Federated Model Search via Reinforcement Learning

Dixi Yao*, Lingdong Wang*, Jiayu Xu, Liyao Xiang*, Shuo Shao, Yingqi Chen, Yanjun Tong
John Hopcroft Center, Shanghai Jiao Tong University, Shanghai, China

***Abstract*—Federated Learning (FL) framework enables training over distributed datasets while keeping the data local. However, it is difficult to customize a model fitting for all unknown local data. A pre-determined model is most likely to lead to slow convergence or low accuracy, especially when the distributed data is non-i.i.d.. To resolve the issue, we propose a model searching method in the federated learning scenario, and the method automatically searches a model structure fitting for the unseen local data. We novelly design a reinforcement learning-based framework that samples and distributes sub-models to the participants and updates its model selection policy by maximizing the reward. In practice, the model search algorithm takes a long time to converge, and hence we adaptively assign sub-models to participants according to the transmission condition. We further propose delay-compensated synchronization to mitigate loss over late updates to facilitate convergence. Extensive experiments show that our federated model search algorithm produces highly accurate models efficiently, particularly on non-i.i.d. data.**

***Index Terms*—Edge Cloud Computing, Neural Architecture Search, Reinforcement Learning**

I. Introduction

The recent progress of deep learning has witnessed a variety of machine learning applications such as objective recognition, voice assistant, machine translation, product recommendation, gaining popularity in everyday life. To protect user privacy, federated learning is proposed to enable computational parties to collaboratively learn a shared model while keeping all training data local, relieving the need for centralized storage. It also takes full advantage of the distributed computational resources such as mobile devices.

In typical federated learning frameworks, the model structure is usually given in advance. Mobilenet [1], Yolo [2], shufflenet [3], and many other Deep Neural Network (DNN) models may be good models for deployment, with or without verified performance on particular datasets. However, real-world data are far more complicated than a well-behaved training set, and their distributions cannot be foreknown. It has been recently pointed out that due to pervasive and unpredictable deviations from i.i.d. datasets, real-world data is particularly hard to train [4], [5], and models with pre-determined structures often fail to converge or result in sub-optimal solutions [6]. Therefore, it is essential to search models customized for the distributed data without violating the privacy constraint of local datasets.

Conventional neural architecture search (NAS) algorithms are not designed for distributed frameworks. NAS methods aim to automatically search for neural network structures customized for the underlying distribution of the dataset. Many approaches such as gradient-based methods [7]–[9], evolutionary algorithms [10]–[12], and reinforcement learning-based [13], [14] approaches have been proposed, and the searched models usually outperform the hand-designed ones. However, they are all designed to run on centralized datasets. It remains a problem on how to perform model search efficiently in a distributed setting, in particular, taking the mobility of the participants into consideration.

We propose a reinforcement learning based model search method in the setting of federated learning, which searches for model structures on a supernet, particularly when the data follows non-i.i.d. distribution. Previous works [15]–[18] have made some progress by proposing distributed versions of evolutionary or gradient-based NAS approaches. However, those methods pose significant overhead on the decentralized devices, either consuming long GPU hours or occupying large network resources. For instance, in FedNAS [17] and DP-FNAS [18], each participant needs to search the entire supernet to obtain a model. Transferring the entire supernet would incur a heavy transmission burden on the local device.

We design a lightweight approach for the distributed devices to jointly search for a suitable model structure. In particular, we describe the search space by hyperparameters, generate sub-models sampled from the search space, train the policy to maximize the expected reward (accuracy) for the generated models. By converting the discrete search to search over probabilistic space, only the sampled sub-models, which are much smaller than the supernet, are transferred and computed on each participant, largely alleviating the local computational and communication burden.

Another critical issue is that the model structure search in distributed settings takes a long time to converge. Hence we propose two approaches to facilitate training. We observe that sub-models sampled are of different sizes and hence can be assigned to participants according to the transmission condition. Moreover, the local computation time varies from participant to participant depending on the computation workload, device resource, network bandwidth, etc. Thus it would cause a significant delay if a simple blocking strategy is adopted waiting for the straggler. In extreme case, the search process

*Dixi Yao and Lingdong Wang contributed equally to this work. Liyao Xiang (xiangliyao08@sjtu.edu.cn) is the corresponding author.

This work was partially supported by NSF China (61902245, 62032020, 61960206002), and the Science and Technology Innovation Program of Shanghai (19YF1424500, 19YF1424200).

would be blocked forever if a participant loses connection with the server. Asynchronous methods have been widely discussed in previous literature [15]–[17], which unfortunately are not feasible for our method — the update on supernet weights and the global search controller can hardly be parallelized due to the serial computational bottleneck.

Facing the challenge, we propose a soft synchronization scheme. In our scheme, the update of the search controller and the supernet are done on the server. In each round, the server only waits until most sub-models are updated, ignoring the stragglers. To fully take advantage of the previous updates from late-coming participants, we propose a delay-compensated approach that uses a second-order Taylor expansion to approximate the fresh update with stale data. Compared with throwing away or directly using the stale data, our soft synchronization scheme with delay-compensation offers better-searching performance, almost comparable to the hard synchronization scheme.

Highlights of our contributions are as follows: we propose an efficient RL-based federated model search algorithm to achieve low communication and computation costs at the participant end. To speed up convergence, we adaptively distribute sub-models according to the transmission conditions. We also develop a soft synchronization scheme with delay compensation to fully utilize the stale update to improve searching performance. A variety of experiments in different settings are conducted to verify our design. It is shown that our algorithm searches models of high accuracy on non-i.i.d. dataset, yet with few communication rounds.

II. Related Work

Our work is most related to the following literature.

A. Neural Architecture Search

The technique of neural architecture search (NAS) [19] aims for automating the design of neural networks and can be categorized according to search space, search strategy, and the performance estimation strategy used. Among them, the search strategy receives the most attention. Common search strategy includes reinforcement learning (RL) [13], [14], gradient-based algorithms [7]–[9], and evolutionary algorithms [10]–[12]. The search strategy of our work falls into the reinforcement learning category.

RL-based NAS typically uses an RL controller to produce a policy according to which a model architecture is sampled in each round and the policy would be adjusted correspondingly to the reward to guide the searching process. The controller can be a recurrent neural network (RNN) as in ENAS [13] which treats the model as a DAG and uses an LSTM as the controller. ProxylessNAS [14] adopts an architecture parameter matrix as a controller to find the optimal operations that maximize a reward. We also use an architecture parameter matrix as a controller in our work, but novelly update the controller in a distributed manner.

Gradient-based NAS transforms the discrete search space of model structures into a continuous space and optimizes the search objective with gradient-based algorithms. A representative algorithm is DARTS [7], which searches the optimal structure of a cell and stacks cells into a complete model. DARTS+ [8] and FairDARTS [9] reveal the performance collapse phenomenon caused by overlaying skip-connection operation with other convolutional operations on the same edge. Our work inherits the search space of DARTS but intentionally lets sub-models have one operation per edge, and thus avoids the performance collapse issue.

