FINCH: Enhancing Federated Learning with Hierarchical Neural Architecture Search

Jianchun Liu, *Member, IEEE,ACM,* Jiaming Yan, Hongli Xu, *Member, IEEE,* Zhiyuan Wang, Jinyang Huang, Yang Xu, *Member, IEEE,*

Abstract—Federated learning (FL) has been widely adopted to train machine learning models over massive data in edge computing. Most works of FL employ pre-defined model architectures on all participating clients for model training. However, these pre-defined architectures may not be the optimal choice for the FL setting since manually designing a high-performance neural architecture is complicated and burdensome with intense human expertise and effort, which easily makes the model training fall into the local suboptimal solution. To this end, Neural Architecture Search (NAS) has been applied to FL to address this critical issue. Unfortunately, the search space of existing federated NAS approaches is extraordinarily large, resulting in unacceptable completion time on the resource-constrained edge clients, especially under the non-independent and identically distributed (non-IID) setting. In order to remedy this, we propose a novel framework, called FINCH, which adopts hierarchical neural architecture search to enhance federated learning. In FINCH, we first divide the clients into several clusters according to the data distribution. Then, some subnets are sampled from a pre-trained supernet and allocated to the specific client clusters for searching the optimal model architecture in parallel, so as to significantly accelerate the process of model searching and training. The extensive experimental results demonstrate the high effectiveness of our proposed framework. Specifically, FINCH can reduce the completion time by about 30.6%, and achieve an average accuracy improvement of around 9.8% compared with the baselines.

Index Terms—*Edge Computing, Federated Learning, Non-IID data, Neural Architecture Search.*

1 INTRODUCTION

With the rapid proliferation of mobile devices in Internet of Things (IoT), more and more data are accumulated at the network edge (*e.g.*, gateway, switch) [1]. With more data and advanced applications (*e.g.*, autonomous driving, virtual reality), machine learning (ML) tasks will be a dominant workload in edge computing (EC) systems [2]. To alleviate the network bandwidth burden and protect privacy, federated learning (FL) becomes an efficient solution to analyze and process the distributed data on edge nodes for those ML tasks [3]. In some traditional FL schemes (*e.g.*, *FedAvg* [3], *CFL* [4]), the parameter server (PS) first distributes the updated global model to the edge clients for training on their local datasets. Then, the clients send the local updated gradient (or model) to the server for global aggregation. Since the server will not directly access the local data of the clients, the risk of privacy leakage in the clients can be significantly relieved.

We observe that the existing FL schemes usually adopt a pre-defined model architecture with human expertise, bringing two primary disadvantages. First, the pre-defined architecture easily makes the model training fall into the local sub-optimal solution, leading to low training performance. Second, it is complex to develop an accurate and small enough model to be deployed at clients, which demands intense human effort in an iterative trial and error process [5]. In other words, manually designing more efficient architectures heavily relies on human experts' experience. In order to mitigate the above disadvantages, we make the attempt to adopt Neural Architecture Search (NAS) [6] technique to design the proper network architecture for FL, which automates the design process of the model architecture without inefficient manual attempts. The searched NAS architectures have outperformed the best expert-designed architectures on many computer vision and natural language processing tasks [7].

To implement efficient FL with NAS in the edge computing, the following challenges should be taken into considerations.

- **Non-IID Data:** Both FL and NAS rely on the stochastic gradient descent (SGD) algorithm [8], which is widely used to train neural networks with good empirical performance. The gradient-based algorithms are often based on the main assumption that data is independent and identically distributed (IID) [9]. However, it is impractical to assume that the local data on each client is always IID. For example, two surveillance cameras in different geographical locations may capture quite different views, *e.g.*, pedestrians and vehicles. Therefore, the training performance degrades severely when the local datasets are highly Non-IID [10].
- **Limited Communication Resource:** Different from the dat-

- *J. Liu, J. Yan, H. Xu, Z. Wang and Y. Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: jcliu17@ustc.edu.cn, jmyan@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, cswangzy@mail.ustc.edu.cn, xuyangcs@ustc.edu.cn.*
- *J. Huang is with the S^2AC Laboratory, School of Computer and Information and Key Laboratory of Knowledge Engineering with Big Data, Hefei University of Technology, Anhui 230002, China. E-mail: hjy@hfut.edu.cn*

TABLE 1: The comparison of the previous solutions and ours.

Schemes	Non-IID Data	Limited Resource
FedNAS [5] One-shot NAS [16]	●	○
OneNas [17]	○	●
FedorAS [18]	◐	◐
Ours	●	●

acenters with sufficient communication resources, the bandwidth between the server and the workers is always limited in edge computing [11]. For example, the bandwidth within typical WANs is only 5$\sim$25Mb/s and is much less than that within datacenters (over 10Gb/s) [12]. Since the model parameters need to be frequently transmitted between the server and clients through WAN, FL demands a large amount of communication resources in contrast to the conventional centralized learning paradigm. As the size of VGG16 for the ImageNet dataset reaches 512MB [13], FL requires to consume more than 50GB of bandwidth if there are hundreds of workers, leading to network congestion easily.

- **Limited Computation Resource:** It has been proved that one-shot NAS method can achieve excellent performance in model searching [14], [15], which is also adopted in this work. In order to search for the optimal model architecture, a large number of subnets are sampled from a pre-trained supernet. Assume that a supernet containing 10 choice nodes (*i.e.*, neurons) and the number of operations (*e.g.*, convolution, pooling) between each node pair is 4, there would be 4^{10}= 1,048,576 subnets in total. Thus, the cost of training the supernet could be multiple times that of a single subnet, which will result in an unacceptable computation overhead especially for the resource-constrained edge clients, leading to a long completion time.

Recently, several efforts have been made to tackle these challenges, which can be divided into two categories. A natural solution (or the *first* category) is to perform training on the whole model architecture, *i.e.*, supernet-based [5], [16]. For example, He *et al.* [5] consider the solution, called FedNas, of searching for a better deep neural network (DNN) architecture to improve the training performance under non-IID settings. Zhang *et al.* [16] advocate one-shot NAS as a basis to deal with the non-IID problem. However, their demand for great computational power contrasts with the low processing capacity on edge clients, leading to a long completion time. The *second* category [17] [18] tries to sample some subnets from the supernet to reduce the search space which consists of a lot of network architectures. In OneNAS [17], a randomly sampled subnet is transmitted to a number of randomly sampled clients for training without reinitialization, so as to reduce the communication and computation costs. FedorAS [18] samples a one-shot path for every client cluster based on the resource budgets for model search and training. Nevertheless, the process of fine-tuning or personalizing will introduce the extra computation cost. Moreover, both these schemes ignore the influence of data imbalance, *i.e.*, non-IID data, on model searching and training. We compare the previous solutions with ours in Table 1. In a nutshell, none of the aforementioned works can fully address these critical challenges for FL with NAS.

To mitigate the challenges of non-IID data and limited network resource, we propose a novel framework, called FINCH, to enhance federated learning with hierarchical neural architecture search in edge computing. Specifically, we first divide all participating clients into different clusters according to the data distribution on each client, *i.e.*, the data distribution in each cluster is close to independent identically distributed (IID). Then, several subnets will be sampled from the supernet and allocated to the proper client clusters for model searching and training, so as to significantly reduce the resource (*e.g.*, computation and communication) cost and completion time. According to the testing results in Section 2.3, the training performance of FL mainly depends on the client clustering and subnet allocation. Thus, *how to properly divide the clients into clusters and how to allocate the subnets to them* are the key challenges in FINCH. The main contributions of this paper are:

- We propose a novel framework, called FINCH, which adopts hierarchical neural architecture search (NAS) to accelerate the process of federated learning, and formally proves the convergence of FINCH.
- We design an efficient algorithm (termed DCSA) to adaptively perform client clustering and subnet allocation, so as to achieve less completion time and resource usage for model searching and training.
- The extensive experimental results demonstrate the high effectiveness of our proposed framework. Specifically, FINCH can improve the test accuracy by about 9.8% under the resource constraints, and reduce the time cost by about 30.6%, compared with the benchmarks.

The rest of this paper is organized as follows. Section 2 introduces some preliminaries and proposes the novel framework of FINCH. The problem is formalized and the efficient algorithms are proposed for FINCH in Section 3. We give the convergence analysis in Section 4. In Section 5, the experiments are conducted and the corresponding results are presented. The related works are summarized in Section 6. Finally, we conclude the paper in Section 7.

2 PRELIMINARIES AND PROBLEM FORMULATION

2.1 Background

2.1.1 *Federated Learning*

FL is a variant of distributed machine learning (DML) and its objective function can be decomposed into a linear combination of K local objective functions, where $K = |\mathcal{V}| > 1$ and $\mathcal{V}$ is the set of clients participating in the model training. Each local objective function depends on the private data hosted by a client and a set of shared parameters ω. Considering a standard supervised learning task (*e.g.*, L-class classification) where the predictive model is modeled as a fixed deep neural network $\mathcal{A}$ with learnable parameters ω, the expected loss on each client $v_k \in \mathcal{V}$ is given as:

$$\mathcal{F}_k\left(\omega, \mathcal{A} \mid \mathcal{D}_k\right) = \frac{1}{N_k} \sum_{i \in \mathcal{D}_k} f(x_i, y_i; \omega, \mathcal{A}) \tag{1}$$

where $\mathcal{D}_k := \{(x_j, y_j)\}_{j=1}^{N_k}$ is the local training data comprising input-output tuples (x, y) and N_k is the local data size on the

TABLE 2: Key Notations

Symbol	Semantics
K	a set of local objective functions
T	the total number of training iterations
$\mathcal{V}$	a set of participating clients
$\mathcal{A}$	the neural network architecture
ω	the model parameters
δ	the convergence threshold of the model training
$\mathcal{D}_k$	the local dataset on the client k
N	the total data size on all clients
N_k	the local data size on the client v_k
$\mathcal{D}_k$	the local dataset in the client k
$\mathcal{Q}$	the set of aggregators
B	the bandwidth budget on the server
b_r^t	the bandwidth consumption by aggregator r for transmitting the model to the server at the cluster $\mathcal{C}_j$
h_i^t	the completion time of the cluster $\mathcal{C}_j$ at iteration t
$\mathcal{F}(w)$	the empirical loss function

client v_k. The total data size on all clients is denoted as $N = \sum_{v_k \in \mathcal{V}} N_k$. The learning problem is to find the optimal parameter vector ω^*, and the objective function of FL is defined as:

$$\min_{\omega} \mathcal{F}(\omega, \mathcal{A}) \overset{\text{def}}{=} \min_{\omega} \sum_{k=1}^{K} \frac{N_k}{N} \cdot \mathcal{F}_k(\omega, \mathcal{A} \mid \mathcal{D}_k) \tag{2}$$

Due to non-convexity of the loss functions, the widely used optimization methods of network parameters are gradient based, *e.g.*, stochastic gradient descent (SGD) [9]. Gradient descent methods take derivatives of loss function according to the model parameters, then move the parameter values in the negative of the gradient. Some important notations are listed in Table 2.

