculates the recommended aggregation frequency rf_i^h and computation time rt_{comp}^i for each child node in the next aggregation (lines 6-9). This method, which utilizes the predicted value for calculation, is tantamount to an upgrade of the RAF algorithm [16].

Adaptive Aggregation Frequency Decision (AAF-D). In this subsection, we propose an adaptive aggregation frequency decision-making method. The method is implemented by both training nodes and non-root aggregation nodes. What they should do is determine their own aggregation frequency by referring to the recommended aggregation frequency and computing time provided by the parent node, in conjunction with their own actual situation.

During the initial round of training, or when a new parent-child node relationship is established, the child node n_i^h has

Algorithm 4 Adaptive Aggregation Frequency Decision

Input: Recommended aggregation frequency rf_i^h, recommended computation time rt_{comp}^i, appropriate timeout ratio $\rho_a(\rho_a > 1.0)$, prevention timeout ratio $\rho_p(\rho_p < 1.0)$
Output: Actual aggregation frequency f_i^h, actual total update time consumed $t_{totalcomp}^i$

1: $t_{totalcomp}^i = 0$
2: Updated times $u_i^h = 0$
3: **repeat**
4: Update once and record the time consumed t_{update}^i
5: $t_{totalcomp}^i = t_{totalcomp}^i + t_{update}^i$
6: $u_i^h = u_i^h + 1$
7: Predict the time consumed $t_{predcomp}^i$ for the next update
8: Predict the total time consumed after the update $t_{predtotal}^i = t_{totalcomp}^i + t_{predcomp}^i$
9: **until** $(u_i^h < rf_i^h$ **and** $t_{predtotal}^i > rt_{comp}^i \times \rho_a)$ **or** $(u_i^h \geq rf_i^h$ **and** $t_{predtotal}^i > rt_{comp}^i \times \rho_p)$
10: $f_i^h = u_i^h$

not received rf_i^h and rt_{comp}^i from the parent node. Consequently, the aggregation frequency f_i^h is set to 1. After a single update, the node uploads the model parameters and other data to the parent node.

Upon receiving rf_i^h and rt_{comp}^i from the parent node, the child node can employ Algorithm 4 to adaptively determine the aggregation frequency. Specifically, the node executes an update and records the time consumed t_{update}^i. Subsequently, it calculates the total time $t_{totalcomp}^i$ and the number of updates u_i^h (lines 4-6). The time $t_{predcomp}^i$ for the next update is predicted based on the time of recent updates, using the exponential smoothing method. The total time $t_{predtotal}^i$ after the next update is then predicted (lines 7-8). Following this, the node decides whether to continue with the update (line 9). If u_i^h has not reached rf_i^h, and if the time taken slightly exceeds rt_{comp}^i, the device is still permitted to perform the next update. If u_i^h has reached rf_i^h, the device is allowed to overtrain while preventing a timeout. After the loop is exited and the update is completed, the device uploads the data to the parent node.

Due to the heterogeneity of resources in the dynamic edge computing environment and the relationships between devices in the aggregation structure, the method might set the aggregation frequency either too high or too low for some nodes. A significant difference in aggregation frequency could introduce excessive residuals, leading to model oscillation and a decrease in convergence speed. To mitigate this, we could constrain the range of the aggregation frequency. When the aggregation frequency reaches the upper limit, the updates are suspended, and the decision algorithm ceases to determine whether to continue with the update.

Combination of DSF and AAF Method. In Subsection IV-A, we proposed the DSF method for the aggregation structure. Consider nodes n_i^h and n_j^{h+1}, which share a parent-child relationship, with n_i^h as the child node and n_j^{h+1} as the parent node. If neither v_i nor v_j is scheduled for removal, the parent-child relationship between n_i^h and n_j^{h+1} will not change. The DSF method aims to preserve as much aggregation frequency data as possible.

The combination of DSF and AAF is detailed as follows:

After the initial construction of the aggregation structure, each child node is assigned $rf_i^h = 1$ and $rt_{comp}^i =$ None. During the first training round, each node uploads data after a single update. Upon receiving the data, the parent node provides rf_i^h and rt_{comp}^i to the child nodes following the AAF-R method. In the subsequent updates of the child nodes, the actual aggregation frequency f_i^h is adaptively adjusted according to the AAF-D method.

After the adjustment of the aggregation structure following the DSF method, rf_i^h and rt_{comp}^i are discarded for any parent-child relationships that were removed during the adjustment process. For any new parent-child relationships established during the adjustment process, because there has not been any previous transmission of model parameters between these two nodes, there is no transmission time information available. Therefore, the computational time and aggregation frequency cannot be recommended as described in lines 7-8 of Algorithm 3. Consequently, each child node is assigned $rf_i^h = 1$ and $rt_{comp}^i =$ None. Despite the possibility that these nodes might have had a parent-child relationship in the past, the dynamic nature of edge computing environments implies that factors such as device computational power and channel bandwidth are continually changing. Relying on outdated time values or considering only computational time without accounting for transmission time would likely result in the node becoming a straggler. Conversely, initially using an aggregation frequency of 1 is a more cautious approach. Data is uploaded after a single update before the child node receives new rf_i^h and rt_{comp}^i recommended by the parent node.

Timeout Abandon Mechanism. In a dynamic edge computing environment, the training of edge devices may be affected by various factors. For example, they might prematurely exit training due to failures or power issues, or they may have limited computational resources if occupied with other tasks. Despite the method AAF enabling the device to adaptively adjust without strictly following the recommended value, it may still fall behind. For instance, irrespective of whether Algorithm 4 is employed for decision-making, the device must complete at least one update before uploading data. If this process is time-consuming, the device could become a straggler. If the aggregation node waits until all child nodes upload data, it may be delayed in aggregating due to the presence of stragglers and might even never receive the data of a child node. This could result in a decrease in model training efficiency and even a halt in model training.

Therefore, it is essential to integrate a timeout abandonment mechanism into hierarchical model training. We introduce a straightforward timeout abandonment mechanism. In Algorithm 3, the aggregation node predicts the computation time t_{pred}^{m+1} for the next aggregation during the m-th aggregation. The predicted value can serve as the benchmark during the $(m+1)$-th aggregation waiting period of the node. We can establish a timeout abandon ratio $\rho_t(\rho_t \geq 1.0)$, and the timeout abandonment threshold is defined as:

$$t_{timeout}^{m+1} = t_{pred}^{m+1} \times \rho_t. \tag{14}$$

If the aggregation node waits beyond $t_{timeout}^{m+1}$ and still does not receive data from some child nodes, it ceases the wait and only utilizes the received model data for aggregation.

C. Theoretical Analysis of EASF

To investigate the impact of device dynamics on model training, we conducted a convergence analysis of the EASF method, using a three-level aggregation structure as a representative example. Given that DSF focuses on constructing and adjusting aggregation structures to align the data distribution within each cluster closer to the global, we introduce the variable Δ_r to quantify its effectiveness, representing the discrepancy in data distribution in the r-th round:

$$\Delta_r = \frac{1}{K_r} \sum_{l=1}^{K_r} \|\boldsymbol{\omega}_l^r - \boldsymbol{\omega}_G^r\|^2, \tag{15}$$

where K_r denotes the number of clusters at level 0, and $\boldsymbol{\omega}_l^r$ is the model parameters of the partial aggregation for cluster l in the r-th training round. For convenience, we denote the total sum of samples of all devices in the cluster l in the r-th round as D_l^r, and the total sum of samples of all devices involved in training as $D_G^r = \sum_{l=1}^{K_r} D_l^r$.