Evolution-based NAS [10]–[12] views each model structure as an individual of a species, and improves the performance of the species by evolutionary algorithms. Unfortunately, evolutionary algorithms always suffer from low efficiency and requires exorbitant computation time.

B. Federated Model Search

In federated learning, participants' data is usually non-i.i.d. and can be highly dynamic. A pre-defined model may not serve the data distribution best, resulting in a sub-optimal solution. To deal with the problem, federated model search is proposed to automatically design DNN models for FL frameworks.

It is challenging to design a decentralized NAS method as most existing ones are developed in a centralized setting. The following works have made some progress in this area. By applying various compression techniques to a well-trained (backbone) model, Xu *et al.* [15] generate a group of new models which are trained locally by users, and the best one will become the backbone model in the next round. Although DNN models get automatically evolved in the method, it still requires a pre-defined model as the initial backbone. Therefore, its flexibility and searching performance are unsatisfying. Zhu *et al.* [16] adopt an evolutionary algorithm to search for a DNN model in the FL scenario. Since evolutionary NAS keeps a group of DNN models as species, it is natural to distribute the training of each model to the participants and apply a global evolution on the central server with the retrieved results. However, suffering from the low efficiency of the evolutionary algorithm, this method can only run in a simple search space and output a relatively inaccurate model. FedNAS [17] develops a gradient-based federated NAS method based on DARTS. While DARTS represents its search space as a supernet and searches on the supernet to optimize the accuracy, FedNAS sends the training task on the supernet to participants and performs global updates with the averaged gradients. Similar to FedNAS, DP-FNAS [18] adopts the search space of DARTS and distributes the supernet's computation tasks to participants. These two methods can generate DNN models almost comparable with traditional centralized NAS methods. However, it is impractical to send the giant supernet to each participant considering the exorbitant communication cost and user-end computation consumption, especially when the participants are resource-intensive mobile devices. Our work searches in the same design space with DARTS and FedNAS, but novelly proposes a synchronization strategy sending lightweight sub-models to participants, which is significantly more efficient in communication and computation costs.

Different from previous works, we resolve federated model search by RL-based approach. Our method can search efficiently in a well-defined search space and brings minimal burden to the participants. We exceed the previous federated model search schemes while meeting the stringent resource requirement of the participants.

III. Preliminaries

In this section, we introduce some backgrounds for ease of understanding our work.

A. Federated Learning

Federated Learning (FL) is a decentralized framework for training neural network models. The most commonly used FL algorithm is Federated Averaging (FedAvg) [20], which distributes training to each participant and performs a global update with retrieved results.

FedAvg algorithm is composed of two parts. On the server side, the server initializes the global model parameter θ, and select n participants out of K according to a pre-defined proportion. The server sends the global model to those participants, and retrieve models $\theta_{t+1}^k, k \in \{1, \ldots, n\}$ trained by participants with their local data. The weighted average of all retrieved models become the new global model θ_{t+1} for the next round of updates. On the participant side, each receives a model θ from the server at the beginning of each round. In the following training iterations, the participant performs batch gradient descent on its local dataset to update its copy of the global model. After several training epochs, participant k returns the updated model θ_{t+1}^k back to the server and receives the new global model for the next round of training.

In another version of FedAvg, participant k computes the gradient $g_{t+1}^k = \nabla \mathcal{L}(\theta)$, and uploads it to the server. The global model on the server can be updated by the average of the gradients $\theta_{t+1} \leftarrow \theta_t - \eta \sum_{k=1}^{n} \frac{1}{n} g_{t+1}^k$.

B. Reinforcement Learning

Reinforcement Learning (RL) is the process of agents learning a policy to take actions in an environment to maximize rewards [21]. At each step, the RL agent observes states and performs action a_t. By a_t, s_t transits to the next state s_{t+1} and the agent receives a reward. The agent's decision making procedure at each time is characterized by a policy $\pi(s, a, \theta) = P\{a_t = a | s_t = s, \theta\}$, which is parameterized by θ, and can be described by state transition probability distributions. The policy network calculates probability per action and updates the policy to maximize the reward function $f(\cdot)$:

$$\nabla_\theta \mathbb{E}_\pi[f(s,a)] = E_\pi[f(s,a) \nabla_\theta \log \pi(s,a,\theta)] \tag{1}$$

The RL method has a natural advantage to be applied in a distributed setting: each action can be sampled and its reward can be computed individually.

C. Delay-Compensated ASGD

In decentralized machine learning frameworks, it is of the utmost importance to perform parallel stochastic gradient descent (SGD) across different training entities to guarantee convergence. There are two types of parallelization — synchronous SGD (SSGD) and asynchronous SGD (ASGD) [22]–[24]. In the former, a global update can only be executed after all participants finishing their local updates. But SSGD is quite inefficient as stragglers may block the entire process. Hence ASGD is proposed, in which a global update is allowed as soon as one of the participants has done training, eliminating the need for awaiting other participants.

A critical issue of ASGD is the staleness of data. In ASGD, likely, the global model parameters have already been altered by one participant while another participant has not finished the computation on the previous parameters. It is often the case that the server receives a participant's update from a past round. A naive solution is to throw the stale data away. However, it would be a waste of participants' computation resources, especially when there is a large proportion of slightly stale data. Another solution is to use the stale data anyway but it would harm the training process. DC-ASGD [24] accommodates stale data by utilizing the second-order Taylor expansion to approximate the fresh gradient with stale data. However, their method does not directly apply to our work since ours poses a bilevel optimization problem where both the model weights and architecture parameters need to be updated on stale information.

IV. RL-based Federated Model Search

In this section, we show the problem of model search in the federated learning setting, and introduce our reinforcement learning based solution.

We formulate the model search task as an optimization problem that seeks the optimal neural architecture on the union of participants' datasets to minimize the total loss. Conventional federated learning does not take the model structure into consideration whereas the model search is usually performed in a centralized setting. Our objective is as follows.

$$\underset{\alpha,\theta}{\text{minimize}} \sum_{k=1}^{K} \sum_{i \in \mathbb{D}_k} \mathcal{L}(x_i, y_i; \alpha, \theta). \tag{2}$$

K is the total number of participants and $\mathbb{D}_k$ is the local dataset of participant $k \in \{1, \ldots, K\}$. Architecture parameter α represents the model structure, and model weights are denoted by θ. $\mathcal{L}$ is the loss function on the model where x_i, y_i are data instances and labels respectively. The objective is to minimize the overall loss of a model on the distributed datasets by choosing appropriate architecture parameters and model weights.

We face two challenges: first, $\mathbb{D}_k$ for all k are most likely to be non-i.i.d. as the data distribution varies from participant to participant. Each participant keeps fitting the local model on their local samples. If we use a model with a structure designed for i.i.d. data, the model would treat local samples

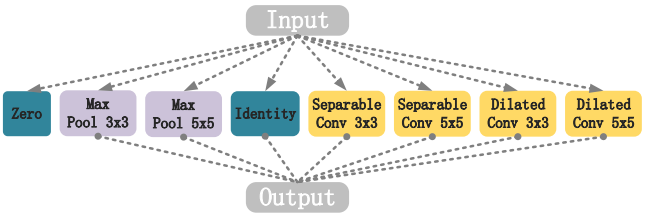

Fig. 1. Candidate Operations

as i.i.d. data, and hence fail to adapt to the divergence across participants' model weights or lead to overfitting in some cases. Second, both the hyperparameters α and parameters θ need to be optimized at the same time in a distributed fashion on resource-stringent devices. We propose a scheme where both α and θ are optimized over distributed dataset: θ is optimized with the regular SGD algorithm on the local dataset and α is treated as controller parameters and updated with reinforcement learning-based strategies.