2.1.2 *Neural Architecture Search*

Neural Architecture Search (NAS) is proposed to automately design the proper network architecture for FL without inefficient manual attempts [6]. There are three major components in NAS as summarized in [19], namely search space, search policy, and performance evaluation. Considering a wide range of IoT applications, *e.g.*, video surveillance, we focus on a computational vision problem which always adopts the convolution neural network (CNN) model for training. Each node (*i.e.*, neuron in CNN) is a latent representation (*e.g.*, a feature map in convolutional networks) and each directed edge between two nodes is associated with some operations (*e.g.*, convolution, max pooling, *zero*) that transform the node. A learned cell, which is a directed acyclic graph (DAG) consisting of an ordered sequence of some nodes, could either be stacked to form a convolutional network or recursively connected to form a recurrent network. For convolutional cells, the input nodes are defined as the cell outputs in the previous two layers [5]. The network architecture of the pre-trained model on the server is usually composed of numerous amount of cells, leading to a large search space.

In order to accomplish the architecture search, early pieces of literature mainly adopt several search policies, *e.g.*, reinforcement

Algorithm 1 FL with Hierarchical NAS (FINCH)

```
1:  Initialize parameter $\omega$ and architecture $\mathcal{A}$ of the supernet
2:  **Processing at the PS**
3:  **if** the requirements of model training is met **then**
4:      **for** each global update **do**
5:          **while** number of received cluster models $< M$ **do**
6:              Waiting for the cluster models from the
                selected aggregators
7:          Update the global parameter $\omega$ and architecture $\mathcal{A}$
8:          Sample the subnets from the supernet
9:          Distribute the subnets to the cluster $\mathcal{C}_j$ according
            to the proposed algorithm in Section 3
10:         Update the resource budgets
11: **Processing at aggregator** $q_j$
12: **for** $t' = 1$ to $T/\lambda_1$ **do**
13:     Receive $\omega^i_{t'}$ and $\mathcal{A}^i_{t'}$ from each client $v_i \in \mathcal{C}_j$
14:     Obtain the cluster model $\tilde{\omega}^j_{t'}$ and $\tilde{\mathcal{A}}^j_{t'}$ by Eq. (5)
15:     **if** $t'$ mod $\lambda_2$ == 0 **then**
16:         Send $\tilde{\omega}^j_{t'}$ and $\tilde{\mathcal{A}}^j_{t'}$ to the parameter server
17:         Receive the fresh global model from PS
18:         Updated the local cluster model
19:     Distribute $\tilde{\omega}^j_{t'}$ and $\tilde{\mathcal{A}}^j_{t'}$ to all clients in $\mathcal{C}_j$
20: **Processing at client** $v_i$
21: **for** $t'$=1 to T **do**
22:     Obtain local model $\omega^i_{t'}$ and $\mathcal{A}^i_{t'}$ by Eq. (4)
23:     **if** $t'$ mod $\lambda_1$ == 0 **then**
24:         Upload $\omega^i_{t'}$ and $\mathcal{A}^i_{t'}$ to the aggregator $q_j$
25:         Receive the updated model from $q_j$
```

learning (RL) [20] or evolutionary algorithms [17]. Nevertheless, these two solutions often use hundreds of GPUs for computation and take a large volume of GPU hours to finish the searching [7]. For the sake of searching efficiency, Liu *et al.* [21] propose a differentiable NAS variant, in which a one-shot over-parameterized supernet is regarded as a full graph and all candidate architectures are derived as its sub-graphs (or subnets). To solve the objective function Eq. (2), previous works [22], [23] choose a fixed model architecture $\mathcal{A}$ and then design various optimization techniques to train the parameters ω. Our goal is to jointly learn the architecture $\mathcal{A}$ and the parameters ω within all the mixed operations (*e.g.*, parameters of the convolution filters). Thus, the objective function Eq. (2) can be reformulated as:

$$\min_{\omega,\mathcal{A}} \mathcal{F}(\omega, \mathcal{A}) \overset{\text{def}}{=} \min_{\omega,\mathcal{A}} \sum_{k=1}^{K} \frac{N_k}{N} \cdot \mathcal{F}_k(\omega, \mathcal{A} \mid \mathcal{D}_k) \tag{3}$$

Different from Eq. (2), the parameters ω and architecture $\mathcal{A}$ will be optimized simultaneously through solving the objective function Eq. (3).

2.2 FL with Hierarchical NAS

The previous works of FL with NAS mainly focus on finding an optimal candidate model (*e.g.*, architecture and parameter) from a huge search space, leading to an enormous communication/computation overhead and an unacceptable completion time

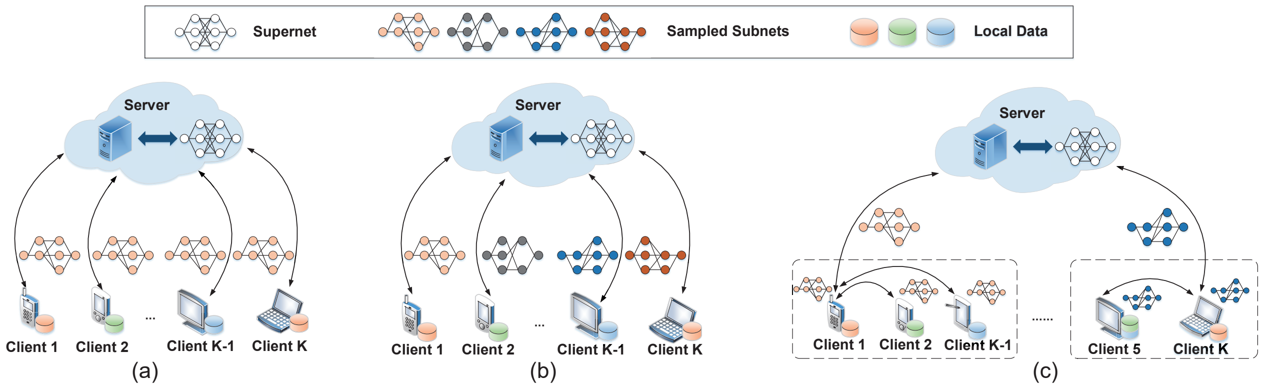


Fig. 1: Illustration of different NAS approachs. (a) OneNAS [17]: the same subnet is allocated to all clients; (b) DecNas [6]: different subnets are allocated to different clients; (c) Our hierarchical NAS: different subnets are allocated to each client cluster and the client in each cluster adopts the same subnet for searching and training.

[24]. Although some works try to conquer these disadvantages, they always ignore the impact of non-IID data on the model searching and training, resulting in low performance. In order to tackle these challenges, we propose a novel framework to accelerate the training process of **f**ederated learning with h**i**erarchical **n**eural ar**c**hitecture searc**h** (FINCH) in edge computing.

FINCH aims to efficiently search for better model architecture and parameters within a given resource budget, considering data distribution among the clients. In order to significantly reduce the search space and accelerate the training process, we divide the clients into multiple clusters according to the data distribution and allocate proper subnets from the pre-trained supernet to these specified client clusters for model searching and training. The framework of FINCH includes the client side, the aggregator side and the parameter server side from bottom up. Specifically, the overall procedure of FINCH is illustrated in Alg. 1, including client clustering, local model searching and training, intra-cluster aggregation and inter-cluster aggregation.

1) *Client Clustering:* The clients in the network are organized into M clusters, *i.e.*, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_M\}$, satisfying $\cup_{j=1}^{M}\mathcal{C}_j = \mathcal{V}$ and $\mathcal{C}_i \neq \varnothing$, $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing, \forall i \neq j$. In each cluster, a client will be randomly selected as the aggregator. Each client sends its local updated model parameters and architecture to the selected aggregator q_j in the cluster $\mathcal{C}_j$, instead of directly sending them to the remote server for global aggregation. The set of all aggregators is denoted as $\mathcal{Q} = \{q_1, q_2, ..., q_M\}$ with $M = |\mathcal{Q}|$. Note that the detailed clustering algorithm will be introduced in Section 3.

2) *Local Model Searching and Training:* In the t-th iteration, each client v_i performs model updating of parameters and architecture over its local dataset by the mini-batch SGD algorithm [7] (Line 21-25):

$$\begin{cases} \omega_t^i = \omega_{t-1}^i - \eta_\omega \nabla\mathcal{F}_{\mathrm{tr}}(\omega_{t-1}^i, \mathcal{A}) \\ \mathcal{A}_t^i = \mathcal{A}_{t-1}^i - \eta_\mathcal{A}\left[\nabla\mathcal{F}_{\mathrm{tr}}(\omega, \mathcal{A}_{t-1}^i) + \pi\nabla\mathcal{F}_{\mathrm{val}}(\omega, \mathcal{A}_{t-1}^i)\right] \end{cases} \tag{4}$$

where ω_{t-1}^i is the latest model available to the i-th client at the start of the t-th iteration. $\mathcal{F}_{\mathrm{tr}}(w, \mathcal{A})$ and $\mathcal{F}_{\mathrm{val}}(w, \mathcal{A})$ denote the loss with respect to the local training data and validation data with ω and $\mathcal{A}$, respectively. η is the learning rate and π is a non-negative regularization parameter that balances the importance of the training loss and validation loss. $\nabla\mathcal{F}(\cdot)$ denotes the gradient computed on the batch of randomly-sampled local data.

3) *Intra-cluster aggregation:* Let λ_1 denote the number of local updating iterations before each intra-cluster aggregation, *i.e.*, each client sends its local model updates to the aggregator if the number of iterations t is an integer multiple of λ_1 (Line 13). The aggregator q_j will aggregate the local updated model from the clients $v_i \in \mathcal{C}_j$ [5] (Line 14):

$$\tilde{\omega}_t^j = \sum_{v_i\in\mathcal{C}_j} \frac{N_i}{N^j}\omega_t^i$$
$$\tilde{\mathcal{A}}_t^j = \sum_{v_i\in\mathcal{C}_j} \frac{N_i}{N^j}\mathcal{A}_t^i \tag{5}$$

where N^j denotes the total data size of clients in the cluster $\mathcal{C}_j$.

