

# SuperFedNAS: Cost-Efficient Federated Neural Architecture Search for On-Device Inference

Alind Khare<sup>1</sup>, Animesh Agrawal<sup>1</sup>, Aditya Annavajjala<sup>1</sup>, Payman Behnam<sup>1</sup>,  
Myungjin Lee<sup>2</sup>, Hugo Latapie<sup>2</sup>, and Alexey Tumanov<sup>1</sup>

<sup>1</sup> Georgia Institute of Technology, Atlanta, USA

<sup>2</sup> Cisco Research, USA

**Abstract.** Neural Architecture Search (NAS) for Federated Learning (FL) is an emerging field. It automates the design and training of Deep Neural Networks (DNNs) when data cannot be centralized due to privacy, communication costs, or regulatory restrictions. Recent federated NAS methods not only reduce manual effort but also help achieve higher accuracy than traditional FL methods like FedAvg. Despite the success, existing federated NAS methods still fall short in satisfying diverse deployment targets common in on-device inference like hardware, latency budgets, or variable battery levels. Most federated NAS methods search for only a limited range of neuro-architectural patterns, repeat them in a DNN, thereby restricting achievable performance. Moreover, these methods incur prohibitive training costs to satisfy deployment targets. They perform the training and search of DNN architectures repeatedly for each case. SuperFedNAS addresses these challenges by decoupling the training and search in federated NAS. SuperFedNAS co-trains a large number of diverse DNN architectures contained inside one supernet in the FL setting. Post-training, clients perform NAS locally to find specialized DNNs by extracting different parts of the trained supernet with no additional training. SuperFedNAS takes  $O(1)$  (instead of  $O(N)$ ) cost to find specialized DNN architectures in FL for any  $N$  deployment targets. As part of SuperFedNAS, we introduce MaxNet—a novel FL training algorithm that performs multi-objective federated optimization of a large number of DNN architectures ( $\approx 5 * 10^8$ ) under different client data distributions. Overall, SuperFedNAS achieves upto 37.7% higher accuracy for the same MACs or upto 8.13x reduction in MACs for the same accuracy than existing federated NAS methods.

## 1 Introduction

Federated Learning (FL) is increasingly used in numerous applications [1, 3, 11, 15, 21, 32]. In FL, a large number of clients collaboratively participate in a distributed training of a deep neural network (DNN) while keeping their data private [10, 22, 23, 29, 36]. FL offers three key benefits: a) smaller communication costs, b) massive parallelism, and c) privacy preservation. Despite achieving notable success, the majority of FL works [2, 25, 29, 36] rely on manually designed predefined DNN architectures, a practice that can often be sub-optimal. These

predefined architectures often struggle to adapt to the nuances present in diverse data distributions across clients [16], leading to sub-optimal accuracy. When these DNN architectures get optimal accuracy, it comes at the expense of increased model complexity [38] as they are primarily designed to increase accuracy. This makes manually designed DNNs unfit for clients' on-device inference: they don't provide optimal accuracy under different deployment targets such as battery conditions, hardware, latency/MACs, memory constraints prevalent in on-device inference [14, 39, 40, 42]. To perform efficient inference, there is a need to automate the design and training of DNN architectures in FL.

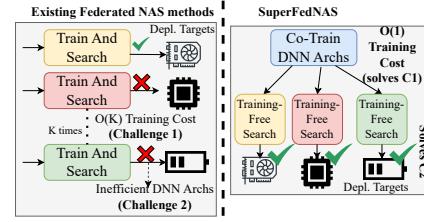
Recent FL works partially address these limitations by proposing neural architecture search (NAS) methods [16, 18]. These methods automatically find the most accurate DNN. By adapting the DNN architecture to different clients' data distributions, these methods improve accuracy over traditional FL methods like FedAvg [29]. However, when it comes to providing optimal DNN architectures for efficient inference, these methods face the following challenges:

- (C1) Existing federated NAS methods are prohibitively expensive to satisfy multiple deployment targets in on-device inference.

The existing federated NAS methods [16, 18] only satisfy one deployment target at a time (Fig. 1). These methods perform the search and training procedures simultaneously [16]. In the end, the output is a single best-performing architecture subject to a deployment target. Therefore, for  $N$  deployment targets, they need to repeat the entire process of search and training  $N$  times. Hence, their communication and computational cost in training becomes  $O(N)$ . These methods are simply not scalable to satisfy increasingly diverse deployment targets. This motivates the need to make federated NAS methods *scalable*. Indeed, diverse deployment targets at inference are increasingly common [14, 39, 40, 42]. For instance, the GBoard application [15] that uses the FL-trained DNNs for next-word prediction runs on a range of hardware from Apple's M-series chips with a dedicated neural engine [4] to old hardware like Pixel XL (gen1) [37]. Even on the same hardware, variable resource availability (e.g., battery levels) at inference makes optimal DNN architectures significantly different.

- (C2) Existing federated NAS methods struggle to produce optimal DNN architectures under inference deployment targets.

Federated NAS methods like FedNAS [16] tend to sub-optimally increase latency/MACs or memory to achieve better accuracy. Precisely, these methods only search for a building block in the DNN architecture and then repeatedly stack the most performing block to create the final DNN. Repeated stacking not only restricts the block diversity but also sub-optimally inflates latency or memory. Instead, to find optimal DNNs under deployment constraints, the federated NAS



**Fig. 1:** SuperFedNAS vs Existing Federated NAS methods [16, 18]

methods need to support rich diversity in DNN architectures, including diversity in blocks. This implies that federated NAS methods need to search for optimal DNNs from a larger search space of DNN architectures.

To solve the aforementioned challenges (**C1**, **C2**), we propose SuperFedNAS: a federated method that performs NAS over a rich diversity of DNN architectures for efficient on-device inference (Fig. 1). Taking inspiration from centralized NAS techniques [6], SuperFedNAS addresses **C1** by decoupling the training and search of DNN architectures in FL. SuperFedNAS has one FL training stage that co-trains a large number of DNNs simultaneously. It achieves this by performing FL-training of the supernet [6]: a DNN that contains multiple smaller DNNs (subnets) with varied shapes and sizes within it. These subnets partially share their weights. Post the FL training stage, a client performs NAS locally without any additional training, and thereby has negligible cost. Given a deployment target, local NAS finds specialized DNNs (subnets) using predictor-guided search [28]. Overall, SuperFedNAS reduces the cost to find specialized DNNs in FL for  $N$  deployment targets to  $O(1)$ . SuperFedNAS solves **C2** as its DNN architectures support different depths and block diversity<sup>3</sup>. In fact, SuperFedNAS performs federated NAS over  $\approx 5 * 10^8$  diverse DNN architectures.

However, decoupling the training and search of DNN architectures in federated NAS is non-trivial. It involves federated co-training of a large number of subnets in the supernet across multiple clients. Such training needs to perform multi-objective federated optimization of the shared weights in the supernet and optimize the accuracy of each subnet. Minimizing naive proxy objectives for the multi-objective federated optimization doesn't address challenges **C1**, **C2**. For instance, a straightforward proxy objective to train the supernet in FL is a linear combination of subnets' losses across all the data partitions. However, minimizing this objective leads to interference: a phenomenon where smaller subnets interfere with larger subnets. The interference results in sub-optimal accuracy of subnets and doesn't solve **C2**. Moreover, it also leads to slow convergence increasing the training cost (doesn't solve **C1**).