Assumption 1 (L-Smoothness). *Assume that the loss function F is L-smooth, where $L > 0$:*

$$F(\boldsymbol{\omega}_2) \leq F(\boldsymbol{\omega}_1) + \nabla F(\boldsymbol{\omega}_1)^\top (\boldsymbol{\omega}_2 - \boldsymbol{\omega}_1) + \frac{L}{2}\|\boldsymbol{\omega}_2 - \boldsymbol{\omega}_1\|^2. \tag{16}$$

Assumption 2 (Boundedness Below). *Assume that the global loss function F is bounded from below, i.e., there exists a value F_{inf} such that $F(\boldsymbol{\omega}) \geq F_{inf}$ for all $\boldsymbol{\omega}$.*

Assumption 3 (Dynamic Discrepancy Decay). *Assume that the discrepancy Δ_r in data distribution satisfies the following changes as the aggregation structure is adjusted by DSF:*

$$\Delta_r \leq \lambda \Delta_{r-1} + \varepsilon, \quad 0 < \lambda < 1,\ \varepsilon > 0, \tag{17}$$

where λ quantifies DSF's adjustment capability, and ε represents the perturbations stemming from the dynamic nature of edge devices.

Based on the aforementioned assumptions, we analyze the convergence bounds for non-convex objectives under the dynamics of the devices.

Theorem. *Under Assumptions 1-3, for a learning rate α satisfying $0 < \alpha < \frac{1}{L}$, after R rounds of training, the average squared gradient norm of the global model satisfies:*

$$\frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}\left[\|\nabla F(\boldsymbol{\omega}_G^r)\|^2\right] \leq \frac{2(F(\boldsymbol{\omega}_G^0) - F_{inf})}{\alpha R} + \frac{\kappa \Delta_0 (1-\lambda^R)}{R(1-\lambda)} + \frac{\kappa\varepsilon}{1-\lambda}. \tag{18}$$

As $R \to \infty$, the expression simplifies to:

$$\lim_{R\to\infty} \frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}\left[\|\nabla F(\boldsymbol{\omega}_G^r)\|^2\right] \leq \frac{\kappa\varepsilon}{1-\lambda}, \tag{19}$$

where $\kappa > 0$ is a constant.

Proof. Based on the EASF method, the global model update can be expressed as:

$$\boldsymbol{\omega}_G^{r+1} = \frac{1}{D_G^r} \sum_{l=1}^{K_r} D_l^r \boldsymbol{\omega}_l^r. \tag{20}$$

We interpret this update as an approximate gradient descent step:

$$\boldsymbol{\omega}_G^{r+1} = \boldsymbol{\omega}_G^r - \alpha\left(\nabla F(\boldsymbol{\omega}_G^r) + \boldsymbol{\epsilon}_r\right), \tag{21}$$

where $\boldsymbol{\epsilon}_r$ is the error term due to the discrepancy between the cluster models and the global model. To quantify $\boldsymbol{\epsilon}_r$, we define the virtual gradient:

$$\begin{aligned} \tilde{\nabla} F(\boldsymbol{\omega}_G^r) &= \frac{1}{\alpha}\left(\boldsymbol{\omega}_G^r - \frac{1}{D_G^r}\sum_{l=1}^{K_r} D_l^r \boldsymbol{\omega}_l^r\right) \\ \boldsymbol{\epsilon}_r &= \tilde{\nabla} F(\boldsymbol{\omega}_G^r) - \nabla F(\boldsymbol{\omega}_G^r). \end{aligned} \tag{22}$$

We assume that: $\mathbb{E}\|\boldsymbol{\epsilon}_r\|^2 \leq \kappa \Delta_r$, where κ captures the influence of the clustering structure and device dynamics on the error term $\boldsymbol{\epsilon}_r$. Its value is affected by the aggregation quality and the perturbations caused by edge device fluctuations.

Using the L-smoothness, we have:

$$\begin{aligned} F(\boldsymbol{\omega}_G^{r+1}) \leq & F(\boldsymbol{\omega}_G^r) - \alpha \nabla F(\boldsymbol{\omega}_G^r)^\top \left(\nabla F(\boldsymbol{\omega}_G^r) + \boldsymbol{\epsilon}_r\right) \\ & + \frac{L\alpha^2}{2}\|\nabla F(\boldsymbol{\omega}_G^r) + \boldsymbol{\epsilon}_r\|^2. \\ \leq & F(\boldsymbol{\omega}_G^r) + \left(-\alpha + \frac{L\alpha^2}{2}\right)\|\nabla F(\boldsymbol{\omega}_G^r)\|^2 \\ & + \left(L\alpha^2 - \alpha\right)\nabla F(\boldsymbol{\omega}_G^r)^\top \boldsymbol{\epsilon}_r + \frac{L\alpha^2}{2}\|\boldsymbol{\epsilon}_r\|^2. \end{aligned} \tag{23}$$

For the cross term, when $0 < \alpha < \frac{1}{L}$, applying Young's inequality:

$$\begin{aligned} \left(L\alpha^2 - \alpha\right)\nabla F(\boldsymbol{\omega}_G^r)^\top \boldsymbol{\epsilon}_r \leq & \frac{\alpha - L\alpha^2}{2}\|\nabla F(\boldsymbol{\omega}_G^r)\|^2 \\ & + \frac{\alpha - L\alpha^2}{2}\|\boldsymbol{\epsilon}_r\|^2. \end{aligned} \tag{24}$$

Substituting back, we get:

$$F(\boldsymbol{\omega}_G^{r+1}) \leq F(\boldsymbol{\omega}_G^r) - \frac{\alpha}{2}\|\nabla F(\boldsymbol{\omega}_G^r)\|^2 + \frac{\alpha}{2}\|\boldsymbol{\epsilon}_r\|^2. \tag{25}$$

Rearranging the terms, taking the expectation, and applying the bound on the error term gives:

$$\frac{\alpha}{2}\mathbb{E}[\|\nabla F(\boldsymbol{\omega}_G^r)\|^2] \leq \mathbb{E}[F(\boldsymbol{\omega}_G^r) - F(\boldsymbol{\omega}_G^{r+1})] + \frac{\alpha\kappa}{2}\mathbb{E}[\Delta_r]. \tag{26}$$

Summing the inequality over all rounds from $r = 0$ to $R-1$ leads to:

$$\begin{aligned} & \frac{\alpha}{2}\sum_{r=0}^{R-1}\mathbb{E}[\|\nabla F(\boldsymbol{\omega}_G^r)\|^2] \\ & \leq \sum_{r=0}^{R-1}\mathbb{E}[F(\boldsymbol{\omega}_G^r) - F(\boldsymbol{\omega}_G^{r+1})] + \frac{\alpha\kappa}{2}\sum_{r=0}^{R-1}\mathbb{E}[\Delta_r]. \end{aligned} \tag{27}$$

The first term on the right is a telescoping sum, bounded by Assumption 2:

$$\begin{aligned}&\sum_{r=0}^{R-1}\mathbb{E}[F(\boldsymbol{\omega}_G^r)-F(\boldsymbol{\omega}_G^{r+1})]\\&=\mathbb{E}[F(\boldsymbol{\omega}_G^0)-F(\boldsymbol{\omega}_G^R)]\leq F(\boldsymbol{\omega}_G^0)-F_{inf}.\end{aligned}\tag{28}$$

