Resource-Aware Federated Neural Architecture Search over Heterogeneous Mobile Devices

Jinliang Yuan, Mengwei Xu, Yuxin Zhao, Kaigui Bian, Gang Huang, Xuanzhe Liu, and Shangguang Wang

Abstract—Federated learning has been recently proposed for many clients to collaboratively train a machine learning model in a privacy-preserving manner. However, it also amplifies the difficulty in designing good neural network architecture, especially considering heterogeneous mobile devices. To this end, we propose a novel neural architecture search algorithm, namely `FedNAS`, which can automatically generate a set of optimal models under federated settings. The main idea is to decouple the two primary steps of the NAS process, i.e, model search and model training, and separately distribute them on the cloud and devices. It also tackles the primary challenge of limited on-device computational and communication resources through its novel designs: `FedNAS` fully exploits the key opportunity of insufficient model candidate re-training during the architecture search process and incorporates three key optimizations: parallel candidate training on partial clients, early dropping candidates with inferior performance, and dynamic round numbers. Evaluated on typical CNN architectures and large-scale datasets, `FedNAS` is able to achieve comparable model accuracy as a state-of-the-art NAS algorithm that trains models with centralized data, and also reduces the client cost by up to 200× or more compared to a straightforward design of federated NAS.

Index Terms—Federated learning, neural architecture search, heterogeneous devices, resource-constrained.

1 Introduction

Attentions have been recently put onto the privacy concerns in the machine learning pipeline, especially the deep neural networks (DNNs), which are increasingly adopted on mobile devices [1], [2] and often require a large amount of sensitive data (e.g., images and input corpus) from mobile users to train. The recent releases of General Data Protection Regulation (GDPR) [3], California Consumer Privacy Act (CCPA) [4] and Personal Information Protection Law (PIPL) [5], repeactively by European Union, America and China, strictly regulates whether and how companies can access the personal data owned by their users. In parallel, a lot of efforts have been made in the research community to design novel paradigm of machine learning and large-scale data mining that preserves the privacy of end users. One promising direction is the emerging federated learning [6], which aims to train DNN models in a distributed way, collaboratively from the contribution of many client devices, without gathering the private data from individual devices to the cloud.

Training neural network models on decentralized data and devices addresses the privacy issue to some extent at the expense of efficiency. Once a neural network architecture is determined, there are newly developed methods to speed up the decentralized training process [7], [6]. However, in most tasks when the network architecture is not determined a priori, it remains very difficult to search for the optimal network architecture(s) and train them efficiently in a decentralized setup. Indeed, it is well known that designing an efficient architecture is a labor-intensive process that may require a vast number of iterations of training attempts, which could become uninhibitedly time-consuming given decentralized training data. Making it worse, the hardware platforms on mobile devices are highly heterogeneous [8], thus different network architectures are required to manage diverse resource budgets on the hardware. This becomes a bottleneck of practically deploying federated learning, given the increasingly important role of neural architecture search (or NAS) in launching deep learning in reality.

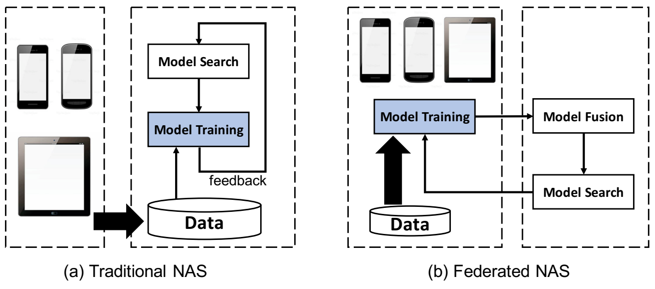


Fig. 1: A high-level comparison between traditional NAS and federated NAS.

To address this major challenge, this paper proposes a new paradigm for DNN training to enable automatic neural architecture search (NAS) on decentralized data, called federated NAS. The major goal is to address both automation and privacy issues while training DNNs with heterogeneous mobile devices. As shown in Figure 1, the basic guiding idea of federated NAS is to decouple the two primary logic steps of NAS process, i.e., model search and model training, and separately distribute them on cloud and clients. Specifically, every single client uses only its local dataset to train and

- *Jinliang Yuan, Mengwei Xu and Shangguang Wang are with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China.*
- *Xuanzhe Liu, Gang Huang, Kaigui Bian and Yuxin Zhao are with Peking University, Beijing, China.*

test a model, while the cloud coordinates all the clients and determines the searching direction without requiring the raw data.

Given the preceding conceptual principles, enabling neural architecture search in a federated setting is fundamentally challenging due to limited on-client hardware resources, i.e., computation and communication. NAS is known to be computation-intensive (e.g., thousands of GPU-hrs [9]), given the large number of model candidates to be explored. Meanwhile, the communication cost between cloud and clients also scales up with the increased number of model candidates. It is also worth mentioning that in the federated NAS paradigm, the data distribution among clients are often non-iid and highly-skewed [6], which can probably mislead the NAS algorithm to select non-optimal DNN candidate. While a few recent studies [10], [11], [12] have preliminarily explored this direction, they mostly target cross-silo scenarios thus ignore the key challenge of high computational and communication loads on devices.

In this work, we present a federated NAS framework, namely *FedNAS*, that is highly optimized for heterogeneous mobile devices. FedNAS starts from an expensive pretrained model, and iteratively adapts the model to a more compact one until it meets a user-specified resource budget. For each iteration, FedNAS generates a list of pruned model candidates, then re-trains (tunes) and tests them collaboratively across the cloud and clients. The most accurate one will be selected before moving to the next iteration. When terminated, FedNAS outputs a sequence of simplified DNN architectures that form the efficient frontier that strikes a balance on the trade-off of model accuracy and resource consumption.

By learning and retrofitting the idea of using proxy task as insufficient candidate re-training from the previous work [13], [14], [15], FedNAS provides several insightful mechanisms, i.e., the parallel tuning of each DNN candidate (across clients), dynamic (across time), and heterogeneous (across models), to make federated NAS practical. **(1)** By parallel tuning, FedNAS works on different model candidates simultaneously and recognizes the available clients into many *groups*. All clients in a group collaboratively train and test a DNN candidate with their results (accuracy, gradients, etc) uploaded and properly fused on the cloud. Different groups work on different DNN candidates in parallel to increase the scalability by involving more available clients. To ensure the generality of each DNN candidate, FedNAS incorporates a principled client partition algorithm with regard to each client' data distribution and data size. **(2)** By dynamic training, FedNAS increasingly trains each candidate with more rounds as iterations go on, instead of using a fixed and large round number as prior NAS works [13]. This is based on the observation that as the model being simplified to smaller, each DNN candidate requires more re-training to regain the accuracy so that FedNAS can adapt the model at the right direction. **(3)** By model heterogeneity, FedNAS early drops the non-optimal candidates during the re-training stages (e.g., 2 rounds), but only the optimal one is trained for the required round number (e.g., 10). This is based on the observation that the optimal DNN candidate often quickly outperforms others even far before the re-training is done.

