Real-Time Federated Evolutionary Neural Architecture Search

Hangyu Zhu and Yaochu Jin, *Fellow, IEEE*

***Abstract*—Federated learning is a distributed machine learning approach to privacy preservation and two major technical challenges prevent a wider application of federated learning. One is that federated learning raises high demands on communication resources, since a large number of model parameters must be transmitted between the server and clients. The other challenge is that training large machine learning models such as deep neural networks in federated learning requires a large amount of computational resources, which may be unrealistic for edge devices such as mobile phones. The problem becomes worse when deep neural architecture search (NAS) is to be carried out in federated learning. To address the above challenges, we propose an evolutionary approach to real-time federated NAS that not only optimizes the model performance but also reduces the local payload. During the search, a double-sampling technique is introduced, in which for each individual, only a randomly sampled submodel is transmitted to a number of randomly sampled clients for training. This way, we effectively reduce computational and communication costs required for evolutionary optimization, making the proposed framework well suitable for real-time federated NAS.**

***Index Terms*—AutoML, communication cost, deep neural networks (DNNs), federated learning, multiobjective evolutionary optimization, neural architecture search, real-time optimization.**

I. Introduction

STANDARD centralized machine learning methods require to collect training data from distributed users and store these data on a single server, which gives rise to a high risk of leaking users' private information. Therefore, a distributed approach called federated learning [1] was proposed to preserve data privacy, enabling multiple local devices to collaboratively train a shared global model while the training data remain to be deployed on the edge devices. Consequently, the central server has no access to the private raw data and the client privacy is protected.

However, federated learning demands a large amount of communication resources in contrast to the conventional centralized learning paradigm, since updating the global model needs to frequently download and upload model parameters between the server and edge clients. To mitigate this problem, a large body of research work has been carried out to reduce the communication costs in federated learning. The most popular approaches include compression and subsampling of the client uploads [2]–[4] or quantization of the weights of the models [5]. Most recently, Chen *et al.* [6] suggested a layer-wise parameter update algorithm to reduce the number of parameters to be transmitted between the server and the clients. In addition, Zhu and Jin [7] used a multiobjective evolutionary algorithm (MOEA) to simultaneously enhance the model performance and communication efficiency.

Little work has been reported on the optimization of the architecture of deep neural networks (DNNs) for federated learning, let alone real-time neural architecture search (NAS) suited to the federated environment. In the field of centralized machine learning, Zoph and Le [8] presented some early work on NAS using reinforcement learning (RL), which, however, consumes a plenty of computational resources. To mitigate this problem, Pham *et al.* [9] introduced a directed acyclic graph (DAG)-based neural architecture representation to significantly accelerate the search speed without much degradation of the learning performance by using the weight sharing technique. Khodak *et al.* [10] suggested a weight sharing method for the optimization of hyperparameters in the federated learning framework to reduce the required computational resources on the clients.

Recently, evolutionary approaches to NAS have received increasing attention [11]–[14], and hybrid methods that combine evolutionary search with the gradient-based method [15] have also been reported [16]. To reduce the computational cost, surrogate ensembles have been introduced into evolutionary NAS, which has been shown promising in reducing the computational complexity without deteriorating the learning performance [17].

Most existing centralized NAS techniques are not well suited to federated NAS for the following two reasons. First, NAS algorithms typically need a plenty of computational and memory resources for model training. However, client devices like mobile phones cannot afford computationally intensive model training and bandwidth restrictions do not allow very large models to be transmitted frequently between the server and clients. Second, many NAS methods search upon cell-based small models [9], [12], [14]–[16], [18] and transfer the found promising cell structures to large models which need to be trained from scratch before the final testing. However, retraining a large model in federated learning increases the communication costs and it might not be beneficial to enhance the performance of federated

Manuscript received January 13, 2021; revised April 2, 2021 and June 3, 2021; accepted July 20, 2021. Date of publication July 26, 2021; date of current version March 31, 2022. *(Corresponding author: Yaochu Jin.)*

The authors are with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: hangyu.zhu@surrey.ac.uk; yaochu.jin@surrey.ac.uk).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2021.3099448.

Digital Object Identifier 10.1109/TEVC.2021.3099448

learning by building large models from the found cell structures.

Note that our previous work on multiobjective evolutionary federated optimization [7] is an offline evolutionary approach to NAS. Similar to the offline evolutionary NAS methods presented in [11], [19], and [20], the parameters of a newly generated offspring model are randomly reinitialized and trained from scratch before the model is evaluated on a validation dataset, requiring a large amount of computational resources. What is worse, the performance of the reinitialized models will dramatically degrade, making the offline optimization approach infeasible for real-time federated learning systems, such as online recommender systems [21]. By real-time or online federated evolutionary NAS, we mean that the neural network models are already in use during the search process. Real-time NAS requires that the performance of the models should not drop suddenly during the search, the clients should not train multiple submodels in one round, and the communication cost should be kept minimum.

To sum up, offline federated evolutionary NAS methods are not directly applicable to real-time applications where the neural network models are already in use during the NAS. Thus, it is highly desirable to develop a framework for real-time federated evolutionary NAS, RT-FedEvoNAS for short.

The main contributions of this work are summarized as follows.

1) To tailor evolutionary NAS methods to real-time federated systems, a double-sampling technique is proposed that randomly samples subnetworks from a supernet, whose parameters are to be transmitted to a subset of randomly sampled local devices without replacement for training. The number of devices to be sampled for each submodel depends on the ratio between the number of connected local clients and the number of individuals in the population. The double-sampling technique brings about two advantages. First, only a subnetwork needs to be trained on local devices, significantly reducing the number of parameters to be uploaded from the local devices to the server. Second, sampling clients without replacement makes sure that each local device needs to train only one subnetwork for once at each generation. The above two features together make RT-FedEvoNAS well suitable for real-time federated systems and substantially different from offline evolutionary NAS, where the whole neural network must be trained on all local devices for fitness evaluations, and each device needs to train all networks in the current population. To the best of our knowledge, it is the first time that a real-time evolutionary NAS algorithm has been developed for the federated learning framework.
2) The weighted averaging strategy in the standard federated learning is applicable only if the architecture of all local models are the same. Federated NAS, however, will result in local models with different neural architectures that cannot be directly aggregated. To address this problem, an aggregation strategy is developed that updates the global model based on the subnetworks sampled and trained at each generation. Thus, in the proposed RT-FedEvoNAS, each generation is comparable to a training round in the traditional federated learning. In addition, the weights of the sampled subnetworks are inherited before it is trained on a local device, accelerating the convergence and avoiding dramatic performance deterioration caused by random reinitialization.

Extensive comparative studies are performed to verify the effectiveness of the proposed RT-FedEvoNAS by comparing the models it obtains with five predefined baseline models in terms of learning performance and computational complexity on both independently and identically distributed (IID) and non-IID data.

Algorithm 1 FedAvg. K Indicates the Total Number of Clients, B Is the Size of Mini-Batches, E Is Equal to the Number of Training Iterations, η Is the Learning Rate, n Is the Total Number of Data Pairs on All Distributed Clients, and n_k Is the Number of Data Pairs on Client k, $k = 1, 2, \ldots, m$

Server Update:
Initialize $\theta(0)$
for each communication round $t = 1, 2, \ldots$ **do**
  Select $m = C \times K$ clients, $C \in (0, 1)$ clients
  Download $\theta(t)$ to each client k
  for each client $k \in m$ **do**
    Wait **Client** k **Update** for synchronization
    $\theta(t) = \sum_{k=1}^{m} \frac{n_k}{n} \theta(t)^k$
  end for
end for

Client k **Update:**
$\theta^k = \theta(t)$
for each iteration from 1 to E **do**
  for batch $b \in B$ **do**
    $\theta^k = \theta^k - \eta \nabla L_k(\theta^k, b)$
  end for
end for
Return θ^k to server

II. Background and Motivation

In this section, a review of federated learning is given at first, followed by an introduction to NAS for DNNs. Then, the basic idea of an MOEA will be introduced. Finally, we briefly reiterate the motivation of the present work and clarify the main differences between the offline and real-time (online) evolutionary optimization frameworks.

A. Federated Learning

As mentioned before, federated learning [1] is an emerging decentralized privacy-preserving model training technology that enables local users to collaboratively train a global model without uploading their local data to a central server. A conventional federated learning algorithm called the federated averaging (FedAvg) algorithm [1] is shown in Algorithm 1. In the following, we briefly introduce this algorithm.

1) Server Side: The model parameters $\theta(0)$ are initialized once at the beginning of the FedAvg algorithm, which is then sent to $m = C \times K$ participating clients, where K is the number of total clients and $0 \leq C \leq 1$ is the participation ratio. After all m clients update and send the updated local model

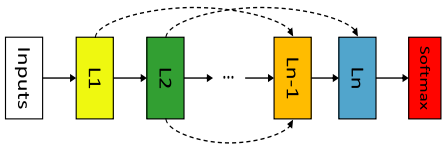


Fig. 1. Illustrative example of a CNN represented in a macro search space.

parameters $\theta^k (k = 1, 2, \ldots, m)$ back to the central server, the parameters $\theta(t)$ of the global model on the server will be replaced by the weighted average of all client's model parameters θ^k.

2) Client Side: The local model parameters θ^k are replaced by the downloaded global model parameters θ_t. Then the local model parameters are updated by the batch stochastic gradient descent (SGD) algorithm [22], where b is the local learning batch size. After local training, the trained model parameters θ^k will be sent back to the central server for global model aggregation.

