DIRECT FEDERATED NEURAL ARCHITECTURE SEARCH

Anubhav Garg*
Cisco Systems

Amit Kumar Saha
Cisco Systems

Debo Dutta †
Cisco Systems

ABSTRACT

Neural Architecture Search (NAS) is a collection of methods to craft the way neural networks are built. We apply this idea to Federated Learning (FL), wherein neural networks with predefined architecture are trained on the client/device data. This approach is not optimal as the model developers can't observe the local data, and hence, are unable to build highly accurate and efficient models. NAS is promising for FL which can search for global and personalized models automatically for the non-IID data. Most NAS methods are computationally expensive and require fine tuning after the search, making it a two-stage complex process with possible human intervention. Thus there is a need for end-to-end NAS which can run on the heterogeneous data and resource distribution typically seen in a FL scenario. In this paper, we present an effective approach for direct federated NAS which is hardware agnostic, computationally lightweight, and a one-stage method to search for ready-to-deploy neural network models. Our results show an order of magnitude reduction in resource consumption while edging out prior art in accuracy. This opens up a window of opportunity to create optimized and computationally efficient federated learning systems.

1 INTRODUCTION

Federated Learning (FL) (McMahan et al. (2017)) is an approach to solving a machine learning problem where many clients generate their own data and collaborate under the orchestration of a central server to train the model, often to preserve privacy of the original data (e.g., in regulated verticals like healthcare). Mathematically, it can be represented by the following optimization problem:

$$\min_{w} \mathcal{F}(w), \quad \text{where } \mathcal{F}(w) \coloneqq \sum_{i=1}^{K} \nu_i \mathcal{F}_i(w) \tag{1}$$

Here K is the total number of clients, $\nu_i \geq 0$ and $\sum_i \nu_i = 1$. $\mathcal{F}_i$ is the local objective function for client i and is typically taken as $\mathcal{L}(x_i, y_i, w, \alpha)$, that is, the loss of the prediction on local data (x_i, y_i) with model parameters w and architecture α. The term ν_i can be considered as the relative effect of each client, and is user defined with two typical values being $\nu_i = 1/n$ or $\nu_i = n_i/n$ where n_i is the number of samples of client i and $n = \sum_i n_i$.

A unique characteristic of FL is that each client's data is private to itself and not exchanged or transferred either with the server or other clients. Here, clients can be organizations like hospitals which have patients' data or edge devices like smart cameras or smartphones, referred to as cross-silo and cross-device setting, respectilvely. We'll use the term clients in this article for simplicity, without loss of generality. Multiple such clients generate data locally, mostly in a non identical and independently distributed (IID) fashion.

To design an efficient and accurate model, model developers need to exhaustively examine the characteristics of the data, which is not possible in FL due to the invisibility of data from all the clients to

*Correspondence to anubhgar@cisco.com
†Work done while author was an employee of Cisco

the machine learning engineers. The current practice in FL is to apply a predefined neural network architecture for the given task. By this we mean a neural network with predefined architecture, and not a pretrained neural network. It involves many iterations of model selection and hyperparameter tuning requiring many rounds of training, which is extremely expensive and difficult to achieve on the limited computational resources and low communication bandwidth scenarios. This makes the objective function in Eqn. 1 hard to optimize. The objective in Eqn. 1 trains a global model for all the clients. If the clients' data is small and IID, the single global model learnt via federated learning should perform better than local models. A core challenge in FL is that the clients continuously generate data in a non-IID manner (Li et al. (2020)), which makes a single predefined model hard to fit for all clients. Moreover, the predefined architecture design may have some modules which are redundant for a particular client's dataset and could lead to useless computation for that client. We refer to Kairouz et al. (2019) for an in-depth discussion.

To solve this problem, we make the use of Neural Architecture Search (NAS), which automatically searches for the optimal neural network architecture for the given task. There are many search methods in NAS, majority of which can be classified as using reinforcement learning, evolutionary algorithms, and gradient descent. Compared to other approaches, gradient based differentiable architecture search methods are order of magnitude cheaper with state-of-the-art performance. These methods employ a one-shot weight-shared model trained using gradient descent. Most of these methods involve a search stage, in which a child architecture is derived from the one-shot model and a retraining stage, in which a bigger network based on the child architecture is trained on the specified dataset. Thus, it is a two-stage process which involves analyses of the target data for building the final network. For example, DARTS (Liu et al. (2019)) based methods search for a cell using a proxy dataset in the architecture search phase. For best performance, this cell is stacked many times according to the final dataset and retrained on it. Moreover, due to the bi-level optimization of DARTS, it is extremely computationally expensive and not fit for edge devices like smartphones (see section C in appendix for further analysis). These methods are developed for centralized training and cannot be used directly in the FL setting. To address these issues, we propose an end-to-end NAS approach developed for the FL setting which can search for the optimal architecture directly on the given task, data, and hardware. It is a one-stage hardware adaptable method which searches for a ready-to-deploy neural network on any type of client hardware, ranging from hospitals having multi-GPU clusters to edge devices like smartphones or other IoT devices. The main contributions of our work can be summarized as follows:

