NebulaFL: Self-Organizing Efficient Multilayer Federated Learning Framework With Adaptive Load Tuning in Heterogeneous Edge Systems

Zirui Lian, Jing Cao, Qianyue Cao, Weihong Liu, Zongwei Zhu, and Xuehai Zhou, *Member, IEEE*

***Abstract*—As a promising edge intelligence technology, federated learning (FL) enables Internet of Things (IoT) devices to train the models collaboratively while ensuring the data privacy and security. Recently, hierarchical FL (HFL) has been designed to promote distributed training in the intricate hierarchical structure of IoT. However, the coarse-grained hierarchical schemes usually fail to thoroughly adapt to the hierarchical environment, leading to high training latency. Meanwhile, highly heterogeneous communication and computation delays due to the device diversity (the system heterogeneity) and decentralized data distribution due to the decentralized device distribution (the data heterogeneity) exacerbate the above challenges. This article proposes NebulaFL, a dual heterogeneity-aware multilayer FL framework, to support efficient distributed training in IoT scenarios. NebulaFL proposes an innovative multilayer architecture organization scheme to adapt the complex hierarchical heterogeneous scenarios. Specifically, through a finer-grained division of the HFL hierarchy, hybrid synchronous-asynchronous training is implemented at both the global system and local device-layer levels. More importantly, to adaptively build a heterogeneity-aware hierarchical training architecture, NebulaFL considers the effect of dual heterogeneity in the architectural organization scheme to determine the optimal location of devices in a multilayer environment. To further improve the training efficiency during the training process, NebulaFL employs an augmented multiarmed bandit technique based on the reinforcement learning to adjust the device-layer training load by evaluating the dynamic training utility and convergence uncertainty feedback. Experiments demonstrate that NebulaFL achieves up to a 15.68× speed-up ratio and a 23.94% increase in the training accuracy compared to the latest or classic approaches.**

***Index Terms*—Automatic layering, edge intelligence, federated learning (FL), heterogeneous training, reinforcement learning.**

I. Introduction

With the rapid expansion of Internet of Things (IoT) systems, the edge smart devices collect increasing amounts of user data to train the artificial intelligent models. However, gathering the large volumes of the device data for the cloud-based model training introduces huge bandwidth costs and privacy leakage risks. Federated learning (FL) [1], [2] offers an efficient solution by utilizing the secure aggregation technologies and diverse privacy policies [3] for the local model training and aggregation on the distributed devices without the need to upload the raw data.

In the traditional FL frameworks, the devices train models with the local data and engage in multiple rounds of synchronous [2] [Fig. 1(a)] or asynchronous [4] [Fig. 1(b)] interactions with the server to aggregate the model parameters. However, at a large scale of devices, this frequent communication results in significant costs and unpredictable delays, especially over the lengthy communication links between the devices and the cloud servers. Recently, *hierarchical FL* (HFL) solutions based on the IoT layered communication structure, including the *cloud* servers, the *gateway* aggregators, and the edge training *devices* have been extensively studied to mitigate the challenges of the device scalability and communication bottlenecks. As shown in Fig. 1(c), HFL establishes short-range communication between the gateways and devices based on the distance or cost, offloading the immense cloud communication load to the gateways closer to the devices, thus reducing the communication overhead.

However, deploying HFL in IoT scenarios encounters significant heterogeneity challenges. From a static perspective, while the hierarchical structures reduce communication distances between the devices and gateways, the variance in devices' configurations (e.g., chips, memory storage, and communication bandwidth) leads to serious straggler problems [5] caused by differences in the execution time, extending the synchronization time required for each training round. Additionally, the diverse data distributions on the edge devices, resulted by environmental or user characteristic differences also introduce inconsistencies in training objectives [6], necessitating more training rounds for convergence. From a dynamic perspective, the unpredictability of communication delays during training and variations in computational speed under the resource

Manuscript received 9 August 2024; accepted 9 August 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62102390; in part by the Special Fund for Jiangsu Natural Resources Development (Innovation Project of Marine Science and Technology) under Grant JSZRHYKJ202218; and in part by the National Key Laboratory of Science and Technology on Space Microwave under Grant HTKJ2022KL504021. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Zongwei Zhu.)*

Zirui Lian, Jing Cao, Qianyue Cao, Weihong Liu, and Xuehai Zhou are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: ustclzr@mail.ustc.edu.cn; congjia@mail.ustc.edu.cn; cqy_1999@mail.ustc.edu.cn; lwh2017@mail.ustc.edu.cn; xhzhou@ustc.edu.cn).

Zongwei Zhu is with the School of Software Engineering and the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: zzw1988@ustc.edu.cn).

Digital Object Identifier 10.1109/TCAD.2024.3443715

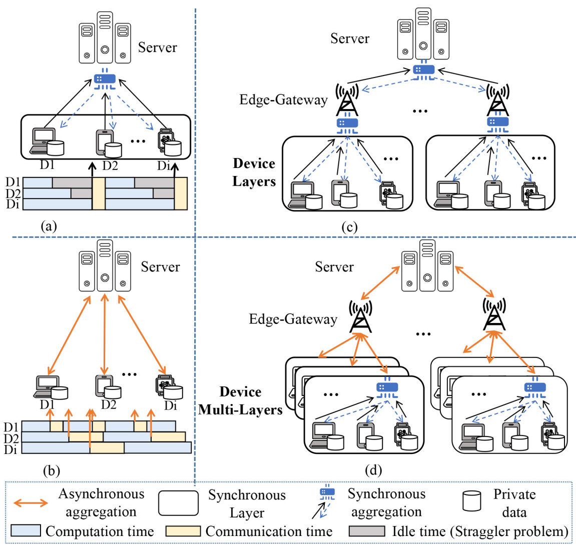


Fig. 1. Architectures and communication mechanisms of different FL frameworks.

constraints significantly impact the training progress and the timeliness of the model updates. Moreover, as the training progresses, the data heterogeneity on the devices at different rounds affects the model convergence variably. Consequently, the devices' utility to the global model dynamically change over time, presenting a complex challenge to achieving efficient HFL.

Recent HFL schemes optimize training efficiency in complex hierarchical heterogeneous environments. However, these schemes usually lack systematic considerations resulting in limited optimization. As shown in Fig. 1(c), classic studies [7] propose *fully synchronous training mechanism* in HFL to mitigate the long-distance communication issues in the cloud–edge layer training architectures. However, the device heterogeneity and unreliable network communication can lead to severe straggler problems. Recent research [8] introduces a *fully asynchronous training mechanisms* in HFL to address the straggler problem. While considering heterogeneity for the architectural organization, the asynchronous method incurs significant communication costs due to frequent model parameter transmission and harms the training efficiency due to severe inconsistencies in training objectives caused by the data heterogeneity. Advanced HFL frameworks introduce the *hybrid synchronous-asynchronous training mechanisms* (the cloud-gateway asynchronous and the gateway-device synchronous) at the global system level [9], [10], significantly enhancing the HFL's training capabilities. However, these methods are designed based on the homogeneity assumption at the device layer. As the scale and distribution differences of the devices under the gateway increase, these efforts still fail to mitigate the main impact of the static heterogeneity, namely the poor synchronization efficiency at the device level. Moreover, all the above studies rarely consider the dynamic effects of heterogeneity.

Considering the shortcomings of the above solutions, we propose an improved HFL architecture, *NebulaFL* as shown in Fig. 1(d). Unlike the previous solutions, NebulaFL employs a synchronous-asynchronous training mechanism at the global system level, innovatively further divides the devices under per gateway into different logical layers (known as the device-layer), and realizes a hybrid synchronous-asynchronous training mechanism at the device logical layers level. NebulaFL has three advantages: 1) asynchronous training between the logical layer of the device and the gateway interaction mitigates the straggler problem caused by the device heterogeneity; 2) synchronous training within the logical layer mitigates the inconsistency of local training objectives caused by the data heterogeneity; and 3) reduces the communication overhead of frequent model transfers compared to the fully asynchronous training. However, there are two significant challenges to realizing efficient NebulaFL as follows.

1) *To mitigate the static impacts of heterogeneity,* how can we determine the optimal placement of devices under specific gateways and within particular device-layers?
2) *To alleviate the dynamic effects of heterogeneity,* how can we maximize the local training utility of device-layers?

Therefore, NebulaFL proposes a series of optimization methods to address the above challenges and nontrivial design ideas as summarized below.

1) We designed novel metrics based on the fine-grained data and the system features to evaluate the device utility, guiding NebulaFL's decision making in heterogeneous scenarios.
2) NebulaFL utilizes the training utility metrics, combined with an improved matching algorithm and the community detection algorithm to adaptively generate efficient architectural organization schemes while mitigating the impact of the heterogeneity's static nature, thereby enhancing the system training efficiency.
3) NebulaFL employs reinforcement learning-based techniques, integrating the training utility and convergence uncertainty factors to adjust the training load of the device logical layers to alleviate the heterogeneity's dynamic impacts and further improve the system training efficiency.
4) Experiments results show that NebulaFL achieves up to 15.68 $\times$ speedup and up to 23.94% improvement in training accuracy compared to the baselines.