Overall, in our RL-based model search framework, the structure of the model θ is a state. The RL-controller generates a policy parameterized by α, according to which the model transits from one state to another. The reward is the accuracy loss over the training data. We formally define the federated model search problem as a Markov Decision Process (MDP) with each component as follows.

A. Problem Formulation

Design Space: We adopt the well-designed search space in DARTS [7]. The search space is composed of two types of cells with each cell having two input nodes and one output node. A cell takes the outputs from the preceding two cells as inputs, and a complete DNN model is made up of stacked cells. A cell can be represented by a Directed Acyclic Graph (DAG). In the DAG, a node indicates an intermediate feature, and an edge denotes a candidate operation. To allow differentiation on operations, we represent an edge as a softmax of a set of possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x). \tag{3}$$

Every operation in the set is represented as $o(\cdot)$ applied on feature x. $\alpha_o^{(i,j)}$ indicates the importance of operation o over all operations on an edge between node i and j. $\mathcal{O}$ means the set of all candidate operations, including convolution, pooling, and so on. We adopt the same $N = 8$ kinds of candidate operations with DARTS as shown in Fig. 1. An edge of a sampled sub-model represents exactly one of the eight candidate operations. Combinations of the operations constitute the architecture search space $\mathcal{O}$. All candidate operations are parameterized by a learnable matrix α where each element represents the softmax likelihood of an operation on an edge. We refer to the complete model with all cells and all 8 operations per edge as a supernet, and we sample sub-models from the supernet according to α. Each sub-model can be considered as a candidate of the DNN model, we search for.

We consider both the architecture and the parameters of the supernet as the state. α denotes the policy variable which chooses an action given a state. The operation of each edge is chosen according to the softmax probability:

$$p_i = \frac{\exp(\alpha_i)}{\sum_{j=1}^{j=N} \exp(\alpha_j)}, \quad \forall i = 1, \ldots, N, \tag{4}$$

where N is the total number of candidate operations. Eq. (4) represents the i-th operation is chosen with probability p_i. We transform the probabilities into binary gates:

$$g = binarize(p_1, \cdots, p_N) = \begin{cases} [1, 0, \cdots, 0], & \text{w/ prob. } p_1, \\ \cdots & \\ [0, 0, \cdots, 1], & \text{w/ prob. } p_N. \end{cases} \tag{5}$$

The one-hot binary mask represents the action of selecting the operation with the corresponding index. Specifically, we apply the binary mask g to the operation vector so that only one operation remains on each edge:

$$\bar{o}(x) = \sum_{i}^{N} g_i o_i(x) = \begin{cases} o_1(x), & \text{w/ prob. } p_1, \\ & \cdots \\ o_N(x), & \text{w/ prob. } p_N. \end{cases} \tag{6}$$

Therefore, with α one can sample a sub-model according to the softmax probability distribution. By jointly considering the transition probability p_i per edge, one can calculate the transition probability from one state to another, which is also the probability of sampling a sub-model $p(\mathcal{N}_k)$.

We design a reward function based on the observations from participants. Formally, the reward function on policy α is

$$J(\alpha) = \mathbb{E}_{g \sim \alpha}[R(\mathcal{N}_g)] = \sum_i p_i R(\mathcal{N}(e = o_i)), \tag{7}$$

where $R(\cdot)$ denotes the training accuracy of a DNN model, $\mathcal{N}$ represents the supernet, $\mathcal{N}_g$ means a sub-model sampled from the supernet with mask g. We implement baseline, a common RL trick, subtracting the moving average to reduce the variance in training [25]–[27]:

$$R(\mathcal{N}_{g^m}) = ACC(\mathcal{N}_{g^m}) - b_{t+1}, \tag{8}$$

where

$$b_{t+1} = \beta \frac{1}{M} \sum_{m=1}^{M} ACC(\mathcal{N}_{g^m}) + (1 - \beta) b_t. \tag{9}$$

B. Optimization

In the search space of Sec. IV-A, we aim to optimize the architecture α and model weights θ of the supernet on distributed $\mathbb{D}_k, k \in \{1, \ldots, K\}$. We observe that the architecture parameters are updated w.r.t. the reward, whereas the reward can be computed in parallel on each sub-model. We take advantage of this property to devise a distributed computation framework.

We maximize the expected reward of all sub-models sampled from the supernet by computing gradients w.r.t. the current α:

$$\begin{aligned}\nabla_\alpha J(\alpha) &= \sum_i R(\mathcal{N}(e=o_i))\nabla_\alpha p_i \\ &= \sum_i R(\mathcal{N}(e=o_i))p_i \nabla_\alpha \log(p_i) \\ &= \mathbb{E}_{g\sim\alpha}[R(\mathcal{N}_g)\nabla_\alpha \log(p(g))] \\ &\approx \frac{1}{M}\sum_{m=1}^{M} R(\mathcal{N}_{g^m})\nabla_\alpha \log(p(g^m)).\end{aligned} \tag{10}$$

The last step of Eq. (10) is the key to our formulation. Here g^m represents the binary mask for model m and $p(g^m)$ is the probability to sample g^m according to Eq. (4). Note that $R(\mathcal{N}_{g^m})$ can be computed on each participant individually. Conventionally, the computation of $\nabla_\alpha \log(p(g^m))$ requires backward propagation on $\log(p(g^m))$. Here we transform it into a form that is easy-to-compute. Since only one operation is chosen for each edge on the sampled sub-model, one entry of a sampled binary mask g is 1. Letting the i-th entry be 1, *i.e.,* $g_i = 1$, we have $p(g) = p(g_i = 1) = p_i$. Because p_i is a function of α, we can directly get the analytical solution of $\nabla_\alpha \log(p_i)$. By Eq. (4), we can obtain the gradient:

$$\frac{\partial \log(p_i)}{\partial \alpha_j} = \delta_{ij} - \frac{\exp(\alpha_j)}{\sum_j \exp(\alpha_j)} = \delta_{ij} - p_j, \tag{11}$$

where

$$\log(p_i) = \alpha_i - \log(\sum_j \exp(\alpha_j)),$$

$$\delta_{ij} = \begin{cases} 0, & i = j, \\ 1, & i \neq j. \end{cases}$$

We can rewrite Eq. (11) to a more compact form:

$$\begin{aligned}\nabla_\alpha \log(p_i) &= \left(\frac{\partial \log(p_i)}{\partial \alpha_1}, \cdots, \frac{\partial \log(p_i)}{\partial \alpha_i}, \cdots, \frac{\partial \log(p_i)}{\partial \alpha_N}\right) \\ &= (-p_1, \cdots, 1-p_i, \cdots, -p_N).\end{aligned} \tag{12}$$

Eq. (12) can be calculated efficiently on the server, which decouples the local computation and server computation in Eq. (10). Hence the gradient of α can be evaluated in two parts, with one part entirely on the participant and the other on the server, which lies the foundation of the distributed NAS framework.