1. To the best of our knowledge, it is the first work to combine federated NAS for both cross-silo and cross-device environment, where the data is partitioned by examples.
2. We provide an efficient plug-and-play solution to search for a ready-to-deploy network in the federated setting. Our one-stage architecture search method eliminates the need for retraining of the derived network on the clients.
3. As shown by experiments, our approach takes significantly less communication rounds and computation resources compared to previous methods. For the same number of communication rounds, our approach achieves an average test accuracy improvement of upto 10% compared to predefined models with FedAvg (McMahan et al. (2017)).

Related work has been discussed in section B of the appendix.

2 Our Approach

2.1 Problem Definition

As mentioned earlier, the definition of a typical federated learning problem is given by Eqn. (1). In previous work, the architecture α of the model is fixed and the optimization is only on model parameters w, as can be seen from Eqn. (1). In contrast, we learn the architecture along with the model weights, α being a variable parameter. The local objective function $\mathcal{F}_i$ of each client therefore is :

$$\mathbb{E}_{Z\sim p_{\alpha(Z)}}[\mathcal{L}_w(x_i, y_i, Z)] \tag{2}$$

Here Z is a matrix of all structural decisions, α is the architecture encoding, $p_\alpha(Z)$ is the distribution of architectures, and $\mathcal{L}$ is the loss function. In other words, the local objective is to find the best α to

optimize the expected performance on the given dataset (x_i, y_i). The global objective is to maximize the predictive performance on the given task by collectively learning the model parameters along with the architecture α.

2.2 DIRECT FEDERATED NAS

2.2.1 PRELIMINARIES

The objective function (Eqn. (2)) can be optimized by searching on a *supernet*, called the parent network. The parent network is a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Nodes $v \in \mathcal{V}$ represents the intermediate representations and edges $e \in \mathcal{E}$ represent some transformation of them (e.g., pooling, convolution). There are multiple edges e_{ij} between pair of nodes v_i and node v_j to comprise all possible architectures in the search space. Every edge has a weight θ, called the architecture parameter. These architecture parameters are trained by a NAS algorithm and for every pair of nodes, the architecture weight with the highest value is chosen to derive the final network, also called the child network.

2.2.2 SEARCHING FOR CHILD NETWORK

Our architecture search method is build upon DSNAS (Hu et al. (2020)). It is a task-specific end-to-end NAS method which searches for the child network in one stage. Algorithm 1 details the specifics of searching for the final network.

Algorithm 1 Client Local Search

Require : parent network, operation parameters w, θ_{th}, total epochs E
for epoch $e = 1, 2, 3, ..., E$ **do**
1. Sample one-hot random variables Z from $p_\theta(Z)$
2. Construct child network with w according to Z, multiply a $\mathbf{1}$ after each feature map X
3. Get a batch from data and forward to get $\mathcal{L}$
4. Backward $\mathcal{L}$ to both w and $\mathbf{1}$, backward $\log p_\theta(Z)$ to θ
5. Update w with $\frac{\partial \mathcal{L}}{\partial w}$, update θ with $\frac{\partial \log p_\theta(z)}{\partial \theta}$ $\frac{\partial \mathcal{L}}{\partial \mathbf{1}}$
6. Prune edges with weight $\theta < \theta_{th}$

end

2.2.3 FEDERATED NAS

The objective in Eqn. 1 can be optimized by updating the global architecture and weight parameters, θ and w respectively by averaging the clients' values. Algorithm 2 in appendix A specifies such an orchestration by a central server.

3 EXPERIMENTS AND RESULTS

To evaluate our approach, we do several experiments for the cross-silo and cross-device setting. We take the target task as image classification in our experiments. We use the FedML (He et al. (2020b)) research library for our experiments. The search space and datasets description is provided in appendix D.