II. Preliminary

A. Federated Learning and Communication Mechanism

FL systems typically consist of an aggregation server and N devices involved in training. Training is initiated after the server broadcasts the training task (model) w. Each device $i \in N$ has a local dataset D_i, containing the data samples $(X_i, y_i) = (X_1, y_1), (X_2, y_2), \ldots, (X_{|D_i|}, y_{|D_i|})$, which $|D_i|$ represents totaling samples and D_i is non-IID (not independently and identically distributed). The training task defines the loss function for each sample (X_j, y_j) as $f(w, X_j, y_j)$, and the local loss function for the device i is defined as $F_i(w) = (1/|D_i|) \sum_{j \in D_i} f(w, X_j, y_j)$. Therefore, the objective of FL will be to minimize the total loss across all the devices

$$F(w) = \sum_{i \in N} \frac{|D_i|}{\sum_{i \in N} |D_i|} F_i(w). \tag{1}$$

Solving $F_i(w)$ involves executing K steps of local iterations using the gradient descent on the device locally. The update rule for the kth step is as follows:

$$w^k = w^{k-1} - \eta \nabla F_i(w)^{k-1} \tag{2}$$

where $\eta > 0$ is a hyperparameter representing the size of the update step. After completing one round locally, the device sends the updated parameters to the server for aggregation. The aggregation update method can be broadly categorized into synchronous and asynchronous updates. Specifically, synchronous updating means the server has to wait for the updates from all the devices in the current round t before performing aggregation. The aggregation rule for round t is

$$W^{K,t} = \sum_{i \in N} \frac{|D_i|}{\sum_{i \in N} |D_i|} w^{K,t}. \tag{3}$$

The asynchronous mechanism does not require waiting for the other devices to arrive, and the aggregation rule is: $W^{K,t} = (1-\alpha)W^{K,t-1} + \alpha w^{K,t}$. It is worth noting that α is introduced to alleviate the challenge of the model staleness [4].

B. Hierarchical Federated Learning

Long-distance communication in the cloud-device dual-layer training architecture causes straggler issues [5] in synchronous mechanisms and high communication costs [4] in asynchronous mechanisms. Therefore, HFL is proposed to solve these problems, and it is divided into three layers: the cloud layer, which provides abundant computing resources and acts as the brain to build the HFL architecture and lead the training process; the gateway layer, consisting of base stations and routers, which serves as the intermediary hub connecting the cloud and edge devices; and the device layer, which contains many heterogeneous devices that are typically constrained by resources and privacy concerns.

Assuming that there are N devices and M gateways, the HFL architecture can be represented as a topology graph $\mathcal{G}_{(M,N)}$, where $M \ll N$. Assuming that the set of devices associated with the gateway m is defined as N_m, the objective of HFL is

$$F_{\text{HFL}}(w) = \sum_{m=1}^{M} \frac{|D_m|}{\sum_{m=1}^{M} |D_m|} F_{\text{HFL}}^{m}(w) \tag{4}$$

where $F_{\text{HFL}}^{m}(w) = \sum_{i \in N_m} (|D_i|/|D_m|) F_i(w)$ denotes the objective on the edge node m. $|D_m|$ is the data size of all the samples in N_m. The approach to address the above objectives is similar to the synchronous training, represented as $W_{\text{HFL}}^{K,t} = \sum_{m \in M} (|D_m|/[\sum_{m \in M} |D_m|]) w_m^{K,t}$. For the devices, aggregation is typically done using a synchronous mechanism, represented as $w_m^{K,t} = \sum_{i \in N_m} (|D_i|/[\sum_{i \in N_m} |D_i|]) w_i^{K,t}$.

III. System Model and Problem Statement

This section first introduces the NebulaFL architecture. Then, it describes the problems and challenges. In addition, we give the design goals and general ideas for its solution.

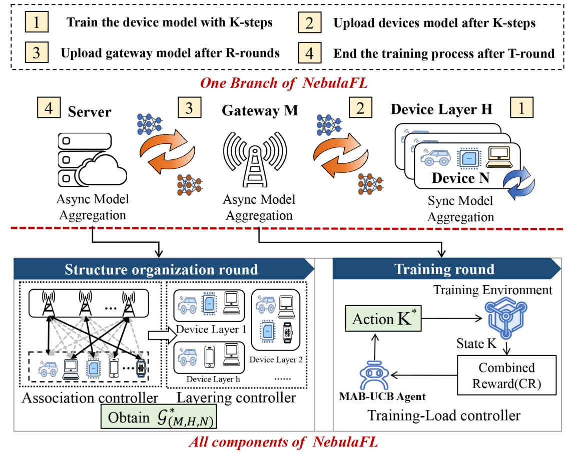


Fig. 2. Training flow in one branch and all the components of NebulaFL. The device organization scheme contains gateway-device association controller and device layering controller. The training-load controller serves as a supplementary weapon for performance enhancement during training.

A. Framework Overview

Unlike the traditional HFL, NebulaFL employs an asynchronous aggregation mechanism at the gateway-to-cloud stage and introduces an additional logical layer per gateway. Devices within the same layer perform synchronous training, and asynchronous aggregation is used between the device logical layers. Furthermore, the topology of the system is $\mathcal{G}_{(M,H,N)}$, where H represents the number of device layers. Meanwhile, the training optimization components are designed, including *device-gateway association* controller, *device layering* controller, and *training load* controller. As shown in Fig. 2, we comprehensively demonstrate a branch training process in NebulaFL and the included optimization components.

The upper part of Fig. 2 shows the shift of NebulaFL from the synchronous training of HFL to an asynchronous approach across the cloud, gateway, and device layers while maintaining the synchronization of the device layer training. *During the training round,* devices initially perform K steps of local training. Subsequently, the devices within the same logical layer transmit their latest models to a gateway or a leader node with optimal communication capabilities (e.g., [9]) for the intralayer synchronous aggregation. Subsequently, the gateway model aggregates updates from the various logical layers asynchronously. After completing R rounds of asynchronous iteration, the gateway forwards the aggregated model to the cloud for the global asynchronous aggregation. The global model is then immediately distributed to all the devices through the gateway for the next round of training. This training process continues for T rounds or until the target accuracy (TA) is achieved.

To support the NebulaFL's structure, we introduces three successive optimization components as illustrated in the lower part of Fig. 2. *During the structure organization round,* devices receive pretraining instructions from the cloud launcher and transmit necessary structure organization information to the cloud server via nearby gateways. It is important to note that, to maintain the original intent of FL, we have carefully designed this information so as not to violate the privacy and security. And then the cloud server designs the structure organization using an association and an hierarchical controller. It then broadcasts the structure information to all the devices, specifying the layer (h) each device (i) is in within the gateway (m). During a training round, each gateway includes a load controller to increase the training load of high-utility devices, thereby accelerating the training progress.

B. Problem Statement and Motivation

This work focuses on the complete training time needed to reach the TA. Therefore, cost analysis of training latency and convergence efficiency is crucial.

System Cost Model: For the NebulaFL topology $\mathcal{G}_{(M,H,N)}$, the time cost of completing a training round comes from three primary sources: 1) device computation latency; 2) device transmission delay; and 3) additional idle overhead for the device synchronization. We model the latency to estimate the theoretical time required for a training round. Specifically, the computational latency of each device i is primarily influenced by the training load K_i, the training hyperparameter batchsize BS, the computation frequency f_i, and the number of clock cycles C_i required for one iteration. The theoretical device latency for one iteration gateway round (R) is $U_i^{\text{Cmp}}(r) = \lfloor (K_i/BS) \rfloor * (C_i/f_i)$. Second, the theoretical communication latency $U_i^{\text{Com}}(r)$ is mainly determined by the device bandwidth B_i and the size of the transmitted information model size $|W|$, additional auxiliary information ε, represented as $U_i^{\text{Com}}(r) = (|W + \varepsilon|/B_i)$. Since, NebulaFL performs the device layering at the end level, where the intralayer communication is synchronous, the latency for each device layer h is determined by the longest time spent by any device within that layer, i.e., $U_h(r) = \max_{i \in h}(U_i^{\text{Cmp}}(r) + U_i^{\text{Com}}(r))$. Therefore, the total time consumption U_{system} of the entire training process can be formalized as

$$U_{\text{system}} = \sum_{t=1}^{T} P_m^t \cdot \sum_{h \in H} U_h(r) \quad \forall m \tag{5}$$

where P_m^t is a binary variable indicating whether the gateway m is involved in the global aggregation at the global round t.