We comprehensively evaluated the performance of FedNAS on two datasets, ImageNet (iid) and Celeba (non-iid), as well as two CNN architectures, i.e., MobileNet and simplified AlexNet. The results show that FedNAS achieves similar model accuracy as state-of-the-art NAS algorithm that trains models on centralized data, and the three novel optimizations above can reduce the client cost by up to two orders of magnitude, e.g., 277× for computation time and 281× for bandwidth usage. FedNAS also provides flexible trade-offs between the generated model accuracy and the client cost.

In summary, the main contributions of this paper are:

- We propose federated neural architecture search, a novel paradigm to automate the generation of DNN models on decentralized data.
- We present FedNAS, a practical framework that enables efficient federated NAS. The core of FedNAS is to fully leverage the insufficient candidate tuning, an intrinsic NAS characteristic, and incorporate key optimizations to reduce on-client overhead.
- We evaluate FedNAS with extensive experiments. Results show that FedNAS is able to generate a sequence of models under different resource budgets with as high accuracy as traditional NAS algorithm without centralized data, and significantly reduce computational and communication cost on clients compared to straightforward federated NAS designs.

2 RELATED WORK

Neural architecture search (NAS) Designing neural networks is a labor-intensive process that requires a large amount of trial and error by experts. To address this problem, there is growing interest in automating the search for good neural network architectures. 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) [14], [9], [16], [17], [18]. In recent years, with more attention on deploying neural networks on heterogeneous platforms, researchers have been developing NAS algorithms [15], [19], [13], [20], [21] to automate model simplifications. The goal is to generate a sequence of simplified models from an expensive one with the best accuracy under corresponding resource budgets, i.e., the pareto frontier of accuracy-computation trade-off. FedNAS is motivated and based on those prior efforts. Despite the remarkable results, conventional NAS algorithms are prohibitively computation-intensive. The main bottleneck is the training of a large number of model candidates, which often takes up to thousands of GPU hours [22]. Recent work has explored weight sharing across models through a hypernetwork [17], [23] or an over-parameterized one-shot model [24], [25] to amortize the cost of training. Those methods, however, target at generating only one model and often break the high parallelism of NAS, making them not suitable to our target scenario, i.e., generating multiple models under different resource budgets in federated settings. A few efforts have proposed distributed systems [26], [27] for automated machine learning tasks. Such work assume the training data is centralized on cloud instead of decentralized on clients. As a comparison, we face some unique challenges: data distributions are non-iid and highly-skewed, client devices are resource-constrained, etc.

Network pruning aims to effectively reduce the computation cost by removing redundant weights in DNN models. Previous work are generally including (1) unstructured pruning [28] and (2) structured pruning [29]. It is generally hard for unstructured pruning to get the desired speedup and loss of accuracy, due to the irregular and high sparsity. Structured pruning enables significant acceleration with a decent accuracy by removing entire filters/channels of weights. It helps to reconstruct the pruned model to a small dense model [29], [30], [31]. In this work, we integrate channel pruning to find personalized compact DNN models with limited resource constraints.

Federated learning (FL) [6], [32] is a distributed machine learning approach to enabling the training on a large corpus of decentralized data residing on devices like smartphones. By decentralization, FL addresses the fundamental problems of privacy, ownership, and data locality. Though our proposed `FedNAS` approach borrows the spirits from FL, all existing FL research focus on training one specific model instead of the end-to-end procedure of automatic architecture search. As a result, their designs miss important optimizations from intrinsic characteristics of NAS such as early dropping out model candidates (more details in Section 3.3). Further enhancements to FL, e.g., improving the privacy guarantees by homomorphic encryption [33], [34], differential privacy [35] and secure aggregation [36], reducing the communication cost among cloud and clients through weights compression [7], [6], are complementary and orthogonal to `FedNAS`.

Federated NAS There are a few related efforts as `FedNAS` that target federating NAS. Most of those work [10], [11], [12] focus on cross-silo scenarios, e.g., sharing financial data among banks. Those approaches cannot apply to our cross-device scenarios where on-device computation and communication resources are much more limited. A concurrent work [37] proposes an evolutionary approach to real-time federated NAS, while our proposal is prunning-based. Overall, the research of federated NAS is still at very preliminary stage where only small-scale datasets are tested, e.g., MNIST and CIFAR10. This work presents a more through study including the highly-optimized design and comprehensive evaluation with ImageNet dataset.

3 METHODOLOGY

This section formally defines our target problem, conceptualizes the federated NAS idea, and identifies the key challenges and opportunities to achieve efficient federated NAS.

3.1 Problem Statement

Objective Intuitively, the goal of our proposed federated NAS framework is to provide optimal models to run on heterogeneous mobile devices in an *automatic* and *privacy-preserving* way. For automation, the framework can begin with a well-known network architecture, e.g., MobileNet, and generate a sequence of simplified models under different resource budgets without any developers' manual

Notation	Definitions or Descriptions
T	Cloud loops for different decayed resource budgets
R	Cloud loops for fusing the gradients from clients
short-term fine-tune	Insufficient re-training of model candidates during neural architecture search without convergence
long-term fine-tune	Sufficient re-training at the end of whole search process
GM	Global model maintained by cloud that achieves best accuracy under certain resource budget
PM	DNN candidate that is simplified from a GM
$\mathcal{P}$	A partition that partition all clients into *groups*
G	A group of available clients
$test\_num$	The number of testing data points on client
acc	The test accuracy of each DNN candidate on client
$train\_num$	The number of training data points on client
$grad$	The gradients of each DNN candidate on client
$Res(M)$	The consumed resource budget by model M
$Acc(M)$	The obtained accuracy of model M
B	The total resource budget

TABLE 1: Terminologies and symbols

efforts. To preserving privacy , the framework requires no training data (e.g., input corpus, images) to be uploaded to a centralized cloud or shared among devices. Such application scenarios are abounding: next-word prediction [35], [38], speech keyword spotting [39], image classification [40], etc. As traditional NAS frameworks do, the goal of federated NAS can be formulated as following:

$$\begin{aligned} \arg\max_{M} \quad & Acc(M) \\ \textit{subject to} \quad & Res_i(M) \leq B_i,\ i = 1, 2...n \end{aligned}$$

where M is a simplified DNN model, n is the number of resource types, $Acc()$ computes the accuracy, $Res_i()$ evaluates the resource consumption of the i^{th} resource type, and B_i is the budget of the i^{th} resource and the constraint on the optimization. The resource type can be computational cost, latency, energy, memory footprint, etc., or a combination of these metrics. The main terminologies and symbols used in this work are summarized in Table 1.

Contributors A typical federated setting assumes that there are substantial distributed devices available for training, e.g., tens of thousands [41]. A device can be a smartphone, a tablet, or even an IoT gadget depending on the target scenario. Each device contains a small number of data samples locally, and limited hardware resources (e.g., computational capacity and network bandwidth).

