A Progressive Subnetwork Searching Framework for Dynamic Inference

Li Yang©, Graduate Student Member, IEEE, Zhezhi He©, Member, IEEE, Yu Cao©, Fellow, IEEE, and Deliang Fan©, Member, IEEE

Abstract—Deep neural network (DNN) model compression is a popular and important optimization method for efficient and fast hardware acceleration. However, the compressed model is usually fixed, without the capability to tune the computing complexity (i.e., latency in hardware) on-the-fly, depending on dynamic latency requirements, workloads, and computing hardware resource allocation. To address this challenge, dynamic DNN with run-time adaption of computing structures has been constructed through training with a cross-entropy objective function consisting of multiple subnets sampled from the supernet. Our investigations in this work show that the performance of dynamic inference highly relies on the quality of subnet sampling. To construct a dynamic DNN with multiple high-quality subnets, we propose a progressive subnetwork searching framework, which is embedded with several proposed new techniques, including trainable noise ranking, channel-group sampling, selective fine-tuning, and subnet filtering. Our proposed framework empowers the target dynamic DNN with higher accuracy for all the subnets sampled with prior works on both the Canadian Institute for Advanced Research dataset with 10 classes (CIFAR-10) and ImageNet datasets. Specifically, compared with United States-Neural Network (US-NN), our method achieves 0.9% average accuracy gain for Alexnet, 2.5% for ResNet18, 1.1% for Visual Geometry Group (VGG)11, and 0.58% for MobileNetv1, on the ImageNet dataset, respectively. Moreover, to demonstrate run-time tuning of computing latency of dynamic DNN in real computing system, we have deployed our constructed dynamic networks into Nvidia Titan graphics processing unit (GPU) and Intel Xeon central processing unit (CPU), showing great improvement over prior works. The code is available at https://github.com/ASU-ESIC-FAN-Lab/Dynamic-inference.

Index Terms—Deep neural network (DNN), dynamic inference, dynamic network, subnet searching.

I. INTRODUCTION

STATE-OF-THE-ART deep neural networks (DNNs) grow into more and more complex structures consisting of deeper layers, larger model size, and denser connections. Such “bulky” models rise challenges to their computing deployment into hardware, for both the edge- and cloud-computing systems. The most common solution is to compress the target DNN for hardware-efficient deployment, optimizing for latency, model size, throughput, etc. [1], [2], [3], [4], [5]. Traditionally, such model compression leads to a static/ixed compressed model that is not capable of adjusting or reconfiguring its computation complexity (i.e., inference structure, latency) in run time, depending on the dynamic hardware resource allocation or environment constraint, after model deployment.

To equip the DNN inference model with run-time adaption, dynamic DNN is proposed with empowered dynamic inference capability. The first approach is called input-dependent dynamic DNN [6], [7], [8], where one specific subnet, sampled from a backbone supernet, is selected as the inference path on-the-fly w.r.t. the current input. Such input-dependent subnet selection can be achieved via the controller module [6], [7], [8] or inserting a cascade of classifiers operating on the features of internal layers [9]. The second approach is called input-agnostic dynamic DNN [10], [11] that aims to only compute proper subnet(s), sampled from a backbone supernet, controlled by run-time dynamic hardware computing resources, application/user speed requirements, dynamic workloads, etc.. Note that unlike input-dependent dynamic DNN, the run-time network architecture adaption of such input-agnostic dynamic DNN does not depend on the varying inputs. This scheme is mainly motivated and required by real-world DNN computing systems and applications. For example, different hardware platforms have nonidentical computing power and resources, which require different degrees of model compression under similar latency requirement, e.g., in real-time data processing. Even for the same hardware platform, the allocated computing resources for DNN application are dynamic, depending on the run-time system configuration. For example, smartphone or laptop may become too hot or run out of battery, resulting in different allocated computing resources for DNN computation. For such complex and dynamic real-world scenarios, it is not practical and efficient to train many different DNN models with different sizes for the same function, to meet all those dynamic computing environments.