The bound for the second term on the right is obtained from Assumption 3:

$$\sum_{r=0}^{R-1}\mathbb{E}[\Delta_r]\leq\sum_{r=0}^{R-1}\left(\lambda^r\Delta_0+\frac{\varepsilon}{1-\lambda}\right)=\frac{1-\lambda^R}{1-\lambda}\Delta_0+\frac{R\varepsilon}{1-\lambda}.\tag{29}$$

Substituting these bounds and dividing by $\frac{\alpha R}{2}$ yields the final result:

$$\begin{aligned}&\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}\left[\|\nabla F(\boldsymbol{\omega}_G^r)\|^2\right]\\&\leq\frac{2(F(\boldsymbol{\omega}_G^0)-F_{inf})}{\alpha R}+\frac{\kappa}{R}\left(\frac{1-\lambda^R}{1-\lambda}\Delta_0+\frac{R\varepsilon}{1-\lambda}\right)\\&=\frac{2(F(\boldsymbol{\omega}_G^0)-F_{inf})}{\alpha R}+\frac{\kappa\Delta_0(1-\lambda^R)}{R(1-\lambda)}+\frac{\kappa\varepsilon}{1-\lambda}.\end{aligned}\tag{30}$$

We note that the convergence bounds are influenced by both ε and λ, which together encapsulate the characteristics of the data distribution, the adjustment effect of the DSF on the aggregation structure, and the dynamic behavior of the device. Lower values of ε and λ correspond to a tighter convergence bound, ensuring the model reaches a more stable stationary point. Additionally, model training factors, including the loss function and learning rate α, play a significant role in shaping convergence behavior. These parameters should be meticulously tailored to the specific scenario to achieve optimal convergence performance.

V. Experimental Results

To evaluate the performance of the EASF method introduced in Section IV, we have conducted experiments in a simulated dynamic edge computing environment.

A. Experimental Setup

Simulated Edge Computing Environment: We simulate a dynamic edge computing environment with the following characteristics: The network topology graph G_{E_t} is randomly generated and incorporates a total number of N edge devices. The actual computing power of each device dynamically fluctuates. During the training period, each device may undergo several busy periods. Training can only resume after these busy periods. Initially, each device has a 0.7 probability of being in a working status, a condition that changes over time. We simulate the time for model transmission and training mainly based on the network bandwidth B_{E_t} and computational power P_{E_t} at time t.

Models: In our experiments, we utilizes four distinct models: a Convolutional Neural Network (CNN) comprising two convolutional-pooling layers and two fully connected layers, a Multilayer Perceptron (MLP) with four fully connected layers, a Long Short-Term Memory (LSTM) network classifier with two LSTM layers, a Transformer classifier featuring two Transformer encoder layers, and a ResNet-18, a deep residual network with 18 layers.

Datasets: We employ the MNIST dataset [34] and EMNIST-Balance (EMNIST) dataset [35]. MNIST includes handwritten digits in 10 classes, while EMNIST consists of handwritten characters in 47 classes. Each sample is a 28×28 pixel image. Additionally, we use the 20newsgroups dataset [36], which contains news articles on 20 topics. We select samples from 10 of these topics for our experiments. Furthermore, we use ImageNet-10, a subset of the full ImageNet dataset [37] containing 10 classes of high-resolution images.

Optimizer: For the CNN model, we use the SGD optimizer with a learning rate of 0.1. For the MLP model, we use the Adam optimizer with a learning rate of 0.0002. For the LSTM and Transformer models, we use the Adam optimizer with a learning rate of 0.01. For the ResNet-18 model, we use the Adam optimizer with a learning rate of 0.0001. The batch size for all optimizers is set to 64, except for the ResNet-18 model, where it is 32.

Data Distribution: To simulate various data distributions, we adopt settings similar to those in [38], assigning a range of classes and a random number of training samples to each device to represent non-IID data. For MNIST, each device is assigned between 200 and 300 samples; for EMNIST, the range is 300 to 500; for 20newsgroups, the range is 400 to 600; and for ImageNet-10, the range is 250 to 350.

Pre-training: The CNN, MLP, and ResNet-18 models are pre-trained for 1 epoch, while the LSTM and Transformer models undergo 3 epochs of pre-training. To ensure fairness in comparisons with methods that exclude this step, we use it solely to measure pre-training accuracy, resetting model parameters to their initial state before formal training begins.

Hardware and Software Setup: Our experiments were conducted on a system equipped with a single NVIDIA GeForce RTX 4060 GPU. The experimental code was developed using Python 3.9 and PyTorch 2.1.0. We emulate the communication between edge devices using multi-thread interaction and estimate the training and communication time for each device, reflecting real-world conditions.

B. Comparative Methods

This section introduces the methods to be compared with the EASF method, which are the Strong Synchronization (Syn) method [39], Hierarchical Federated Learning (HFL) [21], and the KMA-RAF [16] method.

Syn Method: This approach utilizes a distributed method to construct and adjust the aggregation structure. It is originally from [33], taking into account communication delay and hop count. Devices in close proximity are grouped together. The aggregation frequency is strictly synchronized across all nodes, with each node assigned an aggregation frequency of 1. Each node completes a single iteration of the update before uploading the data.

HFL Method: This method is also a type of strong synchronization method, where the aggregation frequency is the same

at the same level. In the experiment, we set the aggregation frequency of the training nodes to 2, while the aggregation frequency of the aggregation nodes remained at 1.

KMA-RAF Method: The KMA algorithm considers both the sample distribution and communication delay of the devices. It is employed to construct and adjust aggregation structures through an additional control node. The initial aggregation frequency for all nodes is set to 1. During the training process, each aggregation node employs the RAF algorithm to determine the aggregation frequency for each child node upon receipt of the aggregation data. When the aggregation structure changes, the method proposed in Subsection IV-B is invoked to maintain the aggregation frequency corresponding to the unchanged parent-child relationship. The remaining aggregation frequencies are reset to 1.

Neither of the three comparison methods integrates the abandonment mechanism. All devices participating in the experiment will send a notification to the parent node upon the termination of their training. Upon receiving such a message from a child node, the parent node will discontinue receiving data from that particular child node.

C. Comparative Experiment: Results and Analysis

To assess the impact of the DSF method for weakly synchronous hierarchical model training, we maintain a relatively stable simulated edge computing environment. The computing power and the working status of each device fluctuate gradually. We configure the number of edge devices to 40 and 80, with each device receiving 2 types of sample data for MNIST. Additionally, we set the number of edge devices to 60, with each device handling 10 and 15 types of sample data for EMNIST. Our proposed method DSF is compared with the KMA algorithm. The optimization of aggregation frequency is achieved by the AAF and RAF algorithms, respectively. Neither of these methods employs the abandon mechanism.