3.2 Simplified Workflow of Federated NAS

Intuitively, any NAS algorithm can be leveraged to work on decentralized data. We base our approach on one of the state-of-the-art: NetAdapt [13]. Besides its superior performance as reported, it has another advantage: NetAdapt generates multiple DNN candidates for each iteration, and selects one of them based on their performance. Those candidates can be trained and tested in parallel without any dependency. Indeed, this suits well into the federated setting where lots of devices run independently.

Figure 2 shows the workflow of NetAdapt (with only left part of the figure) and its federated version (the whole figure). Generally speaking, NetAdapt iterates over monotonically decreasing resources budgets, for each of which it

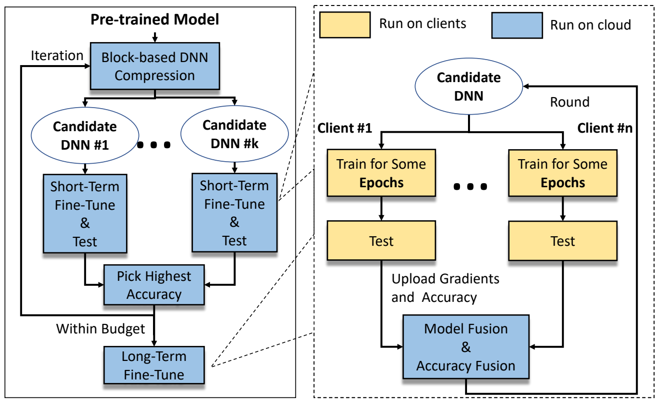


Fig. 2: Simplified workflow of federated NAS and its relationship between original NetAdapt [13]. The left part (inside of the solid box) represents the workflow of original NetAdapt [13] and the right part (inside of the dash box) represents the federated extension of FedNAS.

generates multiple compressed DNN candidates, fine-tunes each candidate, and picks the optimal one with highest accuracy. Finally it performs a long-term fine-tune on the optimal models to convergence. To enable NetAdapt to run on decentralized data, i.e., under federated settings, we can simply replace the training (both short-term and long-term fine-tune) and testing part with a FL-like process, in which a model will be trained at available clients and fused at cloud for many rounds. Section 4 will present more details of how federated NAS works.

3.3 Challenges and Key Optimizations

The major challenge of federated NAS is the heavy on-client computational and communication cost. Consequently, the end-to-end process of federated NAS can be excessively time-consuming. Taking communication cost for short-term fine-tune as an example, the total uplink bandwidth usage can be roughly estimated as

$$\sum_i \sum_j (grad\_size(M_{i,j}) \cdot client\_num(M_{i,j}) \cdot round\_num),$$

where $grad\_size()$ calculates the model gradient size, $client\_num()$ is the number of clients involving training $M_{i,j}$, i and j iterate over all resource budgets (depending on the developer configurations) and DNN candidates (depending on the model architecture), respectively. Communication cost is known to be a major bottleneck in federated learning [6], and it will be further amplified by the large number of DNN candidates and resource budgets to be explored during NAS.

We identify the key opportunity as *how sufficient shall each DNN candidate be tuned during the search process*. While some work realized the tuning can be short-term without getting converged, but they did not explore how long such a process is sufficient. By our study, we find that the tuning of each DNN candidate can be parallel (across clients), dynamic (across time), heterogeneous (across models). In the following sections, we introduce three key optimizations

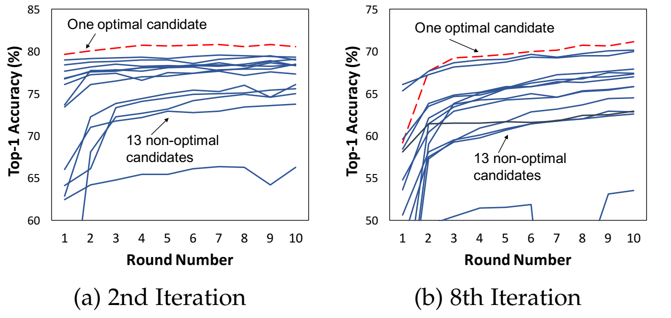


Fig. 3: The accuracy of each DNN candidate as training goes on with more rounds. Each line represents one DNN candidate, and the red dashed one is the optimal one that shall be picked. Dataset: ImageNet, model: MobileNet.

provided by federated NAS, where the first one is to reduce $client\_num$, and the other two are to reduce $round\_num$.

Training candidates on partial clients in parallel One opportunity of speeding up federated NAS comes from the huge amount of client devices that can participate in the training process. In common FL setting, a device is available when it is idle, charged, under unmetered network (e.g., WiFi), and so on. As reported by Google [41], tens of thousands of devices are available for FL at the same time. However, only hundreds of devices can be efficiently utilized in parallel due to the limitation of the state-of-the-art gradients fusion algorithm (e.g., FedAvg). By training and testing DNN candidates on separated groups of clients, we not only reduce the average computational and communication cost of each candidate, but also scale out better with the large number of available clients. It motivates us to organize all clients into a two-level hierarchy for high parallelism (more details in Section 4.2).

Dynamic round number We studied the performance of different DNN candidates at different resource iterations. As illustrated in Figure 3, each line represents the accuracy (y-axis) of a candidate with different training rounds (x-axis), and the red dashed one is the optimal candidate to be picked as it achieves the highest accuracy after all rounds of training done. By comparing the two subfigures, we find that at early iterations the DNN candidates, especially those with higher accuracy, reach stable condition much earlier than later iterations. This is because our algorithm starts with a pre-trained model: as it proceeds the impacts from inefficient tuning accumulate and the model parameters become more and more random. Such insight motivates us towards using dynamic round numbers, e.g., a smaller one for early iterations and keep increasing the number in later stages. While the round number becomes larger, it is worth noting that the model complexity ($grad\_size(M)$) decreases as the algorithm proceeds. It makes the optimization quite effective in reducing clients' overheads.

Early dropping out non-optimal candidates Figure 3 also shows that the optimal candidate (the red dashed line) quickly outperforms others within 1∼3 rounds. It guides us to another optimization: early dropping the candidates while only keeping the optimal one being trained with more rounds. Noting that though optimal candidate has been already picked within several rounds, it still needs to go through more rounds of training. Otherwise the model accuracy will quickly drop to very low and thus misleading the candidate selection afterwards, as confirmed by our experiments in Section 5.4.

In next section we will introduce the details of our federated NAS framework, namely `FedNAS`, which incorporates the aforementioned optimization techniques.

4 THE FEDNAS SYSTEM

Overview The pseudo code of `FedNAS`'s workflow is shown in Algorithm 4.1. `FedNAS` maintains a model called *global model* (GM) among cloud and clients, which starts with an expensive one and will be iteratively adapted until it meets the required resource budget. The network architecture of the initial global model (GM_0) is given by the developers, e.g., MobileNet. It can be either a pre-trained model or actively trained through federated learning as part of `FedNAS`. The goal of each iteration (line 2–11) is to adapt the global model to a smaller one through the cooperation between cloud and clients, i.e., under the budget of $Res(GM_t) - \Delta B_t$ where ΔB_t indicates how much the constraint tightens for the t^{th} iteration (a similar concept of learning rate) and can vary from iteration to iteration. The algorithm terminates when the final resource budget is satisfied. `FedNAS` outputs the final adapted model and also generates a sequence of simplified models at intermediate iterations (i.e., the highest accuracy network picked at each iteration $\langle GM_1, ..., GM_T \rangle$) that form the efficient frontier of accuracy to resource trade-offs.