Data Cost Model: The convergence efficiency of the model is impacted by the data heterogeneity across the devices, typically due to the inconsistent optimization objectives [6] of the local models on different devices. Let's consider under what conditions FL with the data heterogeneity can achieve the accuracy of a centralized model. Using the classic FL algorithm FedAvg, we define the device layer-aggregated model in the rth round of NebulaFL as $w_{h,r}^{\text{fedavg}}$.

1) *Assume That:* $w_{h,r}^{\text{fedavg}} = w_r^*$ is true and $\mathcal{P}_{D_{\text{train}}}^h = \mathcal{P}_{D_{\text{test}}}$, where $\mathcal{P}_{D_{\text{train}}}^h$ refers to the train set distribution at the layer h and $\mathcal{P}_{D_{\text{test}}}$ means the distribution of the balanced test set. According to the solution rule of (3), we can get

$$w_{h,0}^{\text{fedavg}} = \sum_{i \in N} \frac{|D_i|}{\sum_{i \in N} |D_i|} w_0^i = |N| * \frac{|D_{\text{test}}|}{|N| * |D_{\text{test}}|} w_0 = w_0^*. \tag{6}$$

2) *Inductive Assumption:* Assume $w_{h,r}^{\text{fedavg}} = w_r^*$ is true for $r = k$ $(k \in Z^+)$ and $\mathcal{P}_{D_{\text{train}}}^h = \mathcal{P}_{D_{\text{test}}}$. We can get

$$\begin{aligned} w_{h,k+1}^{\text{fedavg}} &= w_{h,k}^{\text{fedavg}} - \eta \nabla_{w_{h,k}^{\text{fedavg}}} \sum_{i=1}^{N} \mathbb{L}\Big(F\Big(X^i; w_{h,k}^{\text{fedavg}}\Big), y^i\Big) \\ &= w_k^* - \eta \nabla_{w_k^*} \mathbb{L}\big(F\big(X^i; w_k^*\big), y^i\big) = w_{k+1}^*. \end{aligned} \tag{7}$$

Therefore, it can be proved by mathematical induction that FL recovers the accuracy of the model when the data distribution of the train set $\mathcal{P}_{D_{\text{train}}}^h$ for each layer is approximately uniformly distributed set $\mathcal{P}_{D_{\text{test}}}$. Similarly, the above conclusions still hold for the group of devices associated with the edge gateway m. Since, the model training object typically aims to minimize test loss, this implies that regardless of the device association or device tiering, *when the data distribution within a region tends toward balance,* it is possible to recover or approach the optimal accuracy under the IID conditions. In NebulaFL, there are primarily two regions: a) the region formed by all the devices under the gateway and b) the region formed by each device layer within the gateway. Therefore, the difference in data distribution within the two regions can be used to quantify the system's overall data cost U_{data}. In other words, the higher the degree of data heterogeneity, the greater the cost and the longer the time required to achieve the TA. We present specific quantitative data distribution metrics to derive U_{data} in Section IV-A.

Training Objective: The system topology $\mathcal{G}_{(M,H,N)}$ is crucial for minimizing the aforementioned system and data costs. However, satisfying both simultaneously is challenging. For instance, devices within the same group may have lower system costs. However, the data distribution within the same group might not be complementary, which is a common scenario in the real world. Therefore, we define V as a certain join relation in the topology, where $V_{(m,h,i)}$ denotes that the ith device is partitioned into the hth device layer and that this device layer is connected to the gateway m. Our ultimate goal is to find a device organization structure $\mathcal{G}_V$ that minimizes the total cost Cost_V as much as possible, formalized as follows:

$$\min_{w \in \{w^t \,:\, t \in [0,T]\}} \text{Cost}_{\mathcal{G}_V} \simeq \min\big\{\beta U_{\text{data}} + (1-\beta) U_{\text{system}}\big\} \tag{8}$$

$$\text{s.t.} \begin{cases} F_{\text{HFL}}\big(w^T\big) - F_{\text{HFL}}(w^*) \leq \varepsilon \\ V_{m,h,i} \in \{0, 1\} \quad \forall m, h, i \\ \sum_{m=1}^{M} \sum_{h=1}^{H} V_{i,h,m} = N \quad \forall m, h, i \end{cases} \tag{9}$$

where w^* is the ideal optimal model parameter. β determines which cost the optimization objective is more inclined toward minimizing. It is worth noting that the load parameter K in the training algorithm affects the system cost and data cost of the heterogeneous training system and needs to be carefully tuned. Hence, to achieve the above objectives, we propose the NebulaFL framework to obtain the optimal $\mathcal{G}_V^*$ and K^* from the perspectives of the system design and algorithm optimization, respectively, to alleviate the cost challenges faced by the NebulaFL simultaneously.

IV. NebulaFL Design

This section focuses on the NebulaFL's system design, which aims to enhance the model's time-to-accuracy performance. Given the complexity of solving (8), NebulaFL employs three sequential optimization components: 1) *training utility evaluation*; 2) *architecture design solutions*; and 3) *training load tuning*. Evaluating training utility is fundamental for the subsequent steps. The final two steps address the static and dynamic effects of heterogeneity optimally.

A. Training Utility Metrics

Despite various metrics for measuring the data heterogeneity in FL, methods relying on the data summaries (e.g., [12] and Section III-B) pose privacy risks. While the algorithms like Oort [11] use training loss effectively for the device selection, they lack deep insights into the data distribution. Moreover, gradient-based metrics often focus only on the immediate gradients [13], ignoring long-term gradient dynamics and diverse data learning capabilities.

NebulaFL utilizes fine-grained gradient information from R rounds of local training on the edge devices to overcome these limitations for the data feature modeling. Inspired by the core-set [14] concept, this method captures gradient variations to reflect the data feature and model convergence. Specifically, we collect gradients from E epochs of local training (where each epoch e involves training on the entire dataset D of the device and $E = \lfloor K/|D| \rfloor$) and obtain the cross-batch gradient feature (σ) and the epoch-based gradient feature (φ):

1) Cross-Batch Feature σ_i: The gradient vector for the kth batch in the eth epoch of training is represented as $g_{k,i}^{e}$. Therefore, the cross-batch difference features σ_i for each device i is defined as the average difference in gradients within a round with the specific formula

$$\sigma_i^e = \frac{1}{K_{e,i}} \sum_{k=1}^{K_{e,i}} \left(\nabla g_{k,i}^e - \nabla g_{k-1,i}^e\right), \quad \text{for} \quad e \in [1, E]. \quad (10)$$

2) Round-Based Feature φ_i: For the device i, its round-based difference feature φ_i measures the cumulative gradient difference from the 1 to the eth epoch. Let $\hat{\nabla g_i^e} = (1/K_{e,i}) \sum_{k=1}^{K_{e,i}} \nabla g_{k,i}^e$ be the average gradient vector for the device i at the end of the eth epoch, and the calculation formula for φ_i^e is

$$\gamma_i^e = \hat{\nabla g_i^e} - \hat{\nabla g_i^{e-1}}, \quad \text{for} \quad e \in [1, E] \quad (11)$$

where $\nabla g_{0,i}^e$ and $\hat{\nabla g_i^0}$ represents the zero gradient vector. Therefore, by integrating the σ_i and φ_i, we can get the data distribution features Γ_i

$$\Gamma_i = \text{flatten}\left(\left[\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^E, \gamma_i^2, \ldots, \gamma_i^E\right]\right) \quad (12)$$

where flatten$(\cdot)$ operation converts the gradient tensors into the 1-D vectors. Γ_i analyses the data distribution in two ways: 1) cross-batch variance to evaluate the training data heterogeneity and model adaptability and 2) round-based variance to assess the impact of the data distribution on the long-term model learning. Notably, by focusing on the gradients of the final layer, Γ can indirectly capture the data distribution characteristics while reducing the dimensions and simplifying the analysis.

In addition, we evaluate the impact of the system heterogeneity on the training efficiency. Given the hyperparameters, the pretraining process is iterated repeatedly to obtain the average computation time U_i^{cmp} required to perform a round of local training. Second, the communication delay is also obtained during the pretraining process, which mainly consists of the device transmission delay U_i^{Com} and the propagation delay $\text{RTT}_{i,m}$ (obtained by tools, such as ping or Traceroute). Considering the uplink from the device to the gateway, the communication time from the device i to the gateway m is $U_{(i,m)}^{\text{Com}} = U_i^{\text{Com}} + [\text{RTT}_{(i,m)}/2]$. Thus, the delay feature per device $\Omega_{(i,m)} = U_i^{\text{cmp}} + U_{(i,m)}^{\text{Com}}$ and the total delay vector to all the gateways m Λ_i can be obtained

$$\Lambda_i = \left[\Omega_{(i,1)}, \Omega_{(i,2)}, \ldots, \Omega_{(i,M)}\right]. \quad (13)$$

The methods for obtaining the system and data characteristics occur during the pretraining phase. Due to the system and data fluctuations, reorganizing the devices during training is often necessary [15]. NebulaFL's flexibility allows it to obtain the organizational features from the previous training round easily. For instance, the data characteristics can be computed from the previously transmitted parameters, and the latency characteristic $\Omega_{(i,m)}$ corresponds to the previous training round's duration.