An alternative to the above approach is to calculate and upload the local gradients only. Then, the local gradients are aggregated on the server by $\theta(t) = \theta(t) - \sum_{k=1}^{m}(n_k/n)g^k$, where g^k represents the local gradients. This method is beneficial to reduce the local computation consumption while achieving the same computing result with the previous one.

B. Neural Architecture Search

NAS is an emerging technique to automatically search for good model structures. In this work, we limit our discussions to convolutional neural networks (CNNs) [23], [24].

Generally speaking, NAS search space can be categorized into macro and micro search spaces [9]. A macro search space (Fig. 1) is over the entire CNN model, for example, the number of hidden layers n, operation types (e.g., convolution), and the link methods for shortcut connections (first used in ResNet [25]), among many others. Different from the macro search space, a micro search space only covers repeated motifs or cells [26], [27] in the whole model structure. And these cells are built in complex multibranch operations [28], [29] as shown in Fig. 2.

Much work has been done that uses RL for NAS [8], [9], [30]–[32]. RL-based search methods always adopt a recurrent neural network (RNN) [33] as a controller to sample a new candidate model to be trained and then use the model performance as the reward score. Then, this score can be used to update the controller for sampling a new candidate model in the next iteration. However, most RL-based NAS methods are computationally intensive due to the large search space. In order to alleviate this issue, Guo *et al.* [34] proposed an efficient curriculum search method that starts from a small search space and gradually expands to a large one, which effectively improves the search efficiency.

Evolutionary algorithms (EAs) have become increasingly popular in NAS. Different from the previous neuroevolution techniques [35] that aim to optimize both the weights and architecture of neural networks, EA-based NAS only optimizes the model architecture itself, and the model parameters are trained using conventional gradient descent methods [11]–[14]. Since EAs are particularly well suited to dealing with multiobjective optimization problems, multiobjective evolutionary NAS has received increased attention. For example, NSGA-net [20] adopted the elitist nondominated sorting genetic algorithm (NSGA-II) to optimize the performance and floating-point operations per second (FLOPs). However, NSGA-net needs to reinitialize a set of newly generated models which are trained from scratch for fitness evaluations, which is computationally very expensive. Interesting ideas to avoid training networks from scratch have been proposed, such as weight sharing [9], network morphism [36], and network transformation [37], [38]. A Lamarckian inheritance-based network morphism mechanism is designed for multiobjective evolutionary NAS to accelerate the search process [39], where both predictive performance and the number of parameters of the models are optimized. Besides, NSGANetV2 [40] uses the weights inherited from the trained supernet [41] as a warm-up to speed up model training. At the same time, a surrogate model is adopted as an accuracy predictor to reduce computation time. In addition, Zhou *et al.* [42] proposed an EA-based method for shallowing DNNs at block levels, which adopts Pareto ensemble pruning [43] to simultaneously maximize the generalization performance and minimize the number of base learners whose parameters are shared within unfolded multipath blocks. More recently, Yang *et al.* [44] proposed a continuous evolution strategy for efficient NAS, in which a modified NSGA-III [45], called pNSGA-III, is used to search for two sets of Pareto-optimal solutions, one simultaneously maximizing the accuracy and minimizing the number of parameters, and the other simultaneously maximizing the increase of accuracy and minimizing the number of parameters, to address the so-called a small model trap phenomenon. Meanwhile, Cai *et al.* [46] designed an once-for-all network that decouples model training from architecture search and adopts a progressive shrinking algorithm to mitigate the interference between the searched subnetworks. After the once-for-all network has been trained, a surrogate model named neural-network-twins is built to predict the latency and

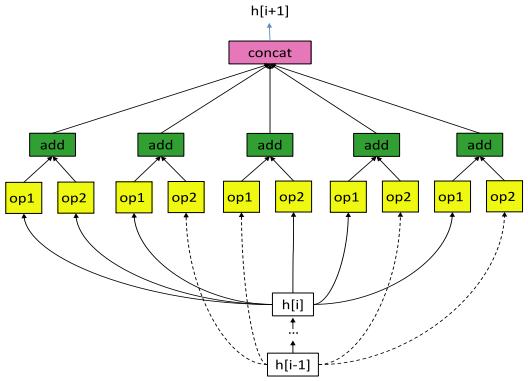


Fig. 2. Typical structure of the normal cell represented in a micro search space, where each block receives the outputs from the previous cell $h[i]$ and the cell before the previous cell $h[i-1]$ as its inputs, which are then connected to two operations, denoted by "op" in the figure. Finally, all the branches are concatenated at the output of this cell.

accuracy for a given model architecture, and an evolutionary search is used to generate a subnetwork upon neural-network-twins according to different requirements of the hardware platform. It should be pointed out, however, that the above methods for avoiding training networks from scratch cannot be directly employed in federated NAS.

The gradient-based NAS methods have become increasingly popular recently, mainly because their search speed is much faster than the RL-based and evolutionary NAS methods. In [15] and [16], relaxation tricks [47] are used to make a weighted sum of the candidate operations differentiable so that the gradient descent can be directly employed upon these weights [48].

However, the gradient-based techniques require much more computation resources than other approaches, since the overall network needs to be jointly optimized. To fix this issue, a sampling strategy is proposed by [41], in which a multibranch supernet is constructed at first. And at each time, only a single path [49] of the whole supernet is uniformly sampled for submodel training. Other techniques, such as Bayesian optimization [50], are also useful approaches to reducing the computational complexity of NAS [51].

Since this research area is becoming increasingly popular, it is of great importance to provide best practices and guidelines for designing and comparing NAS algorithms [52], [53]. Consequently, several NAS benchmarks [54], [55] have also been released to make NAS simulation results more reproducible.

C. Multiobjective Evolutionary Optimization

Federated NAS is naturally a multiobjective optimization problem. For instance, the model performance should be maximized, while the payload transferred between the server and clients should be minimized. In the machine learning community, multiple objectives are usually aggregated into a scalar objective function using hyperparameters. In contrast, the Pareto approach to solving multiobjective optimization problems has been very popular and successful in evolutionary computation [56], which has also been extended to machine learning [57]. The main difference between the Pareto approach and the conventional aggregation-based approach to machine learning is that in the Pareto approach, no hyperparameters need to be defined for aggregating different objective functions and a set of models presenting the tradeoff relationship between the objectives will be obtained. Finally, one or multiple models can be chosen from the tradeoff solutions based on the user's preferences.

NSGA-II is a very popular MOEA based on the dominance relationship between the individuals [58]. The overall framework of NSGA-II is summarized in Algorithm 2.

The main idea of nondominated sorting is to generate a set of ordered fronts based on the Pareto dominance relationship between the objective values of $R(t)$ and solutions located in the same front cannot dominate with each other. Solutions in the first nondominated fronts will have a higher priority to be selected. This sorting algorithm has a computation complexity of $O(mN^2)$, where m is the number of objectives and N is the population size. To promote the solution diversity, a crowding distance that measures the distance between two neighboring solutions in the same front is calculated and those having a large crowding distance have a higher priority to be selected. The computation complexity of crowding distance calculation is $O(mN^2\text{logN})$ in the worst case, when all the solutions are located in one nondominated front. Readers are referred to [58] for more details.

Algorithm 2 NSGA-II. N Is the Population Size, t Is the Generation Index, T Is the Total Number of Generations, P Is the Parents, and Q Is the Offspring

```
Initialize t = 1
Initialize parents P(1) with a population size N
Calculate the objective values for P(1)
for each generation t = 1, 2, ..., T do
    Generate N offspring Q(t) by applying genetic operators on P(t)
    Calculate objective values of Q(t)
    Combine the parent and offspring populations: R(t) ← P(t) + Q(t)
    Sort R(t) based on dominance relationship and crowding distance
    Select the best N individuals from R(t) as new parents P(t + 1)
    t ← t + 1
end for
Return P(t)
```

D. Motivation for Real-Time Federated NAS

There are several practical reasons for developing real-time federated evolutionary NAS algorithms, in which the machine learning models are used online during the NAS. First, federated learning is of great importance not only for privacy preservation but also for real-world applications where the system consists of a large number of subcomponents such as the Industrial Internet of Things. In such systems, there is a large number of edge devices that must solve highly complex learning problems to accomplish their tasks with limited computational and memory resources. Second, compared with gradient-based and RL-based NAS methods, EAs are more powerful in dealing with multiobjective search, and federated learning typically needs to consider multiple objectives, such as maximization of learning performance and minimization of communication costs. Furthermore, EAs are well suited to discrete optimization problems such as NAS, while gradient-based and RL-based methods are originally meant for continuous optimization problems. Finally, although gradient-based methods, such as DARTS [15], converge faster in the search, they require much more hardware memory resources than RL and EA-based methods [59], making them less practical for federated NAS.

Bayesian optimization is recognized as an attractive approach to reducing the computational complexity of NAS. However, Bayesian optimization is not well suited to federated NAS. One reason is that Bayesian optimization relies on a Gaussian process, which suffers from a cubic time complexity and may become computationally prohibitive as the number of training samples increases. In addition, traditional Bayesian optimization is not well suited to dealing with multiobjective optimization problems either.