4) *Inter-cluster aggregation:* Let λ_2 denote the number of intra-cluster aggregations on each aggregator before the inter-cluster aggregation, *i.e.*, global model updating. If the number of iterations t is the integer multiple of $\lambda_1\lambda_2$, the parameter server will perform inter-cluster aggregation to obtain the updated model ω_t^j and architecture $\mathcal{A}_t^j$ (Line 3-7). After that, several subnets will be sampled from the supernet and allocated to the aggregators (Line 8-10). Subsequently, the aggregator q_j distributes the updated model and architecture to each client in the cluster $\mathcal{C}_j$, *i.e.*, $\omega_t^i = \omega_t^j, \mathcal{A}_t^i = \mathcal{A}_t^j$. Then, the client continues model searching with the received model from the aggregator for the next iteration. During the aggregation, the supernet will directly replace its parameters with the subnets' model parameters, if there is no overlap between different subnets. Otherwise, the supernet will aggregate the parameters with different weights, if there is overlap between different subnets.

2.3 Illustration of FINCH

For a better explanation of FINCH, we give an example in Fig. 1. We adopt three different model searching methods (cases I-III) for comparison. As shown in the left plot of Fig. 1 (case I), only one subnet will be sampled and distributed to all participating

TABLE 3: The completion time and test accuracy of FL under different schemes.

Schemes	Fixed Target Accuracy		Fixed Completion Time	
	Same Subnet	Different Subnets	Same Subnet	Different Subnets
Non-Clustered	1.25 GPU/h	1.18 GPU/h	59.7%	62.3%
Clustered	1.06 GPU/h	0.85 GPU/h	67.4%	76.5%

clients in each iteration. Different from case I, each client will be allocated a unique subnet sampled from the supernet for model searching and training in case II, shown in the middle plot of Fig. 1. In FL, the performance of model searching and training (*e.g.*, completion time and test accuracy) will be degraded on each client due to the skewness of the local data [16]. To this end, we divide the participating clients into multiple clusters according to the data on these clients, *i.e.*, data distribution in each cluster is close to IID. For our proposed FINCH framework (case III), several subnets will be sampled and allocated to the proper client clusters for model searching and training, which will significantly accelerate the process of FL. Combining the example in Fig. 1, we perform two groups of tests to illustrate the advantages of FINCH.

We first test the performance (*e.g.*, completion time and test accuracy) of FL under the non-IID setting. The first set of experiments observes the completion time of model searching and training, within a target test accuracy (*e.g.*, 70%). We allocate the same subnet and different subnets to the clients, *i.e.*, cases I and II, respectively. As shown in Table 3, these two cases take relatively long completion time (*e.g.*, 1.25 and 1.18 GPU/hours) to achieve the target test accuracy because the data is Non-IID. To alleviate this problem, we try to perform client clustering such that the data distribution in each cluster is close to IID. When the same subnet is allocated to the clustered clients, the performance of model searching and training is well improved, *e.g.*, the completion time is 1.06 GPU/hours. In order to further improve the time cost, we sample the different subnets to the clustered clients for searching and training in parallel, and the completion time of case III is about 0.85 GPU/hours. In other words, our proposed method (case III) can reduce the completion time by about 27% compared with the other solutions.

We also observe the performance of test accuracy within a fixed completion time (*e.g.*, 1 GPU/hour). Since our proposed method can reduce the search space, less time is needed to complete the model searching and training. Table 3 shows that our proposed method outperforms the other three solutions. Specifically, case III can improve the average test accuracy by about 13.3% compared with cases I and II. Thus, our FINCH can significantly improve the training performance, including completion time and test accuracy, compared with the benchmark cases. Nevertheless, the performance improvement of model searching and training also depends on the different ways of client clustering and subnet allocation, which will be introduced in Section 3.

3 ALGORITHM DESIGN

In fact, the data distribution on the clients may not frequently change [25]. Thus, we first perform client clustering after a given number (*e.g.*, 10) of rounds. We design an efficient algorithm, called DCSA, to obtain the optimal client clustering and subnet allocation, respectively. In DCSA, we first find the proper client clustering scheme according to the data distribution in the network, so as to reduce the impact of non-IID data on the model searching and training performance. Then, we sample the subnets from the pre-trained supernet and allocate them to the client clusters, considering the evaluation performance and the difference between the data distribution of the cluster and the population distribution.

3.1 Client Clustering

According to the test results in [16], the data distribution on each client cluster has a great influence on model searching and training. Therefore, we try to reduce the distance between the data distribution of local data on each cluster and the population distribution of the global dataset. To this end, we adopt earth mover distance (EMD) [26] to measure the difference of data distribution between two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$:

$$\mathcal{S}(\mathcal{D}_1, \mathcal{D}_2) = \sum_{l_i \in \mathcal{L}} \left\| \frac{N_1^i}{N_1} - \frac{N_2^i}{N_2} \right\| \tag{6}$$

where $l_i \in \mathcal{L}$ and $\mathcal{L} = \{l_1, l_2, ..., l_z\}$ is the label space of classification problem. N_j^i denotes the total size of data labeled as l_i in cluster $\mathcal{V}_j$. Then, we adopt $\mathcal{S}_j$ to denote the difference of data distribution between the global dataset and cluster $\mathcal{V}_j$'s dataset:

$$\begin{aligned}\mathcal{S}_j = \mathcal{S}(\mathcal{D}, \mathcal{D}_j) &= \sum_{l_i \in \mathcal{L}} \left\| \frac{N^i}{N} - \frac{N_j^i}{N_j} \right\| \\ &= \sum_{l_i \in \mathcal{L}} \| \psi_i - \phi_j^i \| \end{aligned} \tag{7}$$

where ψ_i and ϕ_j^i denote the proportion of the data labeled l_i in the total data and the cluster $\mathcal{V}_j$'s data, respectively.

Let y_i^j denote whether client v_i belongs to cluster $\mathcal{V}_j$ or not. In other words, $y_i^j = 1$ if q_j is the aggregator of client v_i; $y_i^j = 0$ otherwise. The server will select the aggregator according to the transmission delay for each cluster at a fixed interval. Note that the selected aggregator may not be available or down because of mobility, the server will intermittently communicate with the aggregator to ensure system reliability. The client clustering strategy $\mathcal{Y} = \{y_i^j\}_{v_i \in \mathcal{V}, q_j \in \mathcal{Q}}$ needs to be efficiently determined. During client clustering, we mainly consider two common constraints in FL, *i.e.*, the model convergence and completion time. In DCSA, an aggregator traverses the data distribution of all the unassigned clients and selects the clients whose data distributions can make the cluster's data distribution to be the closest to the population distribution. Accordingly, we define the cluster construction problem for DCSA.

$$\min \max_{q_j \in \mathcal{Q}} \mathcal{S}(\mathcal{D}, \mathcal{D}_j)$$
$$s.t. \begin{cases} \mathcal{F}(\omega) \leq \delta \\ H^j(\mathcal{Y}) \leq H^{max} & \forall q_j \in \mathcal{Q} \\ \sum_{q_j \in \mathcal{Q}} y_i^j = 1, & \forall v_i \in \mathcal{V} \\ y_i^j \in \{0, 1\} & \forall v_i \in \mathcal{V}, q_j \in \mathcal{Q} \end{cases} \tag{8}$$

The first inequality expresses the convergence requirement, where δ is the convergence threshold of the loss value of the learning task. The second set of inequalities represents that the completion time of each cluster should not exceed the given threshold H^{max}. The third set of equalities denotes that each client belongs to a unique cluster. Our goal is to minimize the maximum difference between the data distribution of the cluster and population distribution of the global dataset under clustering strategy $\mathcal{Y}$.

We introduce the client clustering algorithm for FINCH, which is formally described in Alg. 2. Some parameters, *e.g.*, data size, the threshold of completion time and convergence, are first initialized (Lines 1-2). At the beginning, we randomly select a client from $\mathcal{V}'$ for cluster initialization (Line 4) To avoid all clients being in only one cluster, we set the threshold for the maximum number of clients $\mathbf{Q}$ and the completion time H^{max} in each cluster (Line 5). A client is assigned to the cluster each time, considering the data distribution of each client and population distribution (Lines 7-12). The client clustering algorithm will terminate when all clients are assigned to the clusters (Line 13).

Algorithm 2 Client Clustering in DCSA

```
1:  Initialize data size $N_i$, $N_i^j$, $\forall \mathcal{C}_i \in \mathcal{C}$, $l_j \in \mathcal{L}$; the threshold of
    completion time $H^{max}$ and convergence threshold $\delta$
2:  Initialize $\mathcal{Q}' \leftarrow \emptyset; \mathcal{V}' \leftarrow \mathcal{V}$
3:  while $\mathcal{V}' \neq \emptyset$ do
4:      Select a client from $\mathcal{V}'$ for cluster initialization randomly,
        i.e., $q_i' \leftarrow \emptyset, \forall v_i \in \mathcal{V}'$
5:      while $H^{q_i'} \leq H^{max}$ and $|q_i'| \leq \mathbf{Q}$ do
6:          Find the client $\widetilde{v}_{j'}$ with the closest data distribution:
7:          $\widetilde{v}_{j'} \leftarrow \mathrm{argmin}_{v_j \in \mathcal{V}'} \mathcal{S}(\mathcal{D}, \mathcal{D}_{q_i'+v_j})$
8:          Assign the client to the cluster $q_i'$:
9:          $\mathcal{V}' \leftarrow \mathcal{V}' - \{\widetilde{v}_{j'}\}$
10:         $q_i' \leftarrow q_i' + \{\widetilde{v}_{j'}\}$
11:         $y_i^{j'} = 1$
12:     $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup q_i'$
13: return final clustering strategy $\mathcal{Y}$ and client clusters $\mathcal{Q}'$
```

3.2 Subnet Allocation

3.2.1 Search Space Design in NAS

After the client cluster has been constructed, our proposed algorithm samples the subnets from the supernet and allocates them to the specified clusters. In this work, we mainly concentrate our efforts on a computational vision problem as it is a common task for edge clients. Hence, we design our search space, *i.e.*, supernet, based on convolutional neural networks (CNNs) [27]. Following the DARTS approach [21], we define a directed acyclic graph (DAG) where all predecessor nodes are connected to every intermediate node with all possible operations. However, finding an optimal model architecture requires an enormous amount of resource (*e.g.*, computation and time) in such a large search space with hundreds and thousands of nodes in DAG [21]. Assume that there are 100 nodes in the supernet, and 4 optional operations for each node pair. There are totally about 1.6×10^{60} candidate subnets, leading to intolerable completion time (*e.g.*, 1,000 GPU/hours). In order to reduce the search space, some works adopt a cell with several nodes (*e.g.*, 4 or 5) as the granularity for model searching. Then, the optimal model architecture is formed by stacking the searched cell.