B. Architecture Design Solutions

In this section, we utilize the data characteristics Γ_i and the delay features Λ_i from Section IV-A to measure distances between the devices and servers or among devices. Then, we introduce algorithms for the unified device-gateway association and adaptive device layering, focusing on the static effects of heterogeneity. The distance calculation is defined as follows:

$$\begin{cases} \Gamma_m = \frac{1}{|N_m|} \sum_{i \in N_m} \Gamma_i \\ J_{(i,m)} = \frac{1}{2} KL\left(\frac{\Gamma_m}{||\Gamma_m||_2} | \frac{\Gamma_i}{||\Gamma_i||_2}\right) + \frac{1}{2} KL\left(\frac{\Gamma_i}{||\Gamma_i||_2} | \frac{\Gamma_m}{||\Gamma_m||_2}\right) \\ Utils_{(i,m)} = J_{(i,m)} * \left(\frac{\theta_m}{\Lambda_i[m]}\right)^{\mathbb{I}(\theta_m \le \Lambda_i[m]) * \delta} \end{cases} \quad (14)$$

where Γ_m represents the average gradient characteristics of all the devices associated. We use the Jensen–Shannon divergence [16], a method that measures the similarity between the two probability distributions, to calculate the similarity of information vectors between the devices and gateways. This allows us to quantify the overall data cost of the system, $J(i, m)$. The information vectors utilize the data distribution features Γ_i obtained in the previous section rather than the privacy-sensitive data summaries. $Utils_{(i,m)}$ signifies the training utility of the device i being associated with the gateway m. θ_m stands for the maximum delay among the devices in $\Lambda_i[m]$ that are within the top (H/N) of the delay. $\mathbb{I}$ is the indicator function, implying whether the delay of a device exceeds a certain threshold. The experiments will discuss the penalty factor δ as a hyperparameter.

1) Device-Gateway Association: According to (8), the goal of the device-gateway association is to minimize the latency between the devices under the same gateway and to balance the data distribution among the gateways. Previous studies

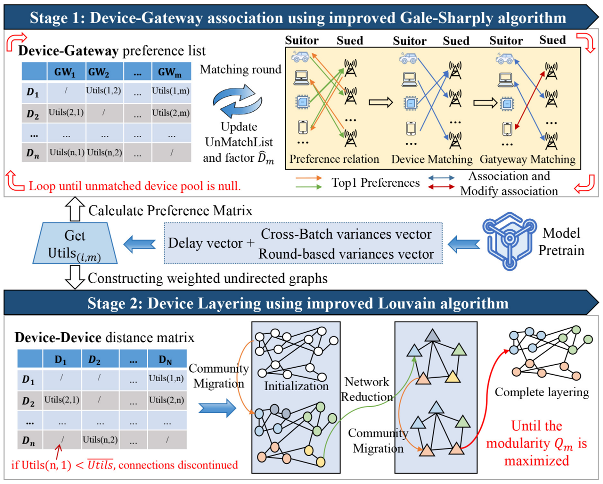


Fig. 3. Introduction to architecture organization processes and algorithms.

usually assume that the gateways possess the data [10], [12] and can serve as the data clustering centers for efficient device association. However, gateways usually lack training data, so the traditional one-time clustering methods cannot associate all the devices. In addition, the new data from the newly associated devices can change the existing data distribution under the gateway, thus affecting the subsequent device association decisions.

Given that the devices and gateways are distinct entities without direct connections between similar types, we frame this challenge as a bidirectional graph-matching problem and improve the Gale–Shapley algorithm [17] to address it. In our modification, the matching dynamic reflects the interplay between "suitor" and "sued" found in the Gale–Shapley algorithm. However, unlike the original one-to-one matching approach, our enhancement allows a gateway to connect with the multiple devices. A straightforward device-gateway association process is shown in Fig. 3 (Stage 1), and the complete algorithm workflow is as follows.

Compute the Preference List: Before each iteration, NebulaFL calculates the training utility $Utils_{(i,m)}$ of the devices in the unmatched pool concerning the gateways according to (14) and generates an ordered preference list where the devices with higher utility values are ranked higher. We further introduce a preference factor to limit the number of devices associated with a gateway by controlling the associated devices' total data samples $|D_m|$. This prevents a few gateways are associated with all the devices. Specifically, during each round of association, we update $Utils_{(i,m)} = \hat{D_m} \times Utils_{(i,m)}$, where $\hat{D_m} = 1 + (|D_m|/|D|)$, to prevent the gateway overload. $|D|$ is the total global data sample. Thus, for each device and gateway, the preference list is denoted as $Plist_i = \{Utils_{\{i,m\}} | m = 1, 2, \ldots, M\}$ and $Plist_m = \{Utils_{\{i,m\}} | i = 1, 2, \ldots, N_m\}$, respectively.

Perform the Matching Process: Initially, when no gateways are matched with any devices, we set $J_{(i,m)}$ and $\hat{D}$ to 1, as the gateways do not have any data characteristics. Subsequently, we divide all the devices in the unmatched device pool into multiple batches, with each batch containing the same number of devices as there are gateways. Devices iteratively send matching requests to their highest-priority gateway based on their preference list $Plist_i$. Gateways then decide whether to accept the match based on their preference list $Plist_m$. In each round, each gateway accepts one device according to its preference list $Plist_m$. If a gateway has already matched with a device but prefers a new incoming device, it will replace the original device and return the replaced device to the unmatched device pool. After completing a round of matching, the devices that failed to match, update their preference lists and send matching requests to the next preferred gateway in the subsequent iteration. This process repeats until all the devices are successfully matched with a gateway.

2) Device Layering: Similar to the association algorithm, layering aims to achieve near uniform latencies for the devices within the same layer while maximally balancing the data distribution across the device layers. The Louvain community detection algorithm [18] is efficient for automatically constructing the communities through the "modularity" optimization. Considering each device layer as an unique community, we have developed an improved Louvain algorithm tailored for the device layering. A straightforward device layering process is shown in Fig. 3 (Stage 2), and the complete algorithm workflow is as follows.

Weighted Network Construction: Construct a fully connected weighted network for all the devices under each gateway, where the nodes in the network correspond to the devices and the weights of the edges are derived from $Utils_{(i,j)}$. Notebly, $Utils_{(i,j)}$ can be equated to the "distance" between the devices i and j, as defined by (14). After that, we introduce a simplification mechanism: connections between any two devices i and j are discontinued if $Utils_{(i,j)}$ surpasses the global average utility $\overline{Utils}$.

Modularity Optimization: Louvain's algorithm detects community structure by optimizing the network's modularity. Modularity is used to measure the quality of community partitioning operations. We define the layered modularity of devices under each gateway m as follows:

$$Q_m = \frac{1}{2\hat{Utils}} \sum \left[Utils_{(i,j)} - \frac{E_i E_j}{2\hat{Utils}} \right] \varpi \left(c_i, c_j\right) \qquad (15)$$

where E_i and E_j are the degrees of nodes i and j, respectively, $\hat{Utils}$ is the sum of all the edge weights in the network, and c_i and c_j are the communities to which nodes the i and j belong, respectively, and ϖ is the Colonic function, which is 1 when $c_i = c_j$ and 0 otherwise.

The iterative process begins with each node being assigned to a community that contains only itself. Next, during the community migration phase, each node is iteratively checked and moved to a neighboring community if it improves the modularity Q_m. Once further improvement of modularity by moving nodes is no longer possible, the network reduction phase begins, where the current communities are merged into single nodes to form a simplified network with edge weights summing up all the edges within the community before the merge. This process is repeated on the simplified network until the modularity degree reaches its maximum, ultimately resulting in the original network being divided into different layers.

C. Training Load Adjustment

Building on the previous design, NebulaFL is divided into multiple asynchronous device layers, with synchronous training within each layer. To further improve the training efficiency of NebulaFL, optimizing the training load can further reduce data and system costs. Although increasing the training load on the devices is commonly used to enhance efficiency and reduce communication costs, the recent studies indicate that uneven device loads can lead to client-drift [6], affecting the effectiveness of synchronized training.

Therefore, overall training load adjustment based on the device-layer becomes a key strategy to improve the efficiency of NebulaFL. Considering the dynamic effects of dual heterogeneity in the training process, it is necessary to introduce an intelligent agent that adjusts the training load based on the real-time feedback from the training. We extend the above idea with a reinforcement learning-based multiarm slot machine (MAB) strategy [19], where the goal is to learn the distribution of rewards for each arm and maximize the total expected reward after a series of actions. In NebulaFL, the training performance may vary with the device performance and data distribution across the training rounds, and the MAB algorithm can adapt to these changes in real time by constantly "trying out" different load configurations to maximize the training gains. MAB faces the problem of finding the optimal balance between exploration and utilization, which is solved using the extended UCB algorithm [19] as follows.