However, conventional offline evolutionary NAS is not suited to real-time federated applications such as federated

recommender systems for the following reasons. First, a neural network model is not allowed to be randomly initialized when it is already in use, because random initialization will seriously deteriorate the performance of the neural network. In addition, some weight inheritance techniques like network morphisms [36], [39] may fail to work in the federated environment for two reasons: 1) random or identity remapping [37], [38] of the newly inserted hidden layers may cause severe model degradation in federated learning and 2) model distillation [60] is much harder to be used in a distributed environment, since the central server does not have the local training data. As a result, distilling the neural network needs to download and upload the model parameters between the server and the clients, thereby increasing communication costs. Even worse, model distillation may cause model divergence if the local data are non-IID. Second, EAs are population-based search methods and at each generation (i.e., time instant), a set of models (depending on the population size) must be assessed, requiring a large amount of computational resources on each client. Furthermore, in an offline evolutionary NAS algorithm, all parameters of multiple models (depending on the population size) must be transferred between the server and the clients, which considerably increases communication costs. Finally, the models in offline evolutionary NAS are usually not fully trained during the optimization to reduce the computation time. Thus, they must be trained again at the end of the evolutionary search, which will incur additional communication costs.

For the above reasons, this work aims to develop a real-time evolutionary NAS framework that is able to flexibly handle multiple objectives within a federated learning system. To satisfy the real-time requirements, the proposed evolutionary NAS framework has the following properties.

1) With the help of submodel sampling together with weight sharing, all local models can maintain a stable performance, which is essential for real-time NAS.
2) A filling strategy is suggested to aggregate submodels with different architectures, making it possible to assemble a global model with consistently good performance.
3) Each local client needs to train one submodel only in each round thanks to the client sampling strategy. This represents one major difference between the proposed real-time evolutionary federated NAS and the offline evolutionary NAS reported in [7].

III. Proposed RT-FedEvoNAS

In this section, we introduce the encoding method and model structure adopted in RT-FedEvoNAS at first. Then, the objectives to be optimized and a double-sampling method for evaluating the objectives are described. Finally, we present the overall framework of RT-FedEvoNAS.

A. Structure Encoding

For real-time NAS, we adopt a light-weighted CNN as the global model, since communication costs are always a primary concern in federated learning. In addition, the search space should not be too large and the total number of layers should be limited to make it appropriate for real-time federated optimization.

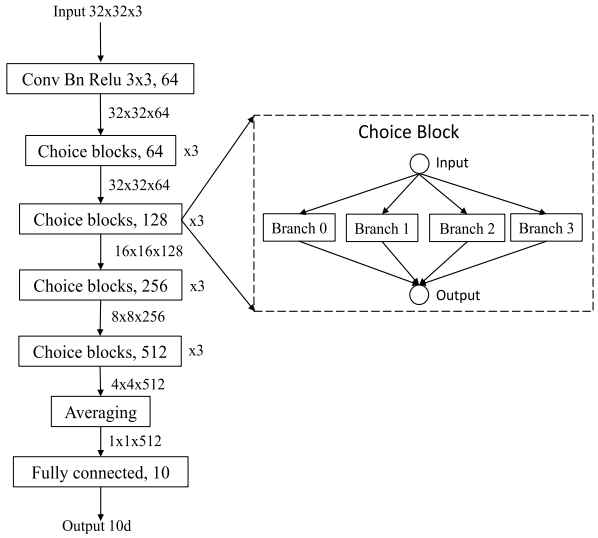


Fig. 3. Example structure of the global model, in which each choice block consists of four branches, where the last digit in each choice block, such as 64 and 128, is the filter channel number.

The global model used in the proposed real-time federated NAS is shown in Fig. 3, where a convolutional block, 3×4 choice blocks (each branch of the choice block contains two convolutional or more advanced depthwise convolutional layers except for the identity block) and a fully connected layer are linked to build a DNN containing a maximum of 26 hidden layers. Specifically, the convolutional block consists of three sequentially connected layers with a convolutional layer, a batch normalization layer [61], and a rectified linear (ReLu) layer. And one choice block is composed of four predefined branches of candidate blocks, namely, identity block, residual block, inverted residual block, and depthwise separable block, as shown in Fig. 4. Thus, there are in total 4^{12} possible one-path subnetworks. In addition, these four candidate blocks are categorized into two groups, one is called a normal block whose input and output share the same channel dimension, whereas the other is called a reduction block whose output channel dimension is doubled, and the spatial dimension is quartered compared to its input.

Identity block directly links its input to its output [Fig. 4(a), left panel], which can be seen as a "layer removal" operation. The right panel of Fig. 4(a) shows a structure reduction block, which operates two branches of pointwise convolution with a stride of 2 at first and then concatenates these two outputs. As a result, the spatial dimensions of the inputs are quartered and the filter channels are doubled through this identity reduction block.

Residual block contains two sequentially connected convolution blocks as shown in Fig. 4(b), where the normal block (left panel) is the same as the residual block used in ResNet [25]. Note that the reduction residual block does not contain shortcut connections while the normal block has.

Inverted residual block [Fig. 4(c)] has an "inverted" bottleneck structure of the residual block, which was first proposed

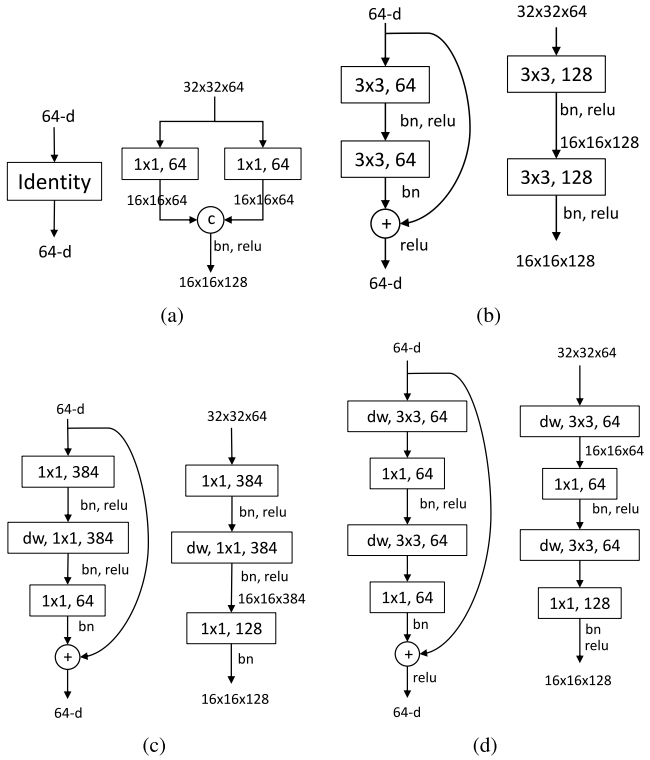


Fig. 4. Four candidate blocks used in RT-FedEvoNAS, where the left panel of each subfigure is the normal blocks and the right panel is the reduction blocks. Symbol c in (a) represents the concatenation operation. Only normal blocks contain shortcut connections. (a) Identity block. (b) Residual block. (c) Inverted residual block. (d) Depthwise separate block.

in MobilenetV2 [62]. This block contains three convolution layers: 1) a 1×1 expanding pointwise convolution layer, followed by a batch normalization layer and a ReLu activation function; 2) a 3×3 depthwise convolution layer [63], [64], followed by a batch normalization layer and a ReLu activation function; and 3) a 1×1 spatially filtered pointwise convolution layer followed by a batch normalization layer.

The intuition of using expanding in the first layer instead of in the last layer as done in the bottleneck layer [25] is that a nonlinear activation function, such as ReLu, may cause layer information loss [65] and using a nonlinear projection upon a high-dimensional space can mitigate this issue. After the tensor is mapped back to a low-dimensional space again through the last pointwise layer, the ReLu function is not needed to prevent information loss.

Depthwise separable block consists of two depthwise convolution operations [63] [Fig. 4(d)] that requires less computation power than the conventional convolution operation. It has been proved in [64] that 3×3 depthwise convolution consumes about one-eighth to one-ninth of the computation time of the standard convolution at the expense of a small deterioration in performance.

For each communication round, only one branch of all 12 choice blocks is sampled from the global model and then downloaded to a client device for reducing communication costs and computation resources required on the local device. This sampled submodel can be encoded into a two-bit binary

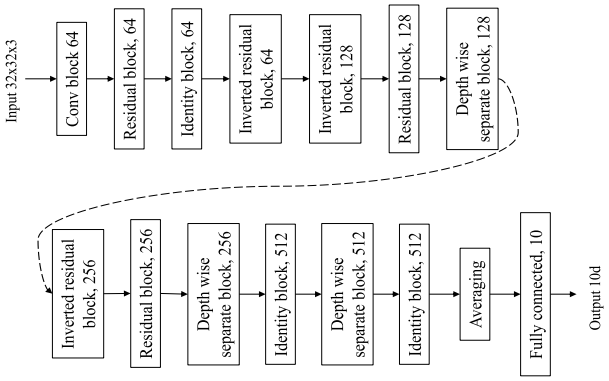


Fig. 5. Submodel represented by the choice key [0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0].

string with a total length of $2 \times 12 = 24$ bits. Every two bits in the code represent one specific branch in the choice block. For instance, [0, 0] represents branch 0, which is the identity block, [0, 1] represents branch 1, i.e., the residual block, [1, 0] represents branch 2, i.e., the inverted residual block, and [1, 1] represents branch 3, which is the depthwise separable block. Therefore, the binary string (also called a choice key) [0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0] can be decoded into a submodel with structure as shown in Fig. 5.

B. Double-Sampling Strategy for Objective Evaluations

Offline evolutionary optimization is intrinsically not suited to federated learning. Although one or multiple light-weighted high-performance models can be found by an offline EA in the last generation [7], a large amount of extra computation and communication resources is required. For instance, for an EA of a population size of N, each client in an offline evolutionary NAS algorithm must evaluate the fitness of N individuals (each representing a model) at each generation, which is $N-1$ times in extra compared to the gradient-based method. In addition, the models are repeatedly randomly reinitialized and trained from scratch on the clients, which may not be good enough to be in the course of evolutionary optimization.