For better illustration, we give an example of a cell in Fig. 2. Inside a cell, several nodes with some potential operations are connected between the input node and output node. For convolutional cells, the input nodes are defined as the cell outputs in the previous two layers. The task of learning the cell therefore reduces to learning the operations on its edges [21]. Although searching a model with a single cell can significantly reduce the completion time, however, the architecture obtained by stacking this cell usually needs a lot of rounds of retraining to achieve better test accuracy, especially under the non-IID setting [28]. To alleviate this problem, we form the cell group as the supernet for model searching. Specifically, the subnets are sampled from the cell group and allocated to the specified clients. After that, the optimal subnet with the best test accuracy is stacked to form the final model architecture. Compared with the single cell scheme, the model architecture obtained by our proposed scheme requires less completion time, and significantly improves the test accuracy.

We further test the impact of the number of cells and groups on the performance of model searching and training. A large number of cells indicate more computation overhead while the edge client is always resource-constrained. In order to meet the resource capacity constraints, let the maximum number of cells be 50 for efficient searching and training [24]. Given a fixed number of iterations, we observe the test accuracy and completion time by changing the number of cells from 1 to 50. As shown in Fig. 3, the test accuracy increases with the number of cells, but with the rapid growth of completion time. When there are more than 20 cells, the test accuracy gradually tends to be stable, however, the completion time still increases sharply. Thus, we choose 20 cells as the size of the target model architecture. We then test the performance of model searching with different numbers of groups. In Fig. 4, we change the number of cells in each group as 1, 2, 4, 10 and 20. The test results show that more cells in a group will significantly improve the training performance (*e.g.*, accuracy), however, leading to higher computation overhead. In order to achieve a balance between the performance and resource consumption, we select 4 cells as a group for model searching. When the optimal subnet in the group is explored, it will be stacked several times to form the final model architecture.

3.2.2 Algorithm for Subnet Allocation

Note that the proper subnet allocation can significantly improve the model searching efficiency and training performance according to the test results in Section 2.3. Intuitively, one straightforward idea is to put more training budgets or data on models that are likely to achieve better performance. According to the testing results in [29], increasing the training budget can significantly improve the training performance of DNN. However, the worse performing models that are usually ignored also have an important influence on the model training [29]. Thus, we should push the performance limits of the worse performing models, which

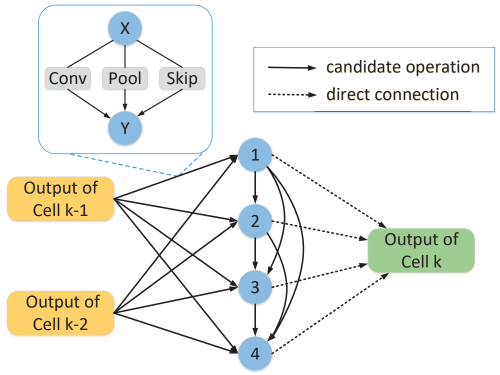


Fig. 2: An example of search space in NAS.

can bring a better optimized weight-sharing graph, such that all trainable components (*e.g.*, channels) reach their maximum potential in contributing to the final performance. Besides, it can lead to more informative gradients and better exploration in the architecture space, thus yielding better NAS performance [29].

We first give the definition of subnet allocation in hierarchical NAS (SA-HNAS) problem. In order to improve the speed of model searching and training, several subnets are sampled and allocated to the client clusters. $x_{i,j}^t$ denotes whether the subnet $(\omega_i, \mathcal{A}_i)$ is allocated to the cluster $\mathcal{C}_j$ or not at iteration t. If $x_{i,j}^t = 1$, the subnet $(\omega_i, \mathcal{A}_i)$ will be allocated to the client cluster $\mathcal{C}_j$ for further searching and training. Otherwise, $x_{i,j}^t = 0$. Without loss of generality, three main kinds of resources, *i.e.*, computation cost, network bandwidth and completion time, are taken into considerations in this work.

Searching and training the deep neural network is always computation-expensive for the resource-constrained clients, especially the larger models are usually selected for better model performance [28]. We believe that the number of floating-point operations (FLOPs) is reasonable to use as a proxy measure for actual computation cost [16]. Let *flops*$(\cdot)$ denote the number of floating points of the sampled architecture which is a commonly used metric to evaluate model complexity.

We use B to represent the bandwidth budget on the server, which is configured by the users at the beginning of model training. Let $b_r^t(\omega_j, A_j)$ denote the bandwidth consumption by aggregator r for transmitting the local model parameters ω_j and architecture $\mathcal{A}_j$ of the cluster $\mathcal{C}_j$ to the server. Thus, the bandwidth consumption of the cluster $\mathcal{C}_j$ at iteration t is

$$B_j^t = \sum_{j \in \mathcal{C}_j} b_{j,r}^t(\omega_j^t, \mathcal{A}_j^t) + b_r^t(\omega_j^t, \mathcal{A}_j^t), \tag{9}$$

where $\sum_{j \in \mathcal{C}_j} b_{j,r}^t(\omega_j^t, \mathcal{A}_j^t)$ denotes the bandwidth consumption of model transmition between the aggregator r and the clients in the cluster $\mathcal{C}_j$.

In practice, some tasks often need to be completed within a deadline. There are two main steps in the proposed scheme, including searching phase and training phase. We use $d_{tr}^t(\omega_j^t, \mathcal{A}_j^t)$ and $d_{val}^t(\omega_j^t, \mathcal{A}_j^t)$ to denote the completion time of model training and searching in the cluster $\mathcal{C}_j$ at iteration t, respectively. Therefore, the completion time of the cluster $\mathcal{C}_j$ at iteration t is:

$$h_j^t = \max\{d_{tr}^t(\omega_j^t, \mathcal{A}_j^t), d_{val}^t(\omega_j^t, \mathcal{A}_j^t)\}, \tag{10}$$

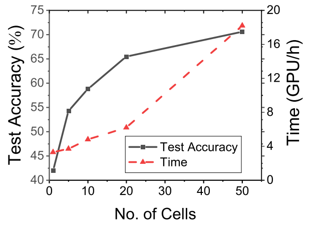


Fig. 3: Test accuracy and completion time with different numbers of cells.

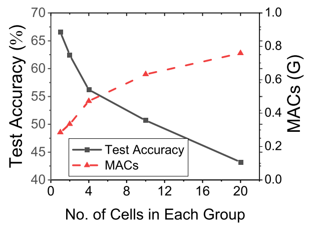


Fig. 4: Test accuracy and FLOPs with different numbers of cells in each group.

Furthermore, the completion time of all participating clients at iteration t can be denoted as $H^t = \max_{j \in M} h_j^t$. Accordingly, we formulate the SA-HNAS problem as follows:

$$\min_{T \in \{1,2,\dots\}} H^t$$

$$s.t. \begin{cases} \mathcal{F}(\omega_T, \mathcal{A}^T) \le \delta, \\ \textit{flops}(\mathcal{A}_i^t) \le \textit{FLOPS}_{max}, & \forall t \\ \sum_{t=1}^T \sum_{j=1}^M B_j^t \le B \\ \sum_{j \in M} x_{i,j}^t = 1, & \forall i, t \\ x_{i,j}^t \in \{0,1\} & \forall i, j, t \end{cases} \tag{11}$$

The first inequality ensures the convergence of model training after T training iterations. The second and third inequalities ensure the computation and communication constraints during the model searching and training, respectively. The last set of constraints tells that each sampled architecture of model will only be assigned to one cluster. The objective of the SA-HNAS problem is to minimize the completion time of FL process.

In order to efficiently solve this problem, we propose a greedy-based algorithm to allocate optimal subnets to the clustered clients, considering the evaluation performance, resource consumption and data distribution. Specifically, the subnet with more resource consumption and worse performance will be allocated to the cluster which has a larger non-IID level of local data, so as to significantly improve the performance of model searching and training. For resource consumption, we mainly consider the communication (bandwidth consumption) and computation (FLOPs) respectively. Accordingly, we define the resource consumption as follows:

$$\mathbf{R}_j^t = \epsilon \cdot \frac{B_j^t}{B} + (1-\epsilon) \cdot \frac{\textit{flops}(\mathcal{A}_j^t)}{\textit{FLOPS}_{max}} \tag{12}$$

where weight $\epsilon \in (0,1)$ is determined by the specific requirements to balance objectives. For instance, when computation consumption dominates the resource cost, a small weight ϵ will be set for computation consumption in the function. Besides, we adopt the difference between the data distribution of cluster $\mathcal{D}_j$ and the population distribution of global dataset $\mathcal{D}$, *i.e.*, $\mathcal{S}(\mathcal{D}, \mathcal{D}_j)$, to represent the level of data imbalance, *i.e.*, non-IID. In other words, the greater the difference of the data distributions, the larger the level of non-IID data on the cluster. During the subnet allocation, each client cluster $\mathcal{C}_j$ is also assigned a weight $\varphi_j \in (0,1]$, along with $\sum_{\mathcal{C}_j \in \mathcal{C}} \varphi_j = 1$. The server will perform weighted aggregation of each subnet $\mathcal{A}_j$ from the client cluster $\mathcal{C}_j$ with the weight φ_j.

Algorithm 3 Subnet Allocation in DCSA

```
1:  Initialize allocation strategy $\mathcal{X} = \{x_{i,j}^0\}$ and resource cost $\mathcal{R} = \{\mathcal{R}_j^0\}, \forall v_i \in \mathcal{V}, q_j \in \mathcal{Q}; \mathcal{V}' \leftarrow \mathcal{V}$; the set of subnets $A$
2:  for each global update $t = T/\lambda_1\lambda_2$ do
3:      Obtain the cost $\mathcal{R}_j^{t-1}$ by Eq. (12) and the evaluation accuracy $c_j^{t-1}$ of subnet $\mathcal{A}_i^{t-1}$ on the cluster $j$ if $x_{i,j}^{t-1} = 1$
4:      Compute the non-IID level for each cluster $q \in \mathcal{Q}$
5:      Range the clusters $q \in \mathcal{Q}$ in increasing order with the non-IID level to form the updated set $\mathcal{Q}'$
6:      Range the subnets $\mathcal{A} \in A$ in the decreasing order with $c_j^{t-1}/\mathcal{R}_j^{t-1}$ to form the updated set $A'$
7:      for $k = 1$ to $|\mathcal{Q}'|$ do
8:          $i \leftarrow A'[k]$
9:          $j \leftarrow \mathcal{Q}'[k]$
10:         $x_{i,j}^t = 1$
11:         $\mathcal{X} \leftarrow \mathcal{X} \cup x_{i,j}^t$
12: return final subnet allocation strategy $\mathcal{X}$
```

We introduce the detailed algorithm for subnet allocation in Alg. 3. At the beginning, the allocation strategy $\mathcal{X} = \{x_{i,j}^0\}$ and the resource cost $\mathcal{R} = \{\mathcal{R}_j^0\}$ are initialized (Line 1), where $x_{i,j}^t$ and $\mathcal{R}_j^0$ denote whether the subnet i is assigned to the cluster j or not and the resource cost on the cluster j at iteration t, respectively. The network information (*e.g.*, resource consumption and evaluation accuracy) is collected by the server (Line 3). Specifically, c_j^{t-1} denotes the evaluation accuracy on the cluster j at iteration $t-1$. Besides, we compute the level of non-IID data in each client cluster (Line 4). Then, we range the client cluster and sampled subnets according to the network information and data distribution, respectively (Line 5-6). Finally, each sampled subnet is allocated to the proper client cluster for model searching and training (Line 7-12). Thus, the model searching and training will be efficiently performed through the client clustering and subnet allocation algorithms in DCSA.