To tackle these challenges, Yu et al. propose the slimmable neural network (S-Net) [10] and its optimized counterpart (United States (US)-Net) [11], which can switch the inference structure (i.e., subnet) among the predefined subnet candidates sampled from a supernet in an input-agnostic and run-time fashion, to dynamically tune computing complexity/latency and inference accuracy. Note that the predefined subnets in S-Net/US-Net [10], [11] are naively sampled by multiplying the original channel width w.r.t. a multiplier. Such channel width multiplier is uniformly applied on all the layers (e.g.,
To enhance searching quality, we propose a new technique to address the above challenges. First, with multiple high-quality subnets, which consists of different subnetwork searching framework to construct dynamic DNN. On small Canadian Institute for Advanced Research dataset, processing unit (GPU) hours for subnet searching of ResNet20. Nevertheless, there still remain several challenges to overcome while incorporating the progressive pruning as the progressive nonuniform subnet searching for dynamic neural network construction. It could be summarized as:

1) Searching Quality: the traditional vanilla progressive pruning normally adopts the $L^p$-norm of weights ($\|W\|_1$ or $\|W\|_2$) as the importance criterion to drop weights channelwise, which can be further optimized for identifying subnets with higher quality (i.e., higher accuracy for the same subnet size) for constructing dynamic DNN.

2) Searching Cost and Scalability: the searching cost of progressive subnet searching is very high e.g., 35.8 graphics processing unit (GPU) hours for subnet searching of ResNet20 on small Canadian Institute for Advanced Research dataset with 10 classes (CIFAR-10), thus counteracting the scalability issue.

In this work, we propose an efficient and fast progressive subnetwork searching framework to construct dynamic DNN with multiple high-quality subnets, which consists of different new techniques to address the above challenges. First, to enhance searching quality, we propose a new trainable-noise-ranking-based searching method, consisting of two steps. The first step is trainable noise injection, where we inject channelwise zero-mean Gaussian noise with trainable magnitude upon weight by fine-tuning the initial model. It is worthy to note that we are the first to discover that such trainable channelwise noise magnitude could serve as an alternative and better criterion to indicate the “importance” level of weight parameters in progressive pruning, compared with the traditional $L^p$-norm. The second step is to leverage such trained variance as a weight importance criterion for conducting fast progressive subnet searching, named as trainable noise ranking. Second, to further boost the searching speed, we also optimize the granularity per searching iteration from single channel to multiple channels as a channel group, combined with selective fine-tuning technique. In summary, our technical contributions include:

1) We propose an efficient, low searching cost, and fast progressive subnetwork searching framework to construct dynamic DNN with multiple high-quality subnets. A series of novel techniques are developed and embedded within this framework, including trainable-noise-ranking-based progressive searching, channel-group sampling, selective fine-tuning, and subnet filtering. Note that those terms will be defined and discussed in Section III. Given a target DNN, our framework could empower it input-agnostic dynamic inference capability which can dynamically trade-off computing complexity (i.e., latency in target hardware platform equivalently) and accuracy on-the-fly via switching among multiple sampled subnets.

2) Taking object classification as a study case, our framework outperforms prior works via comprehensive experiments on both the CIFAR-10 and ImageNet datasets, in terms of subnet accuracy and inference latency measured in real computing devices (central processing unit (CPU) and GPU).

3) Heuristic discussion with experiments is provided to explore optimal subnet structures for different networks, e.g., MobileNet, ResNet, AlexNet, and Visual Geometry Group (VGG).

II. RELATED WORKS AND BACKGROUND
A. Dynamic Neural Network
Dynamic DNNs can mainly be categorized into input-dependent and input-agnostic approach. Many works [6], [7], [8], [9] have been proposed to study the input-dependent dynamic DNN which could adapt the inference structure per input sample. [16] proposed to predictively amplify salient convolutional channels and skip unimportant ones at run time in an input-dependent way by introducing small auxiliary connections to the existing convolutional layers. [17] dynamically removes redundant filters by embedding the manifold information of all instances into the space of pruned networks. In addition, [18] proposed conditionally parameterized convolutions (CondConv), which learn specialized convolutional kernels for each example. Replacing normal convolutions with CondConv enables us to increase the size and capacity of a network, while maintaining efficient inference. Different from that, our work focuses on the input-agnostic dynamic DNNs first defined by S-Net [10] and its optimized counterparts [11], [19], [20], [21].