Table 1 compares the test accuracy and search cost of our method with FedNAS (He et al. (2020a)), which applies MiLeNAS (He et al. (2020c)) architecture search for federated learning. We achieve remarkable computational efficiency improvement. As our method is one-stage, we record the total time taken to search and train. We would like to point out that though the parameters of the final network is more for our approach, the memory consumption during search is orders of magnitude lesser making our approach fit for low memory hardware. Table 2 compares the accuracy with different number of clients and time taken. It can be seen that the approach is robust to different clients and scale. Although FedNAS shows it can beat model-predefined FL during searching, its architecture search method is based on two-stage MiLeNAS which may not work well during inference on more difficult dataset such as ImageNet and CINIC-10.

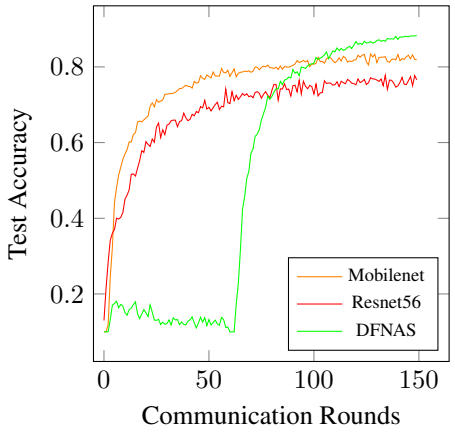

Figure 1: Average test accuracy vs. communication rounds for the non-IID CIFAR-10 data distribution on 8 clients for 150 rounds. DFNAS achieves an accuracy of 88.31%, MobileNetV1 83.41%, and ResNet56 77.78% with FedAvg for same number of rounds.

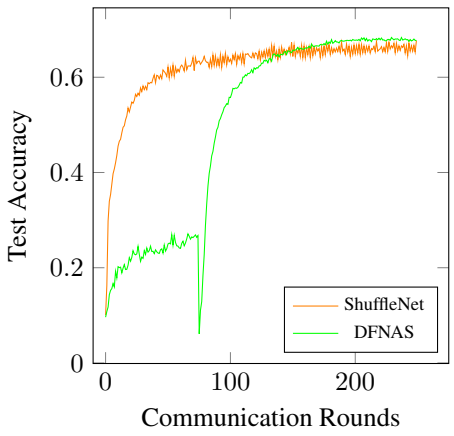

Figure 2: Average test accuracy vs. communication rounds for the non-IID CINIC-10 data distribution on 16 clients for 250 rounds. DFNAS achieves an accuracy of 68.35%, while ShuffleNetV1 with FedAvg 67.70% for same number of rounds.

Figure 1 compares our approach with FedAvg algorithm on two pre-defined neural networks. DFNAS achieves more than 10% higher test accuracy on Resnet56 (He et al. (2016)) and 5% on MobileNet (Howard et al. (2017)) for same number of communication rounds. The lower accuracy of DFNAS for a few rounds is because it searches for the best architecture from the search space while for other curves it is fixed. Figure 2 compares DFNAS with *ShuffleNetV1* (Zhang et al. (2018)) using the FedAvg algorithm on the cross-device setting. The total number of clients are 16, both the methods have equal parameters ($< 1M$) and the data distribution is non-IID. For same number of communication rounds, we achieve better performance than the pre-defined architecture.

Table 1: DFNAS achieves state-of-the art performance on CIFAR-10 dataset. The results are for 8 clients having non-IID data distribution. Since our method is one-stage, the total time consisting of search and training is recorded.

Method	Test Accuracy (%)	Params (M)	Total time (GPU Days)	Memory (MB)
FedNAS	91.43 ± 0.13	0.33	1	10,793
DFNAS (ours)	92.11 ± 0.1	2.1	0.18	1,437

Table 2: Comparison of our approach with different clients on the CIFAR-10 dataset. The data distribution is IID for more than one clients.

Clients	Test Accuracy (%)	Params (M)	Total time (GPU Days)
2	93.53 ± 0.10	2.1	0.83
4	93.31 ± 0.12	2.1	0.43
8	92.79 ± 0.15	2.1	0.18

4 Conclusions

In this paper, we addressed the inefficiency of the current practice of applying predefined neural network architecture to federated learning systems. We presented a plug-and-play solution DFNAS, which is as simple as training one single neural network yet provides an improvement over

current approaches in terms of accuracy, computation and communication bandwidth. Our work highlights the inefficiency of applying the DARTS based NAS methods directly on the federated learning setting. We believe that this general approach could be applied to a variety of federated learning problems.

REFERENCES

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Fabio Maria Carlucci, Pedro Esperanca, Rasul Tutunov, Marco Singh, Victor Gabillon, Antoine Yang, Hang Xu, Zewei Chen, and Jun Wang. Manas: multi-agent neural architecture search. *arXiv preprint arXiv:1909.01051*, 2019.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019a.

Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019b.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019.