4 Convergence Analysis

To show the feasibility of the proposed framework, we prove that FINCH can achieve a constant convergence bound. In FINCH, the model searching and training of the sampled subnet in each cluster are performed simultaneously. We first introduce some lemmas which have been proved in [30].

Lemma 1. *Let $L > 0$. The function $\mathcal{F}_i$ of each subnet on client v_i searched by the NAS algorithms satisfies the block-wise Lipschitz smoothness for any model weight x and y:*

$$\left\|\frac{\partial \mathcal{F}_i}{\partial x} - \frac{\partial \mathcal{F}_i}{\partial y}\right\| \le L \left\|x - y\right\| \tag{13}$$

Lemma 2. *Let $\sigma > 0$. The gradient variance of searched subnet by NAS is bounded by:*

$$\mathbb{E}\left\|\frac{\partial \mathcal{F}_i}{\partial x} - \mathbb{E}\frac{\partial \mathcal{F}_i}{\partial x}\right\|^2 \le \sigma^2 \tag{14}$$

The varied subnets can perform model searching and training independently on multiple client clusters in parallel [31], [32]. Combining the Lemmas 1-2, we then make the following assumptions on the functions $\mathcal{F}_i, \forall v_i \in \mathcal{V}$ for analysis [33].

Assumption 1. *(Smoothness) $\mathcal{F}$ is L-smooth with $L > 0$, i.e., it always holds $\mathcal{F}(y) - \mathcal{F}(x) \le \langle \nabla\mathcal{F}(x), y - x\rangle + \frac{L}{2}\left\|y - x\right\|^2$ for any two model parameters x and y.*

Assumption 2. *(Strong Convexity) $\mathcal{F}_i$ is μ-strongly convex with $\mu > 0$, i.e., it always holds $\mathcal{F}_i(y) - \mathcal{F}_i(x) \ge \langle \nabla\mathcal{F}_i(x), y - x\rangle + \frac{\mu}{2}\left\|y - x\right\|^2$ for any two model parameters x and y.*

Assumption 3. *(Bounded Gradient Variance) The variance of gradients at each client cluster $\mathcal{C}_j \in \mathcal{V}$ is bounded: $\mathbb{E}\left\|\nabla\mathcal{F}(\omega) - \nabla\mathcal{F}_j(\omega)\right\|^2 \le \sigma^2$, where σ is a positive number.*

Assumption 4. *(Existence of Global Optimization) Assume that there exists at least one solution, denoted as ω^*, to achieve the global minimum of the loss function $\mathcal{F}(\omega)$.*

We first introduce some parameters (*e.g.*, λ_1 or λ_2) for convergence analysis. Our FINCH framework adopts hierarchical NAS for model searching and training. In FINCH, intra-cluster aggregation may occur over LANs, while inter-aggregation usually occurs over WANs with more bandwidth consumption [34]. In order to save the resource cost, the frequency of intra-cluster aggregation should be increased (*e.g.*, $\lambda_1 = 1$), while that of inter-cluster aggregation needs to be decreased for efficient model searching and training. We first prove that the aggregated cluster model is equal to training on the cluster's virtual aggregated dataset with the searched architecture $\mathcal{A}$ at each iteration t (Theorem 3). Then, we prove the convergence of model training and obtain the convergence bound after T iterations (Theorem 4).

Theorem 3. *If the number of local updating iterations is 1, i.e., $\lambda_1 = 1$, the aggregated cluster model $\tilde{\omega}_t^j$ satisfies:*

$$\tilde{\omega}_t^j = \omega_{t-1}^j - \eta\nabla\mathcal{F}_j(\omega_{t-1}^j) \tag{15}$$

where $\mathcal{F}_j$ is the loss function on dataset $\mathcal{D}_j$.

Proof. By Eqs. (4) and (5), we have

$$\begin{aligned}\tilde{\omega}_t^j &= \sum_{v_i \in \mathcal{C}_j} \rho_j^i(\omega_{t-1}^i - \eta\nabla\mathcal{F}_i(\omega_{t-1}^i)) \\ &= \sum_{v_i \in \mathcal{C}_j} \rho_j^i\omega_{t-1}^i - \eta \sum_{v_i \in \mathcal{C}_j} \rho_j^i\nabla\mathcal{F}_i(\omega_{t-1}^i) \\ &= \omega_{t-1}^j - \eta \sum_{v_i \in \mathcal{C}_j} \rho_j^i\nabla\mathcal{F}_i(\omega_{t-1}^i)\end{aligned} \tag{16}$$

where $\rho_j^i = \frac{N_i}{N^j}$ and $\nabla\mathcal{F}_i(\omega_{t-1}^i) := \nabla\mathcal{F}_{\mathrm{tr}}(\omega_{t-1}^i, \mathcal{A})$. Due to the linearity of the gradient operator, we can derive that

$$\begin{aligned}\sum_{v_i \in \mathcal{C}_j} \rho_j^i\nabla\mathcal{F}_i(\omega_{t-1}^i) &= \nabla \sum_{v_i \in \mathcal{C}_j} \rho_j^i\mathcal{F}_i(\omega_{t-1}^i) \\ &= \nabla\mathcal{F}_j(\omega_{t-1}^i)\end{aligned} \tag{17}$$

Taking Eq. (17) into Eq. (16), we have

$$\tilde{\omega}_t^j = \omega_{t-1}^j - \eta\nabla\mathcal{F}_j(\omega_{t-1}^i) \tag{18}$$

□

FINCH searches for the optimal model architecture and trains the model parameters at the same time. Assume that FINCH performs a total of T training iterations, and the number of inter-cluster aggregations is $G = T/\lambda_2$. Combining Theorem 3, we prove the convergence of our FINCH framework.

Theorem 4. *Let ω_0 be the initial global model. Thus, the trained global model satisfies*

$$\mathcal{F}(\omega^T) - \mathcal{F}(\omega^*) \leq \tau(\mathcal{F}(\omega^0) - \mathcal{F}(\omega^*)) + \frac{(1-\tau)\delta}{2L\mu} \tag{19}$$

where $\tau = (1 - L\mu\eta)^T$.

Proof. For $q_j \in \mathcal{Q}$, $t = g\lambda_2$, it holds that

$$\tilde{\omega}_t^j = \tilde{\omega}_{g\lambda_2}^j = \omega_{g\lambda_2-1}^j - \eta\nabla\mathcal{F}_j(\omega_{g\lambda_2-1}^j) \tag{20}$$

where $g \in \{0, 1, ..., G\}$ is the index of inter-cluster agregations.

According to Assumption 1, it is obvious that $\mathcal{F}$ is L-smooth. If $0 < L \leq 1$, it follows

$$\begin{aligned}
&\mathcal{F}(\tilde{\omega}_{g\lambda_2}^j) - \mathcal{F}(\omega^*) \\
\leq&\mathcal{F}(\omega_{g\lambda_2-1}^j) - \mathcal{F}(\omega^*) + \langle\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j), \tilde{\omega}_{g\lambda_2}^j - \omega_{g\lambda_2-1}^j\rangle \\
&\quad + \frac{L}{2}\|\tilde{\omega}_{g\lambda_2}^j - \omega_{g\lambda_2-1}^j\|^2 \\
=&\mathcal{F}(\omega_{g\lambda_2-1}^j) - \mathcal{F}(\omega^*) - \eta\langle\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j), \nabla\mathcal{F}_j(\omega_{g\lambda_2-1}^j)\rangle \\
&\quad + \frac{L\eta^2}{2}\|\nabla\mathcal{F}_j(\omega_{g\lambda_2-1}^j)\|^2 \\
\leq&\mathcal{F}(\omega_{g\lambda_2-1}^j) - \mathcal{F}(\omega^*) + \frac{L\eta}{2}\|\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j) - \nabla\mathcal{F}_j(\omega_{g\lambda_2-1}^j)\|^2 \\
&\quad - \frac{L\eta}{2}\|\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j)\|^2
\end{aligned} \tag{21}$$

Substituting ω in Assumption 3 by $\omega_{g\lambda_2-1}^j$, we have

$$\|\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j) - \nabla\mathcal{F}_j(\omega_{g\lambda_2-1}^j)\|^2 \leq \sigma^2 \tag{22}$$

Combining Assumptions 2 and 4, it follows

$$\begin{aligned}
\|\nabla\mathcal{F}(\omega_{g\lambda-1}^j)\|^2 &= \|\nabla\mathcal{F}(\omega_{g\lambda_2-1}^j) - \nabla\mathcal{F}(\omega^*)\|^2 \\
&\geq 2\mu[\mathcal{F}(\omega_{g\lambda_2-1}^j) - \mathcal{F}(\omega^*)]
\end{aligned} \tag{23}$$