In order to address the above issues, a double-sampling technique is proposed here to develop a real-time evolutionary NAS algorithm, in which at each generation, the model for each individual is subsampled uniformly from a common *global model*, whereas the clients for training the model of the individual is subsampled also uniformly from the participating clients. Specifically, a choice key is generated for each individual to randomly sample a subnetwork from the supernet (global model) as this individual's model. Then, the sampled submodel of this individual will be downloaded to a randomly sampled subset of the participating clients. The number of clients to be chosen, say L, for training one submodel of an individual is determined by the ratio between the number of individuals and the number of participating clients, i.e., $L = \lfloor m/N \rfloor$, where $m = CK$ is the number of participating clients, K is the total number of clients, and C is the participating ratio in the current round. Here, we assume that

the number of clients is equal to or larger than that of the population size.

In this work, two objectives are to be optimized: one is the model performance such as the validation error of each sampled submodel, and the other is model complexity, such as the FLOPs or the number of parameters of each submodel. Note that it is fairly straightforward to include additional objectives in evolutionary multiobjective optimization [66], e.g., in the comparative studies, we include an experiment that three objectives, classification accuracy, FLOPs, and the number of parameters in the model are optimized.

Population initialization in RT-FedEvoNAS is composed of the following four main steps.

1) Initialize the global model. Generate the parent population containing N individuals, each representing a submodel sampled from the global model using a choice key. Sample L clients for each individual without replacement. That is, each client should be sampled only once.
2) Download the submodel of each parent individual to the L selected clients and train the submodel using the data on these clients. Once the training is completed, upload the L local submodels to the server for aggregation to update the global model.
3) Generate N offspring individuals using crossover and mutation and generate a choice key for each offspring individual. Download the corresponding submodel (just choice key from the second generation) of each offspring individual to L randomly sampled participating clients and train it on these clients. Upload the trained local submodels and aggregate them to update the global model.
4) Finally, download the global model together with the choice keys of all parents and offspring to all participating clients to evaluate the objectives. Upload all the objective values to the server and calculate the weighted averaging of validation errors for each individual.

Once the objective values of all parent and offspring individuals are calculated, environmental selection can be carried out to generate parent individuals of the next generation based on the elitist nondominated sorting and crowding distance, as discussed in Algorithm 2 in Section II-C.

In the following generations, similar steps as described above will be carried out, except that the global model is updated only once at each generation after the local submodels of all offspring individuals are trained on the sampled clients and the resulting local uploads are aggregated. It should be emphasized that at each generation, the global model shared by all individuals needs to be downloaded to all participating clients only once for evaluating the objectives. Then only the generated choice keys need to be downloaded to the sampled clients, since the whole global model has already been downloaded to the clients at the last generation.

Note that model aggregation is different from that in the conventional federated learning. The reason is that different clients may be sampled for different individuals, and different individuals may have different model structures, which cannot be directly aggregated. Fig. 6 shows an illustrative example, where the global model has two choice blocks C_1 and C_2. There are two individuals, and the choice keys for the two individuals are [0, 0, 0, 1] and [1, 0, 1, 1], respectively. The resulting submodels are B_0 and B_1, and B_2 and B_3. We further assume that client 1 is chosen for training submodel B_0 and B_1 and client 2 for $B2$ and $B3$. Then, each client updates its model parameters according to the available local data and then uploads the trained model to the server, which is denoted by the shaded square. And then two global models are reconstructed by filling in the submodels that are not updated, which are denoted by the white squares. Since the reconstructed global models have the same structure, they can be easily aggregated using, i.e., the weighted averaging. Besides, this aggregation method does not require extra communications, since this operation is only performed on the server. The pseudocode for model aggregation is presented in Algorithm 3.

Algorithm 3 Model Aggregation, m Is the Total Number of Client Uploads. k Is the Client Index. I Is the Total Number of Hidden Layers of the Global Model, i Is the Hidden Layer Index of the Model, B Is the Total Number of Branches in One Choice Block, b Is the Branch Block Index, n Is the Number of Data Samples on All Clients, n_k Is the Number of Data on Client k, and t Is the Communication Round

$\theta(t-1)$ is the parameters of the global model in the last communication round, $\theta_b^i(t)$ is the parameters of the global model for the b-th branch in the i-th hidden layer.
Receive client k model parameters w_k of and choice key C_k, where w_k^i is the model parameters and C_k^i is the choice of i-th hidden layer.
Server Aggregation:
Let $\theta(t) \leftarrow 0$
for each $i \in I$ **do**
  for each $k \in m$ **do**
    if w_k^i is not in choice blocks **then**
      $\theta^i(t) \leftarrow \sum_{k=1}^{m} \frac{n_k}{n} w_k^i$
    else
      for each branch $b \in B$ **do**
        if $C_k^i == b$ **then**
          $\theta_b^i(t) \leftarrow \theta_b^i(t) + \frac{n_k}{n} w_k^i$
        else
          $\theta_b^i(t) \leftarrow \theta_b^i(t) + \frac{n_k}{n} \theta_b^i(t-1)$
        end if
      end for
    end if
  end for
end for
Return $\theta(t)$

From the above description, we can see that the double-sampling strategy fits perfectly well with the population-based evolutionary search so that the objective values of all individuals at one generation can be evaluated within one communication round, seamlessly embedding a generation of architecture search into one round of training in federated learning.

C. Overall Framework

We use NSGA-II to optimize both the submodel complexity and validation performance for the real-time evolutionary federated NAS framework. The framework is illustrated in Fig. 7 and the pseudocode is listed in Algorithm 4.

Algorithm 4 Real-Time Federated NAS Using NSGA-II, N Is the Population Size, t Is the Number of Generations, K Indicates the Total Numbers of Clients; B Is the Size of Mini-Batch, E Is the Training Iterations and η Is the Learning Rate, n Is the Total Number of Data Points on All Clients, and n_k Is the Number of Data Points on Client k

1: ***// Double sampling method used here***
2: **Server:**
3: Initialize $\theta(0)$
4: $\theta(t) \leftarrow \theta(0)$
5: $t \leftarrow 1$
6: ***// Server global model sampling (model sampling)***
7: Randomly sub-sample parent choice keys $C_P(t)$ with a population size N
8: **for** each communication round $t = 1, 2, ...$ **do**
9: ***// For real-time optimization, the generation is equal to the communication round***
10: Convert all $C_P(t)$ choice keys into binary codes $Cb_P(t)$
11: Generate $Cb_Q(t)$ with the size of N by genetic operators
12: Convert $Cb_Q(t)$ into offspring choice keys $C_Q(t)$
13: $C_R(t) \leftarrow C_P(t) + C_Q(t)$
14: Select $m = C \times K$ clients, $C \in (0, 1)$ clients
15: **if** $t \leq 1$ **then**
16: Generate sub-models $\theta^p \in \theta(t)$, $p \in C_P(t)$
17: ***// Client sampling (clients sampling)***
18: Randomly sub-sample m clients to N groups
19: **for** $p \in C_P(t)$, $g \in N$ ($|C_P(t)| = N$) **do**
20: Download θ^P to all clients in group g
21: **end for**
22: **for** each client $k \in m$ **do**
23: Wait **Client** k **Update** for global model aggregation
24: $\theta(t) \leftarrow$ *Do server aggregation in* Algorithm 3
25: **end for**
26: **end if**
27: Generate sub-models $\theta^q \in \theta(t)$, $q \in C_Q(t)$
28: ***// Client sampling (clients sampling)***
29: Randomly sub-sample m clients to N groups
30: **for** $q \in C_Q(t)$, $g \in N$ ($|C_Q(t)| = N$) **do**
31: **if** $t > 1$ **then**
32: Download the choice key q to all clients in group g
33: **else**
34: Download θ^q and the choice key q to all clients in group g
35: **end if**
36: **end for**
37: **for** each client $k \in m$ **do**
38: Wait **Client** k **Update** for global model aggregation
39: $\theta(t) \leftarrow$ *Do server aggregation in* Algorithm 3
40: **end for**
41: ***// Perform NSGA-II optimization***
42: Calculate the complexity of all sub-models in $C_R(t)$
43: **for** each client $k \in m$ **do**
44: Download global model $\theta(t)$ and choice keys $C_R(t)$ to client k
45: Calculate the validation errors for all sub-models in $C_R(t)$
46: Upload them to the server
47: **end for**
48: Calculate the weighted average of the validation errors based on the local data size to achieve the final validation errors of all sub-models in $C_R(t)$
49: Do fast non-dominated sorting
50: Do crowding distance sorting
51: ***// New solutions are generated within each communication round***
52: Generate new parent choice keys $C_P(t)$ from $C_R(t)$
53: $t \leftarrow t + 1$
54: **end for**
55:
56: **Client** k **Update:**
57: **if** Receive one choice key q **then**
58: Sub-sample $\theta(t)$ based on the choice key to generate θ^k
59: **else**
60: $\theta^k \leftarrow \theta^p or \theta^q$
61: **end if**
62: **for** each iteration from 1 to E **do**
63: **for** batch $b \in B$ **do**
64: $\theta^k = \theta^k - \eta \nabla L_k(\theta^k, b)$
65: **end for**
66: **end for**
67: return θ^k to the server

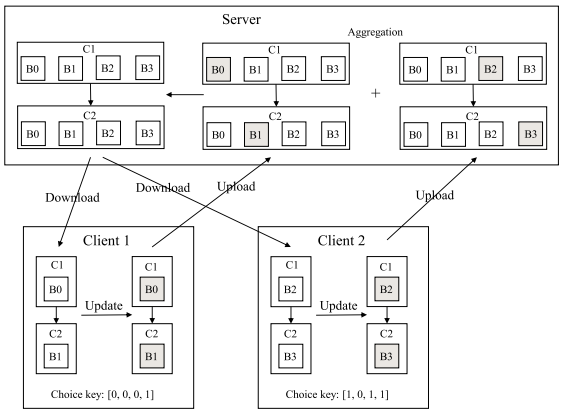


Fig. 6. Illustrative example of model aggregation. The global model contains two choice blocks (C_1, C_2) and B_0, B_1, B_2, and B_3 are four different branches in a choice block. The two sampled submodels are downloaded to client 1 and client 2, respectively, for training. After the updated submodels are uploaded, they are filled with the remaining submodels (those not updated in this round for this individual) to reconstruct the global model before all reconstructed global models are aggregated.