1) Define Combined Rewards: In order to accurately implement a multiarmed slot machine (MAB) and effectively capture the dynamics of the training process, we introduce a combined reward (CR) mechanism that combines both the data reward (DR) and the system reward (SR). This concept draws on the treatment of training loss in Oort [11], which holds that a higher training loss reflects the training value of the device in the current configuration. Specifically, for the hth device layer, the DR DR_h is defined as the ratio of the average training loss of the layer to the average loss of the entire gateway expressed as

$$\mathrm{DR}_h(t) = \frac{\frac{1}{R_h}\sum_{r\in R_h}\mathbb{L}_{(h,r)}\left(w^t\right)}{\frac{1}{H}\sum_{h\in H}\mathbb{L}_h(w^t)} \tag{16}$$

where R_h denotes the interaction count between the hth device layer and the gateway. $\mathbb{L}_{(h,r)}(w^t)$ denotes the average training loss of the hth device layer in the tth round of iterations over the rth round of the gateway iterations. $\mathbb{L}_h(w^t)$ is the average loss of the hth device layers in round t.

Similarly, the SR of the hth device layer is defined as the ratio of the average delay of the device layer h over s rounds of iterations to the average delay of the gateway, i.e.,

$$\mathrm{SR}_h(t) = \frac{\frac{1}{S_h}\sum_{s\in S_h}U_{(h,s)}}{\frac{1}{H}\sum_{h\in H}U_h} \tag{17}$$

where $U_{(h,s)}$ is the average delay of the hth device layer in the sth iteration. U_h is the average delay of the hth device layer over all the iterations. Thus, the CR is a combination of the DR and the SR, which we set

$$\mathrm{CR}_h(t) = \frac{\mathrm{SR}_h(t)}{\mathrm{DR}_h(t)}. \tag{18}$$

2) UCB Selection Strategy: The UCB strategy considers the historical average returns of the selection process (utilization) as well as the uncertainty (exploration) to compute the upper confidence bound value to make a decision. The UCB upper bound for each device layer h is calculated as follows:

$$\mathrm{UCB}_h(t) = \overline{\mathrm{CR}_h} + \sqrt{\frac{2ln\triangle(t)}{\triangle_h(t)}} \tag{19}$$

where $\overline{\mathrm{CR}_h}$ represents the average CR up to the current round. This means that the adjustment of the load depends on the estimation of long-term rewards, in order to reduce the impact of short-term system volatility. $\triangle(t)$ represents the total number of selections up to round t, and $\triangle_h(t)$ represents the number of times device layer h has been selected. Notably, a device layer may be constantly rewarded for having a high $\mathrm{CR}_h(t)$. The UCB algorithm introduces an exploratory factor through $\sqrt{([2ln\triangle(t)]/[\triangle_h(t)])}$, which provides the device layer that is rewarded less often with the opportunity to catch up with the other device layers in order to resolve the uncertainty factor in the convergence process.

3) Define "Arms": An increase in load is considered a reward for the device layer, and each "arm" indicates whether or not a load reward is applied. NebulaFL sorts the device layers in descending order by $\mathrm{UCB}(t)$ and looks for the maximum interval, finding all device layers before the maximum interval and rewarding them. In each round, the load of the selected device layer is increased to $K_h * (1+\pi)$, while the load of the unselected device layer is decreased to $K_h * (1-\pi)$, ensuring that the load never falls below the initial level. π as a hyperparameter will be discussed in the experiments. In addition, $\triangle(t)$ and $\triangle_h(t)$ are updated accordingly.

Navigating the Tradeoff: The CR in (18) achieves a balance between DR and SR. As the model improves, the CR reduces the DR per round through feedback on the training load, since the training loss decreases over time. However, during the convergence, training loss can overfit and remain nearly constant [20], making it difficult to limit the load for the devices with high DRs. The SR effectively addresses this by increasing with the training load, which raises the computational delays. This increase in SR then reduces the CR, preventing the training load from continuously rising. Fig. 4 shows the more precise feedback tuning process.

V. Experiment

A. Implementation and Setup

1) Experimental Platform: Due to the limited number of the physical devices, we first build a heterogeneous simulation platform to conduct the comparative experiments involving many devices. Subsequently, we build a realistic physical platform to implement NebulaFL.

Simulation Platform Setup: We set up the simulation platform on a computing cluster of eight *Nvidia A100* and

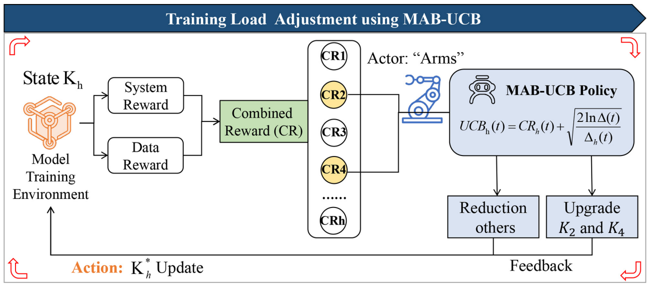


Fig. 4. Device layer-based feedback load tuning method (load controller).

TABLE I
DATASET STATISTICS AND TASK CONFIGURATIONS

Platform	Dataset	Samples	Sample size	Use Model	Partition	Devices
Simulation System	MNIST	60000	28×28 Grey	SimpleCNN (1.6MB)	Bias(2) Dir(0.5)	120 devices 6 gateway
	FMNIST	60000	28×28 Grey	SimpleCNN (1.7MB)	Bias(2) Dir(0.5)	120 devices 6 gateway
	CIFAR10	50000	32×32 RGB	ResNet18 (43MB)	Bias(2) Dir(0.5)	80 devices 5 gateway
	Shake-speare	422615	80×1 Text	LSTM (208KB)	Natural	120 devices 5 gateway
	CIFAR10	50000	32×32 RGB	SimpleCNN (1.7MB)	Dir(0.5)	500 devices 5 gateway
Practical System	CIFAR10	50000	32×32 RGB	VGG11 (136.32MB)	Dir(0.1/0.3/0.6/0.8) and IID	12 devices 3 gateways(PC)

TABLE II
DEVICE CONFIGURATION ON PHYSICAL PLATFORMS

Mode	Frequency	Cores	Number	Total
0 (Jetson TX2)	0.85GHz	256 CUDA cores	3	8
1 (Jetson TX2)	1.12GHz	256 CUDA cores	3	
2 (Jetson TX2)	1.30GHz	256 CUDA cores	2	
3 (Xavier NX)	0.80GHz	48 Tensor cores	2	4
4 (Xavier NX)	1.10GHz	48 Tensor cores	2	

four *Nvidia V100* GPUs. The platform configuration includes *Ubuntu 20.04, CUDA 11.4, and Python 3.9*. We implemented the NebulaFL framework using the *PyTorch distributed communication library* and compared it with the advanced HFL methods. We used an FL discrete event network simulator [21], which extracts the real latitude and longitude from different devices to obtain a realistic latency distribution. To simulate network uncertainty, we added the log-normal latency on the top of the average delay of each local training round, similar to what is done in *Async-HFL*. In this setup, we built three heterogeneous edge systems for different training tasks as shown in Table I "devices."

Physical Platform Setup: We constructed a physical platform to validate the effectiveness of our approach in the real-world scenarios. As shown in Table II, we utilized *Nvidia Jetson TX2* devices with three different computing frequencies and *Jetson Xavier NX* devices with two different frequencies, highlighting the computational heterogeneity. Additionally, we deployed three gateways and one server across four PCs. All the devices are connected via WLAN to a 100 Mb/s Ethernet network. By applying the bandwidth limitations, each device is configured with different network interface bandwidths to simulate a realistic edge environment. All the devices are set up using *JetPack 5.1.3, which includes Ubuntu 20.04, CUDA 11.4, and PyTorch 1.8*.

2) Data and Model Setup: As shown in Table I, we conduct experimental evaluations on the four typical datasets, referencing similar work [7], [9], [10], including *MNIST* [22], *FMNIST* [23], *CIFAR10* [24], and *Shakespeare* [25]. For the data heterogeneity, we employ two commonly used partitioning methods [26]: 1) *Bias* and 2) *Dir*. In bias (2), each device is assigned two preferred classes while maintaining an equal data volume across the devices. In contrast, Dir(0.5) uses the *Dirichlet distribution* parameters to flexibly adjust the statistical heterogeneity, resulting in different clients having varied sample sizes and class distributions. We trained the MNIST and FMNIST datasets using the *SimpleCNN* [27] and the CIFAR10 dataset using the *ResNet18* [28] and *VGG11* [29]. In addition, since the Shakespeare is the textual data, a *two-layer LSTM* is used for the training.