By taking Eqs. (22) and (23) into Eq. (21), we obtain

$$\begin{aligned}
&\mathcal{F}(\tilde{\omega}_{g\lambda_2}^j) - \mathcal{F}(\omega^*) \\
&\leq (1 - L\mu\eta)\mathcal{F}(\omega_{g\lambda_2-1}^j) - \mathcal{F}(\omega^*) + \frac{L\eta\sigma^2}{2} \\
&\leq (1 - L\mu\eta)^{\lambda_2}[\mathcal{F}(\omega_{(g-1)\lambda_2}^j) - \mathcal{F}(\omega^*)] \\
&\qquad + \frac{[1 - (1 - L\mu\eta)^{\lambda_2}]\sigma^2}{2L\mu} \\
&\leq (1 - L\mu\eta)^{\lambda_2}[\mathcal{F}(\omega_{(g-1)\lambda_2}) - \mathcal{F}(\omega^*)] \\
&\qquad + \frac{[1 - (1 - L\mu\eta)^{\lambda_2}]\sigma^2}{2L\mu}
\end{aligned} \tag{24}$$

where $\omega_{(g-1)\lambda_2}^j = \omega_{(g-1)\lambda_2}$ because the global model updating is performed with the cluster models when the index of the current iteration is an integer multiple of λ_2. Let $\Phi_j = N_j/N \in (0, 1]$. We can obtain the difference between $\mathcal{F}(\omega_T)$ and $\mathcal{F}(\omega^*)$ by the global updating of G times in Eq. (3) as follows:

$$\begin{aligned}
&\mathcal{F}(\omega_T) - \mathcal{F}(\omega^*) = \mathcal{F}(\omega_{G\lambda_2}) - \mathcal{F}(\omega^*) \\
&\leq \sum_{q_j\in\mathcal{Q}} \phi_j(\mathcal{F}(\tilde{\omega}_{G\lambda_2}^j) - \mathcal{F}(\omega^*)) \\
&\leq (1 - L\mu\eta)^{\lambda_2}(\mathcal{F}(\omega_{(G-1)\lambda_2}) - \mathcal{F}(\omega^*)) \\
&\qquad + \frac{[1 - (1 - L\mu\eta)^{\lambda_2}]\sigma^2}{2L\mu} \\
&\leq (1 - L\mu\eta)^{G\lambda_2}(\mathcal{F}(\omega_0) - \mathcal{F}(\omega^*)) \\
&\qquad + \frac{1 - (1 - L\mu\eta)^{G\lambda_2}}{1 - (1 - L\mu\eta)^{\lambda_2}} \frac{[1 - (1 - L\mu\eta)^{\lambda_2}]\sigma^2}{2L\mu} \\
&\leq \tau(\mathcal{F}(\omega_0) - \mathcal{F}(\omega^*)) + \frac{(1-\tau)\sigma^2}{2L\mu}
\end{aligned} \tag{25}$$

where $\tau = (1 - L\mu\eta)^T$. □

Thus, we complete the proof of convergence analysis and conclude that our FINCH framework can achieve a constant convergence bound.

5 Performance Evaluation

5.1 Performance Metrics and Benchmarks

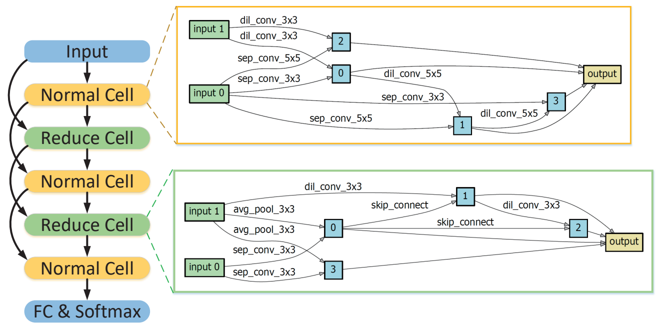
In this paper, we evaluate the performance of different schemes by adopting five metrics as follows: (1) *Training loss* is used to validate whether an FL algorithm can efficiently guarantee the convergence or not. (2) In each round, we will evaluate the global model on test dataset and record the *test accuracy*. (3) Network traffic, *i.e.*, *bandwidth consumption*, is the total size of the model transmitted through the network for model distribution and aggregation among all clients, which quantifies the communication cost in training. (4) *Completion time* is the time duration until training terminates, which evaluates the training speed. (5) *Multiply-accumulate operations* (MACs) is always adopted for specifying computation cost, *i.e.*, FLOPs, on the edge client.

We compare FINCH with five classical FL schemes, *i.e.*, HFL [23], FedNas [5], DecNas [6], OneNas [17] and FedorAS [18]. In order to evaluate the efficiency of NAS, we compare FINCH with HFL [23], which adopts the pre-defined CNN model architecture in hierarchical federated learning paradigm to handle the non-IID problem. Furthermore, we compare FINCH with the following four solutions to test the performance improvement of hierarchical NAS. Specifically, FedNas [5] first searches for the cell architecture as the proxy task and then transfers the learnt structure to the target dataset. DecNas [6] generates a group of candidate neural architectures which are trained locally on clients, and the best one will become the backbone model in the next iteration. In OneNas [17], one-shot NAS is introduced as a basis to deal with limited resource issue, where a large network is adopted as the global model, including all candidate network architectures. The main idea of FedorAS [18] is to sample a subspace for each client considering its resource budgets, focusing on the device heterogeneity and communication efficiency. Finally, we adopt the scheme Random which randomly select the subnet for each client cluster to show the effectiveness of our proposed algorithm SA-HNAS in FINCH.

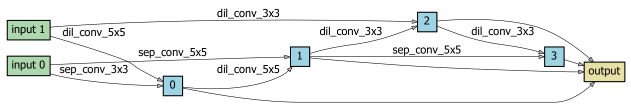
5.2 Simulation Evaluation

5.2.1 Evaluation Settings

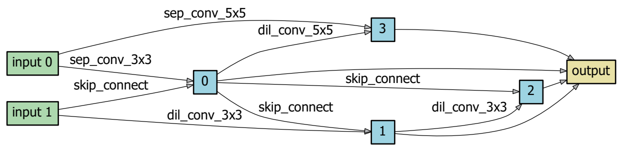
We build an FL experimental environment on an AMAX deep learning workstation (CPU: Intel 671 (R) E5-2620v4, GPU: NVIDIA TITAN RTX), upon which the baselines are implemented with PyTorch [35]. Our experiments are performed on Ubuntu 18.04, CUDA v10.0, cuDNN v7.5.0. On each client, the batch



(a) The network architecture searched for the CIFAR10 dataset.



(b) The normal cell searched for the ImageNet100 dataset.



(c) The reduction cell searched for the ImageNet100 dataset.

Fig. 5: The model architecture serached by FINCH.

size is 50 and the iteration number for local model training is 5. Let the initial learning rate be 0.05. We use the cossin rule as a decay strategy for the learning rate [36]. During the training of the supernet, we adopt Adam [37] to optimize α and momentum SGD to optimize ω with a weight decay of 4×10^{-5}. As suggested in [38], in order to efficiently simulate the training processing in FL of our proposed solution and baselines, a total of 100 edge nodes are generated in the simulation, and 10 of them are randomly activated to participate in the model searching and training. We train the supernet on 10 activated clients for 100 iterations.

Datasets: To evaluate the performance of our proposed method, we adopt two popular benchmark datasets, *i.e.*, CIFAR10 [39] and ImageNet ILSVRC-2012 [40], for image classification. CIFAR-10 contains a training set with 50,000 samples and a test set with 10,000 samples. Each sample in CIFAR-10 is a $32\times32\times3$ RGB image from 10 categories. ImageNet is a widely used dataset for visual recognition which consists of 1,281,167 training images, 50,000 validation images and 100,000 test images from 1,000 categories, and each sample in ImageNet is a $224\times224\times3$ image. Considering the constrained resource on edge devices, we create ImageNet100, a subset of ImageNet that consists of 100 out of 1,000 categories, and each image is downsized with the shape of $144\times144\times3$.

Data Partition: In the experiments, we mainly consider the following five different cases, including IID data and four different levels of non-IID data, to verify the effect of data distributions on the model searching and training performance. For IID federated simulations, all image data are evenly and randomly distributed to each local client without overlaps. For example, 5,000 images per client for CIFAR10 or 17,000 images per client for ImageNet100. To simulate various degrees of non-IID among the clients, we adopt two types of non-IID data distributions, *i.e.*, latent Dirichlet allocation (LDA) and label skewed, which are well explored in previous literature [38]. (1) LDA for CIFAR10: A ζ proportion of the training samples on each client belong to a unique class and the remaining $1-\zeta$ of the data belong to other classes (ζ= 0.1, 0.2, 0.4, 0.6 and 0.8). (2) Label skewed for ImageNet100: Each client lacks a fraction (ζ) of 100 classes of data samples (ζ= 0, 0.1, 0.2, 0.3 and 0.4). Particularly, $\zeta = 0$ represents uniform data distribution. By default, the data distributions of CIFAR10 and ImageNet100 are both non-IID-0.4, *i.e.*, $\zeta = 0.4$.

5.2.2 Simulation Results

We perform five groups of simulations to verify the efficiency of our proposed framework.

1) **Performance Improvement of NAS and SA-HNAS:** We first conduct a set of experiments to test the performance improvement of NAS during federated training. The pre-defined CNN model in HFL easily makes the training fall into the local optimal solution, leading to low test accuracy, especially under the non-IID setting. On the contrary, our proposed framework will search for the proper model architecture and improve the training performance. As illustrated in Fig. 5(a), the normal cell (*e.g.*, conv 3×3 and conv 5×5) and the reduction cell (*e.g.*, avg pooling 3×3) connect alternately to form the final network architecture for the CIFAR10 dataset. Besides, the normal cell and reduction cell searched for the ImageNet100 dateset are shown in Figs. 5(b)-5(c), respectively. Table 4 shows that FINCH can search for a smaller model (*e.g.*, number of parameters and size) with a little accuracy degradation ($<$ 1%) compared with HFL and other classical models, *e.g.*, AlexNet, VGG16. Thus, the network architecture searched by FINCH is more lightweight compared with the pre-defined models, which allows the model to be deployed on the resource-constrained edge clients.

TABLE 4: The model size and test accuracies of pre-defined netowrk architectures and ours.

Model	Parameter (M)	Model Size (MB)	Accuracy
HFL	20.6	78.5	77.8%
AlexNet	3.8	14.6	73.6%
VGG16	15.3	58.2	77.4%
ResNet18	11.2	42.6	73.2%
FINCH	**3.12**	**11.9**	**76.9%**

From Fig. 6, given a fixed number of iterations (*e.g.*, 200), the test accuracy of HFL and FINCH is about 62.6% and 68.4%, respectively. In other words, our proposed framework can improve the test accuracy by about 5.8% compared with the method which adopts the pre-defined model. Besides, we test the resource consumption of two schemes with different target accuracies. Our proposed method with NAS will search for a smaller network

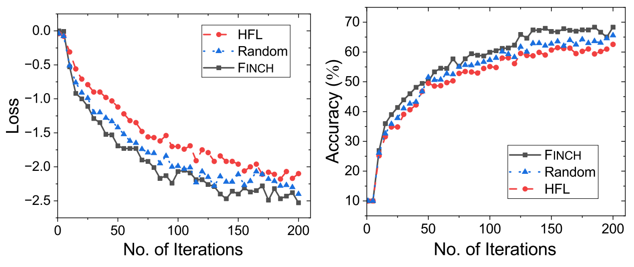


Fig. 6: Loss and test accuracy of three schemes under non-IID setting.