With fixed α, the supernet's model weights θ are to be optimized. We found that it is possible to update the weights of each sub-model locally, and perform gradients averaging on each weight, according to FedAvg [20] and the parameter-sharing scheme in ENAS [13].

The update of θ is as follows. In each epoch, the server samples a group of sub-models from the supernet and sends them to participants. Participants train the sub-models on their respective local datasets and return the gradient of each weight. The server collects the gradients of sub-model weight from participants and averages them to obtain the update to the corresponding supernet weight. Then, the server updates the supernet weight. Since, likely, an operation on one edge is never sampled by any sub-model, we define the gradient of such an operation as zero.

Adaptive transmission. Considering the mobility of participants, the transmission condition for each participant differs by the environment. Hence we assign the sampled sub-models by their respective sizes and the transmission condition. We sort the sub-models by model size and sort the participants by their data rate. By allocating larger sub-models to participants with better transmission conditions, we can reduce the overall latency by saving communication time.

The **overall algorithm** works as follows. For simplicity, we assume that there are always K participants online. The server-side algorithm works as follows. In each round, the server samples a binary mask g_k for each participant k according to Eq. (4). Then we prune the supernet with g_k and obtain sub-model θ_k with only one operation on each edge. The server sends the sub-model to the participant according to transmission condition and retrieves the reward $R(\theta_k)$ as well as the gradient of weights $\nabla_{\theta_k} L_k$. Given that, the server computes $\nabla_\alpha J$ according to Eq. (10) and Eq. (12) to update the architecture parameter α. Finally, the server computes the averaged gradient $\nabla_\theta L$ based on sub-models' gradients $\nabla_{\theta_k} L_k$, and updates the supernet weights θ.

The participant-side algorithm is essentially a DNN model training task. In each round, the participant receives a sub-model sampled from the supernet and trains the sub-model on its local dataset using batch gradient descent. Through one backward propagation, the gradients $\nabla_{\theta_k} L_k$ and the reward $R(\theta_k)$ is computed and sent back to the server.

Our algorithm is highly efficient in comparison with other works. The FL participant only needs to train the sub-model in each round. FedNAS [17] and DP-FNAS [18] adopt the same search space with us but send the whole supernet to participants. In contrast, the sub-model distributed in our algorithm has only one operation per edge, reducing the model size to $\frac{1}{N}$ of a supernet. As the communication cost and participant-end training cost are proportional to the model size, our method is approximately N times more efficient in terms of participants' costs.

V. Delay-Compensated Federated Model Search

In the practical implementation of federated RL for NAS, we found that it is inefficient to have the server wait for the completion of each participant for every round of update, as it would lead to low efficiency or even permanent blocking. Especially in the federated learning framework, the network connectivity of each participant differs on the environment. Some participants could be in poor network condition and stragglers will affect the whole system's performance. On another hand, an asynchronous scheme is not feasible in our case due to the serialization bottleneck in updating the global parameters.

To resolve the issue, we design a delay-compensated scheme to apply soft synchronization in our algorithm. Specifically,

in each round, the server only waits for the completion of most participants and ignores the stragglers for the time being. Updates of the stragglers would be processed later when they arrive.

Since the mechanism introduces staleness to the optimization, it is our goal to take advantage of the stale update without hurting the model performance. In practice, it is very likely that a parameter is altered by a participant while another participant still works on its previous version. In this situation, the slower participant's update is based on the stale data. We are inspired by DC-ASGD [24], which utilizes the second-order Taylor expansion of the loss function to approximate the fresh update with stale data, to design a delay-compensated soft synchronization scheme.

Different from the standard ASGD scenarios, our federated model search runs under a soft synchronization framework due to serialization bottlenecks. Besides, we face a more complicated optimization problem in the framework — both architecture parameters and models' weights can be stale in the distributed training. Hence we apply approximation both on the weights and architecture parameters.

A. Update over Staleness

We introduce the delay-compensation scheme for weights θ. As each sub-model contains a part of θ, we use w to denote the sub-model's weights. The subscript denotes the computation round of w and the superscript represents the computation round of α which w is sampled from. For example, $w_{t+\tau}^{t}$ means a sub-model inherits its weights from the $t+\tau$-th round's θ and its structure is sampled according to t-th round's α.

Assume we are at round $t+\tau$. A fresh gradient for optimizing θ is $h(w_{t+\tau}^{t+\tau})$, but we only have a stale gradient $h(w_t^t)$. Since α affects w through an implicit way, we cannot directly represent w as a function of α. Hence we only approximate the fresh gradient by alleviating the staleness of θ but ignore the staleness in α. In detail, we approximate $h(w_{t+\tau}^{t+\tau})$ using $h(w_{t+\tau}^{t})$, the gradient of a sub-model with fresh weight but sampled from a stale distribution:

$$\begin{aligned} h(w_{t+\tau}^{t+\tau}) &\approx h(w_{t+\tau}^{t}) \\ &\approx h(w_t^t) + \lambda h(w_t^t) \odot h(w_t^t) \odot (w_{t+\tau}^t - w_t^t) \end{aligned} \tag{13}$$

In Eq. (13), $h(w_t^t)$ and w_t^t can be locally computed on each participant while the server has a copy of $w_{t+\tau}^t$.

According to Eq. (10), the gradient of α is computed with sub-model rewards and a probability distribution of α. For consistency, we express the reward $R(\mathcal{N}_{g^m})$ as $R(w)$. And we use g_t^m to represent $g^m \sim \alpha_t$, *i.e.,* a binary mask g^m sampled according to the t-th round α, we have $\mathcal{N}_{g_t^m} = w_t^t$. Since the reward is computed locally on each participant, the server only utilizes the reward of the stale model. We choose to approximate the gradient of α by:

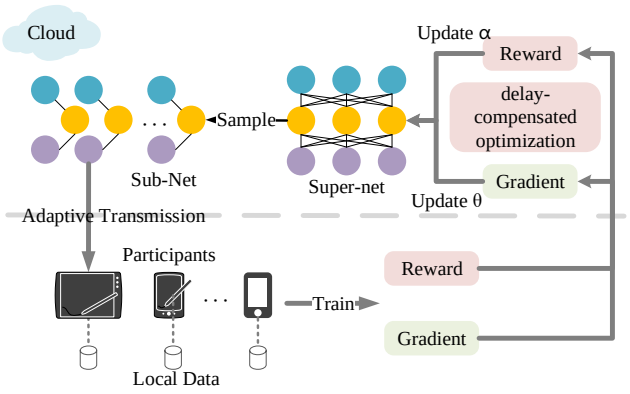

Fig. 2. The framework of delay-compensated federated RL for model search.