3) Baseline Setup: Based on the above platform and data configurations, we compare *NebulaFL* with six classical or state-of-the-art HFL methods. *HierFAVG* is an HFL scheme based on communication latency, performing synchronous aggregation at both the gateway and cloud levels. *FedCH* is a latency-based HFL scheme that performs synchronous aggregation at the gateway and asynchronous aggregation in the cloud, using a random device selection method called *FedCH-random*. Additionally, we combine it with the state-of-the-art device selection algorithm *Oort*, referred to as *FedCH-Oort*, to optimize for data heterogeneity. *Hier-SAFA* is another latency-based HFL scheme that performs semi-asynchronous aggregation at the gateway layer and asynchronous aggregation in the cloud. The semi-synchronous period, which is a challenging hyperparameter, is set to the optimal result from our experiments. *Async-HFL* is a state-of-the-art fully asynchronous FL scheme, carefully optimized for device association schemes and device selection in hierarchical asynchronous training. The *NebulaFL* scheme proposed in this paper, along with its extension *NebulaFL+*, includes a multi-arm bandit-based device workload adjustment controller, supporting the ablation study of *NebulaFL*.

4) Hyperparameters and Metrics: For a fair comparison, all the experiments used consistent hyperparameters, such as SGD as the optimizer, a learning rate of 0.01, a batch size of 32 per client, and a momentum of 0.9. In each round, devices perform 100 local iterations (with *NebulaFL+* implementing adaptive adjustment) before aggregating with the gateway. The edge gateway interacts with the cloud server every ten communication rounds. With a carefully designed association mechanism, the framework reassociates every 20 rounds.

B. Experimental Results

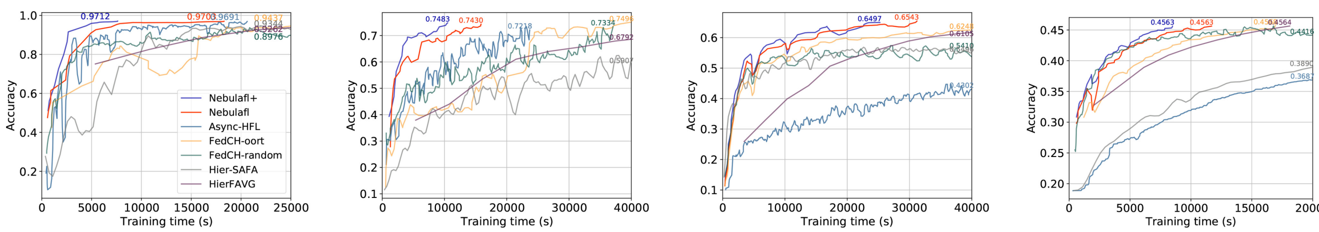
1) Model Convergence Accuracy: Table III compares the highest accuracy achieved by various HFL baseline methods within a specific time threshold (TT) and assesses the speed-up ratio relative to the fully synchronous baseline HierFAVG upon reaching the TA.

Compared to the other training schemes, the NebulaFL method achieved the best training accuracy and speed-up ratio (data highlighted in bold) across most tasks, thanks to its improved HFL architecture that balances the latency and data

TABLE III
CONVERGENCE ACCURACY AND SPEEDUP RATIO OF ALL HFL METHODS ARE EVALUATED IN VARIOUS TASKS

Task	Partition	Method		HierFAVG	Async-HFL	Hier-SAFA	FedCH	FedCH-Oort	**NebulaFL**	**NebulaFL+**
MNIST (SimpleCNN)	Bias	Acc(%)	TT:30000	93.75	97.00	95.39	91.31	94.14	**97.09**	**97.19**
		SpeedUp	TA:95%	1.00×	**12.65×**	1.80×	1.98×	2.36×	5.27×	12.35×
	Dir	Acc(%)	TT:20000	95.61	**97.35**	96.17	95.16	96.23	97.01	97.24
		SpeedUp	TA:95%	1.00×	14.86×	2.36×	3.47×	5.59×	6.13×	**15.68×**
Fashion-MNIST (SimpleCNN)	Bias	Acc(%)	TT:30000	65.99	74.55	40.64	63.12	72.74	74.27	**75.10**
		SpeedUp	TA:70%	1.00×	10.67×	2.79×	4.55×	4.11×	10.61×	**13.04×**
	Dir	Acc(%)	TT:20000	67.33	74.67	70.65	71.56	74.51	**74.85**	**76.33**
		SpeedUp	TA:70%	1.00×	**13.28×**	1.92×	5.77×	12.29×	10.25×	12.68×
CIFAR10 (ResNet-18)	Bias	Acc(%)	TT:30000	41.07	45.00	56.89	54.17	60.78	**64.49**	**65.01**
		SpeedUp	TA:55%	1.00×	–	1.61×	1.72×	1.75×	**4.74×**	**5.85×**
	Dir	Acc(%)	TT:20000	46.98	47.66	56.54	59.95	63.23	**65.64**	**68.35**
		SpeedUp	TA:55%	1.00×	1.12×	1.67×	2.44×	2.51×	**5.23×**	**5.89×**
Shakespeare (LSTM)	Natural	Acc(%)	TT:10000	40.92	31.92	33.63	43.61	44.56	**45.11**	**45.63**
		SpeedUp	TA:43%	1.00×	inf	inf	1.11×	1.69×	1.66×	**2.02×**

[1] All experiments conduct three times, with results averaged for reliability. TT means to reach a target time, while TA means to reach target accuracy.
[2] Bolded data indicates optimal performance for the given task. When NebulaFL and NebulaFL+ achieve optimality, they are all highlighted in bold.



(a) (b) (c) (d)

Fig. 5. Convergence process and accuracy performance of different HFL methods under different tasks. (a) MNIST. (b) FMNIST. (c) CIFAR. (d) Shakespeare.

distribution. Additionally, the comparison between NebulaFL and NebulaFL+ showed that the adaptive load balancing at the device layer also played a significant role. Specifically, for the MNIST dataset, Async-HFL, and NebulaFL reached the best accuracy of 97.35% and 97.19% in the bias and dir partitions, respectively. At the same time, Async-HFL and NebulaFL had comparable speeds in the bias and dir partitions, achieving the best speed-up ratios of 12.65 and 15.68× relative to HierFAVG, thanks to the asynchronous advantage of Async-HFL conducting more iterations within a specific TT.

However, the underlying success factors for these two methods differ significantly. With complex datasets like CIFAR-10 and Shakespeare, Async-HFL's convergence advantage diminishes, particularly in the bias partition, indicating its limitation with the complex datasets despite its effectiveness with simpler ones like MNIST. This suggests the need for the synchronous training mechanisms for more complex datasets. NebulaFL enhances convergence efficiency by implementing synchronous training within the device layers under the same gateway. Furthermore, FedCH-random and FedCH-Oort show stable training across all the tasks, with speed-ups of 5.77 and 12.29×, respectively, in FMNIST's Dir partition training. However, they only consider latency during the association process, overlooking how the data distribution affects accuracy. Additionally, Hier-SAFA's challenge in optimizing its semi-period hyperparameter leads to poorer accuracy.

2) Model Convergence Behavior: We further studied the convergence behavior of the above methods. These training tasks employed a biased partitioning method. As shown in Fig. 5(a) and (b), NebulaFL excels in accelerating MNIST and FMNIST tasks, maintaining a significant performance advantage throughout training. Async-HFL performs well on these simpler datasets due to its rapid asynchronous iterations. FedCH's convergence fluctuates due to its latency-based rules but stabilizes with more iterations. HierFAVG, using a fully synchronous strategy, achieves high-quality iterations but suffers from the straggler issues limiting its iteration count and slowing down training. For the complex tasks like CIFAR-10 and Shakespeare [Fig. 5(c) and (d)], convergence is naturally slower and more complex. Here, the performance heavily depends on how the algorithms handle the data heterogeneity, communication bottlenecks, and synchronization. Async-HFL and Hier-SAFA struggle with the Shakespeare task and often fail to reach TA in time. In contrast, FedCH-random performs better, particularly with the Oort device selection algorithm that considers the latency and data heterogeneity. NebulaFL and its extensions consistently deliver the best overall results.