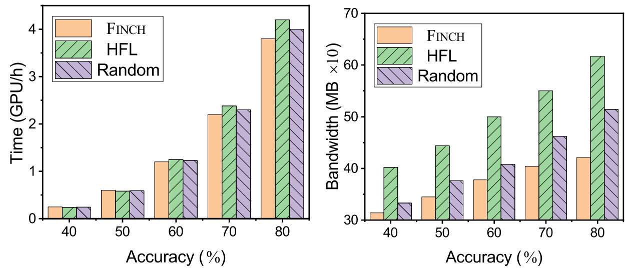


Fig. 7: Resource cost of two schemes with varied target accuracies.

architecture than the pre-defined model, reducing the computation and communication cost. Fig. 7 reveals that the cost of computation and communication gradually increase with the increasing accuracy in HFL and FINCH. However, the increasing ratio of FINCH is slower than that of HFL. For example, given the target accuracy of 70%, the completion time and bandwidth cost of FINCH are about 2.2 GPU/h and 404 MB, while those of HFL are about 2.38 GPU/h and 552 MB, respectively. FINCH can reduce the completion time and communication cost by about 7.6% and 26.8% compared with HFL. Thus, our proposed framework with NAS will significantly improve the performance of model searching and training for FL.

In order to test the effectiveness of the algorithm SA-HNAS in FINCH, we compare it with the scheme of Random, which randomly assign the subnet to each client cluster for model search and training. The randomly selected subnets may not be suitable for the client clusters, leading to a longer search time and poor training performance under resource constraints. The test results in Fig. 6 show that FINCH can achieve better performance, including training loss and test accuracy, compaerd with Random. For example, given 200 training iterations, the test accuracy of FINCH is about 68.4%, which that of Random is about 65.2%. In other words, FINCH can improve the accuracy performance by about 3.2% compared with Random. In addition, the required computing time and communication cost of FINCH is less than that of Random. As shown in Fig. 7, FINCH can reduce the bandwidth cost by about 22% compared with Random when achieving the target accuracy of 80%. Thus, the experimental results reveal that our proposed algorithm SA-HNAS in FINCH outperforms Random in terms of training performance and resource cost.

2) **Convergence Performance:** In the second set of experiments, we observe the performance of model convergence under IID and non-IID data settings. We adopt CIFAR10 and ImageNet100 datasets for model searching and training. Our proposed FINCH framework will perform more iterations compared with these baselines under the same resource budgets (*e.g.*, network bandwidth). As shown in Table 5, FINCH improves the performance of model training, *i.e.*, test accuracy, under both IID and non-IID settings. For CIFAR10, the test accuracy of FINCH is about 80.7%, while that of FedNas, DecNas, OneNas and FedorAS is 78.5%, 77.2%, 76.4% and 78.7% under IID setting, respectively. Besides, we observe that the test accuracy of all the methods show diverse degrees of decline under the non-IID setting. However, with efficient client clustering and subnet allocation, FINCH outperforms the other four schemes on all the datasets. Concretely, FINCH, FedNas, DecNas, OneNas and FedorAS separately achieve the accuracy of 55.3%, 50.6%, 46.2%, 39.7% and 51.4% on the ImageNet100 dataset. The experimental results significantly demonstrate the effectiveness of our proposed FINCH framework.

TABLE 5: Test accuracy (%) of four schemes under different data distributions.

	CIFAR10		ImageNet100	
	IID	non-IID	IID	non-IID
FedNas	78.5	71.3	60.2	50.6
DecNas	77.2	67.6	57.4	46.2
OneNas	76.4	65.2	54.9	39.7
FedorAS	78.7	73.2	61.7	52.4
FINCH	80.7	76.8	64.7	55.3

TABLE 6: Resource consumption, *i.e.*, Time (GPU/h), MACs (G), Bandwidth (G) and Parameters (M), of four different schemes under non-IID setting.

	Time	MACs	Bandwidth	Parameters
FedNas	1.55	0.79	4.68	5.19
DecNas	1.24	0.33	3.47	8.48
OneNas	1.62	0.46	3.88	11.75
FedorAS	1.32	0.41	3.29	7.96
FINCH	1.05	0.12	1.64	3.42

3) **Resource Consumption:** We observe the resource cost (*e.g.*, network bandwidth, FLOPs and completion time) of four schemes given a target accuracy (*e.g.*, 80%) under the non-IID setting ($\zeta = 0.2$). The CIFAR10 dataset is adopted for model searching and training. In FINCH, the parameter server will perform global model updating after receiving the local updated models from all aggregators, instead of the edge clients. Thus, FINCH requires less network bandwidth than other schemes during model transmission. In Table 6, the bandwidth consumption of FINCH is about 1.64 GB, while that of FedNas, DecNas, OneNas and FedorAS is about 4.68 GB, 3.47 GB, 3.88 GB and 3.29 GB. In other words, our proposed framework reduces the communication cost by about 64%, 52.8%, 57.7% and 50.2% compared with the baselines, respectively. Besides, we need less completion time to search and train the subnet in FINCH, even on the resource-constrained edge clients. Table 6 reveals that FINCH

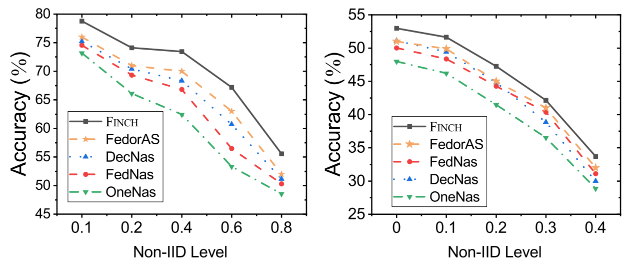


Fig. 8: Test accuracy over the CIFAR10 and ImageNet100 datasets with different non-IID levels.

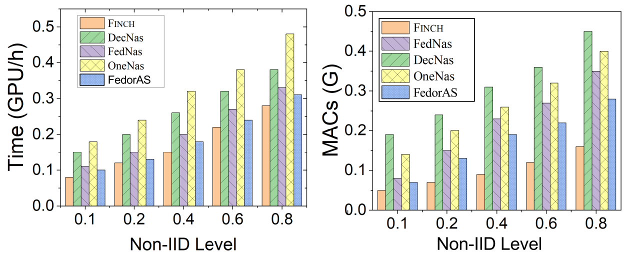


Fig. 9: Resource consumption over the CIFAR10 dataset with different non-IID levels.

can significantly reduce the completion time and computation cost (FLOPs) compared with the baselines. For example, given the target test accuracy of 80%, the completion time of FINCH is about 1.05 GPU/h, while that of FedNas, DecNas, OneNas and FedorAS is about 1.55 GPU/h, 1.24 GPU/h, 1.62 GPU/h and 1.32 GPU/h, respectively. Thus, our proposed framework can significantly accelerate the training process of FL.

4) **Impact of non-IID Levels:** The fourth set of experiments tests the training performance (*e.g.*, test accuracy and resource cost) of the CIFAR10 and ImageNet100 datasets under different non-IID levels. The results in Fig. 8 show that the test accuracy will be degraded with the increasing non-IID level. The adaptive client clustering according to the data distribution significantly improves the weight divergence caused by the data drift. Therefore, FINCH can achieve better training performance than the other schemes, especially under a large non-IID level. For instance, the test accuracy of FINCH is about 73.4% if the non-IID level is 0.4, while that of FedNas, DecNas OneNas and FedorAS is about 66.8%, 68.3%, 62.4% and 69.5%, with 200 iterations. In other words, FINCH improves the test accuracy by about 6.6%, 5.1%, 11% and 3.9% compared with the four baselines, respectively.

In addition, we record the completion time and MACs of federated training over CIFAR10 with different non-IID levels. The results in Fig. 9 show that the completion time and MACs of model searching and training all increase with the increasing non-IID levels. However, the increasing ratio of our proposed FINCH framework is much slower than that of the other schemes. FINCH requires less completion time and MACs than FedNas, DecNas, OneNas and FedorAS. For example, given the level of non-IID of 0.6, the completion time of FINCH is about 2.2 GPU/h with 200 iterations, while that of FedNas, DecNas, OneNas and FedorAS is about 3.2 GPU/h, 2.7 GPU/h, 3.8 GPU/h and 2.4 GPU/h, respectively. Therefore, FINCH reduces the completion time of model training by about 31.2%, 18.5%, 42.1% and 8.3%, respectively. Moreover, FINCH also saves massive computation cost (MACs), *e.g.*, 66.3%, 55.4%, 62.5% and 45.4%, compared with the four baselines.

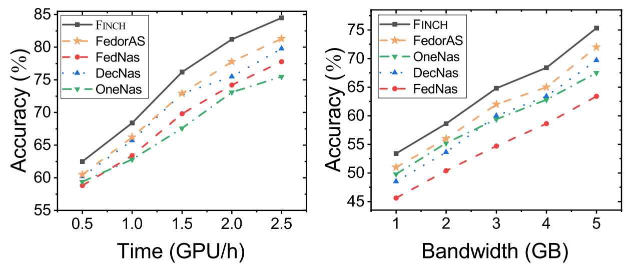


Fig. 10: Test accuracy over the CIFAR10 dataset with completion time and bandwidth constraint.

5) **Effect of Resource Constraints:** The last set of experiments tests the training performance of CIFAR10 with varied resource constraints (*e.g.*, bandwidth and time). Parallel search with several subnets on the client clusters in FINCH will significantly accelerate the process with fewer iterations compared with FedNas, DecNas, OneNas and FedorAS, which will reduce the bandwidth consumption and completion time of federated training. We record the test accuracy of all schemes with different bandwidth budgets in Fig. 10. The results show that the test accuracy of FINCH is always better than that of the other four baselines. For example, given the bandwidth budget of 5GB, the test accuracy of FINCH is about 75.3%, while that of FedNas, DecNas, OneNas and FedorAS is about 63.4%, 69.7%, 67.5% and 72.3%, respectively.

In Fig. 10, the test accuracy of all solutions will be significantly improved with more completion time budget. However, given the same completion time constraint, FINCH can achieve better performance, *i.e.*, higher test accuracy, than the other four schemes. For example, the accuracy of FINCH is about 81.2%, while that of FedNas, DecNas, OneNas and FedorAS is about 74.2%, 75.5%, 73.1% and 77.8% when the time constraint is 2 GPU/h. Thus, FINCH can improve the test accuracy by about 7%, 5.7%, 8.1% and 3.4% compared with FedNas, DecNas, OneNas and FedorAS, respectively.