$$\begin{aligned} \nabla_\alpha J(\alpha) &= \frac{1}{M}\sum_{m=1}^{M} R(w_{t+\tau}^{t+\tau}) \nabla_{\alpha_{t+\tau}} \log(p(g_{t+\tau}^m)) \\ &\approx \frac{1}{M}\sum_{m=1}^{M} R(w_t^t) \nabla_{\alpha_{t+\tau}} \log(p(g_{t+\tau}^m)), \end{aligned} \tag{14}$$

where $\nabla_{\alpha_{t+\tau}} \log(p(g_{t+\tau}^m))$ can be approximated with the late gradient and the current $\alpha_{t+\tau}$:

$$\begin{aligned} &\nabla_{\alpha_{t+\tau}} \log(p(g_{t+\tau}^m)) \approx \nabla_{\alpha_t} \log(p(g_t^m)) + \\ &\lambda \nabla_{\alpha_t} \log(p(g_t^m)) \odot \nabla_{\alpha_t} \log(p(g_t^m)) \odot (\alpha_{t+\tau} - \alpha_t). \end{aligned} \tag{15}$$

B. Delay-Compensated Federated RL

Our model search algorithm is presented in Alg. 1. We replace the hard synchronization with a soft synchronization and apply the delay-compensated scheme in Line 17 to 29. Memory pools are used for storing stale αs, θs, and masks gs.

The participant computes the gradients on its sub-model as well as the reward, and sends them to the server. In the server-side algorithm, at the start of each round, the super-net's weights θ and architecture parameters α are saved into memories. The server samples binary masks and distributes sub-models to the participants.

Fresh updates from participants will be directly adopted, while for each late participants, the server throws away any update exceeding the staleness threshold since they are almost useless. The server retrieves the fresh sub-model weights θ_m^t and stale sub-model weights $\theta_m^{t'}$ using stale α, θ, g in the memory. Note that it would be more efficient to store these data than to save all past sub-models. Then the server applies the delay-compensated update to both $\nabla_\theta L$ and $\nabla_\alpha J$. And an averaged gradients are adopted to update α and θ. Finally, the data exceeding the staleness threshold is kicked out of the memory pool. By removing the hard synchronization barrier, our soft synchronization scheme is more flexible at the extra memory and computation costs of the server. The overall delay-compensated federated RL framework is given in Fig. 2.

Algorithm 1 Delay-Compensated Federated Model Search

Input: Global weights θ, global architecture parameters α, weights memory Θ, architecture parameters memory $\mathbb{A}$, binary masks memory $\mathbb{G}$, the number of participants K, mini-batch size B, learning rate η_L η_J, staleness threshold Δ.

Sever Update:
Initialize θ, α
for round $t = 1, 2, \dots$ **do**
  save θ^t into Θ, α^t into $\mathbb{A}$
  for participant $k = 1, 2, \dots, K$ **do**
    $g_k^t \leftarrow binarize(p_1, \dots, p_N)$
    save g_k^t into $\mathbb{G}$
    $\theta_k^t \leftarrow prune(\theta^t, g_k^t)$
  end for
  sort participants by bandwidth
  sort θ_k^t $(k = 1, 2, \dots, K)$ by $\|\theta_k^t\|$
  for participant $k = 1, 2, \dots, K$ **do**
    Participant Update(t, k, θ_k^t)
  end for
  soft synchronization, wait for most participants
  for participants $m = 1, 2, \dots, M$ **do**
    receive $R(\theta_m^{t'}), \nabla L_m$
    if $t = t'$ **then**
      $\nabla_\theta L \leftarrow \nabla_\theta L + \eta_L \nabla L_m$
      $\nabla_\alpha J \leftarrow \nabla_\alpha J + \eta_J R(\theta_m^t) \nabla \log p(g_m^t)$
    else
      if $t - t' > \Delta$ **then**
        ignore update, $M \leftarrow M - 1$
      else
        get $\theta^{t'}$ from Θ, $\alpha^{t'}$ from $\mathbb{A}$, $g_m^{t'}$ from $\mathbb{G}$
        $\theta_m^t \leftarrow prune(\theta^t, g_m^{t'})$, $\theta_m^{t'} \leftarrow prune(\theta^{t'}, g_m^{t'})$
        $\nabla_\theta L \leftarrow \nabla_\theta L + \eta_L(\nabla L_m + \lambda \nabla L_m \odot \nabla L_m \odot (\theta_m^t - \theta_m^{t'}))$
        $\nabla_\alpha J \leftarrow \nabla_\alpha J + \eta_J R(\theta_m^{t'})(\nabla \log p(g_m^{t'}) + \lambda \nabla \log p(g_m^{t'}) \odot \nabla \log p(g_m^{t'}) \odot (\alpha^t - \alpha^{t'}))$
      end if
    end if
  end for
  $\nabla_\theta L \leftarrow \nabla_\theta L / M$, $\nabla_\alpha J \leftarrow \nabla_\alpha J / M$
  update α with $\nabla_\alpha J$, update θ with $\nabla_\theta L$
  remove $\theta^{t-\Delta}$ from Θ , remove $\alpha^{t-\Delta}$ from $\mathbb{A}$
  remove $g_k^{t-\Delta}$ from $\mathbb{G}$ for $k = 1, 2, \cdots, K$
end for
Participant Update(t, k, θ_k^t)**:**
$\mathbb{B} \leftarrow$ Split local dataset into batches of size B
Randomly sample a batch $b \in \mathbb{B}$
$\nabla_{\theta_k^t} L_k \leftarrow \nabla_{\theta_k^t} \mathcal{L}(\theta_k^t, b)$
Compute $R(\theta_k^t)$ through the same backward propagation
Return $R(\theta_k^t), \nabla_{\theta_k^t} L_k$ to the server

VI. Experiment

A. Implementation Detail

Setup. We choose image classification on CIFAR10, SVHN, and CIFAR100 as our target tasks. Real-world datasets are mostly non-i.i.d. and it is our main motivation to devise a federated framework to search for a best-fit model. Hence we compose non-i.i.d. CIFAR10 and SVHN dataset according to FedNAS [17]. For each class, we distribute its data over all participants according to the Dirichlet distribution $Dir_J(0.5)$. The new datasets have distributions different from the i.i.d. ones and hence require fast adaptation. All experiments are done on servers with GTX 1080 Ti GPUs. Our algorithm is implemented by `Pytorch`, and the communication between servers and participants is implemented by Distributed RPC. The default hyperparameters are listed in Table I.