Here, we first explain the S-Net [10], where a user can switch the inference structure among $N$ predefined subnets ($N = 4$ in [10]). In S-Net, each subnet owns different computing complexity and accuracy. Generally, the accuracy of each subnet is proportional to its model size and computing complexity. The method adopted by S-Net is quite straightforward and described as follows:

1) Uniform Subnets Generation: The S-Net first generates $N = 4$ subnets from the initial full-size model (aka. supernet).
Each $i$-indexed ($i \in \{1, 2, 3, 4\}$) subnet is acquired by uniformly applying a multiplier $m_i \in \{0.25, 0.5, 0.75, 1.0\}$ on the input/output channels of the entire supernet. Taking one fully connected layer in the supernet with weight matrix $W \in \mathbb{R}^{p \times p}$ as an example, the weight matrices in this layer of four subnets $\{W_1, W_2, W_3, W_4\}$ are in the shape of

$$\{\mathbb{R}^{(0.25 \times p)^2}, \mathbb{R}^{(0.5 \times p)^2}, \mathbb{R}^{(0.75 \times p)^2}, \mathbb{R}^{(1.0 \times p)^2}\}$$

s.t. $W_1 \subseteq W_2 \subseteq W_3 \subseteq W_4$ (1)

where $p$ is the input and output size. For simplicity, the weight tensor sets of the $i$th subnets is denoted by $\{W_i\}$. 2) Multiobjective Training: To enable the S-Net switch among subnets with uncompromised individually inference accuracy, the S-Net trains the target DNN by conventional back-propagation with a multiterm objective function, which can be expressed as

$$\min \mathbb{E}_X \left( \sum_{i=1}^{N} \mathcal{L}(f(X, \{W_i\}); T) \right)$$

where $X$ is the minibatch of inputs with the corresponding targets $T$. $\mathcal{L}(\cdot; \cdot)$ calculates the cross-entropy loss of DNN output and target. $f(X, \{W_i\})$ computes the output of the subnet parameterized by $\{W_i\}$.

Through the above two sequential steps, with single DNN as supernet, the S-Net can switch among its subnets on-the-fly. To further improve S-Net performance, Yu et al. also proposed several techniques in their extended work called universally slimmable networks (US-Net) [11], including in-place distillation, poststatistics of batch normalization, etc. Based on S-Net and US-Net, [22] further designed a double-headed dynamic gate function with an attention head and a slimming head upon slimmable networks to achieve input-dependent dynamics.

B. Model Pruning

Model pruning [23] is an important technique in DNN model compression, which aims to reduce the model size to execute in hardware with less computation workload. Pruning can be generally categorized into unstructured pruning and structured pruning. The unstructured methods [23], [24], [25] aim to prune the model elementwise, which can achieve higher sparsity than structured pruning under the same accuracy. However, the specialized hardware or instruction library is needed to demonstrate real hardware level speedup due to the irregular sparsity pattern. Differently, structured pruning [1], [2], [3], [4], [5], [26] prunes the unimportant weights in structured pattern (e.g., channelwise, filterwise) that can show actual speedup in a general-purpose processor. In this work, we focus on the channelwise structured pruning methods, where the basic weight group to be dropped is the entire output channel for convolution and fully connected layers. Recently, many important metrics for channelwise pruning have been proposed. [1], [2], [3], [4], [5] normally applied a weight penalty term (e.g., group Lasso [2], [27]) in the objective function or differentiable masking function on weights, during the training process. [28] proposed a filter pruning method which determines the relative importance of filters by observing the rank of feature maps. [29] proposed an interchannel perspective-motivated metric to evaluate the importance of filters for network pruning.

More related to this work, progressive pruning [30], [31] usually takes a pretrained model as initialization, and then gradually shrinks the full-size model, which can generate a family of pruned subnets with different model sizes. Given a target DNN with $L$ layers and its $i$th layer ($i = 1, 2, \ldots, L$) with $p_i$ output channels, the weight tensors of the entire DNN $\{W_j\}_{j=1}^{L}$ can be refactorized into $\{W_j\}_{j=1}^{L}$. For each iteration of progressive pruning, it attempts to zero-out one channel of weights $W_k, k \in \{1, 2, \ldots, \sum p_i\}$, while minimizing the potential accuracy degradation. Such process can be described as

$$\arg \min_k \text{ACC}_\text{val}(f(X, \{W_j\}_{j=1}^{L} \setminus W_k); T)$$

where \ $\setminus$ denotes the element exclusion from set. As the method adopts brute force to prune the target DNN channel by channel, the computation cost is $\sum p_i (\sum p_i + 1)/2$. Note that such progressive pruning is similar to iterative pruning [32] which iteratively prunes the model until reaching the target accuracy within the hardware constrain (e.g., Flops, memory). The progressive pruning owns the following merits: 1) it can produce multiple pruned subnets since the full-size supernet is progressively shrunk and 2) the smaller pruned network $A$ is certainly the subset of its larger counterpart $B$ (i.e., $W_A \in W_B$). Due to this, Li et al. proposed a nonuniform subnets’ sampling method [21] based on S-Net by adapting regularization-based channelwise pruning [27]. Such method needs to repeteadly fine-tune the model for each subnet, which suffers from large training cost and has limited subnets’ capacity (i.e., 4). More recently, inspired by the multiobjective training method, once-for-all net (OFA) [31] is proposed to construct a dynamic DNN that samples much larger number of subnets across many more dimensions (depth, width, kernel size, and resolution).