To efficiently perform multi-objective federated optimization of a large number of subnets, we propose MaxNet: an FL training algorithm in SuperFedNAS that trains supernets with reduced interference for better accuracy (for **C2**) in a single FL training stage with lower communication/computation costs (for **C1**). The key idea in MaxNet is to optimize a novel objective that explicitly minimizes the loss of worst-performing subnets on each data partition. Optimizing this objective enables MaxNet to adapt DNN architectures to different client data distributions. Moreover, improving the worst-performing subnets on each data partition improves performance of best-performing subnets on every data partition due to weight-sharing and reduces interference. To effectively optimize this novel objective, MaxNet innovates on subnet sampling and supernet's parameter aggregation. In summary, our contributions are as follows:

- SuperFedNAS: A one-stage federated NAS method that trains a rich diversity of DNN architectures for efficient on-device inference.

---

<sup>3</sup> Each block allowed to have different width.

- SuperFedNAS produces specialized DNN architectures for  $N$  deployment (hardware/latency/MAC) targets with  $O(1)$  cost. Post the training stage of SuperFedNAS, the search doesn't require any additional training.
- MaxNet: An FL training algorithm that optimizes a novel objective to train supernets in FL and reduce interference.

SuperFedNAS outperforms existing federated NAS methods across multiple image (CIFAR10/100, CINIC-10) and text datasets (Shakespeare derived from LEAF [8]), degrees of non-iidness, and client participation. It achieves upto 37.7% higher accuracy for the same MACs or upto 8.13x MACs reduction for the same accuracy than existing federated NAS methods with 11x training cost reduction to satisfy 20 deployment targets.

## 2 Related Work

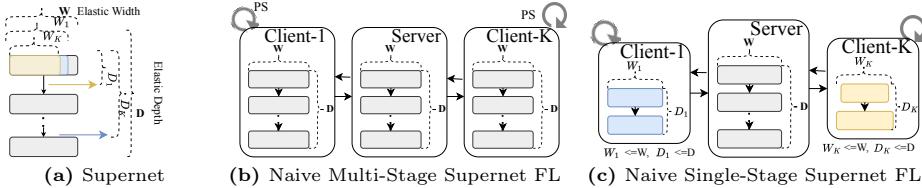
Tab. 1 compares SuperFedNAS with existing FL approaches.

**NAS in FL.** Existing federated NAS [16, 18, 44] methods simultaneously search and train DNN architectures, which becomes prohibitively expensive to satisfy multiple inference deployment targets (Fig. 1). SuperFedNAS decouples the training from search. It only performs training once and enables training-free search to scale to multiple deployment targets. Apart from this key difference, existing federated NAS approaches also differ w.r.t. DNN architectures. The architecture search space of FedNAS [16] remains considerably restricted: its architecture space does not include DNNs that differ per layer or in depth due to repeated stacking (§1). FedNASMobile [44] uses DNN pruning. Its DNN architectures only differ in width and not depth. FedPNas [18] keeps the base architecture the same among all clients, but supports the DNNs that differ in local layers appended to the base architecture. The common base architecture shared by all DNNs restricts architecture diversity. In contrast, SuperFedNAS's architecture space allows DNNs to differ at layer granularity by supporting different width per layer and varied number of layers per stage. The resulting diverse architecture space is essential to produce MACs/latency-efficient DNNs for different hardware (C2).

**NAS in Centralized Setting.** Recent centralized NAS methods [6, 7, 30, 34, 38] produce DNNs suited for inference. They find the most accurate DNN architectures under latency/FLOPs targets on different hardware. Among these methods, Once-for-all NAS methods like OFA [6] and CompOFA [30] reduce the training cost to find optimal DNNs by decoupling the training and search stages. In the training stage, OFA jointly optimizes many DNN architectures (subnets) contained inside the supernet with a multi-staged training algorithm. Once the supernet is trained, OFA performs a search with no additional training to find specialized DNN architectures for target latency/FLOPs. However, all these NAS methods are designed to work in centralized data settings where there are no communication cost restrictions, and techniques like OFA [6] can afford to have multi-staged training (doesn't target C1). In contrast, SuperFedNAS performs NAS in a single-stage FL-training setting and incurs less communication cost.

**System Heterogeneity in FL.** Many works in FL support system heterogeneity [9, 13, 26, 43]. These works perform federated optimization under training time

Feature	FedNAS [16]	FedPNAS [18]	FedNASMo. [44]	ScaleFL [43]	InCo [9]	SuperFedNAS
Weight Sharing						✓
Utilizing NAS	✓	✓		✓		✓
Training Cost for $N$ Deployment	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(1)$
Satisfying Diverse Deployment at Inference						✓

**Table 1:** Comparing existing FL approaches with SuperFedNAS**Fig. 2: Supernet and its Naive FL-Training Methods.** (a) Supernet consists of subnets within it that differ in depth/width. (b) Multi-Stage method sends entire supernet to clients. (c) Single-Stage Supernet FL-Training sends subnets to clients.

constraints like low bandwidth etc. This goal is complementary to SuperFedNAS’s goal that targets satisfying deployment targets at inference (post training). Fundamentally, these works don’t perform NAS (as done in SuperFedNAS): they don’t search for optimal DNN architectures based on clients’ data distributions and latency/MAC deployment targets at inference. Incorporating training time constraints in SuperFedNAS is left as future work.

### 3 Method

SuperFedNAS consists of an FL training stage that co-trains many subnets contained inside the supernet cost efficiently (for **C1**). Once the supernet is globally FL-trained, the clients perform NAS locally to extract optimal subnets (DNN architectures) subject to their diverse deployment targets (for **C2**) with no additional training. Since the local NAS does not require any training and is decoupled, SuperFedNAS’s search is significantly faster than prior federated NAS methods. We begin this section with the description of SuperFedNAS’s training stage including its problem formulation. We explain MaxNet: a training technique that co-trains subnets in SuperFedNAS with reduced interference and compare it with other naive supernet FL-training approaches. Later, we dive deeper into SuperFedNAS’s local NAS stage.

#### 3.1 Problem Formulation

**SuperFedNAS’s DNN Architecture Space.** In SuperFedNAS, DNN architectures differ in depth and width and share their weights as part of a single supernet. These DNN architectures are formally defined by the elastic dimensions of the supernet namely elastic depth and width. Following the common practice of many DNN models [17, 19, 20, 33], the supernet consists of multiple stages and each stage consists of multiple blocks. Fig. 2a illustrates a stage in the supernet. Elastic depth decides the number of blocks selected in each stage for a specific DNN architecture. Elastic width decides the width (e.g. number of convolution channels) selected in each block. In our experiments, the supernet has four stages, the depth in each stage is chosen from {1, 2, 3}, the width expand ratio is chosen from {0.1, 0.14, 0.18, 0.22, 0.25} that roughly equals  $(5^1 + 5^2 + 5^3)^4 \approx 5 * 10^8$  diverse DNN architectures. We use  $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_M\}$  to denote the ordered

set of  $M$  DNN architectures (subnets,  $M \approx 5 * 10^8$ ) w.r.t. MACs. These subnets share the same weights ( $W$ ) and only require 287.1 MB for storage.