TABLE I
Default Experimental Settings

Name	Value	Name	Value
batch size	256	# participant (K)	10
learning rate (θ)	0.025	learning rate (P3, centralized)	0.025
momentum (θ)	0.9	momentum (P3, centralized)	0.9
weight decay (θ)	0.0003	weight decay (P3, centralized)	0.0003
gradient clip (θ)	5	gradient clip (P3, centralized)	5
learning rate (α)	0.003	learning rate (P3, FL)	0.1
weight decay (α)	0.0001	momentum (P3, FL)	0.5
gradient clip (α)	5	weight decay (P3, FL)	0.005
baseline decay (α)	0.99	# warm-up steps	10000
cutout [28]	16	# searching steps	6000
random clip	4	# training epochs	600
random horizontal flapping	0.5	# FL training steps	6000

Our algorithm goes through four phases: the first one (P1) is the warm-up phase, where we fix α and only train θ. Since operations such as convolution have far more parameters than some operations like pooling, it is only fair to compare their performance until the weights nearly converge before searching. The second (P2) is the actual searching phase where our algorithm is performed to seek the optimal α and θ. At the end of this phase, we obtain a DNN model architecture. In the third phase (P3), we re-initialize the searched model structure and train the model from the scratch. We have two approaches for this phase: the searched model is trained in a centralized fashion and in a federated learning setting. In the final phase (P4), the trained model is evaluated on the testing set. Since the algorithm converges slower on non-i.i.d. datasets, we conduct the searching phase for 10000 steps on CIFAR10 and 4000 steps on SVHN. We only use 16 cells at the P4 for SVHN.

We use the average accuracy of participants' models as the metric to gauge searching performance. The performance of the warm-up phase and searching phase is given in Fig. 3 and 4. The blue lines show the average training accuracy of 10 participants' models and the orange line is the moving average with a 50-step window. Both training processes converge in the two phases, and we further investigate the impact of α on searching. We found that fixing θ and updating α alone would result in the failure of convergence and much lower accuracy than Fig. 4, as shown in Fig. 5. Hence it is critical to seek the optimal α and θ at the same time. The searching performance on non-i.i.d. data (Fig. 6) is similar to the one on i.i.d. data (Fig. 4) but only with a slower convergence rate.

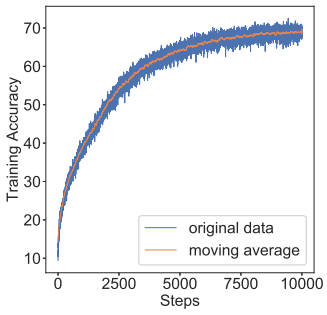

Fig. 3. Warm-up Phase on i.i.d. CIFAR10

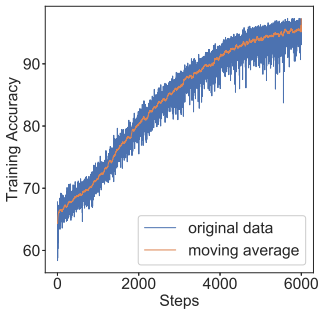

Fig. 4. Searching Phase on i.i.d. CIFAR10

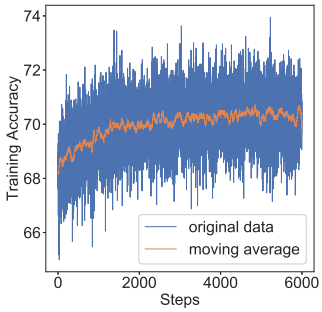

Fig. 5. Updating α with θ fixed .

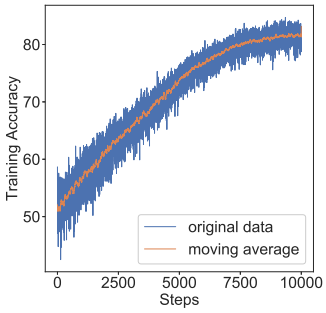

Fig. 6. Searching Phase on Non-i.i.d. CIFAR10

TABLE II
CENTRALIZED EVALUATION ACCURACIES OF SEARCHED MODELS ON CIFAR10

Method	Error(%)	Param(M)	Strategy	FL	NAS
RL-based Federated Model Search					
DARTS (1st order) [7]	3.00	3.3	grad		✓
DARTS (2nd order)	2.81	3.3	grad		✓
ENAS [13]	2.89	4.6	RL		✓
Ours	**2.62**	3.6	RL	✓	✓
Delay-Compensated Federated Model Search					
use (70% staleness)	2.84	3.2	RL	✓	✓
throw (70% staleness)	3.00	4.0	RL	✓	✓
Ours(70% staleness)	**2.72**	3.2	RL	✓	✓
Ours (10% staleness)	**2.59**	**2.7**	RL	✓	✓

TABLE III
FEDERATED EVALUATION ACCURACIES OF SEARCHED MODELS ON CIFAR10

Method	Error(%)	Param(M)	Strategy	FL	NAS
RL-based Federated Model Search					
FedAvg [20]	15.00	-	hand	✓	
EvoFedNAS(big) [16]	13.32	-	evol	✓	✓
EvoFedNAS(small)	16.64	-	evol	✓	✓
Ours	13.36	3.6	RL	✓	✓
Delay-Compensated Federated Model Search					
Ours (10% staleness)	**13.25**	2.7	RL	✓	✓

The extra searching cost compared to the i.i.d. dataset can be considered as the price paid for non-i.i.d. distributions.

B. Accuracy

We use the testing accuracy in the evaluation phase (P4) to gauge the performance of the searched model. We report our searching results on i.i.d. CIFAR10 dataset in Table II where the model is trained in a centralized way at P3. For a clear comparison, we compare the accuracy with other centralized NAS methods such as DARTS and ENAS in Table II. Compared with centralized NAS methods DARTS (gradient-based) and ENAS (RL-based), our algorithm finds models achieving state-of-the-art accuracy performance within the same search space with DARTS, which shows our algorithm can effectively find models with good performance. We also evaluate the accuracy where the searched model is trained distributedly at P3 in Table III along with pre-defined models (FedAvg) and decentralized NAS (EvoFedNAS). Pre-defined models perform the worst. Our algorithm has approximately the same accuracy as EvoFedNAS but EvoFedNAS has a much larger model size than our results. To sum up, the searching capability of our method surpasses previous decentralized NAS methods, and generally achieves the performance of the state-of-the-art centralized NAS. When the searched models are trained in an FL setting, the ones searched by our algorithm also enjoy superior performance than previous works [16], [20].

On non-i.i.d. datasets, we train the model distributedly on the same non-i.i.d. dataset using federated learning at P3. As shown in Table IV, our algorithm achieves higher evaluation accuracy than previous works [16], [17], [20], yet with much smaller model sizes. It can be concluded that our algorithm can find a well-performed model under the non-i.i.d. data distribution of the participants.

C. Efficiency

As mentioned before, the complexity of our algorithm at the local participant is proportional to $\frac{1}{N}$ of the super-net, which is N times more efficient than previous works such as FedNAS [17]. Empirically, the super-net in our experiment is 1.93MB, while our sub-model is 0.27MB on average, which is sufficiently light for deployment on mobile devices.

To verify the efficiency, we record the training time at the searching phase. We deploy our algorithm in a distributed setting, using a GTX 1080 Ti GPU as the server. We use GTX 1080 Ti GPUs and NVIDIA Jetson TX2 (TX2) as participants respectively, and the latter is a common edge device powered by GPU. We compared the search time in Table V. It only takes less than 2.5 hours on 1080 Ti and less than 10 hours on IoT devices, supporting that our algorithm is lightweight and can potentially be deployed on real-world IoT devices.