III. PROGRESSIVE SUBNETWORK SEARCHING

Fig. 1 illustrates the overview of the proposed method, which can be divided into three successive steps: trainable noise injection, progressive sampling, and fused training. In this work, we aim to search multiple high-quality subnets with minimized searching cost for dynamic inference. We propose different novel techniques to overcome the below three challenges in this process.

1) How to search multiple subnets?
2) How to improve the quality (i.e., higher quality means higher accuracy with the same model size or computing [floating-point operations per second (FLOPS)] of identified subnets?)
3) How to improve the searching speed, while maintaining the subnet quality?

First, we propose to leverage progressive pruning as the backbone technique of subnet searching, as it can produce multiple subnets while the smaller pruned network $A$ is the subset of its larger counterpart $B$ (i.e., $W_A \in W_B$). This property is vital to maximize the accuracy of the target dynamic neural network, since the weights of all the subnets are
Then we illustrate the steps of the proposed framework as shown in Fig. 1 in the following three successive steps.

1) In the first step, we fine-tune the initial model with trainable noise injection channelwise, which serves as a criterion to indicate the importance level of channels in progressive sampling.

2) In the second step, given a supernet, we progressively sample nonuniform subnets with different model sizes to form a subnet pool for next step dynamic network training. During progressive sampling, for any iteration, each candidate subnet is obtained through pruning one channel group (consisting of certain number of channels depending on setting) from one layer, where the proposed trainable noise ranking method is used to determine the priority (i.e., higher trainable noise factor has a higher priority to be pruned) of channel group to be pruned for each layer. Meanwhile, its corresponding inference accuracy after pruning is also recorded. Thus, for each iteration, the number of candidate subnets equals to the number of layers. After collecting all the pruned candidate subnets for this iteration, the one with the maximum evaluated inference accuracy as mentioned in (3) will be the winner of this iteration and fed into the subnet pool. Then, it will enter the next progressive sampling iteration with this winner as the new supernet to generate the next winner subnet sent to the subnet pool. The progressive sampling will stop when it reaches the predefined lower bond (25% for each layer in this work following the same setting in [11]). At the end of progressive sampling step, a subnet pool will be generated for the fused training for dynamic inference in the next step.

3) In the third step, we further optimize the subnet pool to guarantee that the evaluated accuracy of larger subnet is always higher than smaller subnets, which we call subnet reselection. Then, the initial model which includes these filtered subnets is trained via an ensemble loss for multiple objective optimizations as expressed in (2). Note that all the subnets partially share the weights of the initial model as indicated by the overlapped channel index. Finally, the trained model can act as a dynamic model whose subnets can perform inference independently at different speed and accuracy.
A. Step-1: Trainable Noise Injection

Training network with weight noise injection has been used as an effective technique to perform model regularization, thus improving model robustness against adversarial input noise injection [33], [34]. But different from aiming to improve model robustness, we are the first to propose that the magnitude of such trainable weight noise can also be used for subnet sampling or pruning from a given DNN. In practice, we introduce the channelwise Gaussian noise to the pretrained model for both the convolutional and fully connected layers, and then the trainable noise variance (i.e., magnitude) is used for channel importance ranking in subnet sampling. The weight noise injection during the initial model fine-tuning can be mathematically described as

\[
\tilde{w}_{l,i} = w_{l,i} + \beta_{l} \cdot \eta_{l,i}; \quad \eta_{l,i} \sim N(0, \sigma_{w_{l,i}}^2) \tag{4}
\]

where \( w_{l,i} \) is the \( i \)th channel of the noise-free weight \( w_l \) in the \( l \)th layer. \( \eta_{l,i} \) is the noise term samples from the Gaussian distribution with zero mean. It shares the same variance \( \sigma_{w_{l,i}}^2 \) of the weight \( w_{l,i} \) in training. \( \beta_{l} \) is the channelwise noise magnitude that scales the injected noise \( \eta_{l,i} \). Note that we adopt the scheme that \( \eta_{l,i} \) shares the identical weight variance with \( w_{l,i} \) as in (equation 4), and thus, the injected additive noise relies on \( \beta_l \) and the distribution of \( w_{l,i} \) simultaneously.