Fig. 4 presents the experimental results. The aggregation structure of KMA is controlled by a control node, making it challenging to adjust promptly when the working status of the devices changes, necessitating the completion of a whole round of training. Simultaneously, many original parent-child relationships may be destroyed during the adjustment, and the aggregation frequency information may be lost, leading to a decrease in the global model's training efficiency. If the control node is faulty, it may halt the overall training progress. In contrast, DSF performs through a distributed approach. Each subtree of the structure is controlled by an aggregation node. When the device leaves or joins, the aggregation structure can be adjusted immediately. Even if it is a central node, the rest of the devices in the cluster can quickly re-select the central node. Simultaneously, the aggregation frequency information is retained as much as possible during the adjustment of the aggregation structure. Regardless of whether RAF or AAF is used to optimize the aggregation frequency, the model training efficiency is higher than KMA. It is evident that DSF is more suitable for constructing and adjusting the aggregation structure of hierarchical model training.

To evaluate the effectiveness of the overall method EASF across various edge computing environments, we reconfigure

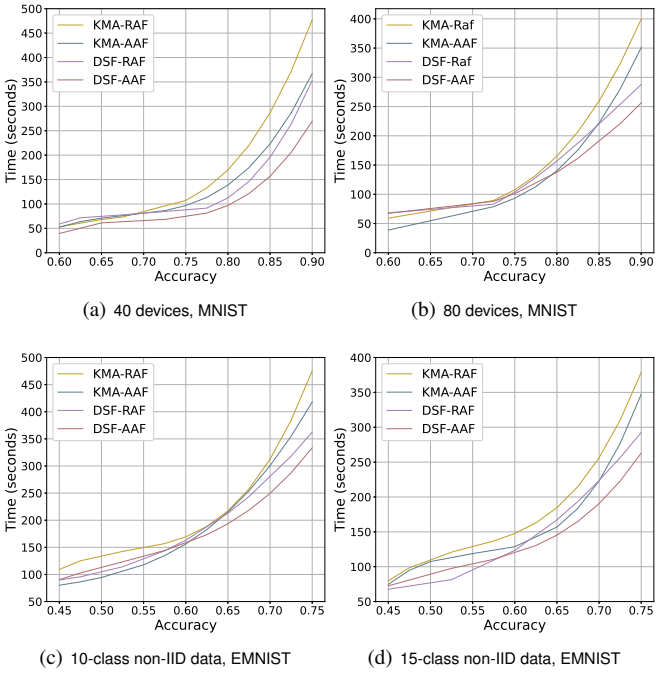


Fig. 4. Performance comparison of different aggregation structure construction and adjustment methods on MNIST and EMNIST.

the simulated edge computing environment to more closely resemble real-world conditions. Fig. 5 and 6, along with Table I and II depict the performance of each method in simulated edge computing environment with a diverse number of devices (30, 60, 100, 150, 200). This experiment is conducted on MNIST, where each device possesses 2 classes of non-IID data. Fig. 7, Table III, and IV showcase the performance of each method with varying data distribution on EMNIST. The number of edge devices is fixed at 50, and the number of classes with samples is set to 4, 8, 16, and 47(IID).

As depicted in the right-hand figures of Fig. 5 and 7, the model accuracy of both EASF and KMA-RAF methods is comparable and superior to that of Syn and HFL. This is mainly due to the consideration of data distribution and weak synchronization in these two methods. However, EASF achieves the highest model accuracy within the same time-frame. This can be attributed to the following two reasons:

Number of training iterations within each round: Both EASF and RAF establish the aggregation frequency for each node. EASF strives to maintain the parent-child relationship in the original structure when the aggregation structure needs adjustment, thus preserving the set aggregation frequency. In contrast, KMA resets the aggregation structure, which could lead to substantial alterations in the tree-based structure and the loss of numerous set aggregation frequencies. Consequently, the subsequent training round becomes similar to strong synchronization, where the aggregation frequencies are predominantly 1. In an environment where changes are frequent and the aggregation structure often requires adjustment, KMA-RAF rarely optimizes the aggregation frequency, resulting in fewer training iterations.

Duration of each round: As shown in Fig. 6, KMA-RAF has

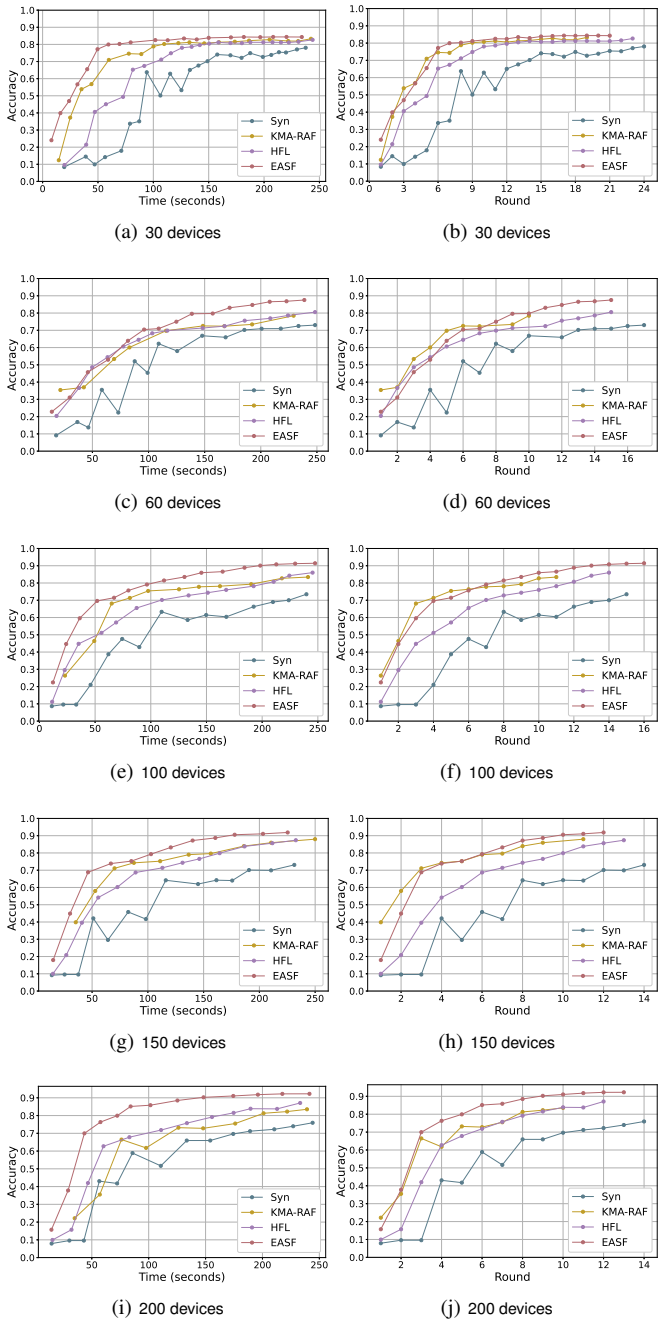


Fig. 5. Performance comparison of methods with different number of Edge Devices on MNIST.

the longest average duration per training round, while EASF and Syn demonstrate shorter average durations, which are relatively similar. Consequently, within a constrained training time, EASF and Syn can execute more training rounds. This outcome can be attributed to two primary factors: firstly, EASF enables nodes to adjust their aggregation frequency based on the actual situation, often leading to more frequent updates within the computing time recommended by the parent node. Conversely, RAF mandates the child node to update according to the aggregation frequency set by the parent node, which may cause the child node to lag due to factors such as device computing power. This leads to longer training rounds for the