Adaptive Transmission. We choose 4G/LTE Bandwidth Logs [30] to simulate our network conditions. The 4G/LTE Bandwidth Logs collect real-world bandwidth measurements in 4G networks. The types of settings include on-foot, bicycle, train, bus, tram, and car, where mobile devices and IoT devices are most commonly used. We use the bandwidth in these logs as our participants' transmission conditions and the total number of participants is 10. For each round of the searching phase, we estimate the maximal and average

TABLE IV
FEDERATED EVALUATION ACCURACIES OF SEARCHED MODELS ON NON-I.I.D. DATASETS

Method	Error(%)	Param(M)	Strategy	NAS
	Non-i.i.d. CIFAR10			
FedAvg * [20]	22.40	58.2	hand	
FedNAS [17]	18.76	4.2	grad	✓
EvoFedNAS(big) [16]	18.73	-	evol	✓
EvoFedNAS(small)	21.06	-	evol	✓
Ours (non i.i.d.)	**18.56**	**3.9**	RL	✓
	Non-i.i.d. SVHN			
FedAvg *	10.78	58.2	hand	
Ours (non i.i.d.)	**10.23**	**2.5**	RL	✓

* Using ResNet152 [29] as the base model.

TABLE V
SEARCH TIME ON CIFAR10

Method	Search Time (hours)	Sub-net Size (M)
FedNAS * [17]	<5	1.93 [1]
EvoFedNAS [16]	16.1	4.23 [1]
Ours (1080Ti)	**<2.5**	**0.27**
Ours (TX2)	**<10**	0.27

* FedNAS [17] use16 RTX2080Ti GPUs as participants, and 1 RTX2080Ti as the server.
[1] The average size of sub-nets.

latencies of sending sub-models over all participants. Besides our adaptive approach, we implement two other methods for comparison: sending the average-sized models and randomly sending models. In particular, sending sub-models of the same size is adopted by previous works [16]–[18]. Fig. 7 shows our approach is superior to baselines in terms of the maximal latency. The average latency has the same trend and thus is omitted for saving space.

Delay-Compensation. Due to the complicated environmental factors and the mobility of participants, staleness persists in the real-world federated learning settings. To verify our delay-compensation scheme on federated model search, we design data distributions with staleness to simulate the real-world scenarios. 0% staleness means the server adopts hard synchronization and tolerates no staleness. 100% staleness means all data are out of date.

The first data distribution represents the case with severe staleness. We have 30% up-to-date data, 40% and 20% of the data respectively stale for 1 and 2 rounds, and the rest data exceeds the staleness threshold. We compare our delay-compensated scheme with two other common techniques — directly using stale data (use) and throwing it away (throw), as well as our same method without any staleness. The moving-average of the training accuracy in the searching phase is given in Fig. 8. Note that all the methods in Fig. 8 share the same warmed-up super-net, and hence the training curves have the same starting point as shown by the subgraph. The throwing-away strategy throws away a large proportion of updates on training data and hence yields the least accurate model among all. The algorithm is superior when directly using stale data,

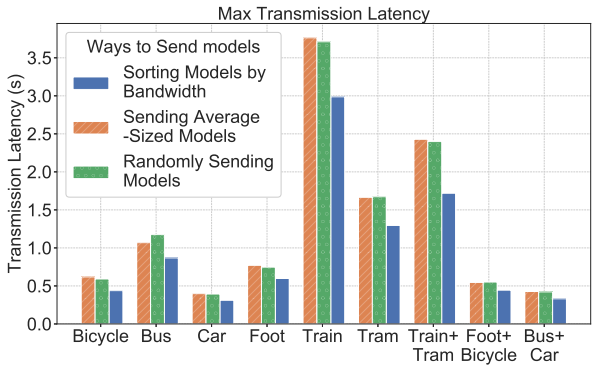

Fig. 7. Maximal transmission latency when sending a sub-net from the cloud to a participant in various network environments. "Bus+Car" means half of the participants are on buses and the other half are in cars.

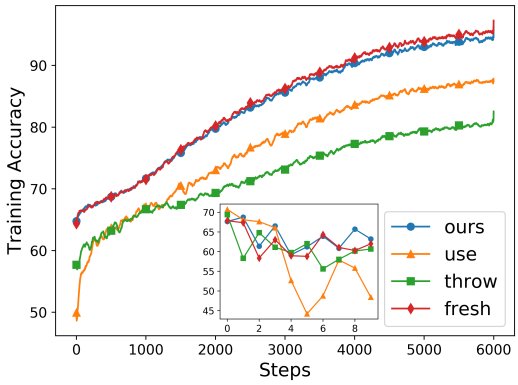

Fig. 8. Searching-Phase Performance on Stale Data (CIFAR10)

but is still inferior to our delay-compensated scheme. It is clear that our delay-compensated federated RL alleviates staleness and has the best searching performance among all, which is very close to the staleness-free case.

The second data distribution represents the case with slight staleness. We have 90% of data up-to-date, 9% and 0.9% of the data respectively stale for 1 and 2 rounds, while the rest data exceeds the staleness threshold. In this setting, our delay-compensated scheme brings limited searching performance enhancement but surprisingly finds the model with 2.59% testing error and 2.7MB size, which even outperforms the 2.62% error rate without any staleness. We think it may be because the introduction of delay-compensation may improve the generalization capability of the searched model.

The testing accuracy on the searched model in the evaluation phase is given in the second section of Table II and III. It is obvious that with delay-compensation, federated RL is still able to find highly accurate and small-sized models outperforming previous works with less time.

Not only is our algorithm efficient at the searching phase, but its searched models have better convergence performance overall. For straightforward illustration, we show the average training and validation accuracies of the participants versus the communication rounds over non-i.i.d. data at P3 where

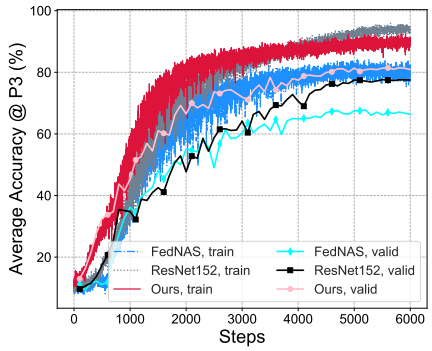

Fig. 9. Average Accuracy v.s. Rounds on Non-i.i.d CIFAR10

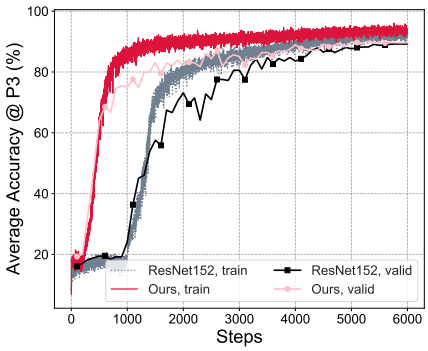

Fig. 10. Average Accuracy v.s. Rounds on Non-i.i.d SVHN

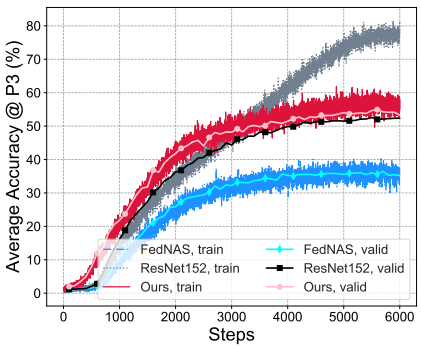

Fig. 11. Average Accuracy v.s. Rounds when Transferring models to Non-i.i.d. CIFAR100

