Neural Architecture Search over Decentralized Data

Mengwei Xu
Peking University

Yuxin Zhao
Peking University

Kaigui Bian
Peking University

Gang Huang
Peking University

Qiaozhu Mei
University of Michigan

Xuanzhe Liu
Peking University

ABSTRACT

To preserve user privacy while enabling mobile intelligence, techniques have been proposed to train deep neural networks on decentralized data. However, decentralized training makes the design of neural architecture quite difficult as it already was. Such difficulty is further amplified when designing and deploying different neural architectures for heterogeneous mobile platforms. In this work, we propose an automatic neural architecture search into the decentralized training, as a new DNN training paradigm called Federated Neural Architecture Search, namely federated NAS. To deal with the primary challenge of limited on-client computational and communication resources, we present FedNAS, a highly optimized framework for efficient federated NAS. FedNAS fully exploits the key opportunity of insufficient model candidate re-training during the architecture search process, and incorporates three key optimizations: parallel candidates training on partial clients, early dropping candidates with inferior performance, and dynamic round numbers. Tested on large-scale datasets and typical CNN architectures, FedNAS achieves comparable model accuracy as 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.

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 [29, 31] and often require a large amount of sensitive data (e.g., images and input corpus) from mobile users to train. As one of the many typical examples, the recent release of General Data Protection Regulation (GDPR) [3] by European Union 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 [24], which aims to train DNN models in a decentralized 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 [17, 24]. 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 [1], 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.

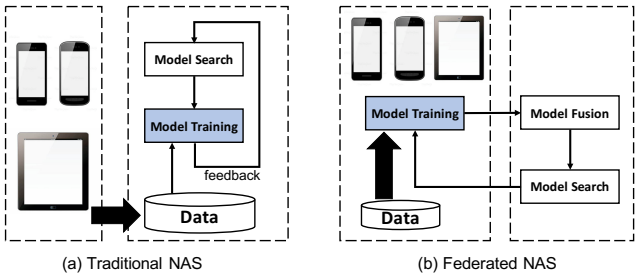


Figure 1: A high-level comparison between traditional centralized NAS and federated NAS.

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 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 [36]), 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 [24], which can probably mislead the NAS algorithm to select non-optimal DNN candidate.

We present the first framework for federated NAS, named FedNAS. FedNAS starts from an expensive pre-trained 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 [8, 20, 28, 34], 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 [34]. 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:

- To our best knowledge, we are the first to 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 FedNASwith 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) [20, 36]. In recent years, with more attention on deploying neural networks on heterogeneous platforms, researchers have been developing NAS algorithms [13, 28, 30, 34] 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*. FedNASis motivated and based on those prior efforts.

Accelerating NAS 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 [37]. As a trade-off, many NAS algorithms [8, 13, 20, 28, 34] propose to search for building blocks on proxy tasks, such as training for fewer epochs, starting with a smaller dataset, or learning with fewer blocks. Our work also utilizes and retrofits such proxy tasks as insufficient tuning during the search process. Recent work has explored weight sharing across models through a hypernetwork [7, 26] or an over-parameterized one-shot model [4, 9] 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., generate multiple models under different resource budgets in federated settings. A few efforts have proposed distributed systems [12, 27, 35] 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.

Federated learning (FL) [24] 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 differential privacy [25] and secure aggregation [6], reducing the communication cost among cloud and clients through weights compression [17, 24], are complementary and orthogonal to FedNAS.

3 METHODOLOGY

We define our target problem and identify the challenges.

3.1 Problem Statement

Objective Intuitively, the goal of our proposed federated NAS framework is to provide optimal models to run on mobile devices

Notation	Definitions or Descriptions
iteration	Cloud loops for different decayed resource budgets (T)
round	Cloud loops for fusing the gradients from different clients (R)
epoch	Client loops for training on local dataset each round (E)
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

Table 1: Terminologies and symbols

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 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 [14, 25], speech keyword spotting [18], image classification [22], etc. As traditional NAS frameworks do, the goal of federated NAS can be formulated as following:

$$\begin{aligned} \arg\max_{DNN} \quad & Acc(DNN) \\ subject\ to \quad & Res_j(DNN) \leq Bud_j, j = 1, 2...n \end{aligned}$$

where DNN is a simplified NN model, $Acc()$ computes the accuracy, $Res_i()$ evaluates the resource consumption of the i^{th} resource type, and Bud_i is the budget of the i^{th} resource and the constraint on the optimization. The resource type can be computational cost (MACs), 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. For simplicity, we only consider one resource type in this work (i.e., n=1).