3) Analysis of Time Savings: Fig. 6 compares the training time required for various HFL methods to achieve the target accuracies across different tasks. Blue ⋆ and red ⋆ represent the training times for different target accuracies, while the black × indicates failure to reach the target within the given time. The position of the blue stars highlights NebulaFL's significant time efficiency advantage across all the tasks with the red stars showing its robust convergence in later training stages. For example, on the MNIST dataset, NebulaFL saved up to 1.56× the training time compared to the next best method, Async-HFL, and up to 6.6× compared to the least efficient baseline. The introduction of step size adjustment in NebulaFL+ further enhanced these savings. For the challenging CIFAR-10 dataset, NebulaFL reduced training time by 1.42× compared to the next best method, FedCH-Oort. Meanwhile, asynchronous methods (Async-HFL) and methods with random associations

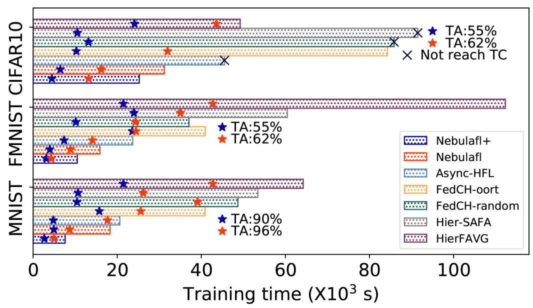


Fig. 6. Time spend to achieve TA under different tasks.

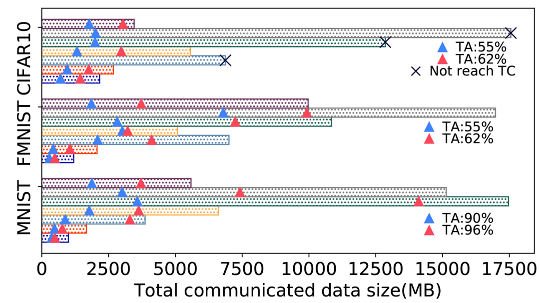
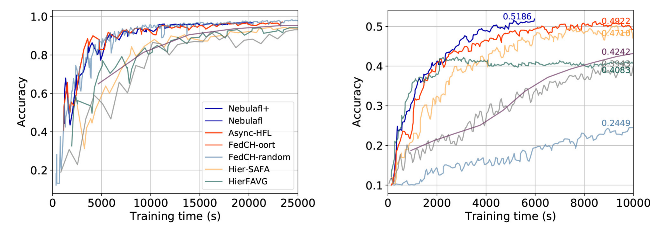


Fig. 7. Communication spend to achieve TA under different tasks.



(a) (b)

Fig. 8. Robustness analysis. (a) Impact of device drops. (b) Impact of large scale devices. (a) MNIST with SimpleCNN. (b) Cifar10 with SimpleCNN.

(HierFAVG or Hier-SAFA) failed to reach the TA within the allocated time.

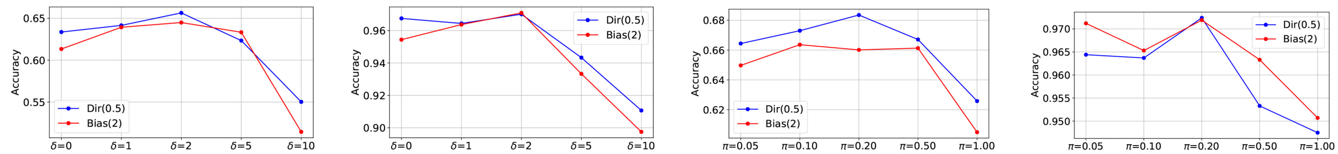
4) Analysis of Communications Savings: We record the communication overhead in Fig. 7, which is defined as the total amount of the model information transmitted during the training. Blue (▲) and red (▲) indicate the communication overhead needed to achieve a specific TA. Communication savings are generally related to the training rounds and the duration of rounds. Compared to Async-HFL, NebulaFL achieves higher quality per training round, reducing the number of rounds needed for the convergence and thus saving the communication costs. While the asynchronous communication requires more rounds than the synchronous training, it promotes rapid convergence, reducing the total latency. In the CIFAR10 training task, NebulaFL reduced communication overhead by at least 618.75 MB compared to the next best method, FedCH-Oort. For FedCH and Hier-SAFA, the lack of consideration for data heterogeneity requires more iterations, leading to increased communication overhead. For example, in the MNIST and FMNIST training tasks, FedCH's communication overhead increased by 2398.92 and 3103.12 MB compared to NebulaFL.

5) Analysis of Robustness: Fig. 8(a) shows the convergence behavior after 20 000 s of training on MNIST with a 20% dropout rate per round. When devices drop out, the synchronization mechanism cannot detect it immediately, often requiring a long wait time. We set the synchronization wait time due to device dropout to 1.2×, the maximum device delay under normal operation. The asynchronous baseline shows a more significant convergence effect because when a device drops out, the fully asynchronous method (Async-HFL) does not affect the other devices. In contrast, NebulaFL limits the impact to a synchronous device layer under the gateway. Additionally, NebulaFL may become the preferred choice for the large-scale hierarchical training systems with the random offline devices due to its high performance in the complex tasks. To further validate NebulaFL's high performance in larger systems, we conducted CIFAR10 training using SimpleCNN on 500 nodes and achieved similar convergence results as shown in Fig. 8(b). The results once again demonstrate NebulaFL's effectiveness in large-scale operations.

6) Hyperparametric Ablation: We explore the effects of the utility penalty factor δ in (14) and the load factor π in Section IV-C, respectively. The δ moderates the data and statistical heterogeneity among the devices and affects the utility value $Utils(\cdot)$. Fig. 9(a) and (b) show that the $\delta = 1, 2, 5$ setting maintains robust accuracy. In contrast, $\delta = 0$ causes the system to prioritize the data distribution, leading to longer training time per round. When $\delta = 10$, the system over-penalizes slower devices, thus increasing the number of training rounds required. Therefore, an appropriate δ can improve the convergence efficiency. Fig. 9(c) and (d) also investigate the tuning of π, and the results show that the accuracy initially improves as π increases but decreases beyond a certain threshold. This suggests that there exists an appropriate π range that improves accuracy without causing the performance degradation [6] due to longer training time per round or increased load on the device layer.

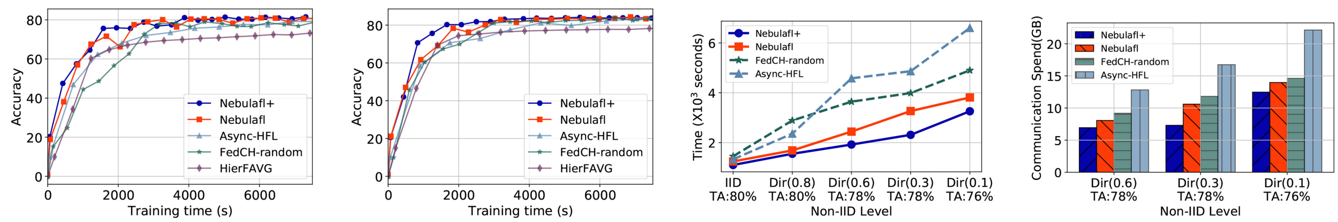
7) Results on Physical Platform: We trained the CIFAR10 dataset using a larger VGG11 model on a physical platform. The runtime is set to 8000 s, with each gateway executing $R = 10$ rounds per cloud cycle and each device performing local training for $E = 2$ epochs. The hyperparameters were consistent with those used in the simulation experiments. Fig. 10(a) and (b) summarize the convergence process over wall-clock time (recording 15 time-accuracy points for each method), while Fig. 10(c) and (d) display resource consumption. Notably, because of the limited number of devices, each device could be allocated enough training data, resulting in relatively similar performance across the methods. However, NebulaFL still demonstrated a significant convergence advantage. As shown in Fig. 10(b), NebulaFL+ achieved 80% accuracy in less than 1755 s, whereas the next best baseline took over 2806 s to reach the same accuracy.

Regarding time consumption as shown in Fig. 10(c), NebulaFL used the least training time across various data heterogeneity levels and maintained stable performance as heterogeneity changed. Regarding communication resource consumption, NebulaFL generally incurred lower overhead due to its faster convergence. For example, at a Dir(0.6) data heterogeneity level, NebulaFL saved 1.1 GB of traffic



(a) (b) (c) (d)

Fig. 9. Performance at different penalty factors δ and load tuning factors π. (a) δ value in CIFAR10. (b) δ value in MNIST. (c) π value in CIFAR10. (d) π value in MNIST.



(a) (b) (c) (d)

Fig. 10. Left two figures: convergence performance on physical platform. Right two figures: resource consumption on physical platform. (a) CIFAR10 with Dir(0.3). (b) CIFAR10 with Dir(0.6). (c) Time consumption. (d) Communication overhead.

compared to the next best baseline, FedCH, to reach a TA of 78%. This efficiency is attributed to the NebulaFL's device organization scheme, which accounts for the dual heterogeneity. Additionally, NebulaFL+ performed even better, thanks to intelligent load adjustment.