The objective evaluations are done for both parent and offspring populations at each generation, which is comparable to a communication round in the proposed RT-FedEvoNAS. For fitness evaluations, both the global model and all choice keys $C_R(t)$ are downloaded to all participating clients. Thus, we do not need to download any model parameters for training submodels in the next round and it is sufficient to download the choice keys only (refer to lines 32–34 in Algorithm 4. After that, each client uploads the updated local submodels to the server for model aggregation. As a result, the proposed model sampling method can reduce both local computation complexity and communication costs for uploading the models.

It should be noticed that the parent submodels are trained only at the first generation. In the subsequent evolutionary optimization, only offspring submodels need to be trained. However, all $2N$ submodels sampled by $C_R(t)$ need to be evaluated to calculate the validation errors for objective evaluations at each generation, because training the offspring submodels will also affect the parameters of the parent submodel since parent and offspring submodels always share weights of the global model. In addition, we do not need to reinitialize the model parameters of the sampled offspring submodels before training starts. Due to the client sampling strategy, the population size N does not affect the communication costs for objective evaluations, since the entire global model is downloaded from the server, and sampling of the global model can be done on the clients.

Since NSGA-II is a Pareto-based multiobjective optimization algorithm which can find a set of optimal

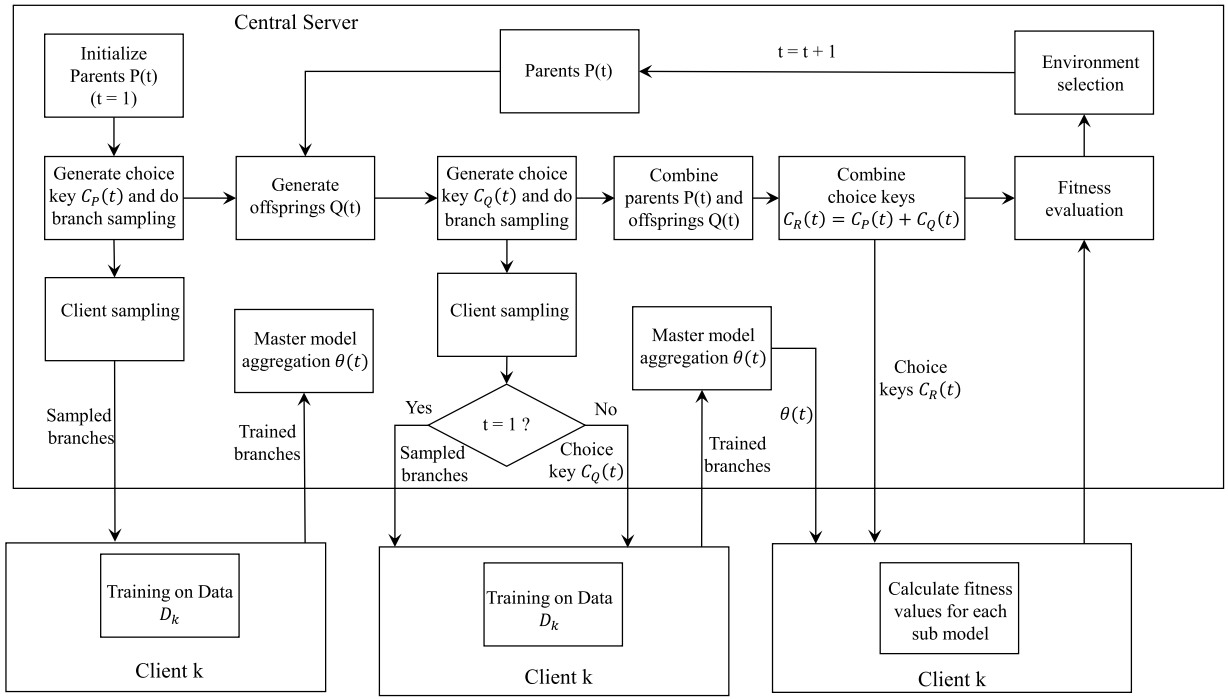


Fig. 7. Overall framework for multiobjective real-time evolutionary federated NAS, where t is the communication round, $P(t)$ is the parent population, and $C_P(t)$ is the corresponding choice keys, $Q(t)$ is the offspring population and $C_Q(t)$ is its choice keys, $R(t)$ is the combined population and $C_R(t)$ is the choice keys of the combined population, and $\theta(t)$ is all trainable parameters of the global model.

models that tradeoff between conflicting objectives such as model accuracy and computational complexity. Therefore, for real-time applications, the user needs to articulate preferences to select one or some of the Pareto-optimal solutions from the parent population in each round. In practice, the Pareto-optimal solutions with the highest validation accuracy and the knee point [67]–[69] are preferred unless there are other strong user-specified preferences.

IV. Experimental Results

The purpose of the experiments we conduct here is not meant to show that RT-FedEvoNAS can compete with state-of-the-art of NAS algorithms. Instead, we aim to empirically verify that the proposed algorithm is successful in accomplishing real-time federated NAS and to understand its performance in learning IID and non-IID data, its computational complexity, and the models it can obtain, in comparison with other search methods such as simple random search. The detailed experimental settings are shown in Tables S1 and S2 of Sections I and II in the supplementary document.

A. Datasets

Five datasets are used in our experiments, namely, CIFAR10, CIFAR100 [70], Street View House Numbers (SVHN), Pathmnist [71], and Tiny Imagenet [72].

CIFAR10 contains 50 000 training and 10 000 testing 32×32 RGB images with ten different kinds of objects. CIFAR100 contains 50 000 training and 10 000 testing 32×32 RGB images with 100 different kinds of objects. Therefore, CIFAR100 is a more challenging dataset compared to CIFAR10.

The SVHN dataset is obtained from house numbers in Google street view images and contains 73 257 training, 26 032 testing, and 531 131 additional 32×32 RGB digits with ten different classes. Each class label represents one digit, e.g., label "1" represents digits "1" and label "0" represents digits "10." Only training and testing digits are used in our experiments.

The description of Pathmnist and Tiny Imagenet and the corresponding simulation results are presented in Sections VIII and IX, respectively, of the supplementary document.

To simulate federated learning on IID data, all training image data are evenly and randomly distributed to each local client without overlap. For experiments on non-IID data, each client has images with five classes of objects for both CIFAR10 and SVHN datasets, and 50 classes of objects for the CIFAR100 dataset. In all experiments, 20% of the training data on each client are used for validation and the rest are used for training.

Note that we do not apply any data augmentation [73] in our experiments, because augmentation techniques consume additional computation resources on the clients.

B. Global and Baseline Models

The global model used in this work is a supernet with multiple branches containing a total of 28 layers (12 choice blocks, each containing two convolution layers). The number of channels for all block layers is [64, 64, 64, 128, 128, 128, 256, 256, 256, 512, 512, 512]. The overall structure of the global model is shown in Fig. 3. Note that both trainable parameters and moving statistics in the batch normalization layers are disabled for simplicity in our simulations, since they might slow down the convergence at the early stage of the

weight sharing training paradigm [49], [74], [75]. Our empirical results related to the role of the trainable parameters and moving statistics in federated NAS are given in Section VII of the supplementary document.

The submodels found by RT-FedEvoNAS are compared with five baseline models in terms of their objective values over the entire optimization process. One baseline model is a typical ResNet [25] containing 18 hidden layers. More light-weighted models designed for edge devices, i.e., MobileNet [76] containing 15 hidden layers with depthwise and pointwise convolutions and MobileNetv2 [77] containing 17 inverted residual blocks are also compared in the experiments. In addition, two smaller models, PnasNet-1 and PnasNet-2 with 1 and 2 blocks in each repeated cell, respectively, found by PNASNET [78] are also included as our baseline models for comparison. All the five baseline models are implemented according to the published papers, while minor structure changes are applied to MobileNetV2 to make it suited for 32×32 RGB images (the original model structure is designed for the ImageNet dataset [24]).

C. Results on Different Numbers of Clients

In this set of experiments, we examine the performance of the obtained submodels given different numbers of clients. To this end, we adopt three different numbers of clients (10, 20, 50, and 100) and optimize two objectives, i.e., the validation error and FLOPs of the submodels on CIFAR10.

The obtained Pareto-optimal solutions on both IID and non-IID data over every 50 communication rounds (generations) are shown in Fig. 8 to visualize the objective values (validation accuracy and FLOPs) of the evolved submodels during the optimization.

The following three observations can be made. First, the classification accuracies of the optimized models on IID data are better than those on non-IID data. Second, the smaller the number of clients, the better the classification performance. These two phenomenons are reasonable, since learning on IID data is much easier than learning on non-IID data in federated learning. On the other hand, the more the number of clients, the less data there is on each client, as the amount of data in total is given. Third, the population converges well during the optimization and the Pareto-optimal solutions in the last generation are evenly distributed.