Contributors A typical federated setting assumes that there are substantial distributed devices available for training, e.g., tens of thousands [5]. 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 Federating State-of-the-Art NAS Algorithm

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 [34]. 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 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

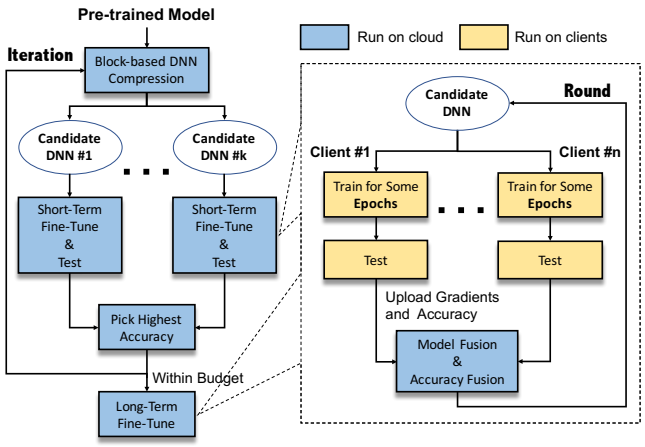


Figure 2: Simplified workflow of FedNAS. The left part (outside of the dash box) also represents the workflow of original NetAdapt [34], the basis of FedNAS.

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 FedNAS works.

3.3 Challenges and Key Optimizations

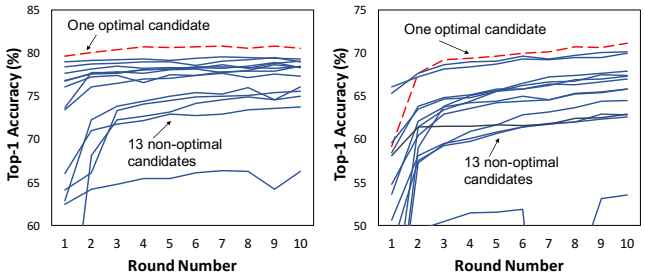
The major challenge of federated NAS is the heavy on-client computational and communication cost. Consequently, the end-to-end process of federated NAS can be excessively time-consuming. Taking communication cost for short-term fine-tune as an example, the total uplink bandwidth usage can be roughly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

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

where $grad\_size()$ calculates the model gradient size, $client\_num()$ is the number of clients involving training $DNN_{i,j}$, 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 [24], 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 provided by FedNAS, 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 [5], tens of thousands of devices are available for FL at the



(a) 2nd Iteration **(b) 8th Iteration**

Figure 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.

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 ($grad\_size(DNN)$) 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, 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 $Res(GM_t) - \Delta R_t$ where ΔR_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 [32]. 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 [33].

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 [15].

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

input : the init model GM_0, the resource budget R, iteration number T, candidate drop ratio $\alpha\%$
output: a list of models (GM_1, ...GM_T) under different resource budgets (R_1, ..., R_T where $R_T = R$)