TABLE I
HIGHEST ACCURACY ACHIEVED WITHIN A CERTAIN TIME (250 SECONDS) WITH DIFFERENT NUMBERS OF EDGE DEVICES ON MNIST

Method Name	Number of Edge Devices				
	30	**60**	**100**	**150**	**200**
Syn	0.7805	0.7302	0.7348	0.7308	0.7594
KMA-RAF	0.8329	0.7846	0.8344	0.8798	0.8353
HFL	0.8266	0.8054	0.8601	0.8737	0.8709
EASF	0.8436	0.8755	0.9143	0.9186	0.9225

TABLE II
TIME TO FIRST EXCEED A CERTAIN ACCURACY (0.68) WITH DIFFERENT NUMBERS OF EDGE DEVICES ON MNIST

Method Name	Number of Edge Devices				
	30	**60**	**100**	**150**	**200**
Syn	148.90	185.39	209.66	190.69	174.31
KMA-RAF	59.80	115.62	64.89	70.10	126.15
HFL	107.14	103.31	109.99	89.00	110.90
EASF	49.57	96.97	51.86	46.40	43.50

RAF method. Secondly, EASF incorporates an abandonment mechanism, which allows aggregation nodes to disregard data from slower nodes, preventing a delay in overall progress. This feature is also a crucial reason why EASF can occasionally surpass Syn in terms of the number of training rounds.

Considering the impact of varying numbers of edge devices, Table I reveals that the model accuracy of EASF does not significantly deviate from that of the other methods when the number of devices is relatively small. However, as the number of devices escalates, the model accuracy of EASF markedly outperforms that of the other two methods. Table II further underscores that the time required for Syn, KMA-RAF, and HFL methods to attain a specified accuracy varies considerably with the number of edge devices. In contrast, EASF not only necessitates the least time to achieve a predetermined accuracy but also exhibits relative stability in this aspect. It demonstrates superior adaptability to fluctuating numbers of edge devices, effectively handling edge environments with a large number of edge devices.

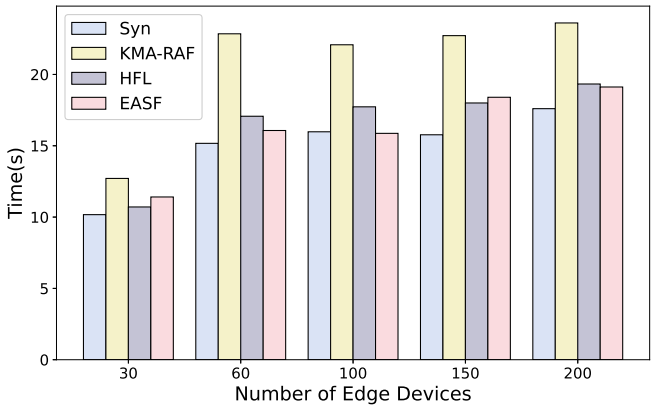
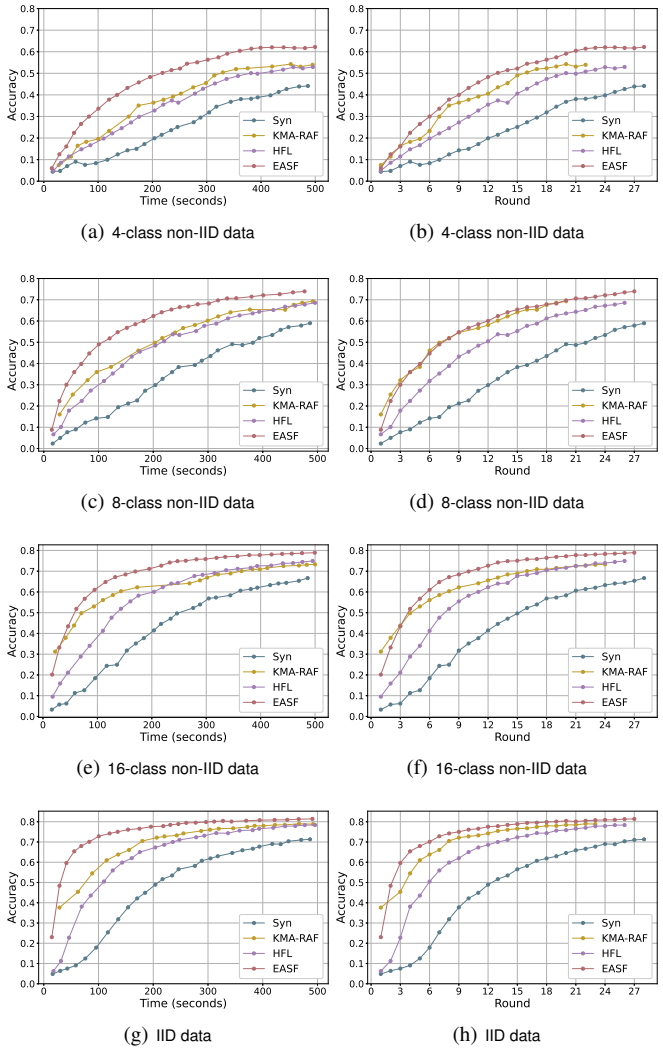


Fig. 6. Comparison of average training time per round of different edge devices and methods on MNIST.



(a) 4-class non-IID data (b) 4-class non-IID data

(c) 8-class non-IID data (d) 8-class non-IID data

(e) 16-class non-IID data (f) 16-class non-IID data

(g) IID data (h) IID data

Fig. 7. Performance comparison of methods with different sample distributions on EMNIST.

Considering the data distribution varies, Tables III and IV indicate that the more extreme the non-IID situation of data distribution, the slower the improvement in model accuracy for each method, and the lower the maximum accuracy achievable within a specified timeframe. In all instances of data distributions, EASF exhibits superior convergence efficiency compared to the other methods. This underscores that EASF possesses enhanced adaptability to various data distributions and can effectively handle more extreme non-IID data distributions.

To investigate the time utilization of each node, we performed a statistical analysis in the scenario of training the CNN model on MNIST. In the dynamic edge computing environment, where devices may join or leave during training, we only counted the time utilization of each node in the training rounds it fully participated. The statistical results are shown in Table V. For training time, due to the AAF method, each node trains as much as possible within a limited time. The average training time proportion of EASF nodes is significantly higher than that of other comparison methods.

TABLE III
HIGHEST ACCURACY ACHIEVED WITHIN A CERTAIN TIME (500 SECONDS) WITH DIFFERENT SAMPLE DISTRIBUTION CLASSES ON EMNIST

Method Name	Number of Sample Distribution Classes			
	4	8	16	47
Syn	0.4413	0.5898	0.6669	0.7131
KMA-RAF	0.5426	0.6930	0.7331	0.7898
HFL	0.5294	0.6856	0.7495	0.7835
EASF	0.6218	0.7395	0.7893	0.8139

TABLE IV
TIME TO FIRST EXCEED A CERTAIN ACCURACY (0.44) WITH DIFFERENT SAMPLE DISTRIBUTION CLASSES ON EMNIST

Method Name	Number of Sample Distribution Classes			
	4	8	16	47
Syn	486.36	317.82	216.99	185.25
KMA-RAF	298.47	172.84	70.51	62.47
HFL	315.20	175.49	125.99	110.09
EASF	174.86	83.19	61.33	28.68

Transmission time is mainly determined by the number of training rounds. Since RAF does not consider the dynamic edge environment, setting a high aggregation frequency statically may lead to stragglers, resulting in fewer total training rounds within a limited time. The number of training rounds of EASF is similar to that of the Syn and HFL methods because EASF considers communication factors in the DSF method to reduce transmission time. Overall, EASF not only maximizes trainable time but also reduces transmission time through the optimization of the aggregation structure, resulting in significantly lower overall idle waiting time compared to other methods and achieving higher training efficiency.