Chaoyang He, M. Annavaram, and A. S. Avestimehr. Fednas: Federated deep learning via neural architecture search. *ArXiv*, abs/2004.08546, 2020a.

Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020b.

Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020c.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

A. Howard, Menglong Zhu, Bo Chen, D. Kalenichenko, W. Wang, Tobias Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

Shoukang Hu, S. Xie, Hehui Zheng, C. Liu, Jianping Shi, Xunying Liu, and D. Lin. Dsnas: Direct neural architecture search without parameter retraining. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12081–12089, 2020.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, 2019.

P. Kairouz, H. McMahan, B. Avent, Aurélien Bellet, Mehdi Bennis, A. Bhagoji, Keith Bonawitz, Z. Charles, Graham Cormode, R. Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, D. Evans, Josh Gardner, Zachary A. Garrett, A. Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Z. Harchaoui, Chaoyang He, Lie He, Z. Huo, B. Hutchinson, Justin Hsu, Martin Jaggi,

T. Javidi, Gauri Joshi, M. Khodak, Jakub Konecný, A. Korolova, F. Koushanfar, O. Koyejo, T. Lepoint, Yang Liu, P. Mittal, M. Mohri, R. Nock, Ayfer Özgür, R. Pagh, Mariana Raykova, Hang Qi, D. Ramage, R. Raskar, D. Song, Weikang Song, S. Stich, Ziteng Sun, A. T. Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, L. Xiong, Zheng Xu, Q. Yang, F. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *ArXiv*, abs/1912.04977, 2019.

Purushotham Kamath, Abhishek Singh, and Debo Dutta. Fast neural architecture construction using envelopenets. *arXiv preprint arXiv:1803.06744*, 2018.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pp. 2016–2025, 2018.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37:50–60, 2020.

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

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1eYHoC5FX`.

H. McMahan, Eider Moore, D. Ramage, S. Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pp. 1975–1985, 2019.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2902–2911. JMLR. org, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Ishika Singh, Hao-Yi Zhou, Kunlin Yang, M. Ding, B. Lin, and Pengtao Xie. Differentially-private federated neural architecture search. *ArXiv*, abs/2006.10559, 2020.

Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *International Conference on Machine Learning*, pp. 4771–4780, 2018.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rylqooRqK7`.

Mengwei Xu, Yuxin Zhao, K. Bian, Gang Huang, Q. Mei, and X. Liu. Federated neural architecture search. *arXiv: Learning*, 2020a.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020b.

Tien-Ju Yang, A. Howard, Bo Chen, X. Zhang, A. Go, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *ArXiv*, abs/1804.03230, 2018.

X. Zhang, X. Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

Hangyu Zhu and Y. Jin. Real-time federated evolutionary neural architecture search. *ArXiv*, abs/2003.02793, 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR 2017*, 2017.

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

A FEDERATED ARCHITECTURE SEARCH

Algorithm 2 Direct Federated NAS

Require : Hardware configuration, the total number of rounds T
Build the parent network based on hardware configuration
Initialize w, θ of parent network. Call it w_0 and θ_0
for *round* $t = 0, 1, 2, ..., T$ **do**
  Select K clients S_t
  for each client $i \in S_t$ **in parallel do**
    Send w_t, θ_t to client. Run Client Algorithm (Algorithm 1 from paper) and get updated parameters, w_{t+1}^i and θ_{t+1}^i
  end
  Update architecture $\theta_{t+1} \leftarrow \sum_i^K \frac{n_i}{n} \theta_t^i$
  Update weight $w_{t+1} \leftarrow \sum_i^K \frac{n_i}{n} w_t^i$
end

The above algorithm requires the device type as input, which can be GPU or CPU, and the parent network is constructed accordingly. The server initializes the weight and architecture parameters of the parent network. For every round, the server selects K clients from the pool of available clients. In each round, the the local search process is run by every client in parallel and the architecture and weight parameters are trained on their respective dataset. After the clients have finished the local search, the architecture θ and weight w parameters of the parent network is updated by the formulae in Algorithm 2. This process is repeated for all rounds or until the model is converged.

This method gives a ready-to-deploy network with no need of retraining a larger network constructed by stacking the DAG's learnt on a small network. Our method is hardware agnostic, which can search on the target hardware type directly by constructing the parent network accordingly. It learns the architecture directly on the federated datasets and target hardware without any proxy or manual intervention.