Function *Cloud_Operation()*: ▷ run on cloud
  for $t \leftarrow 1$ **to** T **do**
    $R_t \leftarrow$ next resource budget
    $round\_num \leftarrow$ round number at current iteration
    $groups \leftarrow$ partition all available clients into groups
    $PMs \leftarrow$ generate a list of simplified models by pruning different layers of GM_{t-1}
    for $k \leftarrow 1$ **to** $round\_num$ **do**
      $PMs \leftarrow Cloud\_One\_Round(PMs, groups)$
    end
    $GM_t \leftarrow PMs[0]$
  end
  perform FL on GM_T or other GMs as user specify
Function *Cloud_One_Round(PMs, groups)*:
  for *each* PM_i *in* PMs ***in parallel*** **do**
    $G_j \leftarrow$ wait to get a free group from $groups$
    lock(G_j)
    for *each client* C *in* G_j ***in parallel*** **do**
      send PM_i to C
      invoke $C.Client\_Operation(PM_i)$
      collect acc, $test\_num$ from C
    end
    $PM_i\_acc \leftarrow$ fuse all collected acc based on $test\_num$
    unlock(G_j)
  end
  $drop\_num \leftarrow max(\alpha\% \cdot PMs.len(), PMs.len() - 1)$
  remove $drop\_num$ models with lowest PM_acc from PMs
  for *each* PM_i *in* PMs ***in parallel*** **do**
    $G_j \leftarrow$ the group that runs PM_i
    collect $grad$, $train\_num$ from all clients within G_j
    $grad \leftarrow$ fuse all $grad$ based on $train\_num$
    $PM_i \leftarrow PM_i + grad$
  end
  return PMs
Function *Client_Operation(Model)*: ▷ run on remote clients
  split local dataset into training and validation set
  $grad \leftarrow$ train $Model$ on local training dataset for E epoches
  $acc \leftarrow$ test $Model$ on local validation dataset
  store $grad$, acc, $train\_num$, $test\_num$ locally

Algorithm 4.1: The proposed FedNAS framework

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.

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

Here, $dist\_dist()$ calculates the distance between the data distributions of two groups, G_{all} is an imaginary group including the data from all clients, r 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 [19].

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 (E) to reduce round number and communication cost [24]. 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_j, once the cloud receives all accuracy of the clients within the same group G_j, it combines the accuracy into a new one by weighting the testing data numbers on the same client:

$$PM_i\_acc = \frac{\sum_s (test\_num_{j,s} \cdot acc_{j,s})}{\sum_s test\_num_{j,s}}, \quad s = 1...g_j$$

where $test\_num_{j,s}$ and $acc_{j,s}$ are the testing data number and testing accuracy reported by the s^{th} client of j^{th} group correspondingly, g_j is the client number of j^{th} group.

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* (i.e., the unpruned model at the beginning of this iteration):

$$acc\_degradation = \frac{Acc(prev\_GM) - PM_i\_acc}{Res(prev\_GM) - Res(PM_i)}$$

Model fusion For a given PM_i and its scheduled group G_j, FedNAS fuses the gradients from all clients within G_j by weighting the training data numbers used in local-tune.

$$fused\_grad_i = \frac{\sum_s (train\_num_{j,s} \cdot grad_{j,s})}{\sum_s train\_num_{j,s}}, \quad s = 1...g_j$$

$$PM_i \leftarrow PM_i + fused\_grad_i$$

where $train\_num_{j,s}$ and $grad_{j,s}$ are the training data number and gradients uploaded from the s^{th} client of j^{th} group correspondingly.

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* [24]. When multiple *GMs* are demanded, FedNAS can still utilize the partitioned clients to train them in parallel.

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.

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 [11] (iid) and Celeba [23] (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 [10], 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).

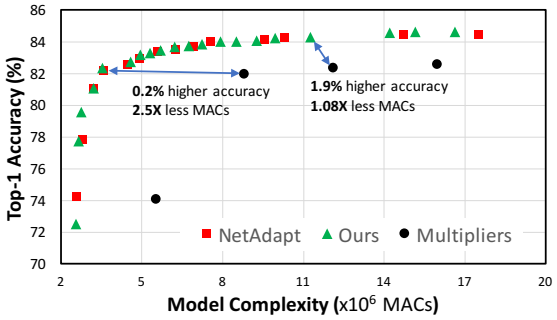
Models We applied FedNAS on two models: MobileNet [16] (for ImageNet, 224x224 input size), a widely used CNN network for mobile applications; A simplified AlexNet, which we call ConvNet (for Celeba, 128x128 input size) with sequential CONV, Pooling, and final FC layers. We did not apply FedNAS on larger 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.