Before starting subnets’ searching, to get the trainable parameter \( \beta \) (i.e., noise injection as shown in Fig. 1), we first fine-tune the initial DNN model using the above discussed trainable noise injection method as shown in (4). In this procedure, \( \beta \) can be quickly converged in a few epochs. In this work, we only fine-tune 20 epochs with 0.01 learning rate in all our experiments. Then, this trained \( \beta \) is used as the ranking metric to quickly sample subnets. Note that \( \beta \) is fixed during the following progressive subnets sampling step.

Theoretical Discussion: In terms of model robustness, the previous work [35] proves that the additive weight noise can indicate the sensitivity of the loss function w.r.t. the learned weight, which can be formulated as

\[
\mathcal{L}(w + \epsilon) \approx \mathcal{L}(w) + \epsilon \cdot \nabla_w \mathcal{L}(w) \tag{5}
\]

where \( \epsilon = \beta \cdot \eta \) in our case as defined in (4). When we minimize the loss function \( \mathcal{L}(w + \epsilon) \), the second term on the right-hand side would minimize the sensitivity of the function w.r.t. the corresponding weight that results in model robustness improvement. In our case, since we define the scaling factor \( \beta \) as a channelwise learnable parameter, it could indicate the sensitivity of the weight channels. In practice, smaller \( \beta \) means the corresponding weight channels are robust and the second terms would help reduce the loss error.

B. Step-2: Progressive Sampling

The progressive sampling aims to iteratively generate a subnet pool with different subnet model sizes. To achieve this, for each searching iteration, we first sample the candidate subnets through pruning one channel group with largest trainable noise factor for each layer. Meanwhile, the corresponding evaluated inference accuracy will be recorded. Then, for each iteration, \( l \) number of pruned candidate subnet will be generated for a supernet with \( l \) layers. Next, the pruned candidate subnets with the maximum evaluated accuracy will be the winner for this iteration and will be sent to the subnet pool for fused training as shown in the progressive sampling step of Fig. 1. In the end, the current subnet winner of current iteration will become the new supernet for the next iteration sampling. Note that we define the lower bound of the channel multiplier as 0.25× for all the layers consistently, meaning that one specific layer will stop sampling if it is pruned to 0.25× of its full channel numbers. As a result, the whole progressive sampling step will stop if all the layers are pruned to 0.25×. Next, we will introduce several new techniques we proposed during progressive sampling.

1) Trainable Noise Ranking: Using noise magnitude \( \beta \) to prune weight channels is guided by the following two properties: 1) it is a channelwise parameter that can be automatically updated according to the current weight distribution during initial fine-tuning. Thus, the trainable noise magnitude \( \beta \) is different among channels and 2) the impact of \( \beta \) is to scale the magnitude of the corresponding noise, which represents the strength of noise. For example, the weight channel with larger value of \( \beta \) means that stronger regularization is needed to keep robustness and accuracy. It implies that this weight channel is not as important as the one with smaller trained noise magnitude. So we conjecture that “the larger value of coefficient \( \beta_{l,i} \) represents the corresponding weight channel is less important.” Based on this hypothesis, we apply the channelwise \( \beta \) to guide subnet searching. As shown in Fig. 1, the weight channel with larger \( \beta_{l,i} \) is pruned first in each layer.

The progressive searching via trainable noise ranking can be defined as

\[
\arg \min_{k \ast} \text{ACC}_\text{val}(f(X, \{W_j\}_{j=1}^{\sum p_l} \setminus W_k); T) \tag{6}
\]

s.t. \( k \ast = \arg \max_{k} \beta_{l,k} \).

Note that we introduce (4) to the pretrained model, and then only the parameter \( \beta \) is used to select the channel to be pruned. Our experiments show that trainable noise ranking can achieve the same or better performance than typical norm-based ranking in pruning, especially on larger network, which will be discussed in detail in Section IV.