The t^{th} iteration begins with generating a set of pruned models (PMs) as candidates based on GM_{t-1} (§4.1). Each PM will be scheduled to a group of clients (§4.2), on which the model will be repeatedly i) downloaded to each client within the group (line 14–24); ii) trained and tested via the local dataset on that client (line 34–38); iii) collected to cloud and fused into a new model for many rounds (line 27–32). During this process, all PMs except the optimal one will dropped out (line 25–26, §4.3). This picked PM represents the most accurate model under the current resource budget, thus making it the next global model GM_t (line 10). Finally, the cloud performs federated learning on the GM_T or other GMs as specified by developers till convergence (line 12, §4.4).

4.1 Model Pruning

`FedNAS` adapts a GM based on standard pruning approaches. More specifically, `FedNAS` reduces the number of filters in a single CONV (convolutional) or FC (fully-connected) layers to meet the resource budget of current iteration, as CONV and FC are known to be the computationally dominant layers in most NN architectures [42]. To choose which filters to prune, `FedNAS` computes the $\ell 2$-norm magnitude of each filter and the one with smallest value will be pruned first. More advanced methods can be adopted to replace the magnitude-based method, such as removing the filters based on their joint influence on the feature maps [43].

By adapting, `FedNAS` generates K pruned model candidates PMs, where K equals to the sum of CONV and FC layer numbers, e.g., 14 for MobileNet. For larger models, we can also speed up the adaptation process by treating a group of multiple layers as a single unit (instead of a single layer), e.g., residual block in ResNet [44].

```
input : $GM_0$, $B$, $T$, $R$, candidate drop ratio $\alpha$
output: a list of models ($GM_1, ...GM_T$) under different
        resource budgets ($B_1, ..., B_T$ where $B_T = B$)
1  Function Cloud_Operation(): ▷run on cloud
2      for $t \leftarrow 1$ to $T$ do
3          $B_t \leftarrow$ next resource budget
4          $R \leftarrow$ round number at current iteration
5          $\mathcal{P} \leftarrow$ partition all available clients
6          $PMs \leftarrow$ generate a list of simplified models by
             pruning different layers of $GM_{t-1}$
7          for $r \leftarrow 1$ to $R$ do
8              $PMs \leftarrow Cloud\_One\_Round(PMs, groups)$
9          $GM_t \leftarrow PMs[0]$ with the highest $Acc(PM)$
10     perform FL on $GM_T$ or other $GMs$ as user specify
11 Function Cloud_One_Round($PMs$, $groups$):
12     for each $PM_i$ in $PMs$ in parallel do
13         $G \leftarrow$ wait to get a free group from $groups$
14         lock($G$)
15         for each client $C$ in $G$ in parallel do
16             send $PM_i$ to $C$
17             invoke $C.Client\_Operation(PM_i)$
18             collect $acc, test\_num$ from $C$
19         $Acc(PM_i) \leftarrow$ fuse collected $acc$ with $test\_num$
20         unlock($G$)
21     $drop\_num \leftarrow max(\alpha \cdot PMs.len(), PMs.len() - 1)$
22     Sort $PMs$ in reverse order of $Acc(PM_i)$ and
         remove the lowest $drop\_num$ models
23     for each $PM_i$ in $PMs$ in parallel do
24         $G \leftarrow$ the group that runs $PM_i$
25         collect $grad, train\_num$ from all clients in $G$
26         $fused\_grad_i \leftarrow$ fuse all $grad$ with $train\_num$
27         $PM_i \leftarrow PM_i + fused\_grad_i$
28     return $PMs$
29 Function Client_Operation($PM$): ▷run on clients
30     split local dataset into training and validation set
31     $grad \leftarrow$ train $PM$ on local training dataset
32     $acc \leftarrow$ test $PM$ on local validation dataset
33     store $grad$, $acc$, $train\_num$, $test\_num$ locally
```

Algorithm 4.1: The proposed `FedNAS` framework

4.2 Clients Partitioning & Scheduling

The goal of this stage is to partition the clients that are available for training into different groups. Each group contains one or multiple clients, and the number of groups ($\mathcal{K}$) is given by developers. The partition starts once the cloud determines which clients are available and their associated information (i.e., data number and data distribution, see below) has been uploaded. Since the availability of clients is dynamic depending on the user behavior and device status, the partition needs to be performed at each iteration. Each PM will be scheduled to one group for training (i.e., short-term fine-tune) and testing.

How to partition A good partition follows two principles. First, the total data number of each group shall be close and balanced. This is to ensure that each PM is tuned and tested on enough data to make the results trustworthy, and also ensure high parallelism without being bottlenecked by large groups. Second, the data distribution of each group shall be representative of the dataset from all clients. Since in federated setting the data owned by each client is often non-iid, a random partition may lead to groups with biased data and makes the resultant accuracy non-representative. In such a case, our algorithm may choose the wrong candidate.

To formulize the two policies above, we denote a partition $\mathcal{P}$ as $\{G_1, ..., G_{\mathcal{K}}\}$, and the total data number within G_i as d_i which is simply summed over the data number of all clients within G_i.

$$\begin{aligned} &\arg\min_{\mathcal{P}} \quad \frac{1}{\mathcal{K}} \sum_{i=1,...,\mathcal{K}} Dist(G_i, G_{all}) \\ &\textit{subject to} \quad \max(d_j) \leq \gamma \cdot \min(d_j), \quad j = 1, ..., \mathcal{K} \end{aligned}$$

Here, $Dist()$ calculates the distance between the data distributions of two groups, G_{all} is an imaginary group including the data from all clients, γ is a configurable variable that controls how unbalanced FedNAS can tolerate about the data sizes across different groups (default: 1.1). This equation can be approximately solved by a greedy algorithm: first sorting all clients by their data number, then iteratively dispatching the largest one to a group so that the data size balance is maintained (i.e., the inequality) while the smallest average distribution distance is achieved.

For classification tasks, which is the focus of this work, FedNAS uses the normalized number of each class type to represent the data distribution, i.e., a vector $v = (v_1, v_2, ..., v_m)$ where v_i equals to the ratio of data numbers labeled with i^{th} class type. The distribution distance is computed as the Manhattan distance between such two vectors. Note that the ratio of different class types can be considered to be less privacy-sensitive compared to the gradients that need to be uploaded for many times, so it shall not compromise the original privacy level of federated setting. Nevertheless, the distribution vectors can be further encrypted through secure multiparty computation [45].