B RELATED WORK

Federated Learning: The approach to train shared model on decentralized data without transferring or exchanging it by aggregating locally computed updates was termed *Federated Learning* (McMahan et al. (2017)). They introduced the Federated Averaging algorithm, in which the clients use SGD to train the shared model weights which are averaged by the central server. There are four core challenges in achieving the objective in Eq. 1 — statistical heterogeneity of data, hardware heterogeneity of clients, low communication bandwidth, and privacy (Li et al. (2020)). Our method solves for first two of the above mentioned challenges directly and it solves the third challenge indirectly by eliminating the need for large number of communication rounds required due to model refinement. We would like to point out that standard privacy preserving approaches like differential privacy can be applied to our method to preserve server and client privacy.

Neural Architecture Search: Zoph & Le (2017) used a controller RNN and trained it with reinforcement learning to search for architectures. Since then, numerous NAS methods have been studied. Based on major search strategy, NAS methods can be classified into Reinforcement Learning (Baker et al. (2016); Cai et al. (2018); Zhong et al. (2018); Zoph et al. (2018); Pham et al. (2018)), Neuro-Evolution (Real et al. (2017); Suganuma et al. (2018); Liu et al. (2018b); Real et al. (2019); Elsken et al. (2019)) and gradient-based (Liu et al. (2019); Cai et al. (2019); Xie et al. (2019); Nayman et al. (2019); Xu et al. (2020b); Chen et al. (2019a)). Other methods include Random Search (Li & Talwalkar (2019)), Bayesian Optimization (Jin et al. (2019); Kandasamy et al. (2018) and some custom methods (Chen et al. (2019b); Kamath et al. (2018); Carlucci et al. (2019); Liu et al. (2018a)).

Federated Neural Architecture Search (FNAS): There are several works on FNAS which apply some NAS algorithm to the federated learning setting. In (He et al. (2020a)), clients use MiLeNAS (He et al. (2020c)) to search for an architecture for the cross-silo FL setting. In DP-FNAS (Singh et al. (2020)), in each round clients compute weight and architecture parameter gradients on a subset of local data which are sent to server for averaging in two steps, thus requiring double communication rounds than our approach. DP-FNAS utilize DARTS with additionally applying random Gaussian noise to local gradients to preserve differential privacy. Though they achieve differential privacy by adding random noise to gradients, there is a tradeoff between validation accuracy and the level of privacy. Both of these approaches have large memory requirement than our approach. They are two-stage methods which require parameter retraining after the architecture search, thus requiring large number of communication rounds. DecNAS (Xu et al. (2020a)) applies model compression and pruning techniques to a pre-trained model based on the NetAdapt (Yang et al. (2018)) framework for fitting on the given resource budget. Concretely, they prune the convolution filters with the smallest value based on the ℓ_2 norm. DecNAS searched architecture has a performance limitation as that of the pre-trained model used with a key challenge to fit on the non-IID data in the FL setting. (Zhu & Jin (2020)) proposes a multi-objective double sampling evolutionary approach to FNAS which is computationally expensive.

C DISCUSSIONS

The computational complexity of our method is the same as training a single neural network since we sample only one path in the parent supernet DAG at every step instead of considering all the candidate paths, which requires the whole network to be stored in the memory. Specifically, if P, Q and R denote the forward time, backward time and memory consumption of a sampled subnetwork from the search space, then DARTS-like methods take nP, nQ and nR, where n is the number of operations while DFNAS takes the same as a single network, that is, P, Q and R.

D SEARCH SPACE AND DATASET

Search space: Our parent network is built using shuffle blocks (Zhang et al. (2018)). For the cross-silo evaluation, there are 4 choices for each block in the parent network, with kernel sizes 3, 5, 7, and a xception block. For the cross-device experiments, there are 3 candidates for each choice block in the parent network. For further details about our search space, we refer to (Hu et al. (2020)). Our search space comprises of 4^{20} neural networks architectures for the cross-silo setting and 3^{12} for the cross-device setting.

Dataset : For cross-silo setting, we perform our experiments on the CIFAR-10 dataset for both IID and non-IID distribution. For the IID case, we divide the training dataset homogeneously between the clients in each round. We generate the non-IID dataset similar to (He et al. (2020a)), for fair comparison, that is, by splitting the training images into K clients in an unbalanced manner: sampling $p_c \sim \text{Dir}(\alpha)$ where Dir is the dirichlet distribution with $\alpha = 0.5$ and allocating a $p_{c,k}$ proportion of the samples of class c to local client k. Similar to (He et al. (2020a)) we test different methods on the test data held by the central server. CIFAR-10 has 50,000 training and 10,000 test images, all coloured, of size 32x32, and split equally in 10 classes.

For the cross-device setting, we use the CINIC-10 dataset, which has 90,000 training and test coloured images of size 32x32 split equally in 10 classes. The sampling strategy is same as that of the cross-silo setting.