**Prior NAS (Supernet Training) Formulation.** With the SuperFedNAS's architecture space defined, we first elaborate on the prior formulation used in centralized NAS techniques [6, 30] that train supernets in non-federated settings. The objective of centralized NAS techniques [6] is formalized as follows:

$$\min_{W} \mathbb{E}_{\alpha_i \in \mathcal{A}} [L(\mathcal{G}(W, \alpha_i))] \text{ s.t. } \mathcal{P}(\alpha_i \in \mathcal{A}) = \frac{1}{M}, |\mathcal{A}| = M \quad (1)$$

where  $\mathcal{G}(W, \alpha_i)$  denotes a selection (subset) of  $\alpha_i$ 's parameters from shared weights  $W$ .  $L(\mathcal{G}(W, \alpha_i))$  denotes the loss of subnet  $\alpha_i$  on a central dataset. Centralized NAS techniques minimize the expected loss of all DNN architectures.

**FL Notation.**  $K$  clients have their own data partition  $P_k$ . The size of the partition is denoted by  $n_k$  (where  $n = \sum_{k=1}^K n_k$  is the total number of data points).  $L_k(w) = \sum_{i \in P_k} l_i(w)$  denotes the loss for data points of partition  $P_k$ .

**SuperFedNAS's Training Stage Naive Formulation.** SuperFedNAS's training stage performs the federated optimization of the shared weights ( $W$ ) of the architecture space  $\mathcal{A}$ . Thus, the objective function of such training is:

$$\min_{W} \mathbb{E}_{\alpha_i \in \mathcal{A}} \left[ \sum_{k=1}^K \frac{n_k}{n} * L_k(\mathcal{G}(W, \alpha_i)) \right] \text{ s.t. } \mathcal{P}(\alpha_i \in \mathcal{A}) = \frac{1}{M} \quad (2)$$

Obj. 2 finds the weights of the supernet ( $W$ ) that minimize the expected loss of all the DNN architectures in  $\mathcal{A}$  on all data partitions  $\{P_1, P_2, \dots, P_K\}$ . Clearly, Obj. 2 differs from Obj. 1, and can be viewed as a multi-objective federated optimization with the sub-objective as loss minimization of a subnet on every data partition. Obj. 2 is different from the objective of personalized FL [31] that doesn't train DNN architectures globally on all the data partitions.

### 3.2 Naive Supernet FL-Training Algorithms

The training algorithms that train the supernet in FL need to optimize Obj. 2. However, optimizing Obj. 2 is non-trivial due to the expensive expectation over all the DNN architectures' losses on all data partitions. Aggregating gradients of all architectures from all clients is prohibitively expensive and doesn't solve **C1**, especially for the large architecture space considered in SuperFedNAS ( $\approx 10^{18}$ ). We first propose two fundamentally distinct naive methods that reformulate Obj. 2 differently to make it tractable:

**Multi-Staged Supernet FL-Training (PS + FL).** This method optimizes the following objective:

$$\min_{W} \sum_{k=1}^K \frac{n_k}{n} * \underbrace{\mathbb{E}_{\alpha_i \in \mathcal{A}} [\mathcal{L}_k(\mathcal{G}(W, \alpha_i))]}_{\text{Approx. using PS [6]}} \text{ s.t. } \mathcal{P}(\alpha_i \in \mathcal{A}) = \frac{1}{M} \quad (3)$$

Obj. 3 is equivalent to Obj. 2, however, it is easier to approximate the expectation on a given partition using existing NAS methods. Therefore, this naive method uses OFA's training algorithm Progressive Shrinking (PS) [6] and locally runs it in clients to optimize the inner term in Obj. 3 (Fig. 2b). It has been shown in prior works [6] that PS effectively approximates the inner term in Obj. 3 by sampling larger subnets initially and gradually sampling smaller subnets in multiple phases. To adapt PS to the FL setting, we employ multi-stage FL training such that

DNN Arch. $\in \mathcal{A}$	Method	Test Accuracy (%)		
		non-iid=100	non-iid=1	non-iid=0.1
Smallest	FedAvg	$85.25 \pm 0.46$	$83.42 \pm 0.19$	$77.15 \pm 2.5$
	Single-Staged Supernet FL	$84.6 \pm 0.19$	$83.17 \pm 0.12$	$76.28 \pm 1.31$
	Multi-Staged Supernet FL	$84.53 \pm 0.58$	$82.82 \pm 0.34$	$76.26 \pm 2.35$
	MaxNet	$89.42 \pm 0.11$	$88.69 \pm 0.2$	$81.81 \pm 1.59$
Largest	FedAvg	$89.44 \pm 0.67$	$87.88 \pm 0.7$	$81.24 \pm 1.99$
	Single-Staged Supernet FL	$87.14 \pm 0.2$	$86.03 \pm 0.26$	$80.02 \pm 2.07$
	Multi-Staged Supernet FL	$86.45 \pm 0.53$	$85.02 \pm 0.32$	$78.57 \pm 2.48$
	MaxNet	$91.34 \pm 0.3$	$90.91 \pm 0.15$	$84.72 \pm 1.78$

**Table 2: Naive Supernet FL-Training Accuracy Comparison.** Test Accuracy compared on CIFAR10 dataset partitioned with different levels of non-iidness among 20 clients, 40% client-participation. MaxNet outperforms the naive supernet FL-Training methods (single/multi). The naive methods are inferior to FedAvg.

the server provides the entire supernet to each client participating in an FL round along with the PS training phase information (depth, width, etc.). The client trains the supernet locally via PS on its data partition based on the phase provided by the server, updates the supernet’s parameters, and sends the locally trained supernet to the server. The server aggregates the supernet parameters from participating clients using the FedAvg [29] algorithm. Overall, we call this method PS+FL as this method adapts Progressive Shrinking to FL.

**Single-Staged Supernet FL-Training.** To achieve supernet’s FL-training in a single stage, this method optimizes the following objective:

$$\min_W \mathbb{E}_{\mathcal{A}_K \subset \mathcal{A}} \left[ \sum_{\alpha_k \in \mathcal{A}_K} \frac{n_k}{n} * \mathcal{L}_k(\mathcal{G}(W, \alpha_k)) \right] \text{ s.t. } \mathcal{P}(\mathcal{A}_K \subset \mathcal{A}) = \frac{1}{\binom{N}{k} k!}, \quad (4)$$

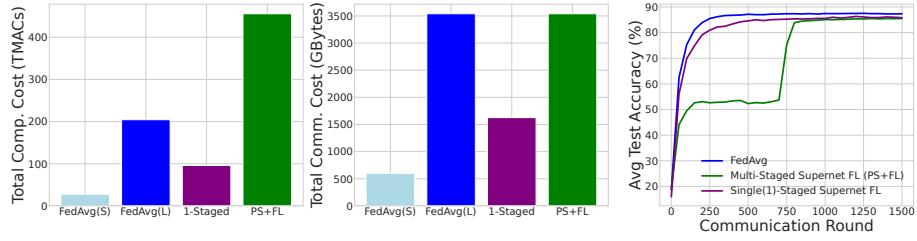
$$\mathcal{A}_K = \{\alpha_i, \dots, \alpha_{i+K}\}, \forall \alpha_i, \alpha_j \in \mathcal{A}_K \alpha_i \neq \alpha_j, |\mathcal{A}_K| = K$$

Obj. 4 minimizes the expected loss of any  $K$  ordered DNN architectures selected from  $\mathcal{A}$  ( $M$  DNN architectures) mapped to their specific data partition ( $\alpha_k$  is uniquely mapped to  $L_k$ ). Note that the probability of selecting  $K$ -ordered DNN architectures is uniform w.r.t. all permutations ( $\mathcal{P}(\mathcal{A}_K \subset \mathcal{A}) = \frac{1}{\binom{N}{k} k!}$ ). Therefore,

in this method, the server uniformly samples  $K^4$  DNN architectures from the architecture space  $\mathcal{A}$ . Then, it randomly assigns the sampled architectures (subnets) to each participating client and sends subnets’ partial parameters ( $\mathcal{G}(W, \alpha_k)$ ), illustrated in Fig. 2c. On receiving a specific subnet from the server, the client trains the subnet locally on its data partition and sends it back to the server. The server receives different subnets that vary in shape and size from different clients. The subnets’ parameters partially overlap with each other due to weight-sharing. Therefore, the server performs cardinal averaging: for each supernet parameter, only the clients’ subnets that share that parameter are averaged.