How to schedule Each PM will be scheduled to a random group for training and testing. If all groups are busy, cloud will wait until one has finished and schedule the next PM to this group.

As an important configuration to be set by the developers, the number of groups ($\mathcal{K}$) makes the trade-offs between the quality of neural architecture selection and the computational cost imposed on client devices. A larger $\mathcal{K}$ promises higher parallelism so that the NAS process can be faster, but also means the training and testing data provisioned to each PM is less. Our experiments in Section 5 will dig into such trade-offs and provide useful insights to developers in determining a proper group number.

4.3 Candidate Dropping and Selection

Short-term fine-tune on decentralized data Each PM will be trained and tested on the scheduled group for many rounds, similar to the methodology of federated learning. At each round, every client within the group downloads the newest PM version, then trains (local-tune) and tests the model. The training and testing datasets are both split from the client's local dataset. The local-tune takes multiple epochs to reduce round number and communication cost [6]. The training and testing results, i.e., gradients and accuracy, associated with the dataset size, will be uploaded to cloud. The gradients will be fused to update the model candidate PM on cloud, and the accuracy will be fused as the metric to pick the optimal model candidate after all rounds.

Guided by our finding in Section 3.3, FedNAS reduces the on-client computational and communication cost during short-term fine-tune process through dynamic round number and early dropping candidates. More specifically, FedNAS increasingly trains each candidate with more rounds as iterations go on. For each round, FedNAS collects the local accuracy from clients and fuse them into a weighted accuracy for each PM. The $\alpha\%$ ones with largest accuracy degradation (defined below) will be dropped and no longer tuned. For the rest of the valid candidates, FedNAS collects the gradients from clients and fuses them into a new PM. As round goes on, fewer and fewer candidates need to be tuned and tested. Noting that the accuracy is fused first so that the gradients of the dropped candidates at this round do not need to be uploaded.

The goal of this short-term fine-tune is to regain accuracy of PMs. This step is important while adapting small networks with a large resource reduction because otherwise the accuracy will drop to zero, which can cause FedNAS to choose the wrong model candidate. One main difference between this stage and a standard FL process is that this stage takes relatively smaller number of iterations (i.e., short-term) without requiring the model to converge.

Accuracy fusion and comparison The accuracy generated by each client will be uploaded to the cloud. For a given PM_i and its scheduled group $G \in \mathcal{P}$, once the cloud receives all accuracy of the clients within the same group G, it combines the accuracy into a new one by weighting the testing data numbers on the same client:

$$Acc(PM_i) = \frac{\sum_c (test\_num_c \cdot acc_c)}{\sum_c test\_num_c}, \quad c \in G, \qquad (1)$$

where c denotes a client in this group G, $test\_num_c$ and acc_c are the testing data number and testing accuracy reported by client c.

With the accuracy of all model candidates computed at each round, FedNAS drops the models with largest accuracy degradation. Note that each PM may have different resource consumptions (Section 4.1), we use the ratio of accuracy degradation to the resource consumption reduction over the previous GM_{t-1} (i.e., the unpruned model at the beginning of iteration t):

$$acc\_degradation = \frac{Acc(GM_{t-1}) - Acc(PM_i)}{Res(GM_{t-1}) - Res(PM_i)} \qquad (2)$$

Model fusion For a given PM_i and its scheduled group G, FedNAS fuses the gradients from all clients within G by weighting the training data numbers used in local-tune. It can be formulated as:

$$fused\_grad_i = \frac{\sum_c (train\_num_c \cdot grad_c)}{\sum_c train\_num_c}, \quad c \in G, \qquad (3)$$

and

$$PM_i = PM_i + fused\_grad_i, \qquad (4)$$

where c denotes a client in this group G, $train\_num_c$ and $grad_c$ are the training data number and update gradients reported by client c.

Dataset	Model	Task	Client number	Data per client
ImageNet (*iid*)	MobileNet (13 CONV, 1 FC)	Image classification	1,500	915.0
Celeba (*non-iid*)	Simplified AlexNet (6 CONV, 1 FC)	Face attrs classification	9,343	21.4

TABLE 2: Datasets and models used in experiments.

4.4 FL-tuning

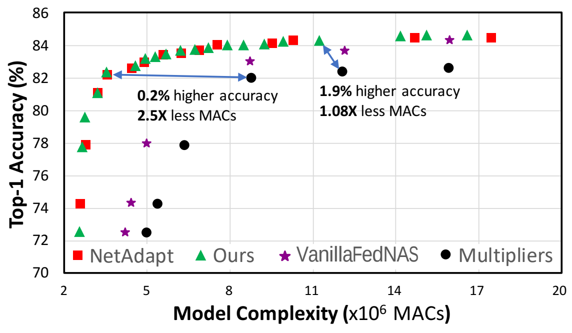
Once FedNAS finishes the model search process above, a sequence of models have been generated, i.e., $GM_1, ..., GM_T$. As the final stage, FedNAS performs a standard federated learning on GM_T or other GMs (called *FL-tune*) if needed by the developer. The goal of this stage is to make the obtained models converge. FedNAS can utilize any existing FL algorithm to run FL-tune and currently it uses one of the state-of-the-art *FedAvg* [6]. When multiple GMs are demanded, FedNAS can still utilize the partitioned clients to train them in parallel.

5 EVALUATION

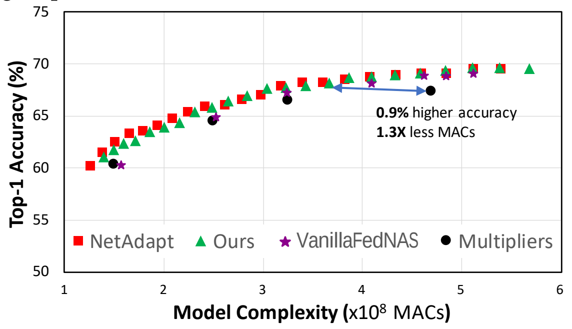
In our experiments, we mainly evaluate three parts of performance: 1) §5.2: does FedNAS generate high accuracy models under different resource budgets? 2) §5.3: what's the computational and communication cost of FedNAS on clients? 3) §5.4: what's the impacts of FedNAS's key designs?

5.1 Experiment Settings

Datasets As shown in Table 2, we tested FedNAS on 2 datasets commonly used for federated learning experiments: ImageNet [46] (iid) and Celeba [47] (non-iid). For ImageNet, we randomly split it into 1,500 clients. For Celeba, we split it into 9,343 clients based on the identities of face images. We re-used the scripts of LEAF [48], a popular federated learning framework, to pre-process Celeba data and generate non-iid data. Each Celeba image is tagged with 40 binary attributes. We randomly select 3 of them (Smiling, Male, Mouth_Slightly_Open) and combine the 3 features into a classification task with 8 classes. The dataset on each client was further split to three parts: training set used for short-term fine-tune, validation set used to test the accuracy of DNN candidates, testing set used to evaluate the final accuracy of each simplified model (6:2:2).
Models We applied FedNAS on two models: MobileNet [49] (for ImageNet, 224x224 input size), a widely used CNN network for mobile applications; A simplified AlexNet, which we call ConvNet (for Celeba, 128x128 input size) with sequential CONV, Pooling, and final FC layers. We did not apply FedNAS on larger networks like ResNet or VGG because small and compact networks are more difficult to simplify; these large networks are also seldom deployed on mobile platforms.
Resource type We mainly used multiply-accumulate operations (MACs) as the metric to specify resource budgets. For MobileNet, we reduce the resource budget by 5% at each iteration with 0.98 decay. For ConvNet, we reduce the resource budget by 5% at each iteration with 0.93 decay.
Alternatives We compare FedNAS with two state-of-the-art automatic network simplification approaches. Note that both of them are performed on centralized data.