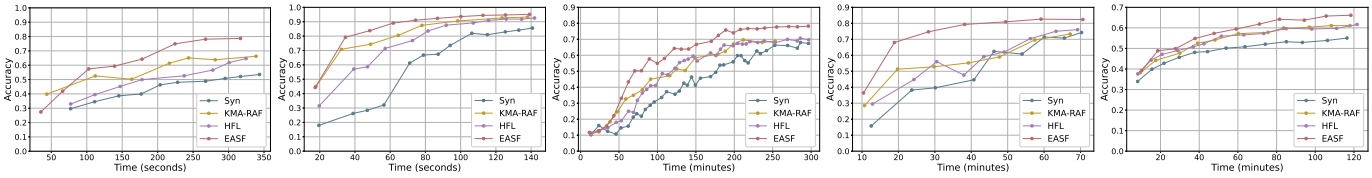
In conclusion, EASF has the capability to construct and adjust the aggregation structure and frequency according to the actual situation, alleviate the communication pressure on the control node, fully utilize device resources, and execute more training rounds within a restricted timeframe. Consequently, EASF can deliver superior model training performance and is adept at handling a variety of edge environments, encompassing a substantial number of edge devices and more extreme non-IID data distributions.

D. Case Study

To further investigate the effectiveness of the EASF method in real-world environments, we conducted a case study. In this section, we enhanced the previously established simulated edge computing environment to make it more realistic, including:

- Some devices may be unable to serve as aggregation nodes due to insufficient bandwidth, power, computational capability, or storage capacity.
- The introduction of a real-world factor, signal strength, as an element affecting transmission time.

In real-world scenarios, edge devices are often required to train with a variety of models and datasets. We trained



(a) MLP; MNIST (b) CNN; MNIST (c) LSTM; 20newsgroups (d) Transformer; 20newsgroups (e) ResNet-18; ImageNet-10

Fig. 8. Performance comparison of the different methods across a variety of models and datasets. Each subfigure's caption is formatted as 'Model; Dataset'.

TABLE V
AVERAGE TIME PROPORTION OF NODES IN TRAINING CNN MODEL ON MNIST

Method Name	Training	Transmisstion	Waiting
Syn	4.9%	29.3%	65.8%
KMA-RAF	8.9%	23.9%	67.2%
HFL	8.3%	27.9%	63.8%
EASF	18.2%	26.7%	55.1%

TABLE VI
IMPACT OF DIFFERENT MODULES ON EASF METHOD PERFORMANCE ON EMNIST

Set	DSF	AAF	Timeout Abandon Mechanism	Max Accuracy (400s) Number of Classes 6	10
#1	×	×	×	0.5757	0.6880
#2	✓	×	×	0.6319	0.6775
#3	×	✓	×	0.6395	0.7457
#4	×	×	✓	0.5956	0.6850
#5	✓	✓	×	0.6647	0.7620
#6	✓	×	✓	0.6706	0.7518
#7	×	✓	✓	0.6729	0.7483
#8(ours)	✓	✓	✓	0.7095	0.7753

Note: ✓indicates the proposed method in this paper is used; × indicates the method is not used.

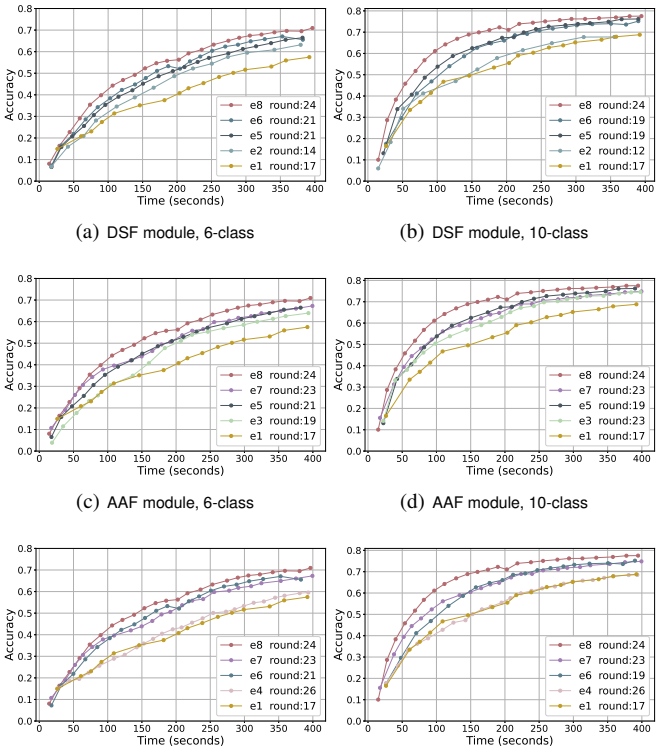
MLP and CNN models on the MNIST dataset, LSTM and Transformer models on the 20newsgroups dataset, and the ResNet-18 model on the ImageNet-10 dataset. The device number was set to 30, with each containing samples from 5 classes.

The experimental results are presented in Fig. 8. The results demonstrate that the EASF method outperforms others across various models and datasets. This is primarily because the aggregation frequency in EASF is primarily determined by model training time and transmission time, without requiring detailed consideration of various factors influencing time. Moreover, it is not directly related to the specific models or datasets being used. In summary, the EASF method is applicable to a wide range of real-world scenarios.

E. Ablation Studies

To evaluate the impact of each component in the EASF method on the hierarchical model's performance during training, we conducted a series of ablation experiments.

EASF is primarily composed of three modules: the DSF method for aggregation structure, the AAF method for aggregation frequency, and a timeout abandonment mechanism. The design and results of the ablation experiment are illustrated in Table VI. The alternative strategy for DSF employs a distributed clustering technique that groups nearby devices. For AAF, the alternative is the RAF algorithm. Regarding the abandonment mechanism, the method proposed in this paper necessitates time predictions from the AAF method to determine the timeout abandonment threshold. Therefore, in this experiment, when AAF is not utilized but the abandonment mechanism is, the threshold is set to a fixed value.



(a) DSF module, 6-class (b) DSF module, 10-class

(c) AAF module, 6-class (d) AAF module, 10-class

(e) abandon mechanism module, 6-class (f) abandon mechanism module, 10-class

Fig. 9. Impact analysis of various modules on the performance of the EASF method on EMNIST.

As depicted in Fig. 9, we present the experimental results in six subfigures, each illustrating the data distribution and the impact of using a particular module.

Analysis of the DSF method: The effectiveness of DSF is demonstrated in Fig. 9a and 9b. In Set #1, where none of the methods of EASF are used, the model training exhibits low accuracy and efficiency. Set #2, which incorporates the DSF method, shows an improvement over Set #1, but lacks stability. The improvement is primarily due to the fact that the clustering method, which takes into account data distribution, is superior

to merely considering delay and hop count in accelerating the model's convergence speed. The instability arises from the preservation of substantial aggregation frequency information during the structure adjustment process, but the aggregation frequency obtained by RAF is not sufficiently accurate, leading to the appearance of stragglers. In Set #2, neither the AAF method nor the abandonment mechanism is present, which may cause the overall progress of model training to be delayed by the stragglers. Set #5 and #6, based on Set #2, introduce the AAF method or a fixed threshold abandonment mechanism. They noticeably improve the training efficiency, but the efficiency and accuracy are still lower than Set #8, which employs all the methods.