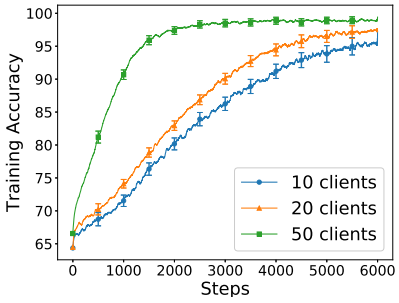

Fig. 12. Searching-Phase Performance v.s. Number of Participants

TABLE VI
RESULTS UNDER DIFFERENT NUMBERS OF PARTICIPANTS

# Part	Err(%)	Para(M)
10	2.62	3.6
20	2.88	2.9
50	2.67	3.8

TABLE VII
RESULTS OF TRANSFERRING FROM I.I.D. CIFAR10 TO I.I.D. CIFAR100

Method	Acc(%)	Para(M)
DARTS	82.99	3.4
FedNAS	80.28	4.2
Ours	**83.31**	3.6

TABLE VIII
RESULTS OF TRANSFERRING FROM NON-I.I.D. CIFAR10 TO NON-I.I.D. CIFAR100

Method	Acc(%)	Para(M)
FedAvg	52.57	58.2
FedNAS	36.01	4.2
Ours	**54.63**	3.9

the model is trained in a federated learning fashion. The results are shown in Fig. 9, Fig. 10, and Fig. 11. We compare our searched models with a pre-defined model (ResNet152) and the model searched by FedNAS. Our searched models converge within fewer communication rounds. In the case where we transfer searched models from CIFAR10 to non-i.i.d. CIFAR100, although our training accuracy is lower than the pre-defined model, our validation accuracy is higher, as the pre-defined model merely overfits the non-i.i.d. dataset. The set of results deliver convincing evidence that our searched models fit non-i.i.d. data better, with higher accuracies and faster convergence.

D. Number of Participants

We study the impact of the number of participants on our proposed algorithm. We choose 10, 20, and 50 participants, and equally, divide the CIFAR10 datasets among them. The moving average of the training accuracy in the searching phase is reported in Fig. 12. Since all participants perform the model search at the same time, the time cost for each round of updates is approximately the same, despite the number of participants.

From Fig. 12, it is easy to conclude that as the number of participants increase, the convergence speeds up. And the final searching-phase accuracy is promoted with more participants. As the error bars of each curve indicates, the fluctuation in participants' model accuracy decreases when there are more participants. We also present the best testing accuracies of the searched models with different numbers of FL participants in Table VI. It shows that the searched models achieve almost the same accuracy performance regardless of the number of participants, and though each local dataset is smaller with more participants. Jointly considering the searching-phase convergence rate, our algorithm performs well in large-scale settings.

E. Transferrability

Since the searching-phase is prolonged, it is often a choice to transfer the model learned on one dataset to another [7]–[9], [31], [32]. We study the generalization capability of the model searched by our RL-based algorithm by transferring the model structure searched on the i.i.d./non-i.i.d. CIFAR10 dataset to the i.i.d./non-i.i.d. CIFAR100 dataset. As shown in Table VII and Table VIII, our method provides satisfying transferability with competitive accuracies against other methods. Hence it is possible to perform our search algorithm over a small dataset and later transfer the model to a larger dataset.

VII. CONCLUSION

Since a pre-determined DNN model often fails to adapt to dynamic, unknown local data distribution under the FL framework, we propose a reinforcement learning based federated model search to automatically search for a best-fit model. Our algorithm adaptively distributes the training tasks of sub-models to participants, which is highly efficient in communica-

tion and computation. We also propose a soft synchronization scheme, in which we design a delay-compensated optimizer based on the second-order Taylor expansion to alleviate the staleness. Supported by abundant experiments, our algorithm outperforms the state-of-the-art methods in terms of efficiency and model accuracy, particularly on non-i.i.i.d. data.

REFERENCES

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2016.

[3] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2018.

[4] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018.

[5] F. Sattler, S. Wiedemann, K. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *CoRR*, vol. abs/1903.02891, 2019. [Online]. Available: http://arxiv.org/abs/1903.02891

[6] P. Kairouz, H. McMahan, B. Avent, A. Bellet, M. Bennis, A. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. D'Oliveira, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. Gibbons, M. Gruteser, and S. Zhao, "Advances and open problems in federated learning," 12 2019.

[7] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[8] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, "Darts+: Improved differentiable architecture search with early stopping," *ArXiv*, vol. abs/1909.06035, 2019.

[9] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," *ArXiv*, vol. abs/1911.12126, 2019.

[10] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 497–504. [Online]. Available: https://doi.org/10.1145/3071178.3071229

[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *arXiv e-prints*, p. arXiv:1802.01548, Feb. 2018.

[12] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," 2017.

[13] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[14] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=HylVB3AqYm

[15] M. Xu, Y. Zhao, K. Bian, G. Huang, Q. Mei, and X. Liu, "Federated neural architecture search," 2020.

[16] H. Zhu and Y. Jin, "Real-time federated evolutionary neural architecture search," *ArXiv*, vol. abs/2003.02793, 2020.

[17] C. He, M. Annavaram, and A. S. Avestimehr, "Fednas: Federated deep learning via neural architecture search," *ArXiv*, vol. abs/2004.08546, 2020.

[18] I. Singh, H.-Y. Zhou, K. Yang, M. Ding, B. Lin, and P. Xie, "Differentially-private federated neural architecture search," *ArXiv*, vol. abs/2006.10559, 2020.

[19] F. Hutter, L. Kotthoff, and J. Vanschoren, "Automated machine learning : Methods, systems, challenges." Springer Nature, 2019.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 4 2017, pp. 1273–1282.

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[22] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems (NIPS) 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231.

[23] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems (NIPS) 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 693–701.

[24] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 4120–4129.

[25] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, 1984, aAI8410337.

[26] R. Williams, "Towards a theory of reinforcement-learning connectionist systems," 01 1988.

[27] L. Weaver and N. Tao, "The optimal reward baseline for gradient-based reinforcement learning," 2013.

[28] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *ArXiv*, vol. abs/1708.04552, 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2016.

[30] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[31] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "Xnas: Neural architecture search with expert advice," in *Advances in Neural Information Processing Systems (NIPS) 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 1977–1987.

[32] X. Zheng, R. Ji, L. Tang, Y. Wan, B. Zhang, Y. Wu, Y. Wu, and L. Shao, "Dynamic distribution pruning for efficient network architecture search," *ArXiv*, vol. abs/1905.13543, 2019.