(a) Celeba. FedNAS setting: 10 rounds, 4 epochs, 20 groups.



(b) ImageNet. FedNAS setting: 100 rounds, 3 epochs, 50 groups.

Fig. 4: Accuracy comparison among FedNAS and the alternatives with MACs as the target resource type.

- NetAdapt [13] is the basis of FedNAS. We directly reused their open code and kept the original parameter setting.
- VanillaFedNAS [10] is a vanilla federated NAS framework that focuses on improve model accuracy towards Non-IID data distribution in FL, but without considering the heterogeneous resources of constrained mobile devices.
- Multipliers [49] are simple but effective approaches to simplify networks. We used Width Multiplier to scale the number of filters by a percentage across all CONV and FC layers.

Hardware of Cloud and Clients All experiments were carried out on a high-end server with 12× P100 Tesla GPUs. To simulate the client-side computation cost, we used DL4J[1] to obtain the training speed of MobileNet and ConvNet as well as each pruned DNN candidate on Samsung Note 10. The training speed is then plugged into to our experiment platform, as a way to simulate the on-client computation cost. The communication cost is also simulated by recording the data transmission between cloud process and client processes.

5.2 Analysis of Accuracy

Figure 4 shows the comparison of the models generated by FedNAS and other alternatives. Overall, FedNAS achieves similar performance as NetAdapt, and both of them significantly outperform Multipliers. Noting that FedNAS trains models on decentralized data with much

1. https://deeplearning4j.org/

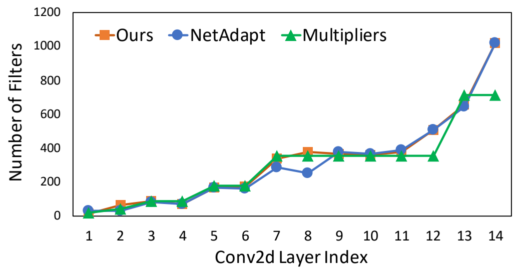
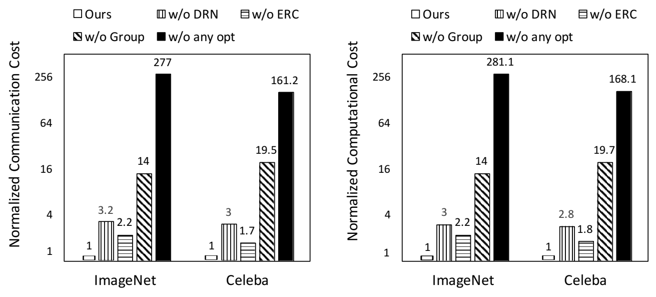


Fig. 5: When adapting MobileNet on ImageNet to 50% MACs, FedNAS and NetAdapt generate similar network architectures, while more different from Multipliers.



(a) Communication cost (b) Computational cost

Fig. 6: The on-client cost reduction brought by our key optimizations ("DRN": dynamic round number; "ERC": early dropping candidates; "Group": parallel training). The setting is the same as Figure 4. The y-axis is logarithmic.

Model	Drop ratio each round	Top-1 Accuracy (%)	Avg uplink cost per client (MBs)
50% ConvNet	0% (no drop)	83.8 (0.0)	59.7 (0.0)
	33% (default)	**83.8 (0.0)**	**12.8 (-79%)**
	50%	83.5 (-0.3)	10.7 (-82%)
	100%	82.1 (-1.7)	8.5 (-85%)
25% ConvNet	0% (no drop)	82.3 (0.0)	138.0 (0.0)
	33% (default)	**82.3 (0.0)**	**29.6 (-77%)**
	50%	81.6 (-0.7)	24.6 (-81%)
	100%	77.8 (-4.5)	19.7 (-88%)
15% ConvNet	0% (no drop)	78.4 (0.0)	209.6 (0.0)
	33% (default)	**78.2 (-0.2)**	**44.9 (-79%)**
	50%	74.1 (-4.1)	37.4 (-81%)
	100%	47.1 (-31.3)	30.0 (-86%)

TABLE 3: The trade-offs from when to drop candidates on the model accuracy and on-client communication (uplink) cost. Dynamic round number is disabled in this experiment.

better user privacy compared with Multipliers. On Celeba, the model generated by FedNAS is up to 2.5× less complex (specified by MACs) with the similar accuracy or 1.9% higher accuracy with the same complexity compared to Multipliers. On ImageNet, the model generated by FedNAS is 1.3× less complex with 0.9% higher accuracy compared to Multipliers. Noting that FedNAS is performed on decentralized data that well preserves user privacy. Compared with VanillaFedNAS, FedNAS achieves similar accuracy, but it takes less complexity to converge. For example on Celeba with Non-IID setting, the model generated by FedNAS is up to 1.8× less complex (specified by MACs) with the similar accuracy compared with VanillaFedNAS. Even on ImageNet with IID setting, the model generated by FedNAS is nearly 1.1× less complex (specified by MACs) with the similar accuracy (e.g., 68%). This significant improvement comes from FedNAS's resource-aware techniques, which can fully exploits the insufficient model candidate.

On ImageNet, we notice an increasing performance gap between FedNAS and NetAdapt while MobileNet is simplified to be smaller architecture. This is because in our current default setting, the short-term fine-tune is conservative to keep the client cost low, so that sometimes the candidate is not sufficiently trained thus misleading the model selection. As we will show later, by varying the system configurations (e.g., round number and group number), the accuracy of FedNAS can be further improved.

We then studied how the network architectures look like when adapting MobileNet to 50% MACs on ImageNet using different approaches. As illustrated in Figure 5, FedNAS generates similar network architecture as NetAdapt but different from Multipliers. This well explains the performance similarity/gap between FedNAS and the alternatives shown above.

5.3 Analysis of Client Cost

We studied how much improvements and trade-offs brought by our key optimizations introduced in Section 3.3. By default, we drop 33% candidates after each round, thus all non-optimal candidates will be dropped after 3 rounds. The dynamic round numbers used are (1-5 iters: 25 rounds; 7-10: 50; 11-15: 75; >15: 100) for ImageNet and (1-5 iters: 2 rounds; 6-10: 5; 11-15: 8; >15: 10) for Celeba. The group numbers for ImageNet/Celeba are 15 and 20, respectively. The settings are consistent with the accuracy experiments in Figure 4. Here we only report the on-client cost for short-term fine-tune during model search, excluding the cost for long-term fine-tune at the last step and the potential federated learning for the initial model. This is because the short-term fine-tune is often more computational intensive, while the latter ones depend on further user specifications, e.g., what models are needed for deployment.