**Comparing Naive Supernet FL-Training Methods.** We compare the two naive federated supernet training methods. To meaningfully understand the accuracy gaps, the naive methods are also compared with FedAvg [29] which

<sup>4</sup> the method can sample DNN architectures less than  $K$ , say  $C * K$  if the client participation ratio is  $C < 1$ .



**Fig. 3: Naive Supernet FL-Training Cost/Convergence Comparison.** Communication/Computational cost (left,middle) are compared for naive supernet FL approaches with FedAvg training of largest/smallest subnet. The right plot compares convergence over 1500 rounds. Naive-methods have slow convergence and high training cost.

trains the smallest and largest subnets (no weight-sharing). The smallest/largest subnets in  $\mathcal{A}$  represent the lower and upper bound of accuracy reached by all DNNs in  $\mathcal{A}$ . Tab. 2 describes the FL setting.

**Training Costs Comparison.** The takeaways w.r.t. training cost are as follows:

- The multi-stage naive method incurs more communication/computation cost than the single-stage naive method (left, middle plot in Fig. 3). This is because the multi-stage method sends the entire supernet ( $|W|$ ) back and forth to each participating client in each round, whereas, the single-stage method only sends partial weights of the supernet (in the form of subnets) to each client that leads to communication cost savings ( $|\mathcal{G}(W, \alpha_i)| \leq |W|$ ). The multi-stage method has more computational cost as PS that runs locally samples more than one architecture in each iteration (PS [6] samples 4 subnets in each minibatch), while, single-stage method trains only one architecture locally (Fig. 2).
- Multi-stage method’s communication cost is the same as the cost of training the largest network using FedAvg. This is because the largest subnet subsumes all parameters of the supernet ( $\max_{\alpha_i} |\mathcal{G}(W, \alpha_i)| = W$ ) and clients receive all parameters of the supernet in both the methods. Single-stage naive method’s communication cost lies between the cost of FedAvg training of the smallest/largest subnet as partial parameters are sent to clients in each round ( $\min_{\alpha_i} |\mathcal{G}(W, \alpha_i)| < 1/M * \sum_{\alpha_i} |\mathcal{G}(W, \alpha_i)| < \max_{\alpha_i} |\mathcal{G}(W, \alpha_i)|$ ).

**Accuracy Comparison.** As seen in Tab. 2, both the naive supernet FL-training methods fail to match the test accuracy of the smallest or largest subnets trained without weight sharing using FedAvg. Hence, the naive methods don’t provide optimal DNN architectures (don’t solve **C2**). We attribute the following reasons to the sub-optimal accuracy of naive supernet FL-training methods:

- *Interference.* The accuracy of the largest subnet in naive supernet FL-training methods is sub-par accuracy compared to FedAvg. We argue this is because of interference: phenomena also observed in centralized supernet training methods [6, 41] where smaller subnets affect the accuracy of larger subnets.
- *Slow Convergence.* Both the naive methods converge slowly compared to FedAvg as seen in Fig. 3 (right plot). The multi-stage method suffers from slow convergence due to the multi-phase approach. It introduces smaller subnets progressively which makes the convergence of smaller subnets slow. The naive single-stage method suffers from slow convergence as all the supernet’s param-

eters ( $W$ ) may not get updated in every FL round, leading to staleness. This is because the method that optimizes Obj. 4 randomly samples  $K$  architectures in each round, which may not span the entire supernet parameters *i.e.*  $\exists \mathcal{A}_K$  s.t.  $\bigcup_{\alpha_k \in \mathcal{A}_K} \mathcal{G}(W, \alpha_k) \neq W$ .

### 3.3 MaxNet: Proposed Supernet FL-Training Algorithm

MaxNet is a supernet FL training algorithm that produces optimal DNN architectures (for **C2**) and trains the supernet in a single stage. The single-stage training offers lower training cost (for **C1**) compared to multi-stage supernet FL training methods (§3.2). MaxNet produces optimal DNN architectures as it optimizes a fundamentally different objective to reduce interference:

$$\min_W \max_{\vec{\gamma}} \mathbb{E} \left[ \sum_{k=1}^K \gamma_{ik} * \frac{n_k}{n} * \mathcal{L}_k(\mathcal{G}(W, \alpha_i)) \right] \text{ s.t. } \mathcal{P}(\alpha_i \in \mathcal{A}) = \frac{|\mathcal{G}(W, \alpha_i)|}{\sum_{i=1}^M |\mathcal{G}(W, \alpha_i)|}, \quad (5)$$

$$\forall i \in [1, M], k \in [1, K] : \gamma_{ik} \in \{0, 1\}, \sum_{i=1}^M \gamma_{ik} = 1, \sum_{i=1}^M \sum_{k=1}^K \gamma_{ik} = K$$

Obj. 5 minimizes the expected loss of worst-performing DNN architectures on each data partition.  $\gamma_{ik}$  is an indicator that represents the DNN architecture  $\alpha_i$  evaluated on data partition  $P_k$ . The maximum over  $\vec{\gamma}$  picks worst-performing DNN architectures (maximum loss) on each data partition. The constraint  $\sum_{i=1}^M \gamma_{ik} = 1$  ensures that only one architecture gets assigned to  $k^{th}$  data partition. Therefore, in MaxNet, each client (data partition) receives one DNN architecture (in expectation) similar to single-staged method in §3.2. Overall, Obj. 5 has two key features:

- *Improving Worst-Performing DNN Architectures on each Data Partition.* The key insight of MaxNet is that improving accuracy (by minimizing loss) of worst-performing DNN architectures on each data partition has a potential to improve the accuracy of best performing DNN architectures across all the data partitions as the weights are shared. This optimization is particularly helpful in non-iid settings where each data partitions differs significantly. Due to the data heterogeneity, arbitrarily different DNN architectures may perform worse on different partition. Under such settings, Obj. 5 enables DNN architectures to adapt to different data distributions by explicitly minimizing the loss of worst performing DNN architectures on each data partition.
- *Weight-Shared Based Sampling Probability.* Instead of uniform sampling, Obj. 5 prioritizes sampling the DNN architectures that share their weights most with the supernet ( $\mathcal{P}(\alpha_i \in \mathcal{A}) = \frac{|\mathcal{G}(W, \alpha_i)|}{\sum_{i=1}^M |\mathcal{G}(W, \alpha_i)|}$ ). This ensures that most of the supernet's parameters get updated as part of the optimization, mitigating the staleness observed in naive single-staged supernet FL-training method (§3.2).