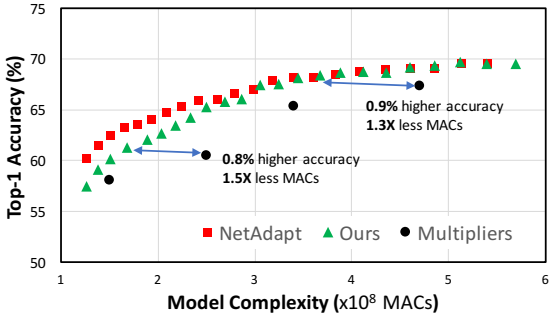
Alternatives We compare FedNAS with two state-of-the-art automatic network simplification approaches. Note that both of them are performed on centralized data.

- *NetAdapt* [34] is the basis of FedNAS. We directly reused their open code and kept the original parameter setting.
- *Multipliers* [16] 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 [2] to obtain the



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



(b) ImageNet. FedNAS setting: 20 rounds, 3 epochs, 14 groups.

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

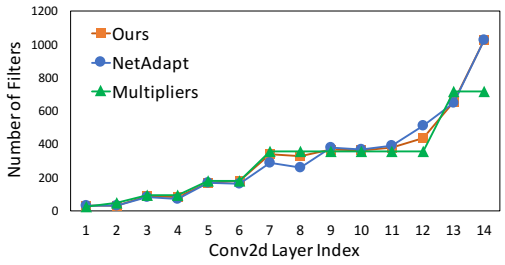
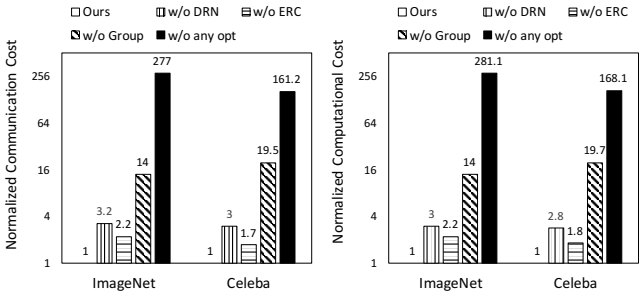


Figure 5: When adapting MobileNet on ImageNet to 50% MACs, FedNAS and NetAdapt generate similar network architectures, while more different from Multipliers.



(a) Communication cost (b) Computational cost

Figure 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.

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 better user privacy. 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 1.5× less complex with 0.8% higher accuracy compared to Multipliers.

On ImageNet, we notice a performance gap between FedNAS and NetAdapt around 2% when MobileNet is simplified by more than 70%. 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 to be closer to NetAdapt.

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: 5 rounds; 7-10: 10; 11-15: 15; >15: 20) 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

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.

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 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 (7) on ImageNet, FedNAS's accuracy doesn't improve much (up to 0.4%) compared to our default setting (14), but incurs much more client cost (e.g., 2× 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 28, the accuracy drops by 1.1% when adapted to 50% complexity, but the uplink cost is also

Model	Group number	Top-1 Accuracy (%)	Avg uplink cost per client (MBs)
75% MobileNet	14 (default)	68.8 (0.0)	70.3 (0.0)
	7	68.9 (+0.1)	140.7 (+100%)
	28	68.6 (-0.2)	35.2 (-50%)
	100	68.5 (-0.3)	9.8 (-86%)
50% MobileNet	14 (default)	67.4 (0.0)	218.1 (0.0)
	7	67.6 (+0.4)	436.2 (+100%)
	28	66.3 (-1.1)	109.0 (-50%)
	100	66.0 (-1.4)	30.5 (-86%)

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.

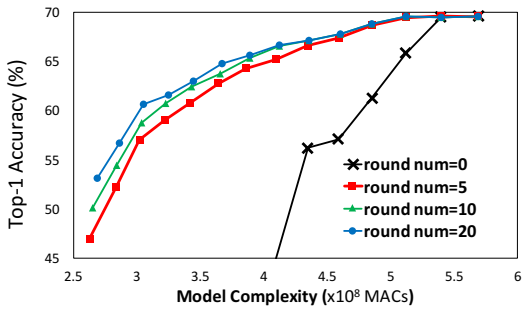


Figure 7: The impacts of short-term fine-tune (round number) on model accuracy (without long-term fine-tune). Other settings are the same as Figure 4. Model: MobileNet.

reduced by 50%. An even larger group number (100) helps reduce the cost by 86% but the accuracy degradation increases up to 1.4%. In a word, the group number provides rich trade-offs between the generated model accuracy and on-client cost. But note 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 currently don't evaluate this neural 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). 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. In this case, 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., 5 and 10), though the model accuracy can be largely preserved but still lower than the default setting. 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.