To take a closer look at the final test performances of obtained models, we present the test accuracies and FLOPs of the submodel having the highest validation accuracy (called *High*) and the knee solution (called *Knee*) in Table I. Note that all test accuracies of the optimized submodels are obtained directly on testing data without retraining them from scratch.

From these results, it is clear to see that solutions with higher model FLOPs always have better final test accuracies. Furthermore, the final test accuracies of obtained two Pareto-optimal submodels become worse as the total number of clients becomes larger (e.g., 84.67% with ten clients and 67.66% with 100 clients on IID data). This indicates that it becomes harder to find an optimal submodel as the amount of data on each client becomes less.

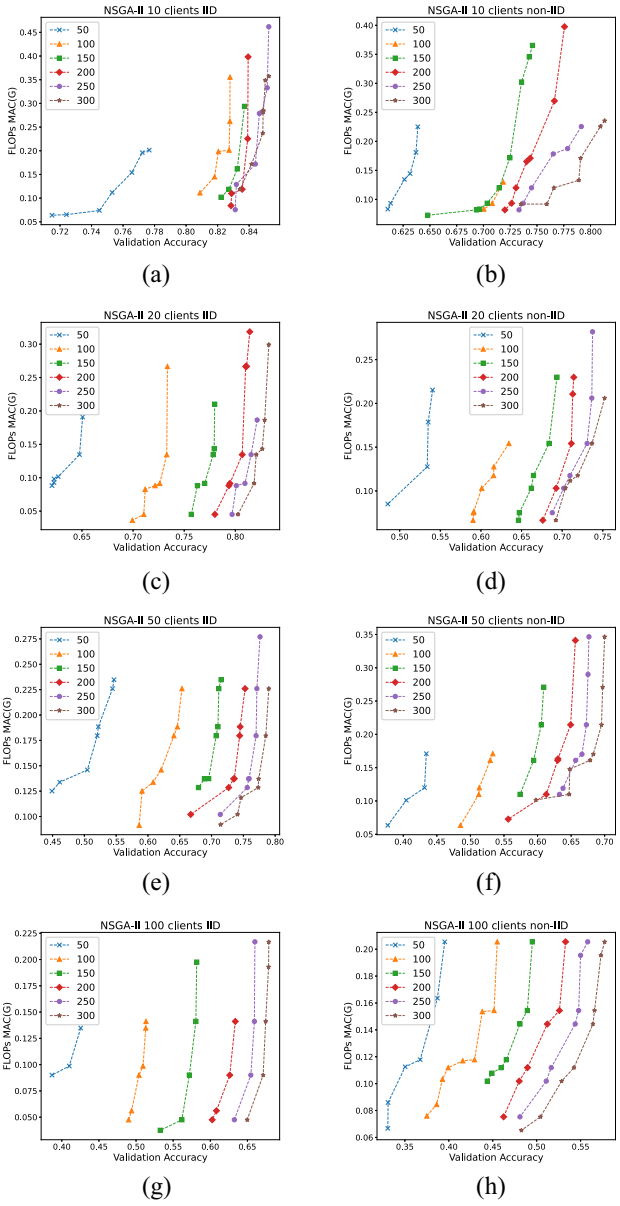


Fig. 8. Pareto-optimal solutions obtained on IID and non-IID CIFAR10 data for 10, 20, 50, and 100 clients. In the figure, the integer numbers in the legend in each subplot represent the number of communication rounds (generations) during the real-time federated evolutionary optimization. (a) 10 clients, IID data. (b) 10 clients, non-IID data. (c) 20 clients, IID data. (d) 20 clients, non-IID data. (e) 50 clients, IID data. (f) 50 clients, non-IID data. (g) 100 clients, IID data. (h) 100 clients, non-IID data.

D. Real-Time Performances

Apart from the final results in the last generation of the optimization, we also compare the real-time performances of the solutions found by RT-FedEvoNAS with those found by the baseline models on the three datasets. In this set of experiments, we extend our previous biobjective optimization formulation of real-time federated NAS into a three-objective optimization problem by including an extra objective, i.e., the number of model parameters.

Similarly, we examine the real-time performances of two preferred Pareto-optimal solutions: one is the model having the highest validation accuracy and the other is the knee solution.

TABLE I
FINAL TEST PERFORMANCES OF THE EVOLVED TWO PARETO SOLUTIONS ON CIFAR10

Model	Clients	IID	Test accuracy	FLOPs (MAC)
High	10	Yes	**84.67%**	0.3572G
Knee	10	Yes	84.38%	**0.2370G**
High	10	No	**76.51%**	0.2356G
Knee	10	No	76.11%	**0.0920G**
High	20	Yes	**82.19%**	0.2992G
Knee	20	Yes	81.53%	**0.1433G**
High	20	No	**78.16%**	0.2060G
Knee	20	No	76.28%	**0.1542G**
High	50	Yes	**78.69%**	0.2260G
Knee	50	Yes	77.12%	**0.1285G**
High	50	No	**73.66%**	0.3465G
Knee	50	No	71.21%	**0.1702G**
High	100	Yes	**67.66%**	0.1930G
Knee	100	Yes	67.53%	**0.1411G**
High	100	No	**63.98%**	0.2055G
Knee	100	No	62.19%	**0.1443G**

For simplicity, we fix the number of participating clients to 20 here, and the results on the CIFAR10, CIFAR100, and SVHN datasets are shown in Figs. 9–11, respectively.

Regarding the real-time performances on CIFAR10, we can clearly see from Fig. 9 that two solutions (High and Knee) found by RT-FedEvoNAS are able to outperform all the baseline models after approximately 100 communication rounds on the IID data with a validation accuracy of 84.34% and 83.47%, respectively, although they are underperformed by ResNet and MobileNetV2 at the early stage. In contrast on the non-IID data, all baseline models except PnasNet-1 converge faster than the two evolved solutions. Nevertheless, the two solutions outperform MobileNet and PnasNet-2 already at the middle stage and solution "High" outperforms all the baseline models with a validation accuracy of 74.37% at the end of communication rounds. We also note that PnasNet-1 and PnasNet-2 do not perform very well compared to other models and PnasNet-1 has the slowest convergence speed on both IID and non-IID data with a validation accuracy of 69.74% and 62.50%, respectively.

The convergence curves resulting from the IID and non-IID CIFAR100 data are similar and the two models found by RT-FedEvoNAS outperform all the baseline models with a validation accuracy of approximately 53% around the middle of the communication rounds. We can also find that the real-time performances of these two solutions are very stable during the evolutionary search, although the knee solution experiences some minor fluctuations. However, ResNet, MobileNet, MobileNetV2, and PnasNet-2 suffer from a performance degradation around the 75th round, implying that they are not well suited to learning hard tasks such as CIFAR100 in the federated learning environment. Note also that PnasNet-1 does not experience a similar performance drop and continues to improve its performance over the rounds.

The results on the SVHN dataset are given in Fig. 11. All models under comparison except for PnasNet-1 converge quickly and perform very well on SVHN and the two models found by RT-FedEvoNAS perform comparably with

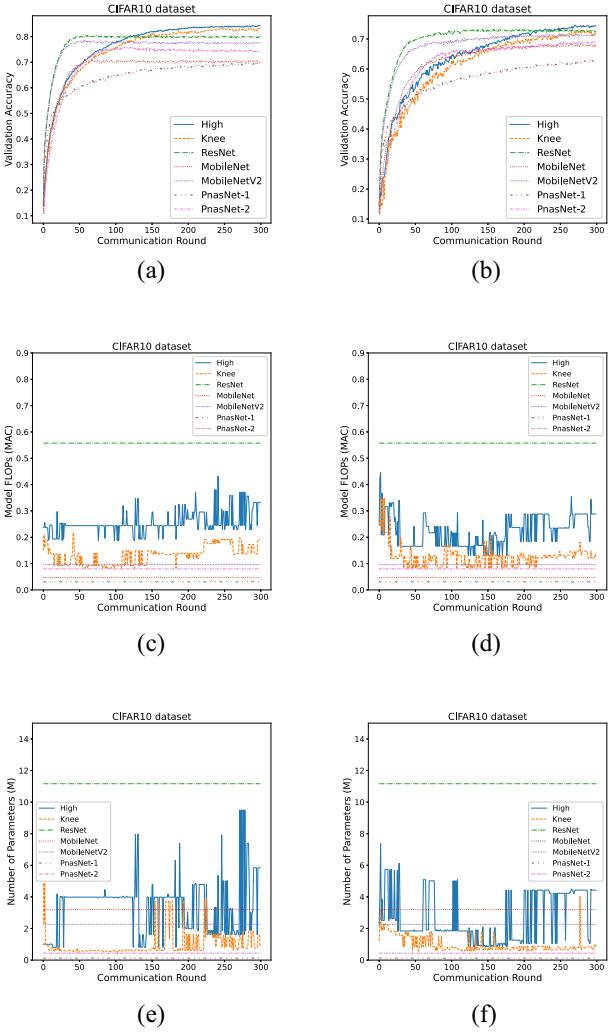


Fig. 9. Real-time performances of two Pareto-optimal solutions and five baseline models over the rounds (generations) on the CIFAR10 dataset. (a) Validation accuracy, IID data. (b) Validation accuracy, non-IID data. (c) FLOPs, IID data. (d) FLOPs, non-IID data. (e) Number of parameters, IID data. (f) Number of parameters, non-IID data.

other models on IID and non-IID data. PnasNet-1 performs worse than other compared models probably because its model complexity is too small.