However, optimizing Obj. 5 remains prohibitively expensive due to SuperFedNAS's large architecture space (§3.1). To make the optimization tractable, MaxNet innovates on DNN architecture sampling and parameter aggregation:

**MaxNet's Subnet Sampling.** MaxNet approximates subnet selection ( $\alpha_i$ ) in Obj. 5. While,  $\max_{\gamma}$  enables selection of smaller subnets as smaller subnets

typically have high loss, the weight-shared based probability prioritizes the sampling of larger subnets. To approximate this, MaxNet samples the largest, smallest and random subnets (uniformly) in  $\mathcal{A}$  in each FL round. The largest and smallest subnets explicitly sampled in MaxNet act as lower/upper bounds of accuracy. Optimizing both the bounds improves performance of other subnets due to weight-sharing. To improve worst-performing DNN architectures on different data partitions, MaxNet gives the smallest/largest subnets to the clients that have received these subnets the least in each FL-round. MaxNet approximates  $\max_{\gamma}$  in Obj. 5 as the clients that receive largest/smallest subnets the least have high loss w.r.t. these subnets on their data partitions.

**MaxNet’s Parameter Aggregation.** The parameter aggregation in MaxNet is based on the optimization dynamics of Obj. 5. Initially, when the loss of all the subnets in  $\mathcal{A}$  is roughly equal, larger subnets get optimized due to weight-shared based probability. As the optimization progresses, the smaller subnets have higher loss and get optimized due to  $\max_{\gamma}$ . To emulate this optimization dynamics, MaxNet performs weighted parameter aggregation. It assigns a weight of  $\beta \in (0, 1)$  to largest subnet’s parameters and  $(1 - \beta)$  to the parameters of rest of the subnets. Initially,  $\beta$  is assigned a high value ( $\beta_0 = 0.9$ ) to prioritize updates from the largest subnet. As FL-rounds progress,  $\beta$  is decayed to make smaller subnets contribute more in the supernet FL training. We provide an extensive evaluation on both the initial  $\beta$ -value and its decay function in §4.

MaxNet’s performance is shown in Tab. 2. MaxNet achieves superior performance compared to the naive supernet FL training methods even under extreme non-iidness (0.1). MaxNet benefits from optimizing Obj. 5 and improving worst-performing subnets on each data partition (helpful in non-iid settings).

### 3.4 SuperFedNAS’s Search Stage

Once the supernet is trained in FL using MaxNet, SuperFedNAS’s search stage finds the optimal DNN architectures subject to client’s deployment targets (**C2**). SuperFedNAS’s search stage is formulated as a constraint optimization problem:

$$\min_{\alpha \in \mathcal{A}} L_{val}(\mathcal{G}(W^*, \alpha_i)) \text{ s.t. } \text{MACs}(\alpha) = \theta \quad (6)$$

$W^*$  is supernet’s parameters trained using MaxNet and  $\theta$  is the MAC (Multiply-and-Add) constraint (or any other depl. target).  $L_{val}$  represents loss on either a local or global (if available) validation dataset. Finding optimal DNN architecture ( $\alpha_*$ ) doesn’t require any re-training. Optimizing Obj. 6 is fast as it simply involves evaluating subnets for accuracy/FLOPs. Applying Obj. 6 to multiple deployment targets ( $\theta_i$ ’s) doesn’t add any training cost. This makes MaxNet O(1) w.r.t. deployment targets. To speed-up the search, SuperFedNAS uses predictors (three layer MLPs) for estimating accuracy and latency of subnets, similar to OFA[6]. SuperFedNAS uses predictor-guided evolutionary search [28] to optimize Obj. 6.

## 4 Experiments

We evaluate whether SuperFedNAS produces efficient DNN architectures (solves **C2**, §1) and generalizes on various FL scenarios: (1) real-world text/image datasets (2) non-iidness (3) client participation ratio ( $C$ ) under multiple MAC targets. We also analyze SuperFedNAS’s scalability to satisfy multiple deployment scenarios

by reporting its training cost (**C1**, §1). We perform a detailed ablation study on MaxNet’s hyper-parameters and its ability to specialize DNN architectures on different hardware/latency targets.

#### 4.1 Setup

**Baselines.** We compare SuperFedNAS with FedAvg [29] and existing federated NAS methods: FedNAS [16], FedPNAS [18]. The comparisons are done over four deployment targets<sup>5</sup> (Multiply-and-Additions (MACs) constraints). The baselines are run repeatedly for each deployment target, SuperFedNAS is run only once.

**Dataset and Models.** Experiments are done on three images and one text datasets: CIFAR10/100 [24], CINIC-10 [12], and Shakespeare derived from LEAF benchmark<sup>6</sup> [8]. For image datasets, SuperFedNAS’s supernet is a ResNet [17] based architecture, containing ResNet-10/26 as the smallest/largest subnets respectively. For the text dataset, the base supernet is a TCN [5] based architecture. For FedAvg, the DNNs are manually chosen following the scaling rule in [35]. The DNN-archs of FedNAS, FedPNAS are kept the same as described in their method. The validation dataset is used for the search phase in all federated NAS methods including SuperFedNAS, the setting is the same as FedNAS [16].

**Non-iid Setting.** Similar to [10, 26], we use the Dirichlet distribution for non-iid training data. The non-iid degree=100 is close to the uniform distribution of classes. Lower non-iid degree (like 0.1) increases per-class differences in data.

**Training Hyper-params.** For image datasets, SuperFedNAS’s setting is similar to [26] using local SGD [27] with no weight decay and a constant learning rate.

#### 4.2 Comparison with Baselines

**Comparison on Image Datasets.** Tab. 3 compares SuperFedNAS with the baselines on three image datasets. For this experiment, we keep the client participation as  $C = 0.4$  and non-iid degree=100. CINIC10/CIFAR10/100 are run for  $R = 1000/1500/2000$  communication rounds. CINIC10 is divided into K=100 partitions (#. clients) and CIFAR10/100 are divided into K=20 partitions.

*Takeaway.* SuperFedNAS achieves **upto 13.1% more accuracy** for target MACs. It finds better DNN architectures across multiple MAC targets compared to the baselines on different datasets. Specifically, SuperFedNAS outperforms the baselines at lower MACs and tougher dataset (CIFAR100). SuperFedNAS’s superior performance comes from co-training a large number of diverse DNN architectures. This enables the search for DNN architectures with arbitrary width and depth to satisfy MAC targets. This DNN architecture diversity remains restrictive in existing federated NAS methods and leads to sub-optimal performance.

**Comparison on Text Dataset.** We evaluate SuperFedNAS on a tough FL setting: text dataset, non-iidness, large number of clients (data partitions) K=660, and 4% client participation. Tab. 4 compares SuperFedNAS with FedAvg on Shakespeare dataset derived from LEAF [8]<sup>7</sup>.

---

<sup>5</sup> FedPNAS could only satisfy two deployment targets at lower MACs

<sup>6</sup> The data is partitioned based on each role in a play in non-IID setting

<sup>7</sup> FedNAS, FedPNAS only release their DNN architectures for image datasets.