C. Discussion: Overhead and Privacy

1) Overhead Analysis: The main reasons for additional overhead are the architectural organization and load-tuning process. In terms of computational overhead, given the limited number of devices and gateways, the Gale–Shapley matching algorithm and the Louvain layering algorithm that we use in our experiments can be solved in a very short period ranging from 3 to 10 s by utilizing the abundant computational power in the cloud infrastructure. As far as the communication overhead is concerned, it mainly consists of exchanging the device characteristics, data attributes, and training loss metrics each round. However, compared to the transmitted model parameters (MB/GB), this overhead (kB) is negligible.

2) Privacy Discussion: NebulaFL optimizes the training architecture to enhance performance but is orthogonal to privacy-preserving algorithms like differential privacy and homomorphic encryption. Differential privacy can be applied at the device level by adding noise during local training, maintaining training efficiency, and can be independently integrated with NebulaFL. Furthermore, NebulaFL's hierarchical architecture, with distributed gateways, reduces the risk of single-point privacy breaches. The information sent from gateways to the server is aggregated model data, not raw data, further minimizing potential risks.

VI. Related Work

FL [2] trains distributed models on the edge devices to create privacy-preserving intelligent systems and is considered a leading technique in the edge intelligence research [31]. FedAvg [2] introduced a synchronous training mechanism, and subsequent research has focused on optimizing the data heterogeneity [32] or the system heterogeneity [33]. Innovations in algorithms and system design, such as asynchronous communication [34], weak synchronization mechanisms [35], and semi-asynchronous methods [30], have improved the training efficiency. Device Selection [11] can also enhance the training efficiency and reduce the communication stress. However, these approaches must address the IoT network challenges, like long-distance and unreliable wide-area communication.

HFL [36] addresses the above challenges with several approaches. HierFAVG [7] deploys an HFL scheme to tackle long-distance communication challenges by incorporating gateways, ensuring synchronized aggregation at both the gateway and the cloud levels. On the other hand, FedCH [9] adopts a device performance-based HFL approach, moving to asynchronous communication between the gateways and the cloud to address the straggler issues and optimize the number of layers. HiFlash [10] follows a similar communication mechanism to FedCH, explicitly focusing on minimizing the model version inconsistencies during asynchronous updates. Async-HFL [8] offers a fully asynchronous FL model, establishing the benchmarks for the device association and selection in asynchronous training despite ongoing challenges in asynchronous interactions. HierFedML [37] and [38] also explore the system cost minimization in multiaccess edge computing (MEC) environments to minimize the training loss and round delay. Meanwhile, strategies like HACCS [12] concentrate on the data distribution, grouping clients with similar data patterns to tackle statistical heterogeneity.

However, the works above often assume that the devices under the same gateway tend to be homogeneous, adopting synchronous or weakly synchronous aggregation for training, inevitably leading to inefficient training bottlenecks. Unlike these works, this article focuses on leveraging the IoT's naturally existing hierarchical architecture to design the efficient communication mechanisms, construct more fine grained and efficient device organization schemes, and related the training optimization algorithms.

VII. Conclusion

This article proposes NebulaFL to improve the training efficiency in hierarchical IoT scenarios through a finer-grained self-organizing layering scheme and load adaptive tuning. Specifically, NebulaFL tackles two critical issues as follows.

1) Optimal device placement by designing the device association schemes and device layering methods considering the static impact of dual heterogeneity.
2) Adjusting the training load of the lowest device layer by considering the dynamic impact of dual heterogeneity.

Experiments show that NebulaFL significantly enhances the training accuracy and speed in both the simulated and physical systems while greatly reducing the communication costs. NebulaFL can be widely applied to various scenarios, such as online training of recommendation systems, multi-device collaborative inference systems, and lifelong learning systems, regardless of device configurations, model structures, or training algorithms. While NebulaFL offers many advantages, differences in hardware and operating systems may require additional compatibility adjustments. We plan to use containerization tools like Docker to simplify deployment and will continue to address engineering challenges in real-world deployments to enhance NebulaFL's usability. We hope the NebulaFL framework will provide guidance and inspiration for distributed training across large-scale devices in the IoT.

References

[1] D. Wu, W. Yang, H. Jin, X. Zou, W. Xia, and B. Fang, "FedComp: A federated learning compression framework for resource-constrained edge computing devices," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 230–243, Jan. 2024.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Stat.*, 2017, pp. 1–10.

[3] B. Mao, J. Liu, Y. Wu, and N. Kato, "Security and privacy on 6G network edge: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1095–1127, 2nd Quart., 2023.

[4] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019, *arXiv:1903.03934*.

[5] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.

[6] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 7611–7623.

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

[8] X. Yu et al., "Async-HFL: Efficient and robust asynchronous federated learning in hierarchical IoT networks," in *Proc. 8th ACM/IEEE Conf. Internet Things Design Implement.*, 2023, pp. 236–248.

[9] Z. Wang, H. Xu, J. Liu, Y. Xu, H. Huang, and Y. Zhao, "Accelerating federated learning with cluster construction and hierarchical aggregation," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 3805–3822, Jul. 2023.

[10] Q. Wu et al., "HiFlash: Communication-efficient hierarchical federated learning with adaptive staleness control and heterogeneity-aware client-edge association," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 5, pp. 1560–1579, May 2023.

[11] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *Proc. Oper. Syst. Design Implement. (OSDI)*, 2021, pp. 1–18.

[12] J. Wolfrath, N. Sreekumar, D. Kumar, Y. Wang, and A. Chandra, "HACCS: Heterogeneity-aware clustered client selection for accelerated federated learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2022, pp. 985–995.

[13] Z. Li et al., "Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching," *IEEE Trans. Big Data*, early access, Nov. 17, 2022, doi: 10.1109/TBDATA.2022.3222971.

[14] R. Tiwari, K. Killamsetty, R. Iyer, and P. Shenoy, "GCR: Gradient coreset based replay buffer selection for continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 99–108.

[15] N. Wang, R. Zhou, L. Su, G. Fang, and Z. Li, "Adaptive clustered federated learning for clients with time-varying interests," in *Proc. IEEE/ACM 30th Int. Symp. Qual. Service (IWQoS)*, 2022, pp. 1–10.

[16] B. Fuglede and F. Topsoe, "Jensen-Shannon divergence and Hilbert space embedding," *Int. Symp. Inf. Theory*, 2004, p. 31.

[17] L. E. Dubins and D. A. Freedman, "Machiavelli and the Gale-Shapley algorithm," *Amer. Math. Month.*, vol. 88, no. 7, pp. 485–494, 1981.

[18] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Generalized Louvain method for community detection in large networks," in *Proc. 11th Int. Conf. Intell. Syst. Design Appl.*, 2011, pp. 88–93.

[19] A. Garivier and E. Moulines, "On upper-confidence bound policies for non-stationary bandit problems," 2008, *arXiv:0805.3415*.

[20] Z. Lian, J. Cao, Z. Zhu, X. Zhou, and W. Liu, "GOFL: An accurate and efficient federated learning framework based on gradient optimization in heterogeneous IoT systems," *IEEE Internet Things J.*, vol. 11, no. 7, pp. 12459–12474, Apr. 2024.

[21] E. Ekaireb et al., "ns3-fl: Simulating federated learning with ns-3," in *Proc. Workshop ns-3*, 2022, pp. 97–104.

[22] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[23] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*,

[24] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[25] S. Caldas et al., "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

[26] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-IID data silos: An experimental study," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, 2022, pp. 965–978.

[27] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 2809–2813.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

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

[30] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "SAFA: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 655–668, May 2021.

[31] B. Liu, N. Lv, Y. Guo, and Y. Li, "Recent advances on federated learning: A systematic survey," 2023, *arXiv:2301.01299*.

[32] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, "Heterogeneous federated learning: State-of-the-art and research challenges," *ACM Comput. Surv.*, vol. 56, no. 3, pp. 1–44, 2023.

[33] O. R. A. Almanifi, C. O. Chow, M. L. Tham, J. H. Chuah, and J. Kanesan, "Communication and computation efficiency in federated learning: A survey," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100742.

[34] C.-H. Hu, Z. Chen, and E. G. Larsson, "Scheduling and aggregation design for asynchronous federated learning over wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 874–886, Apr. 2023.

[35] J. Cao et al., "FedStar: Efficient federated learning on heterogeneous communication networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 6, pp. 1848–1861, Jun. 2024.

[36] J. Yan, T. Chen, B. Xie, Y. Sun, S. Zhou, and Z. Niu, "Hierarchical federated learning: Architecture, challenges, and its implementation in vehicular networks," *ZTE Commun.*, vol. 21, no. 1, pp. 38–45, 2023.

[37] Z. Xu et al., "HierFedML: Aggregator placement and UE assignment for hierarchical federated learning in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 328–345, Jan. 2023.

[38] Y. Cui, K. Cao, J. Zhou, and T. Wei, "Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge cloud computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1518–1531, May 2023.