In the following, we take a look at the complexity of the obtained models in terms of FLOPs and the number of parameters. The FLOPs of the two selected Pareto-optimal solutions together with those of the five baseline models on the three datasets are shown in Figs. 9(c) and (d), 10(c) and (d), and 11(c) and (d), respectively. The average model FLOPs of the High and the Knee solutions are 0.26G and 0.13G, while the average FLOPs of ResNet, MobileNet, MobileNetV2, PnasNet-1, and PnasNet-2 are about 0.56G, 0.05G, 0.10G, 0.03G, and 0.08G, respectively.

The number of model parameters of two solutions together with five baseline models are shown in Figs. 9(e) and (f), 10(e) and (f), and 11(e) and (f), respectively. We see that the number of parameters of the two models found by RT-FedEvoNAS is much smaller than ResNet, although they both outperform ResNet on CIFAR10 and CIFAR100, and comparably

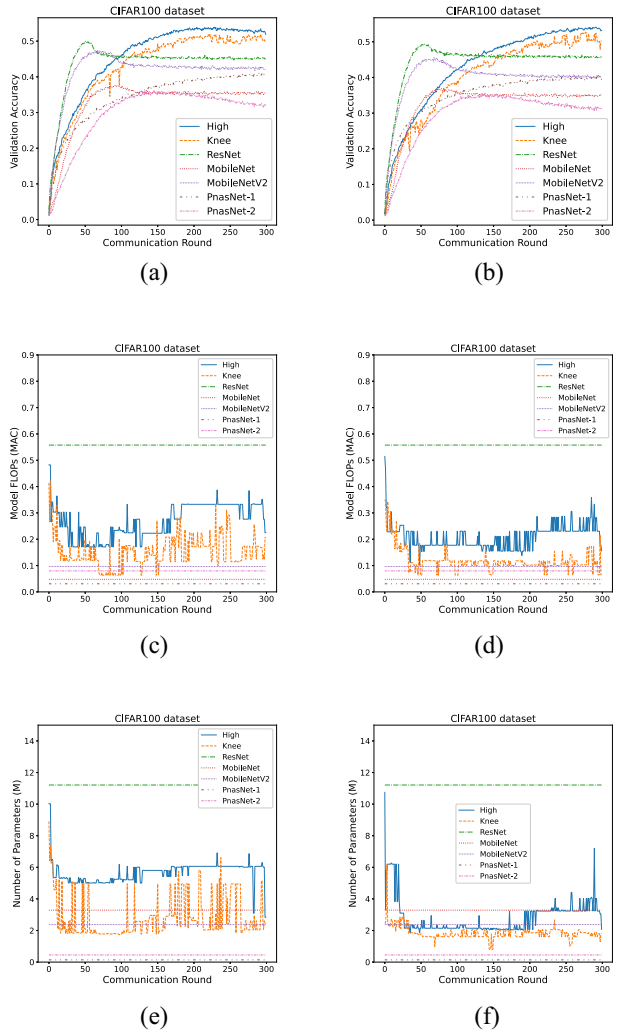


Fig. 10. Real-time performances of the two selected Pareto-optimal solutions and five baseline models over the rounds (generations) on the CIFAR100 dataset. (a) Validation accuracy, IID data. (b) Validation accuracy, non-IID data. (c) FLOPs, IID data. (d) FLOPs, non-IID data. (e) Number of parameters, IID data. (f) Number of parameters, non-IID data.

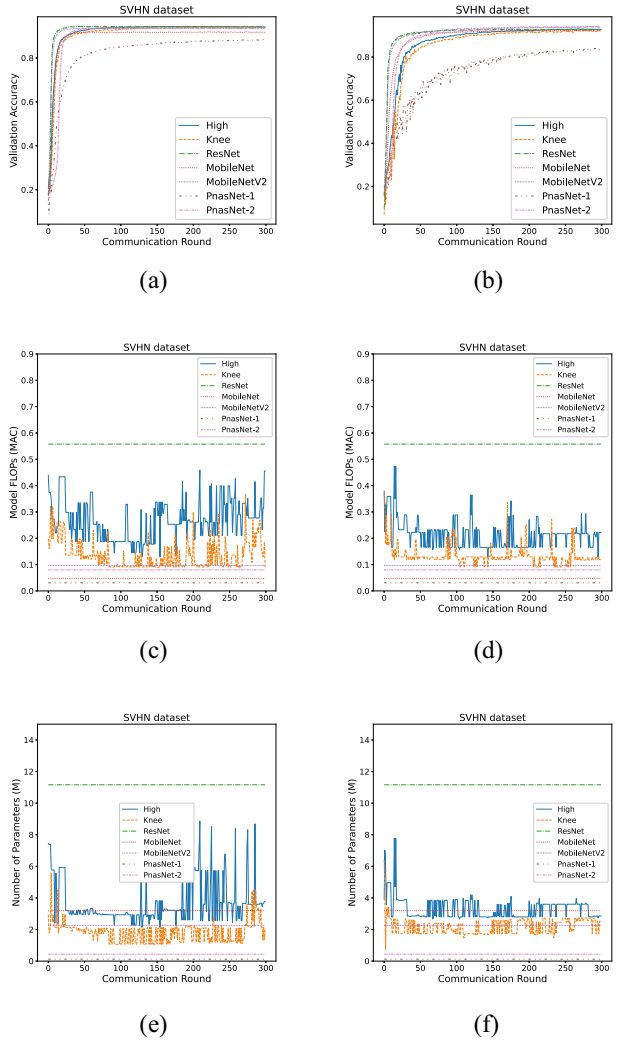


Fig. 11. Real-time performances of the two selected Pareto-optimal solutions and five baseline models over the rounds (generations) on the SVHN dataset. (a) Validation accuracy, IID data. (b) Validation accuracy, non-IID data. (c) FLOPs, IID data. (d) FLOPs, non-IID data. (e) Number of parameters, IID data. (f) Number of parameters, non-IID data.

on SVHN. The real-time model parameters of the High solution fluctuated heavily on CIFAR10 and SVHN, while it does not change much on CIFAR100. This implies that on simpler tasks, such as CIFAR10 and SVHN, a smaller model can perform as well as a larger model, and model with a higher complexity might be needed for CIFAR100. In most of the communication rounds, the number of model parameters of the Knee solution is between MobileNetV2 and PnasNet-2 and mostly smaller than MobileNetV2.

The final test accuracies together with the model FLOPs and parameters are listed in Table S3 of the supplementary document. We can find from these results that, compared to the five baseline models, the High solution exhibits the best test performance on all IID and non-IID datasets, except that it performs slightly worse than ResNet on non-IID SVHN. Although PnasNet-1 has very small FLOPs and the small number of parameters, its final test accuracies are much worse than other compared models on all datasets.

From the above results, we can conclude that the proposed RT-FedEvoNAS is not only able to find light-weighted models but also achieves stable and competitive learning performance during the optimization.

E. Comparison With Offline Federated NAS

We compare our RT-FedEvoNAS algorithm with the offline evolutionary federated NAS algorithm [7]. It should be pointed out that even if we use the same search space, it is hard to make a completely fair comparison between these two methods. This is because the offline approach requires that each individual be trained on all participating clients and the number of communication rounds has to be manually defined. A larger number of communication rounds can ensure each sampled submodel to be well trained, which however, will consume much more communication and local computation resources. In addition, the models found by the offline NAS method often need to be trained from scratch since weight sharing techniques cannot be directly applied in the offline evolutionary search, which

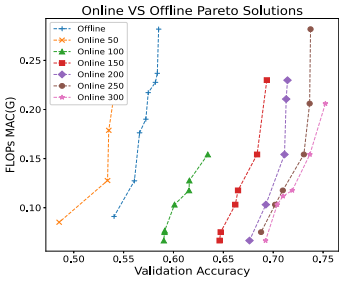


Fig. 12. Pareto-optimal solutions with 20 clients on non-IID data, where the integer numbers in the legend denote the number of generations. The Pareto-optimal solutions obtained by the offline method are plotted together with the solutions obtained by the online method over every 50 generations.

TABLE II
TIME CONSUMPTION AND NUMBER OF DOWNLOADED AND UPLOADED MODEL PARAMETERS FOR RT-FEDEVONAS AND OFFLINE EVOLUTIONARY FEDERATED NAS

Methods	RT-FedEvoNAS	Offline
Avg Time consumption per generation	2min	162min
Avg Client Downloads per generation	31.32M	715.98M
Avg Client uploads per generation	6.19M	715.98M

also incurs extra computation resources and communication costs.

In the experiments, the number of communication rounds for training one submodel in the offline approach is defined to be 10 and the total number of generations is set to 20. In the above setting, it appears that the maximum number of generations of the offline method is much smaller than that of the real-time method, however, the actual total number of communication rounds for the offline search is $10 \times 10 \times 20 = 2000$ (ten individuals in each population, ten communication rounds for each individual, and the evolution is run for 20 generations), which is much larger than the 300 rounds used in RT-FedEvoNAS.

The Pareto-optimal solutions found over the generations by RT-FedEvoNAS and the offline method in the last generation are plotted in Fig. 12. It is clear to see that the Pareto solutions found by the real-time method at generation 100 (also the 100th communication round) dominate the final solutions obtained by the offline method (after a total of 2000 communication rounds). This also explains why the Pareto-optimal solutions found by the offline method require to be retrained from scratch to enhance their performances, while the models found by RT-FedEvoNAS can be directly used without retraining during the search.

The runtime and communication costs per generation (excluding the retraining time in the offline approach) are listed in Table II. The average runtime per generation of the offline method is approximately 81 times larger than that of RT-FedEvoNAS. Besides, the average numbers of model parameters downloaded and uploaded per generation for each client are both 715.98M for the offline approach, which are 23 times and 116 times more than those for RT-FedEvoNAS. Therefore, the proposed RT-FedEvoNAS consumes much less computation and communication resources compared to the offline federated NAS method.