Billion MACs	Method	Test Accuracy (%)		
		CIFAR10	CIFAR100	CINIC10
0.45-0.95	FedAvg	85.25 ± 0.46	43.19 ± 0.54	61.76 ± 0.78
	FedNAS	77.33 ± 0.31	40.92 ± 2.21	58.15 ± 0.18
	FedPNAS	88.83 ± 0.5	45.77 ± 0.68	64.3 ± 0.98
	SuperFedNAS	<b>89.42 ± 0.11</b>	<b>56.35 ± 0.3</b>	<b>73.12 ± 0.77</b>
0.95-1.45	FedAvg	86.36 ± 0.22	43.92 ± 0.57	63 ± 0.17
	FedPNAS	89.27 ± 0.51	47.8 ± 26	66.74 ± 0.32
	SuperFedNAS	<b>90.22 ± 0.31</b>	<b>57.16 ± 0.23</b>	<b>74.5 ± 0.74</b>
1.45-2.45	FedAvg	87.59 ± 0.27	44.4 ± 0.58	64 ± 0.07
	FedNAS	86.41 ± 0.1	55.82 ± 0.29	69.97 ± 0.27
	SuperFedNAS	<b>90.93 ± 0.23</b>	<b>57.85 ± 0.31</b>	<b>75.08 ± 0.7</b>
2.45-3.75	FedAvg	89.44 ± 0.67	45 ± 0.27	66.02 ± 0.13
	FedNAS	89.43 ± 0.36	58.39 ± 0.23	71.93 ± 0.13
	SuperFedNAS	<b>91.34 ± 0.3</b>	<b>58.25 ± 0.39</b>	<b>75.38 ± 0.73</b>

**Table 3: Image Datasets Comparison.** SuperFedNAS compared with FedAvg, FL-NAS methods on image datasets for different MAC targets. SuperFedNAS consistently outperforms the baselines.

Million MACs	Method	Test Accuracy (%)	
		C=0.5	C=0.1
0-0.5	FedAvg	48.52 ± 0.11	
	SuperFedNAS	48.22 ± 0.27	
0.5-1	FedAvg	49.17 ± 0.02	
	SuperFedNAS	<b>49.81 ± 0.16</b>	
1-1.5	FedAvg	51.94 ± 0.03	
	SuperFedNAS	<b>53.26 ± 0.06</b>	
1.5-2.75	FedAvg	53.48 ± 0.09	
	SuperFedNAS	<b>54.59 ± 0.15</b>	
2.75-4.0	FedAvg	53.62 ± 0.1	
	SuperFedNAS	<b>54.61 ± 0.13</b>	

**Table 4: Text Dataset Comparison.**

Comparison on tough FL setting: Shakespeare dataset [8], C=4%, non-iidness, K=660 partitions. SuperFedNAS finds efficient DNN archs under tough FL setting.

Billion MACs	Method	Test Accuracy (%)		
		non-iid=100	non-iid=1	non-iid=0.1
0.45-0.95	FedAvg	85.25 ± 0.46	83.42 ± 0.19	77.15 ± 2.5
	FedNAS	77.33 ± 0.31	71.38 ± 0.37	51.57 ± 3.32
	FedPNAS	88.83 ± 0.5	85.7 ± 0.4	78.73 ± 0.45
	SuperFedNAS	<b>89.42 ± 0.11</b>	<b>88.69 ± 0.2</b>	<b>81.81 ± 1.59</b>
0.95-1.45	FedAvg	86.36 ± 0.22	84.65 ± 0.11	77.99 ± 1.6
	FedPNAS	89.27 ± 0.51	87.53 ± 0.32	81.13 ± 0.4
	SuperFedNAS	<b>90.22 ± 0.31</b>	<b>89.3 ± 0.35</b>	<b>83.27 ± 1.28</b>
1.45-2.45	FedAvg	87.59 ± 0.27	86.14 ± 0.23	79.93 ± 1.34
	FedNAS	86.41 ± 0.1	82.13 ± 0.65	65.03 ± 2.57
	SuperFedNAS	<b>90.93 ± 0.23</b>	<b>90.36 ± 0.21</b>	<b>84.1 ± 1.71</b>
2.45-3.75	FedAvg	89.44 ± 0.67	87.88 ± 0.7	81.24 ± 1.99
	FedNAS	89.43 ± 0.36	85.85 ± 0.35	68.13 ± 5.04
	SuperFedNAS	<b>91.34 ± 0.3</b>	<b>90.91 ± 0.15</b>	<b>84.72 ± 1.78</b>

**Table 5: Non-iidness Comparison.** Comparison across varying non-iidness on CIFAR10. SuperFedNAS outperforms baselines as it adapts DNN archs to non-iidness due to optimization of Obj. 5.

Billion MACs	Method	Test Accuracy (%)	
		C=0.2	C=0.4
0.45-0.95	FedAvg	85.59 ± 0.59	85.25 ± 0.46
	FedNAS	76.23 ± 0.5	77.33 ± 0.31
	FedPNAS	86.63 ± 0.51	88.83 ± 0.5
	SuperFedNAS	<b>89.58 ± 0.3</b>	<b>89.42 ± 0.11</b>
0.95-1.45	FedAvg	87.01 ± 0.24	86.36 ± 0.22
	FedPNAS	87.83 ± 0.21	89.27 ± 0.51
	SuperFedNAS	<b>89.95 ± 0.57</b>	<b>90.22 ± 0.31</b>
1.45-2.45	FedAvg	88.04 ± 0.31	87.59 ± 0.27
	FedNAS	84.65 ± 0.14	86.41 ± 0.1
	SuperFedNAS	<b>90.7 ± 0.48</b>	<b>90.93 ± 0.23</b>
2.45-3.75	FedAvg	89.96 ± 0.65	89.44 ± 0.67
	FedNAS	88 ± 0.38	89.43 ± 0.36
	SuperFedNAS	<b>91.16 ± 0.45</b>	<b>91.34 ± 0.3</b>

**Table 6: Client Participation.** Comparison w.r.t. C=0.2,0.4 on CIFAR10 dataset. SuperFedNAS outperforms baselines. MaxNet’s subnet sampling effectively optimizes Obj. 5 at low C.

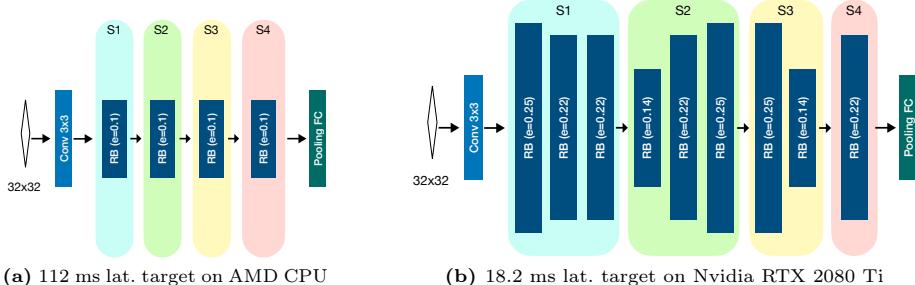
*Takeaway.* SuperFedNAS outperforms FedAvg on shakespeare dataset, is upto **1.29%** more accurate. Even under tough FL settings, SuperFedNAS benefits from automating the design and training of DNN architectures (§1).

**Comparison on Non-iidness.** We evaluate if SuperFedNAS adapts to different data distributions in FL. Tab. 5 compares SuperFedNAS with the baselines on different degrees of non-iidness (0.1, 1, 100). We use CIFAR10 dataset for this experiment divided into K=20 partitions with 40% client participation.

*Takeaway.* SuperFedNAS outperforms the baselines on varied degrees of non-iidness at multiple MAC targets. Particularly, the superiority of SuperFedNAS w.r.t. accuracy is more at extreme non-iidness (0.1). This is because SuperFedNAS benefits from adapting its DNN architectures to different data distributions by optimizing Obj. 5 and explicitly improving the worst-performing subnets (§3.3).