Overall improvements As shown in Figure 6, all three techniques can significantly reduce the on-client cost, i.e., computational and communication. In a naive design of federated NAS with all optimizations disabled, the communication and computational cost are 277× and 281× more on ImageNet, and 161× and 162× more on Celeba, respectively. With one technique disabled, i.e., dynamic round number / early dropping candidates / group hierarchy, the cost can be up to 3.2× / 2.2× / 19.7× more. We observe that the first two optimizations aiming at reducing the round number are more effective at ImageNet. This is because ImageNet task is more complex than Celeba, so the model requires more short-term fine-tuning (round numbers) thus leaves more headroom for optimizations.

Note that, according to the experiments, our optimizations with the default settings have almost *zero* affects at the model accuracy. In fact, Figure 4 shows that FedNAS already achieves the accuracy upper bound defined by NetAdapt. Next, we studied the trade-offs between accuracy and cost from two optimizations (early drop candidates and group hierarchy) by varying the default settings.

Trade-offs from drop round Table 3 shows the trade-offs

Model	Group number	Top-1 Accuracy (%)	Avg uplink cost per client (MBs)
75% MobileNet	10	68.8 (0.0)	723.0 (+400%)
	50 (default)	**68.8 (0.0)**	**140.6 (0.0)**
	100	68.6 (-0.2)	70.3 (-50%)
	200	68.5 (-0.3)	35.2 (-75%)
50% MobileNet	10	67.6 (+0.1)	2181.0 (+400%)
	50 (default)	**67.5 (0.0)**	**436.2 (0.0)**
	100	67.0 (-0.5)	218.1 (-50%)
	200	66.4 (-1.1)	109.1 (-75%)

TABLE 4: The trade-offs from group number on the model accuracy and on-client communication (uplink) cost. All optimizations are enabled in this experiment.

from the timing to drop the non-optimal candidates. The results show that by dropping 33% candidates at each round, FedNAS can reduce the uplink cost by 57% with very little accuracy loss (<0.2%). By more aggressive early dropping, FedNAS further reduces the uplink cost, but sacrifices much more model accuracy. In an extreme case where all non-optimal candidates are dropped immediately before the first round of model fusion (100% drop ratio), the model accuracy degrades by 31.3% when simplifying the ConvNet to 15% complexity. The reason is that with insufficient training (few rounds), the accuracy of candidates are not yet representative of the real performance of the corresponding network architectures, thus leading FedNAS to pick the wrong candidate. The impacts from such misleading accumulate as more iterations go on.

Trade-offs from group number In essence, the group number determines how many clients and data are involved in training each model candidate. As shown in Table 4, with a smaller group number (10) on ImageNet, FedNAS's accuracy doesn't improve much (up to 0.1%) compared to our default setting (50), but incurs much more client cost (e.g., 5$\times$ more uplink network). It confirms our observation as discussed in Section 3.3 that training and testing each model candidate only require partial clients and data to involve. With a relatively larger group number 100, the accuracy drops by 0.2% when adapted to 50% complexity, but the uplink cost is also reduced by 50%. An even larger group number (200) helps reduce the cost by 75% but the accuracy degradation increases up to 1.1%. In a word, the group number provides rich trade-offs between the generated model accuracy and on-client cost. Noting that when the group number is larger than the candidate number, further increasing it doesn't reduce the end-to-end architecture search time because of the dependency between sequential iterations. Due to the limitation of current federated learning platforms, we don't evaluate this architecture search time and leave it as future work.

5.4 Ablation Studies

Impact of short-term fine-tuning Figure 7 shows the model accuracy with different round numbers (without long-term fine-tuning), which demonstrates that short-term fine-tuning can accelerate the search for the optimal network architecture. With a larger round number, it takes little computational cost (e.g., 2.6 $\times$ 10^8 MACs) to get decent model accuracy (e.g., 66%), but increases the convergence time. In an extreme case with zero round number, i.e., all candidates except the optimal one are dropped without model fusion, the accuracy rapidly drops to almost random guess. For example, it takes computation cost up to 4.1×10^8 MACs and only achieved 45% model accuracy. The reason behind is that the algorithm picks the best candidate solely based on noise thus gives poor performance, and the long-term fine-tune cannot save the accuracy because the model architecture is inferior. With a reasonably smaller round number (e.g., 10), it can achieve up to 66% accuracy with the same computation cost (4.1×10^8). Though it is still lower than the default setting (70%), the model accuracy can be largely preserved. It demonstrates that though a small round number is often enough to pick the optimal candidate at current iteration (motivation for early dropping optimization), but still we need more rounds to re-train the picked model before entering into the next round. Otherwise the pruning direction at later iterations will be misled.

Impact of long-term fine-tuning Figure 8 illustrates the importance of performing the long-term fine-tuning using federated learning after global models have been generated. It shows that the short-term fine-tuning can preserve the accuracy well at the beginning, but the accuracy still drops faster as iterations go on due to the accumulation of insufficient training. The long-term fine-tuning can increase the accuracy by up to another 5% at later stages. Though at later iterations the raw accuracy drops faster, FedNAS is still able to pick the good candidate, thus maintains close performance compared to NetAdapt as shown above. Nevertheless, it shows that the training under the default setting has the potential to be further improved by adding more rounds.

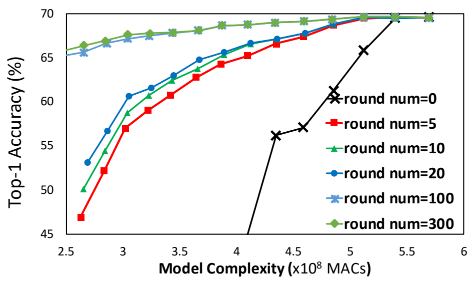


Fig. 7: The impacts of short-term fine-tune (round number) on model accuracy (without long-term fine-tune). Other settings are the same as Figure 4. Model: MobileNet.

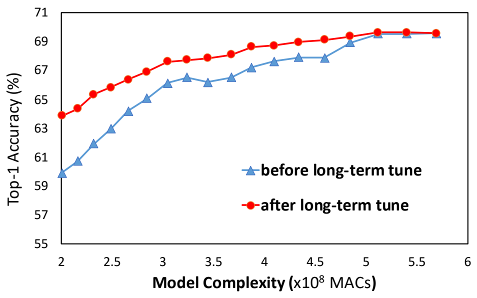


Fig. 8: Long-term fine-tune can substantially increase the generated model accuracy. Model: MobileNet.