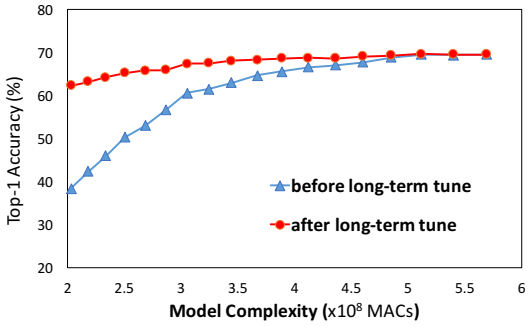


Figure 8: Long-term fine-tune can substantially increase the generated model accuracy. Model: MobileNet.

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 20% 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.

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: parallel model tuning, dynamic training, and candidates early dropping. Tested on both iid and non-iid datasets, FedNAS is able to generate neural networks with similar accuracy compared to training on centralized data, with tolerable computational and communication cost on clients.

REFERENCES

[1] Ai benchmark. http://ai-benchmark.com/index.html, 2019.
[2] Deep learning for java. https://deeplearning4j.org/, 2019.
[3] General data protection regulation (gdpr). https://gdpr-info.eu/, 2019.
[4] Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. V. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), pp. 549–558.
[5] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecny, J., Mazzocchi, S., McMahan, H. B., et al. Towards federated learning at scale: System design. *Proceedings of the 2nd SysML Conference (2019)* (2019).
[6] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 1175–1191.
[7] Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).
[8] Cai, H., Yang, J., Zhang, W., Han, S., and Yu, Y. Path-level network transformation for efficient architecture search. *arXiv preprint arXiv:1806.02639* (2018).
[9] Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).
[10] Caldas, S., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).
[11] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.
[12] Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017* (2017), pp. 1487–1495.
[13] Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 1586–1595.
[14] Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).
[15] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.
[16] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).
[17] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).
[18] Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., and Dureau, J. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), IEEE, pp. 6341–6345.
[19] Lindell, Y. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*. IGI Global, 2005, pp. 1005–1009.
[20] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 19–34.
[21] Liu, X., Hui, Y., Sun, W., and Liang, H. Towards service composition based on mashup. In *2007 IEEE Congress on Services (Services 2007)* (2007), IEEE, pp. 332–339.
[22] Liu, Y., Chen, T., and Yang, Q. Secure federated transfer learning. *arXiv preprint arXiv:1812.03337* (2018).
[23] Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)* (December 2015).
[24] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).
[25] McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings* (2018).
[26] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).
[27] Swearingen, T., Drevo, W., Cyphers, B., Cuesta-Infante, A., Ross, A., and Veeramachaneni, K. ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017* (2017), pp. 151–162.
[28] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 2820–2828.
[29] Wang, J., Zhang, J., Bao, W., Zhu, X., Cao, B., and Yu, P. S. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018* (2018), pp. 2407–2416.
[30] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 10734–10742.
[31] Xu, M., Liu, J., Liu, Y., Lin, F. X., Liu, Y., and Liu, X. A first look at deep learning apps on smartphones. In *The World Wide Web Conference* (2019), ACM, pp. 2125–2136.
[32] Xu, M., Zhu, M., Liu, Y., Lin, F. X., and Liu, X. Deepcache: principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (2018), ACM, pp. 129–144.

[33] Yang, T.-J., Chen, Y.-H., and Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 5687–5695.

[34] Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 285–300.

[35] Zhou, J., Velichkevich, A., Prosvirov, K., Garg, A., Oshima, Y., and Dutta, D. Katib: A distributed general automl platform on kubernetes. In *2019 USENIX Conference on Operational Machine Learning, OpML 2019, Santa Clara, CA, USA, May 20, 2019* (2019), pp. 55–57.

[36] Zoph, B., and Le, Q. V. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* (2017).

[37] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 8697–8710.