**Effect of Client Participation.** We evaluate the efficacy of SuperFedNAS under different client participation in FL (C=0.2,0.4) on CIFAR10 divided into K=20 partitions. We intend to establish whether MaxNet can approximate Obj. 5 under low client participation. Tab. 6 compares SuperFedNAS with the baselines.

*Takeaway.* SuperFedNAS achieves better accuracy than the baselines for different MAC targets even under low client participation (20%). This is because MaxNet’s



**Fig. 5: SuperFedNAS’s DNN Arch. Specialization.** Specialized DNNs found by SuperFedNAS’s search stage on different hardware/latency targets. SuperFedNAS finds a more accurate DNN for RTX 2080Ti GPU (91.56%) compared to AMD CPU (85.25%). SuperFedNAS’s specialized DNNs are: shallow/thin for AMD CPU, wide/deep for GPU.

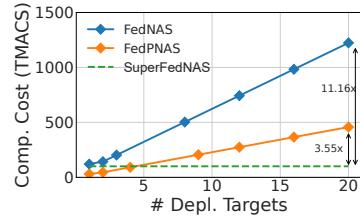
subnet sampling effectively approximates Obj. 5 under different client participation. It sends the smallest/largest subnets to the participating clients that have received these subnets the least (§3.3).

**Training Cost Comparison.** We assess whether SuperFedNAS is scalable to multiple deployment (solves C1, §1). We report the SuperFedNAS’s training cost to satisfy multiple deployment targets. Fig. 4 compares SuperFedNAS’s training cost with baselines w.r.t. #. computations required to satisfy 20 depl. targets.

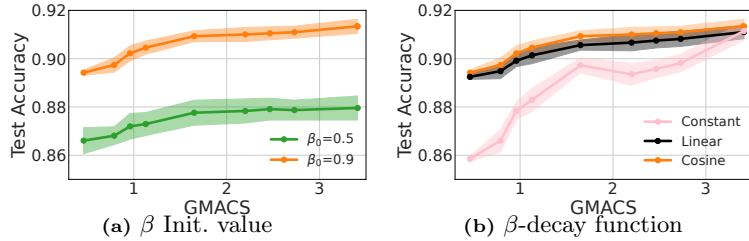
*Takeaway.* SuperFedNAS’s training cost remains  $O(1)$  w.r.t. number of deployment targets and is upto **6x less** than the baselines. This is because SuperFedNAS decouples training from the search in federated NAS, enables search without additional training (§3.4). In contrast, both FedNAS and FedPNAS are run repeatedly to satisfy multiple deployment targets as they train and search simultaneously. This leads to a linear increase in training cost with deployment targets for the baselines.

### 4.3 Ablation Study

**Specialized DNNs for Target Inference Deployments.** We evaluate SuperFedNAS’s ability to specialize DNNs for diverse inference deployment targets. We use the supernet trained on CIFAR10 dataset divided into  $k = 20$  partitions (an experiment in Tab. 3). We use SuperFedNAS’s search stage (§3.4) to find specialized subnets on two different hardware/latency targets: 18.2 ms on RTX 2080Ti GPU and 112 ms on an AMD CPU. As the search doesn’t require re-training, we create a dataset by sampling different subnets and getting their accuracy and latency. We train two different 3-layer MLPs on this dataset that predict accuracy/latency respectively for a given subnet. The MLP takes the subnet’s DNN architecture flattened into a vector as input. Using these predictors the search time reduces to just **2 minutes**.



**Fig. 4: Training Cost Comparison.**



**Fig. 6: MaxNet’s hyperparameters.** **a)** MaxNet approximates Obj. 5 and performs wt. aggregation in the supernet’s parameters.  $\beta$  assigned to maximum subnet and  $(1 - \beta)$  to the rest **b)**  $\beta$  is decayed from  $0.9 \rightarrow$  uniform within the same number of rounds (80% of rounds). Higher  $\beta$  initially with cosine decay produces better accuracy.

*Takeaway.* Fig. 5 shows two different DNN architectures found by SuperFedNAS’s search stage. For AMD CPU as the target hardware, SuperFedNAS finds a shallow and thin DNN at 112 ms latency target. For RTX 2080Ti as the target hardware, SuperFedNAS finds a deep and wide DNN at 18.2 ms latency target. As RTX 2080Ti offers more floating point operations per second than AMD CPU, SuperFedNAS finds a more accurate specialized DNN for RTX 2080Ti (91.56%) compared to the specialized DNN for the AMD CPU (85.25%). SuperFedNAS finds specialized DNNs as it supports both depth and block diversity.

**MaxNet’s Hyperparameters.** We provide ablation on MaxNet that emulates optimization dynamics of Obj. 5. MaxNet introduces two hyperparameters: the weight provided to maximum subnet’s parameter in aggregation ( $\beta$ ) and the decay function that decays  $\beta$  with FL-rounds (§3.3). Fig. 6 compares test accuracy at multiple MAC targets for different intial  $\beta$  values and decay functions on CIFAR10 dataset divided into  $K = 20$  partitions with non-iid degree=100.

*Takeaway.* The initial value of  $\beta$  has a major effect on test accuracy of subnets (Fig. 6a). Higher initial  $\beta = 0.9$  approximates the dynamics introduced via weight-shared sampling probability in Obj. 5 better, resulting in better accuracy. Moreover, decaying  $\beta$  using the cosine function outperforms other decay functions (Fig. 6b). This is because the smaller subnets gradually get high loss, and therefore, get optimized in Obj. 5 due to  $\max_\gamma$ . This optimization dynamic is emulated by a cosine decay function over  $\beta$ , which gradually increases the importance of updates from smaller subnets.

## 5 Conclusion

SuperFedNAS is a scalable federated NAS method that provides efficient DNN architectures for inference deployment targets. It takes  $O(1)$  training cost to satisfy  $N$  deployment targets. SuperFedNAS achieves this by decoupling the training of DNN architectures from their search. SuperFedNAS’s training stage uses MaxNet to co-train a large number of diverse DNN architectures (subnets) as part of a supernet in FL. Once the supernet is trained in FL, clients perform NAS locally with no additional training. MaxNet optimizes a novel objective that improves the performance of worst-performing subnets on each data partition. SuperFedNAS is shown to surpass existing federated NAS methods. It provides optimal DNN architectures for diverse MAC targets with less training cost.