F. Further Comparative Studies

To further verify the performance of RT-FedEvoNAS, we performed several additional comparative studies and all the results are presented in the supplementary document. We first compare RT-FedEvoNAS with three federated NAS algorithms, namely, FedNAS [79], a state-of-the-art gradient-based algorithm FedNAS, a variant of RT-FedEvoNAS that uses submodel sampling only, and a federated NAS using random search. The comparative results are presented in Sections IV–VI of the supplementary document, respectively. In addition, the comparative results with the centralized NAS, a baseline algorithm under the IID setting [1], [80], are provided in Section VII. Finally, we compare the proposed algorithm with two federated learning algorithms, FedAvg and FedProx [81], and the results are given in Section XI of the supplementary document.

V. CONCLUSION AND FUTURE WORK

This article proposes a real-time multiobjective evolutionary method for federated NAS, which can effectively avoid extra communication costs and computational resources and maintain a stable performance of the models during the optimization. This is achieved by a double-sampling strategy that samples submodels from a global model shared by all individuals in the same population and samples the participating clients for training submodels. This way, one generation of evolutionary optimization can be embedded and completed within one single communication round, thereby reducing both communication and computation costs.

Extensive experiments are performed to compare the proposed RT-FedEvoNAS with five baseline models and the offline version of the evolutionary NAS method on three widely used datasets. These results show that the solutions obtained by the proposed method exhibit better or comparable classification performance than the ResNet in the federated learning framework with a much lower computational complexity, thereby significantly reducing the communication costs. In addition, the solutions obtained by RT-FedEvoNAS significantly outperform all compared models having a lower complexity. We show that compared to the conventional offline evolutionary method, the proposed real-time method incurs much less computational and communication costs. Overall, the proposed algorithm offers a realistic federated evolutionary NAS approach suited to real-time applications.

The present work is a first valuable step toward the application of NAS to the federated learning framework. Despite the encouraging empirical results we have achieved, we note that the classification performance of models obtained by the federated evolutionary NAS method is clearly worse than that of the models obtained by the centralized evolutionary NAS algorithm. In the future, we are going to develop federated evolutionary NAS techniques that can further enhance the classification performance without significantly increasing computational and communication costs. In addition, we are going to verify and extend the proposed algorithm for real-time NAS in large-scale federated learning systems. Finally,

new techniques remain to be investigated to deal with data that are vertically partitioned and distributed on the clients.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat.*, 2017, pp. 1273–1282.

[2] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, Monticello, IL, USA, 2015, pp. 1310–1321.

[3] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.

[4] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," 2018. [Online]. Available: arXiv:1812.07210.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015. [Online]. Available: arXiv:1510.00149.

[6] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation," 2019. [Online]. Available: arXiv:1903.07424.

[7] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1310–1322, Apr. 2020.

[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16. [Online]. Available: OpenReview.net

[9] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80. Stockholm Sweden, Jul. 2018, pp. 4095–4104.

[10] M. Khodak, L. Li, N. Roberts, M.-F. Balcan, and A. Talwalkar, "Weight-sharing beyond neural architecture search: Efficient feature map selection and federated hyperparameter tuning," in *Proc. 2nd SysML Conf.*, Palo Alto, CA, USA, 2019.

[11] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 497–504.

[12] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[13] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn. Vol. 70*, 2017, pp. 2902–2911. [Online]. Available: JMLR.org

[14] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[15] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[16] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 1761–1770.

[17] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020. doi: 10.1109/TEVC.2019.2924461.

[18] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 2820–2828.

[19] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.

[20] Z. Lu *et al.*, "NSGA-net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[21] M. Ammad-Ud Din *et al.*, "Federated collaborative filtering for privacy-preserving personalized recommendation system," 2019. [Online]. Available: arXiv:1901.09888.

[22] L. Bottou, "Stochastic gradient learning in neural networks," *Proc. Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[23] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258..

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2012, pp. 1097–1105.

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

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 2818–2826.

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 4700–4708.

[28] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4278–4284.

[29] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.

[30] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

[31] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016. [Online]. Available: arXiv:1611.02167.

[32] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 2423–2432.

[33] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[34] Y. Guo *et al.*, "Breaking the curse of space explosion: Towards efficient NAS with curriculum search," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 3822–3831.

[35] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[36] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Proc. NeurIPS Meta-Learn. Workshop*, 2017.

[37] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794.

[38] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," in *Proc. Int. Conf. Mech. Learn. (ICML)*, 2018, pp. 677–686.

[39] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[40] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 35–51.

[41] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.

[42] Y. Zhou, G. G. Yen, and Z. Yi, "Evolutionary shallowing deep neural networks at block levels," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 26, 2021, doi: 10.1109/TNNLS.2021.3059529.

[43] C. Qian, Y. Yu, and Z.-H. Zhou, "Pareto ensemble pruning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 29, 2015, pp. 2935–2941.

[44] Z. Yang *et al.*, "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 1826–1835.

[45] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

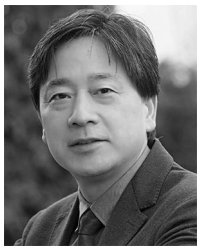
[46] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[47] M. A. Trick, "A linear relaxation heuristic for the generalized assignment problem," *Naval Res. Logist.*, vol. 39, no. 2, pp. 137–151, 1992.

[48] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. Int. Conf. Comput. Stat. (COMPSTAT)*, 2010, pp. 177–186.

[49] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," 2019. [Online]. Available: arXiv:1904.00420.
[50] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[51] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2018, pp. 2016–2025.
[52] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019. [Online]. Available: arXiv:1902.07638.
[53] M. Lindauer and F. Hutter, "Best practices for scientific research on neural architecture search," 2019. [Online]. Available: arXiv:1909.02453.
[54] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[55] A. Zela, J. Siems, and F. Hutter, "NAS-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search," 2020. [Online]. Available: arXiv:2001.10422.
[56] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Hoboken, NJ, USA: Wiley, 2001.
[57] Y. Jin, Ed., *Multi-Objective Machine Learning*. Berlin, Germany: Springer, 2006.
[58] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, 2000, pp. 849–858.
[59] P. Ren *et al.*, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, 2021.
[60] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: arXiv:1503.02531.
[61] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, Jul. 2015, pp. 448–456.
[62] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.
[63] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 1251–1258.
[64] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: arXiv:1704.04861.
[65] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 5927–5935.
[66] C. A. C. Coello, S. G. Brambila, J. F. Gamboa, M. G. C. Tapia, and R. Gómez, "Evolutionary multiobjective optimization: Open research areas and some challenges lying ahead," *Complex Intell. Syst.*, vol. 6, pp. 221–236, Jul. 2020.
[67] G. Yu, Y. Jin, and M. Olhofer, "Benchmark problems and performance indicators for search of knee points in multiobjective optimization," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3531–3544, Aug. 2020.
[68] G. Yu, Y. Jin, and M. Olhofer, "A method for a posteriori identification of knee points based on solution density," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Rio de Janeiro, Brazil, Jul. 2018, pp. 1–8.
[69] X. Zhang, Y. Tian, and Y. Jin, "A knee point-driven evolutionary algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 6, pp. 761–776, Dec. 2015.
[]
[70] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.
[71] J. Yang, R. Shi, and B. Ni, "MedMNIST classification decathlon: A lightweight AutoML benchmark for medical image analysis," 2020. [Online]. Available: arXiv:2010.14925.
[72] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 248–255.
[73] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017. [Online]. Available: arXiv:1712.04621.
[74] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," 2018. [Online]. Available: arXiv:1812.08928.
[75] X. Chu, B. Zhang, R. Xu, and J. Li, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," 2019. [Online]. Available: arXiv:1907.01845.
[76] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861.
[77] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.
[78] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.
[79] C. He, M. Annavaram, and S. Avestimehr, "FedNAS: Federated deep learning via neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshop Beyond Neural Archit. Search*, 2020.
[80] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Red Hook, NY, USA: Curran Associates, Inc., 2020, pp. 14068–14080.
[81] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2018. [Online]. Available: arXiv:1812.06127.

Hangyu Zhu received the B.Sc. degree from Yangzhou University, Yangzhou, China, in 2015, and the M.Sc. degree from RMIT University, Melbourne, VIC, Australia, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Surrey, Guildford, U.K., working on evolutionary neural architecture search for federated learning.



Yaochu Jin (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr University Bochum, Germany, in 2001.

He is currently a Distinguished Chair Professor of Computational Intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K., where he heads the Nature Inspired Computing and Engineering Group. He was a Finland Distinguished Professor, Finland, and a Changjiang Distinguished Visiting Professor, China. He has (co)authored over 300 peer-reviewed journal and conference papers and been granted eight patents on evolutionary optimization. His research interests lie primarily in the cross-disciplinary areas of computational intelligence, computational neuroscience, and computational systems biology. He is also particularly interested in the application of nature-inspired algorithms to solving real-world optimization, learning, and self-organization problems.

Prof. Jin is a recipient of the 2015, 2017, and 2020 *IEEE Computational Intelligence Magazine* Outstanding Paper Award, and the 2018 and 2021 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award. He was awarded the Alexander von Humboldt Professorship for Artificial Intelligence in 2021. He is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and *Complex & Intelligent Systems*. He was an IEEE Distinguished Lecturer from 2017 to 2019 and the Vice President for Technical Activities of the IEEE Computational Intelligence Society from 2014 to 2015. He is a member of Academia Europaea.