6 Conclusion

In this work, we have presented a novel framework, FedNAS, which can automatically generate neural architectures with training data decentralized with a large number of clients. To deal with the heavy cost of on-client computation and communication, FedNAS identifies the key opportunity as insufficient candidate tuning by looking into the NAS intrinsic characteristics, and incorporates three key optimizations. Tested on both iid and non-iid datasets, FedNAS is able to generate neural networks with similar accuracy compared to training on centralized data, but with less computation complexity.

7 Acknowledges

This work was supported by National Key R&D Program of China (NO.2021ZD0113001), NSFC (No.62102045 and No.61922017), Beijing Nova Program (No.Z211100002121118), and Young Elite Scientists Sponsorship Program by CAST (No.2021QNRC001). Kaigui Bian, as one of the authors, was partly supported by the 13th 5-year plan Civil Aerospace Technology Advance Research Project (NO.D030301).

References

[1] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *Proceedings of World Wide Web Conference (WWW)*, pages 2125–2136, 2019.

[2] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2407–2416, 2018.

[3] General data protection regulation (gdpr). https://gdpr-info.eu/, 2019.

[4] California consumer privacy act (ccpa). https://oag.ca.gov/privacy/ccpa, 2018.

[5] Personal information protection law (pipl). https://www.npc.gov.cn, 2021.

[6] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[7] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[8] Ai benchmark. http://ai-benchmark.com/index.html, 2019.

[9] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[10] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Fednas: Federated deep learning via neural architecture search. *arXiv preprint arXiv:2004.08546*, 2020.

[11] Ishika Singh, Haoyi Zhou, Kunlin Yang, Meng Ding, Bill Lin, and Pengtao Xie. Differentially-private federated neural architecture search. *arXiv preprint arXiv:2006.10559*, 2020.

[12] Xinle Liang, Yang Liu, Jiahuan Luo, Yuanqin He, Tianjian Chen, and Qiang Yang. Self-supervised cross-silo federated neural architecture search. *arXiv preprint arXiv:2101.11896*, 2021.

[13] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[14] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[15] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828, 2019.

[16] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Jianyuan Guo, Wei Zhang, Chao Xu, Chunjing Xu, Dacheng Tao, and Chang Xu. Hournas: Extremely fast neural architecture search through an hourglass lens. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10896–10906, 2021.

[17] Yanyu Li, Pu Zhao, Geng Yuan, Xue Lin, Yanzhi Wang, and Xin Chen. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3236–3242, 2022.

[18] Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. Analyzing and mitigating interference in neural architecture search. In *International Conference on Machine Learning (ICML)*, pages 24646–24662, 2022.

[19] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1586–1595, 2018.

[20] Ramtin Hosseini, Xingyi Yang, and Pengtao Xie. DSRNA: differentiable search of robust neural architectures. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6196–6205, 2021.

[21] Edgar Liberis, Lukasz Dudziak, and Nicholas D. Lane. μnas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systemsg Virtual Event (EuroSys)*, pages 70–79, 2021.

[22] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[23] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[24] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 549–558, 2018.

[25] Minghao Chen, Jianlong Fu, and Haibin Ling. One-shot neural ensemble architecture search by diversity-guided search space shrinking. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16530–16539, 2021.

[26] Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. ATM: A distributed, collaborative, scalable system for automated machine learning. In *Proceedings of International Conference on Big Data*, pages 151–162, 2017.

[27] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1487–1495, 2017.

[28] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*, pages 1135–1143, 2015.

[29] Yi Guo, Huan Yuan, Jianchao Tan, Zhangyang Wang, Sen Yang, and Ji Liu. GDP: stabilized neural network pruning via gates with differentiable polarization. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 5219–5230, 2021.

[30] Xiu Su, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Bcnet: Searching for network width with bilaterally coupled network. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2175–2184, 2021.

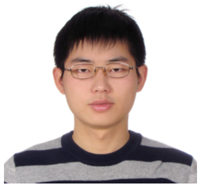
[31] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3977–3986, 2019.

[32] Xiaojin Zhang, Hanlin Gu, Lixin Fan, Kai Chen, and Qiang Yang. No free lunch theorem for security and utility in federated learning. *arXiv preprint arXiv:2203.05816*, 2022.

[33] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-

silo federated learning. In *Proceedings of Annual Technical Conference (ATC)*, pages 493–506, 2020.
[34] Yang Liu, Zhihao Yi, Yan Kang, Yuanqin He, Wenhan Liu, Tianyuan Zou, and Qiang Yang. Defending label inference and backdoor attacks in vertical federated learning. *arXiv preprint arXiv:2112.05409*, 2021.
[35] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.
[36] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of Conference on Computer and Communications Security (CCS)*, pages 1175–1191, 2017.
[37] Hangyu Zhu and Yaochu Jin. Real-time federated evolutionary neural architecture search. *IEEE Transactions on Evolutionary Computation*, pages 364–378, 2022.
[38] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.
[39] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345, 2019.
[40] Yang Liu, Tianjian Chen, and Qiang Yang. Secure federated transfer learning. *arXiv preprint arXiv:1812.03337*, 2018.
[41] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *SysML Conference*, 2019.
[42] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: principled cache for mobile deep vision. In *Proceedings of Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 129–144, 2018.
[43] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5687–5695, 2017.
[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.
[45] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. 2005.
[46] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 248–255, 2009.
[47] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.
[48] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.
[49] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jinliang Yuan is a Ph.D student in the School of Computer Science, Beijing University of Posts and Telecommunication, Beijing, China. His research interests are distributed machine learning, federated learning, and edge computing.



Mengwei Xu is an assistant professor in the computer science department at Beijing University of Posts and Telecommunications. His research interests cover the broad areas of mobile computing, edge computing, artificial intelligence, and system software.

Yuxin Zhao is a master student in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. Her research interests are data mining, artificial intelligence, and mobile computing.

Kaigui Bian is an associate professor in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He received his Ph.D. degree in Computer Engineering from Virginia Tech in 2011, and his B.S. degree in Computer Science from Peking University, Beijing, China in 2005. He was a visiting researcher at Microsoft Research Asia in 2013. He is a senior member of IEEE, and a member of ACM, IET, and CCF. His research interests include networking system, mobile computing, and big data analytics.

Gang Huang is now a full professor in Institute of Software, Peking University. His research interests are in the area of middleware of cloud computing and mobile computing. He is a member of IEEE.

Xuanzhe Liu is an associate professor in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. His research interests are in the area of services computing, mobile computing, web-based systems, and big data analytics.

Shangguang Wang is a Professor at the School of Computer Science and Engineering, Beijing University of Posts and Telecommunications, China. He received his Ph.D. degree at Beijing University of Posts and Telecommunications in 2011. He has published more than 150 papers. His research interests include service computing, mobile edge computing, and satellite computing. He is currently serving as Chair of IEEE Technical Committee on Services Computing (2022-2013), and Vice-Chair of IEEE Technical Committee on Cloud Computing (2020-). He also served as General Chairs or Program Chairs of 10+ IEEE conferences. He is a Fellow of the IET, and Senior Member of the IEEE. For further information on Dr. Wang, please visit: http://www.sguangwang.com