Analysis of the AAF method: Fig. 9c and 9d depict the efficacy of AAF. As this method enables the device to adaptively adjust the aggregation frequency based on the actual scenario, all experiments using this method outperform those without it in terms of model training. In Set #3, the lack of DSF results in a significant loss of frequency information, leading to a minor improvement in model training. Conversely, Set #5, #7, and #8 incorporate at least one of the DSF and timeout abandonment mechanisms. The former helps in preserving the aggregation frequency information, while the latter aids in reducing the lag of the overall progress, thereby significantly enhancing the efficiency of model training.

Analysis of the Timeout Abandonment Mechanism: Fig. 9e, 9f illustrate the experimental results of employing the timeout abandonment mechanism. The presence of this mechanism significantly reduces the average training time per round in Set #4. However, it also necessitates the sacrifice of the trained model parameters from a subset of the devices, leading to a minimal improvement compared to Set #1. Set #6 and #7, which incorporate DSF or AAF, effectively augment the training efficiency. Set #8, which utilizes all the methods of EASF, exhibits markedly superior efficiency and accuracy relative to those that omit some modules.

To sum up, AAF forms the core of the EASF method. It enables the edge devices in the dynamic edge computing environment to adaptively adjust the appropriate aggregation frequency. DSF takes into account the data distribution and strives to retain as much aggregation frequency information as possible during the adjustment of the aggregation structure. The timeout abandonment mechanism effectively mitigates the delay in overall training progress caused by the stragglers. In a real-world edge computing environment, when a device departs from training due to failure or other reasons and may not be able to transmit relevant information promptly, the abandonment mechanism becomes indispensable. The combination of the three components enables the training to achieve optimal performance.

VI. Conclusion

In this paper, we present a distributed hierarchical model training framework, featuring a novel method called EASF. This method constructs and adjusts the tree-based aggregation structure in a distributed manner, taking into account both the data distribution and working status of edge devices. Furthermore, EASF adaptively adjusts the aggregation frequency in accordance with the computational capacity of the devices, the heterogeneity of communication resources, and the real-time conditions of training. We also incorporate an abandonment mechanism to mitigate the impact of slower devices on global training, thereby enhancing the robustness of model training.

EASF requires the prediction of multiple time values. The accuracy of these predictions is crucial as they indirectly affect the actual aggregation frequency. In this study, we mainly employ the exponential weighted method for prediction. Future research could explore other prediction methods to achieve higher accuracy and improve the adaptability of the aggregation frequency.

References

[1] N. Marz and J. Warren, *Big data: principles and best practices of scalable real-time data systems*. Shelter Island, NY: Manning Publications, 2015.

[2] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A Survey on Edge Computing Systems and Tools," *Proc. IEEE.*, vol. 107, no. 8, pp. 1537–1562, Aug. 2019.

[3] D. Xu et al., "Edge Intelligence: Empowering Intelligence to the Edge of Network," *Proc. IEEE.*, vol. 109, no. 11, pp. 1778–1837, Nov. 2021.

[4] J. Mendez, K. Bierzynski, M. P. Cuéllar, and D. P. Morales, "Edge Intelligence: Concepts, Architectures, Applications, and Future Directions," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, pp. 1–41, Sep. 2022.

[5] F. Al-Doghman, N. Moustafa, I. Khalil, N. Sohrabi, Z. Tari, and A. Y. Zomaya, "AI-Enabled Secure Microservices in Edge Computing: Opportunities and Challenges," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1485–1504, Mar. 2023.

[6] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. IEEE.*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[7] Z. Qadir, M. Bilal, G. Liu, and X. Xu, "Autonomous Trajectory Optimization for UAVs in Disaster Zone Using Henry Gas Optimization Scheme," *arXiv:2506.15910*, 2025.

[8] S. Wang et al., "When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning," in *IEEE INFOCOM*, IEEE, Apr. 2018, pp. 63–71.

[9] H. Hu, D. Wang, and C. Wu, "Distributed Machine Learning through Heterogeneous Edge Systems," *AAAI*, vol. 34, no. 05, pp. 7179–7186, Apr. 2020.

[10] W. Li, H. Hacid, E. Almazrouei, and M. Debbah, "A Comprehensive Review and a Taxonomy of Edge Machine Learning: Requirements, Paradigms, and Techniques," *AI.*, vol. 4, no. 3, pp. 729–786, Sep. 2023.

[11] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine Learning at the Edge: A Data-Driven Architecture With Applications to 5G Cellular Networks," *IEEE Trans. on Mobile Comput.*, vol. 20, no. 12, pp. 3367–3382, Dec. 2021.

[12] Z. Wang, S. Liu, B. Guo, Z. Yu, and D. Zhang, "CrowdLearning: A Decentralized Distributed Training Framework Based on Collectives of Trusted AIoT Devices," *IEEE Trans. on Mobile Comput.*, pp. 1–18, 2024.

[13] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–33, Mar. 2021.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. Int. Conf. Artif. Intell. Stat.*, vol. 54. PMLR, Apr. 2017, pp. 1273–1282.

[15] D. Chai, L. Wang, L. Yang, J. Zhang, K. Chen, and Q. Yang, "A Survey for Federated Learning Evaluations: Goals and Measures," IEEE Trans. Knowl. Data Eng., pp. 1–20, 2024.

[16] L. Yang, Y. Gan, J. Cao, and Z. Wang, "Optimizing Aggregation Frequency for Hierarchical Model Training in Heterogeneous Edge Computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 7, pp. 4181–4194, Jul. 2023.

[17] R. Han, S. Li, X. Wang, C. H. Liu, G. Xin, and L. Y. Chen, "Accelerating Gossip-Based Deep Learning in Heterogeneous Edge Computing Platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1591–1602, Jul. 2021.

[18] Z. Tang, S. Shi, B. Li, and X. Chu, "GossipFL: A Decentralized Federated Learning Framework With Sparsified and Adaptive Communication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 909–922, Mar. 2023.