## Bibliography

- [1] Apple. designing for privacy (video and slide deck). apple wwdc (2019), <https://developer.apple.com/videos/play/wwdc2019/708> 1
- [2] Acar, D.A.E., Zhao, Y., Matas, R., Mattina, M., Whatmough, P., Saligrama, V.: Federated learning based on dynamic regularization. In: International Conference on Learning Representations (2021), <https://openreview.net/forum?id=B7v4QMR6Z9w> 1
- [3] Ammad-ud-din, M., Ivannikova, E., Khan, S.A., Oyomno, W., Fu, Q., Tan, K.E., Flanagan, A.: Federated collaborative filtering for privacy-preserving personalized recommendation system. CoRR **abs/1901.09888** (2019), <http://arxiv.org/abs/1901.09888> 1
- [4] Apple: Apple unveils M3, M3 Pro, and M3 Max, the most advanced chips for a personal computer — apple.com. <https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/>, [Accessed 20-02-2024] 2
- [5] Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. CoRR **abs/1803.01271** (2018), <http://arxiv.org/abs/1803.01271> 11
- [6] Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), <https://openreview.net/forum?id=HyIxE1HKwS> 3, 4, 6, 8, 10
- [7] Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018) 4
- [8] Caldas, S., Duddu, S.M.K., Wu, P., Li, T., Konečný, J., McMahan, H.B., Smith, V., Talwalkar, A.: Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097 (2018) 4, 11, 12
- [9] Chan, Y.H., Zhou, R., Zhao, R., JIANG, Z., Ngai, E.C.: Internal cross-layer gradients for extending homogeneity to heterogeneity in federated learning. In: The Twelfth International Conference on Learning Representations (2023) 4, 5
- [10] Chen, H., Chao, W.: Feddistill: Making bayesian model ensemble applicable to federated learning. CoRR **abs/2009.01974** (2020), <https://arxiv.org/abs/2009.01974> 1, 11
- [11] CORDIS., E.: Machine learning ledger orchestration for drug discovery (2019), [https://cordis.europa.eu/project/id/831472?WT.mc\\_id=RSS-Feed&WT.rss\\_f=project&WT.rss\\_a=223634&WT.rss\\_ev=a](https://cordis.europa.eu/project/id/831472?WT.mc_id=RSS-Feed&WT.rss_f=project&WT.rss_a=223634&WT.rss_ev=a) 1
- [12] Darlow, L.N., Crowley, E.J., Antoniou, A., Storkey, A.J.: CINIC-10 is not imagenet or CIFAR-10. CoRR **abs/1810.03505** (2018), <http://arxiv.org/abs/1810.03505> 11
- [13] Diao, E., Ding, J., Tarokh, V.: Heterofl: Computation and communication efficient federated learning for heterogeneous clients. CoRR **abs/2010.01264** (2020), <https://arxiv.org/abs/2010.01264> 4

- [14] Fang, B., Zeng, X., Zhang, M.: Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. CoRR **abs/1810.10090** (2018), <http://arxiv.org/abs/1810.10090> 2
- [15] Hard, A., Kiddon, C.M., Ramage, D., Beaufays, F., Eichner, H., Rao, K., Mathews, R., Augenstein, S.: Federated learning for mobile keyboard prediction (2018), <https://arxiv.org/abs/1811.03604> 1, 2
- [16] He, C., Annavaram, M., Avestimehr, S.: Towards non-i.i.d. and invisible data with fednas: Federated deep learning via neural architecture search. CoRR **abs/2004.08546** (2020), <https://arxiv.org/abs/2004.08546> 2, 4, 5, 11
- [17] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 5, 11
- [18] Hoang, M., Kingsford, C.: Personalized neural architecture search for federated learning (2021) 2, 4, 5, 11
- [19] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR **abs/1704.04861** (2017), <http://arxiv.org/abs/1704.04861> 5
- [20] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017) 5
- [21] Huang, L., Liu, D.: Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. CoRR **abs/1903.09296** (2019), <http://arxiv.org/abs/1903.09296> 1
- [22] Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S.J., Stich, S.U., Suresh, A.T.: Scaffold: Stochastic controlled averaging for federated learning (2021) 1
- [23] Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016) 1
- [24] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009) 11
- [25] Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks (2020) 1
- [26] Lin, T., Kong, L., Stich, S.U., Jaggi, M.: Ensemble distillation for robust model fusion in federated learning. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 2351–2363. Curran Associates, Inc. (2020), <https://proceedings.neurips.cc/paper/2020/file/18df51b97cccd68128e994804f3ecc87-Paper.pdf> 4, 11
- [27] Lin, T., Stich, S.U., Patel, K.K., Jaggi, M.: Don't use large mini-batches, use local sgd. In: International Conference on Learning Representations (2020), <https://openreview.net/forum?id=B1ey01BFPr> 11
- [28] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European conference on computer vision (ECCV). pp. 19–34 (2018) 3, 10

- [29] McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data (2017) 1, 2, 7, 11
- [30] Sahni, M., Varshini, S., Khare, A., Tumanov, A.: Comp{ofa} – compound once-for-all networks for faster multi-platform deployment. In: International Conference on Learning Representations (2021), <https://openreview.net/forum?id=IgIk8RRT-Z> 4, 6
- [31] Shamsian, A., Navon, A., Fetaya, E., Chechik, G.: Personalized federated learning using hypernetworks. In: International Conference on Machine Learning. pp. 9489–9502. PMLR (2021) 6
- [32] Silva, S., Gutman, B.A., Romero, E., Thompson, P.M., Altmann, A., Lorenzi, M.: Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data. In: 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019). pp. 270–274 (2019). <https://doi.org/10.1109/ISBI.2019.8759317> 1
- [33] Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., Zhou, D.: Mobilebert: a compact task-agnostic BERT for resource-limited devices. CoRR **abs/2004.02984** (2020), <https://arxiv.org/abs/2004.02984> 5
- [34] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2820–2828 (2019) 4
- [35] Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. CoRR **abs/1905.11946** (2019), <http://arxiv.org/abs/1905.11946> 11
- [36] Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., Khazaeni, Y.: Federated learning with matched averaging. In: International Conference on Learning Representations (2020), <https://openreview.net/forum?id=BkluqlSFDS> 1
- [37] Wikipedia: Pixel (1st generation) - Wikipedia — en.wikipedia.org. [https://en.wikipedia.org/wiki/Pixel\\_\(1st\\_generation\)](https://en.wikipedia.org/wiki/Pixel_(1st_generation)), [Accessed 20-02-2024] 2
- [38] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 10734–10742. Computer Vision Foundation / IEEE (2019). <https://doi.org/10.1109/CVPR.2019.01099>, [http://openaccess.thecvf.com/content\\_CVPR\\_2019/html/Wu\\_FBNet\\_Hardware-Aware\\_Efficient\\_ConvNet\\_Design\\_via\\_Differentiable\\_Neural\\_Architecture\\_Search\\_CVPR\\_2019\\_paper.html](http://openaccess.thecvf.com/content_CVPR_2019/html/Wu_FBNet_Hardware-Aware_Efficient_ConvNet_Design_via_Differentiable_Neural_Architecture_Search_CVPR_2019_paper.html) 2, 4
- [39] Xu, Z., Yu, F., Liu, C., Chen, X.: Reform: Static and dynamic resource-aware dnn reconfiguration framework for mobile device. In: Proceedings of the 56th Annual Design Automation Conference 2019. DAC '19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3316781.3324696> 2

- [40] Yu, J., Huang, T.S.: Universally slimmable networks and improved training techniques. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019) [2](#)
- [41] Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P.J., Tan, M., Huang, T., Song, X., Pang, R., Le, Q.: Bignas: Scaling up neural architecture search with big single-stage models. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) Computer Vision – ECCV 2020. pp. 702–717. Springer International Publishing, Cham (2020) [8](#)
- [42] Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.: Slimmable neural networks. In: International Conference on Learning Representations (2019), <https://openreview.net/forum?id=H1gMCsAqY7> [2](#)
- [43] Yu, S., Nguyen, P., Abebe, W., Stanley, J., Munoz, P., Jannesari, A.: Resource-aware heterogeneous federated learning using neural architecture search. arXiv preprint arXiv:2211.05716 (2022) [4](#), [5](#)
- [44] Yuan, J., Xu, M., Zhao, Y., Bian, K., Huang, G., Liu, X., Wang, S.: Resource-aware federated neural architecture search over heterogeneous mobile devices. IEEE Transactions on Big Data (2022) [4](#), [5](#)