6 RELATED WORKS

6.1 Federated Learning

In recent years, federated learning (FL) has been widely adopted in both academia and industry fields [22], [41]. The FL paradigm was first proposed in [3], which coordinates multiple edge clients to learn a globally shared model based on their local datasets. Concretely, McMahan *et al.* develop the *FedAvg* algorithm by combining the local stochastic gradient descent (SGD) on each client with a server that performs synchronous model averaging [3]. The *FedAvg* can decouple the model training from the need for direct access to raw data, and the data privacy can be preserved. However, due to the constrained resources in edge computing, FL imposes massive computation and communication overhead, which limits its efficiency in practical deployment. Tran *et al.*

[42] jointly optimize the CPU frequency, transmit power and model accuracy to minimize the weighted sum of energy cost and learning time. Yang *et al.* [43] propose an iterative algorithm to address the problem of energy-efficient transmission and computation resource allocation. Data heterogeneity (*i.e.*, non-IID data) is another challenge to achieving efficient training performance. Li [44] analyzes the convergence of *FedAvg* on non-IID data and establishes a convergence rate for convex and smooth problems. Zhao [45] allows the public available data to be distributed to clients so that the clients' data becomes IID.

6.2 Neural Architecture Search

Originally, NAS is mostly designed to find the single most accurate architecture within a large search space, without regard for the model performance (*e.g.*, size and computations). The works of NAS can be divided into three categories according to the search strategy: reinforcement learning (RL), evolutionary algorithm (EA), and gradient-based (GD) [46]. Early attempts employ evolutionary algorithms (EA) for optimizing neural architectures and parameters. The best architecture may be obtained by iteratively mutating a population of candidate architectures [17]. An alternative of EA is to use reinforcement learning (RL) techniques, *e.g.*, policy gradients and Q-learning, to train a recurrent neural network. Zoph *et al.* [47] first adopt the RL to train a recurrent neural network (RNN) model that generates architectures, which is a pioneering work in the field of NAS [48]. However, EA and RL based methods often require a large amount of computations, which are inefficient in search. We draw inspiration from the differentiable formulation of NAS. Among these, gradient-based methods are the most efficient as they can finish searching in only a few hours, compared to thousands of GPU days with other methods.

6.3 FL with NAS

The goal of federated NAS is to generate a sequence of simplified models from an expensive one with the best accuracy under the resource budgets in the network. FedorAS [18] samples a one-shot path for every client cluster based on the resource budgets for model search and training. Nevertheless, the process of fine-tuning or personalizing will introduce the extra computation cost. Zhu *et al.* [49] propose an offline federated NAS framework using a multi-objective evolutionary algorithm. All participating clients must train each of the architecture of the neural network for fitness evaluations, which significantly increases both computation and communication costs. Client sampling can be used to alleviate this issue, in which only subsets of participating clients contribute to the model training. For example, all the connected clients are divided into different groups and each sampled model uses one group of clients for local training [50]. In addition to client sampling, the authors in [50] also remove subsets of global models to further reduce the communication costs. However, the test accuracy of each global model in the list is calculated before model aggregation, which sometimes cannot represent the real test accuracies, especially for the cases when the clients' data are particularly non-IID. To alleviate this problem, He *et al.* [5] propose FedNas to search for a better DNN architecture based on the whole supernet, improving the training performance under non-IID settings. Zhang *et al.* [16] advocate one-shot NAS as a basis to deal with the non-IID problem. FINCH proposes a novel hierarchical NAS, which can achieve better training performance (*e.g.*, accuracy) and convergence speed (or completion time) within the resource budgets, especially under high-level non-IID settings.

7 Conclusion

In this work, we propose the FINCH framework, which adopts hierarchical neural architecture search (NAS) to address the challenges of non-IID data and limited network resource in edge computing. Specifically, several subnets are sampled from the pre-trained supernet and allocated to the proper client clusters for model searching and training. Besides, we propose an efficient algorithm, called DCSA, to perform the proper client clustering and subnet allocation according to the data distribution and netowrk resource. We conduct extensive experiments on the real-word classical datasets. The results demonstrate the effectiveness of FINCH which can significantly improve the training performance with less resource consumption in edge computing.

References

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Towards an intelligent edge: Wireless communication meets machine learning," *arXiv preprint arXiv:1809.00343*, 2018.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

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

[4] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[5] C. He, E. Mushtaq, J. Ding, and S. Avestimehr, "Fednas: Federated deep learning via neural architecture search," 2021.

[6] M. Xu, Y. Zhao, K. Bian, G. Huang, Q. Mei, and X. Liu, "Federated neural architecture search," *arXiv preprint arXiv:2002.06352*, 2020.

[7] C. He, H. Ye, L. Shen, and T. Zhang, "Milenas: Efficient neural architecture search via mixed-level reformulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 993–12 002.

[8] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 344–353.

[9] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[10] Y. Esfandiari, S. Y. Tan, Z. Jiang, A. Balu, E. Herron, C. Hegde, and S. Sarkar, "Cross-gradient aggregation for decentralized learning from non-iid data," *arXiv preprint arXiv:2103.02051*, 2021.

[11] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[12] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[14] S. You, T. Huang, M. Yang, F. Wang, C. Qian, and C. Zhang, "Greedynas: Towards fast one-shot nas with greedy supernet," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1999–2008.

[15] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3681–3690.

[16] F. Zhang, J. Ge, C. Wong, S. Zhang, C. Li, and B. Luo, "Optimizing federated edge learning on non-iid data via neural architecture search," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[17] H. Zhu and Y. Jin, "Real-time federated evolutionary neural architecture search," *IEEE Transactions on Evolutionary Computation*, 2021.

[18] L. Dudziak, S. Laskaridis, and J. Fernandez-Marques, "Fedoras: Federated architecture search under system heterogeneity," *arXiv preprint arXiv:2206.11239*, 2022.

[19] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[20] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[21] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[22] J. Liu, H. Xu, L. Wang, Y. Xu, C. Qian, J. Huang, and H. Huang, "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[23] N. Mhaisen, A. A. Abdellatif, A. Mohamed, A. Erbad, and M. Guizani, "Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 55–66, 2021.

[24] Y. Li, C. Hao, X. Zhang, J. Xiong, W.-m. Hwu, and D. Chen, "Improving random-sampling neural architecture search by evolving the proxy search space," 2020.

[25] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 59–71, 2020.

[26] A. Andoni, P. Indyk, and R. Krauthgamer, "Earth mover distance over high-dimensional spaces." in *SODA*, vol. 8, 2008, pp. 343–352.

[27] I. Cardoso-Pereira, G. Lobo-Pappa, and H. S. Ramos, "Neural architecture search for resource-constrained internet of things devices," in *2021 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2021, pp. 1–6.

[28] B. Lyu, H. Yuan, L. Lu, and Y. Zhang, "Resource-constrained neural architecture search on edge devices," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 134–142, 2021.

[29] D. Wang, M. Li, C. Gong, and V. Chandra, "Attentivenas: Improving neural architecture search via attentive sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6418–6427.

[30] Y. Shu, W. Wang, and S. Cai, "Understanding architectures learnt by cell-based neural architecture search," *arXiv preprint arXiv:1909.09569*, 2019.

[31] B. Yuan, C. R. Wolfe, C. Dun, Y. Tang, A. Kyrillidis, and C. M. Jermaine, "Distributed learning of deep neural networks using independent subnet training," *arXiv preprint arXiv:1910.02120*, 2019.

[32] F. Liao and A. Kyrillidis, "On the convergence of shallow neural network training with randomly masked neurons," *arXiv preprint arXiv:2112.02668*, 2021.

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

[34] A. M. Bongale, C. Nirmala, and A. M. Bongale, "Energy efficient intra-cluster data aggregation technique for wireless sensor network," *International Journal of Information Technology*, pp. 1–9, 2020.

[35] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[36] C. Zhang, X. Yuan, Q. Zhang, G. Zhu, L. Cheng, and N. Zhang, "Towards tailored models on private aiot devices: Federated direct neural architecture search," *IEEE Internet of Things Journal*, 2022.

[37] I. K. M. Jais, A. R. Ismail, and S. Q. Nisa, "Adam optimization algorithm for wide and deep neural network," *Knowledge Engineering and Data Science*, vol. 2, no. 1, pp. 41–46, 2019.

[38] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1698–1707.

[39] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[41] P. Zhou, Q. Lin, D. Loghin, B. C. Ooi, Y. Wu, and H. Yu, "Communication-efficient decentralized machine learning over heterogeneous networks," *arXiv preprint arXiv:2009.05766*, 2020.

[42] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 1387–1395.

[43] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.

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

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

[46] H. Zhu, H. Zhang, and Y. Jin, "From federated learning to federated neural architecture search: a survey," *Complex & Intelligent Systems*, vol. 7, pp. 639–657, 2021.

[47] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

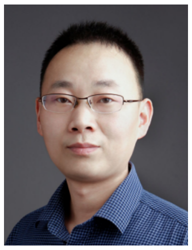
[48] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[49] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 4, pp. 1310–1322, 2019.

[50] M. Xu, Y. Zhao, K. Bian, G. Huang, Q. Mei, and X. Liu, "Neural architecture search over decentralized data," *arXiv preprint arXiv:2002.06352*, 2020.

Jianchun Liu received the Ph.D. degree in School of Data Science from the University of Science and Technology of China in 2022. He is currently an associate researcher in the School of Computer Science and Technology at University of Science and Technology of China. His main research interests are software defined networks, network function virtualization, edge computing and federated learning.

Jiaming Yan received B.S. degree in 2021 from Hefei University of Technology. He is currently studying for a master's degree in the School of Computer Science, University of Science and Technology of China (USTC). His main research interests are edge computing, deep learning and federated learning.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph. D degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

Zhiyuan Wang received the B.S. degree from the Jilin University in 2019. He is currently studying for a master's degree in the School of Computer Science, University of Science and Technology of China (USTC). His main research interests are edge computing, deep learning and federated learning.

Jingyang Huang received the B.Eng. degree from Anhui University in 2017 and the Ph.D. degree at the School of Cyberspace Security from University of Science and Technology of China in 2022. Now, he is a Lecturer at the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), and a member of the HFUT-S2AC Group. His research interests lie in Human-computer interaction, Wireless sensing, Wireless communication, and Machine learning.

Yang Xu (Member, IEEE) is currently an associate researcher in the School of Computer Science and Technology at University of Science and Technology of China. He got his Ph.D. degree in computer science and technology from University of Science and Technology of China in 2019. He got his B.S. degree in Wuhan University of Technology in 2014. His research interests include Ubiquitous Computing, Deep Learning and Mobile Edge Computing.