[19] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-iid data quagmire of decentralized machine learning," in *Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 4387–4398.
[20] C. Hu, H. H. Liang, X. M. Han, B. A. Liu, D. Z. Cheng, and D. Wang, "Spread: Decentralized Model Aggregation for Scalable Federated Learning," in *Pro. Int. Conf. Parallel Process.*, Bordeaux France: ACM, Aug. 2022, pp. 1–12.
[21] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-Edge-Cloud Hierarchical Federated Learning," in *IEEE Int. Conf. Commun.*, IEEE, Jun. 2020, pp. 1–6.
[22] M. S. H. Abad, E. Ozfatura, D. GUndUz, and O. Ercetin, "Hierarchical Federated Learning ACROSS Heterogeneous Cellular Networks," in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., May 2020, pp. 8866–8870.
[23] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing," in *IEEE INFOCOM.*, IEEE, Jul. 2020, pp. 2076–2085.
[24] Y. Fraboni, R. Vidal, L. Kameni, and M. Lorenzi, "Clustered Sampling: Low-Variance and Improved Representativity for Clients Selection in Federated Learning," in *Int. Conf. Mach. Learn.*, PMLR, Jul. 2021, pp. 3407–3416.
[25] S. Liu, J. Yu, X. Deng, and S. Wan, "FedCPF: An Efficient-Communication Federated Learning Approach for Vehicular Edge Computing in 6G Communication Networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 1616–1629, Feb. 2022.
[26] Z. He, L. Yang, W. Lin, and W. Wu, "Improving Accuracy and Convergence in Group-Based Federated Learning on Non-IID Data," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1389–1404, May 2023.
[27] W. Y. B. Lim et al., "Decentralized Edge Intelligence: A Dynamic Resource Allocation Framework for Hierarchical Federated Learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 536–550, Mar. 2022.
[28] C. You, K. Guo, H. H. Yang, and T. Q. S. Quek, "Hierarchical Personalized Federated Learning Over Massive Mobile Edge Computing Networks," *IEEE Trans. Wirel. Commun.*, vol. 22, no. 11, pp. 8141–8157, Nov. 2023.
[29] S. Wang et al., "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," *IEEE J. Select. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
[30] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive Control of Local Updating and Model Compression for Efficient Federated Learnings," *IEEE Trans. on Mobile Comput.*, vol. 22, no. 10, pp. 5675–5689, Oct. 2023.
[31] J. Yan, J. Liu, S. Wang, H. Xu, H. Liu, and J. Zhou, "Heroes: Lightweight Federated Learning With Neural Composition and Adaptive Local Update in Heterogeneous Edge Networks," in *Proc. IEEE INFOCOM 2024-IEEE Conf. Comput. Commun.*, 2024, pp. 831–840.
[32] L. Luo, C. Zhang, H. Yu, G. Sun, S. Luo, and S. Dustdar, "Communication-Efficient Federated Learning With Adaptive Aggregation for Heterogeneous Client-Edge-Cloud Network," *IEEE Trans. Services Comput.*, vol. 17, no. 6, pp. 3241–3255, Nov./Dec. 2024.
[33] Lakshmish Ramaswamy, B. Gedik, and L. Liu, "A distributed approach to node clustering in decentralized peer-to-peer networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 9, pp. 814–829, Sep. 2005.
[34] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE.*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[35] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Int. Joint Conf. Neural Netw.*, IEEE, May 2017, pp. 2921–2926.
[36] K. Lang, "NewsWeeder: Learning to Filter Netnews," in Machine Learning Proceedings 1995, Elsevier, 1995, pp. 331–339.
[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conf. Comput. Vis. Pattern Recog.*, 2009, pp. 248—255.
[38] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," *arXiv:1806.00582*, 2018.
[39] F. Cicirelli, A. Giordano, and C. Mastroianni, "Analysis of Global and Local Synchronization in Parallel Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 988–1000, May 2021.

Xiaolong Xu received the Ph.D. degree in computer science and technology from Nanjing University, China, in 2016. He is currently a Full Professor with the School of Software, Nanjing University of Information Science and Technology. He has published more than 100 peer-review articles in international journals and conferences, including the IEEE TKDE, IEEE TPDS, JSAC, IEEE TSC, IEEE TFS, IEEE T-ITS, IJCAI, ICDM, ICWS, ICSOC, etc. He was selected as the Highly Cited Researcher of Clarivate (2021-2023). He received best paper awards from Tsinghua Science and Technology at 2023, Journal of Network and Computer Applications at 2022, and several conferences, including IEEE HPCC 2023, IEEE ISPA 2022, IEEE CyberSciTech 2021, IEEE CPSCom2020, etc. His research interests include edge computing, the Internet of Things (IoT), cloud computing, and big data.

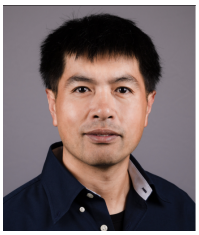
Jiayang Sun received the BEng degree in software engineering from Nanjing University of Information Science & Technology, China, in June 2024. Currently, he is pursuing a postgraduate studies at the School of Software Engineering, Nanjing University of Information Science & Technology. His research interests include edge computing and distributed machine learning.

Guangming Cui received his Master's degree from Anhui University, China, in 2018 and his PhD degree from Swinburne University of Technology, Australia, in 2022, in computer science. Currently, he is an associate professor at Nanjing University of Information Science & Technology, China. His research interests include edge computing, service computing, mobile computing and software engineering.

Lianyong Qi received the Ph.D. degree from the Department of Computer Science and Technology, Nanjing University, China, in 2011. He is currently a Full Professor with the College of Computer Science and Technology, China University of Petroleum (East China), China. He has already published more than 100 articles, including the IEEE JSAC, the IEEE TCC, TBD, FGCS, the Journal of Computational Social Science, CCPE, ICWS, and ICSOC. His research interests include services computing, big data, and the Internet of Things.

Muhammad Bilal received the Ph.D. degree in information and communication network engineering from the School of Electronics and Telecommunications Research Institute (ETRI), Korea University of Science and Technology, Daejeon, South Korea, in 2017. From 2018 to 2023, he was an Assistant Professor with the Division of Computer and Electronic Systems Engineering, Hankuk University of Foreign Studies, Yongin, South Korea. In 2023, he joined Lancaster University, Lancaster LA1 4YW, United Kingdom, where he is currently working as a Senior Lecturer with School of Computing and Communications. His research interests include design and analysis of network protocols, network architecture, network security, the IoT, named data networking, blockchain, cryptology, and future Internet.

Wanchun Dou is a lecturer in the received the Ph.D. degree in mechanical and electronic engineering from the Nanjing University of Science and Technology, China, in 2001. He is currently a Full Professor at the State Key Laboratory for Novel Software Technology, Nanjing University. From April 2005 to June 2005 and from November 2008 to February 2009, he visited the Departments of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, respectively, as a Visiting Scholar. He has published more than 100 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.



Zhipeng Cai received the B.S. degree from Beijing Institute of Technology, Beijing, China, in 2001, and the M.S. and Ph.D. degrees from the Department of Computing Science, University of Alberta, Edmonton, AB, Canada, in 2004 and 2008, respectively. He is currently a Professor with the Department of Computer Science, Georgia State University, Atlanta, GA, USA. His research has received funding from multiple academic and industrial sponsors, including the National Science Foundation and the U.S. Department of State, and has resulted in over 100 publications in top journals and conferences, with more than 14500 citations, including over 80 IEEE/ACM transactions papers. His research expertise lies in the areas of resource management and scheduling, privacy, networking, and big data.

Jon Crowcroft received the degree in physics from Trinity College, University of Cambridge, Cambridge, U.K., in 1979, the M.Sc. degree in computing, and the Ph.D. degree from University College London, London, U.K., in 1981 and 1993, respectively. From 2016 to 2018, he was the Programme Chair with Alan Turing Institute, U.K. National Data Science and AI Institute, London, U.K. He is currently a Researcher with Alan Turing Institute. Since October 200l, he has been the Marconi Professor of communications systems with Computer Laboratory. His research interests include Internet support for multimedia communications, scalable multicast routing, practical approaches to traffic management, the design of deployable end-to-end protocols, opportunistic communications, social networks, privacy preserving analytics, and techniques and algorithms to scale infrastructure-free mobile systems. Dr. Crowcroft is a Fellow of the Royal Society, ACM, British Computer Society, IET and the Royal Academy of Engineering.