2) Channel-Group Sampling: To further reduce the searching space and improve the searching speed, we define the searching granularity per iteration as channel group, which may be set as either one channel or a group of multiple channels. To implement so, we rank the weight channels via trainable noise magnitude \( \beta_{l,i} \) and combine the adjacent several channels to be one group. Then one channel group will be pruned in each searching iteration, instead of one channel. For generalization, we name the group \( G = (\text{total channel number})/(\text{group size}) \) as the total number of groups per layer, which could be the power of 2 (e.g., 4, 8, 16) for efficient computing in hardware, and it is identical for all the layers. The group size will be different for each layer to guarantee \( G \) is the same. Then, given a target DNN with \( L \) layers and \( G \) groups per layer, the searching cost is reduced to \( L \times G \) from \( L \times \sum p_l \).
3) Selective Fine-Tuning: Moreover, after each progressive sampling iteration, it is a common setting to fine-tune the current model to optimize the unpruned weight before going to the next sampling iteration [32]. However, fine-tuning, which retrains the current model with lower learning rate (e.g., 0.01) with a few epochs, is time-consuming especially for a large dataset (e.g., ImageNet). To reduce the fine-tuning time cost, we propose a technique called selective fine-tuning, which sets an accuracy target as a threshold, where the sampled subnets will only be fine-tuned when its accuracy is lower than that. It is motivated by our observation that fine-tuning is not necessary for large subnet with higher accuracy. Our experiments also indicate that such selective fine-tuning method can further reduce the sampling cost without influencing the performance of dynamic inference, which will be elaborated in the experiment section.

C. Step-3: Fused Training

1) Subnets Filtering: After progressive searching, all the pruned subnets are fed into the subnet pool. These subnets are sampled using (6) gradually, which can be considered as “optimal” structures intraiterations, but not interiterations. Due to model redundancy, the accuracy of smaller sampled subnets $\mathcal{A}$ may be higher than a larger subnet $\mathcal{B}$ (i.e., $W_\mathcal{A} \in \mathcal{W}_\mathcal{A}$, but $\text{Acc}_\text{val}(W_\mathcal{A}) > \text{Acc}_\text{val}(W_\mathcal{B})$). We found that it will affect the overall performance of dynamic DNN training. As a countermeasure, we add a constraint, called subnets filtering, to remove the larger subnets with lower accuracy w.r.t. smaller counterparts (e.g., $W_\mathcal{B}$). The constraint can be formulated as

$$\text{ACC}_\text{val}(f(X, W_{[i,...,i-1]}); T) > \text{ACC}_\text{val}(f(X, W_i); T) \quad (7)$$

where $i$ is the subnets’ index.

2) Multiterm Training: Then after subnet filtering, the remaining subnets will be applied to fused training using multiple-term objective optimization for dynamic inference as discussed in (2). Training such a dynamic network also has a common problem of discrepancy of the feature mean and variance across different subnets, leading to inaccurate statistics of shared batch normalization layers [36]. To solve this problem, we follow the same solution in US-Net [11], which assigns dedicated batch norm mean and variance statistics for each sampled (i.e., pruned) subnet during fused training. When a dynamic DNN is finally constructed, benefiting from the subnets’ partial sharing, these subnets can switch between each other on-the-fly and perform inference independently at different speed, accuracy, and power consumption in hardware. The algorithm for the overall proposed method is listed in Algorithm 1.

### Algorithm 1 Proposed Progressive Subnetwork Searching for Dynamic Inference

**Require:** Given a pretrained model $W$ with $L$ layers, the defined group channel $G$, the model size constraint $M^*$, the channel group $\mathcal{G}$, and the accuracy threshold $\gamma$, and noise variance $\beta$.

1: $\text{Acc} = 0$
2: $\text{subnets}_{\text{pool}} = []$
3: $i = 0$
4: $M_i = \text{ModelsizeMeasurement}(W)$
5: while $M_i > M^*$ do
6:   for $j \leftarrow 0, L$ do
7:     $W_{ij} = \text{TrainableNoiseRanking}(W_{ij}, G, \beta_{ij})$
8:     $\text{Acc} = \text{ACC}_\text{val}(f(W_{ij}))$
9:     if $\text{Acc} > \text{Acc}$ then
10:        $\text{Acc} = \text{Acc}$
11:        $W^* = W_{ij}$
12:   end if
13: end for
14: $M_{i+1} = \text{ModelsizeMeasurement}(W^*)$
15: $\text{subnets}_{\text{pool}}.\text{append}(W^*)$
16: if $\text{Acc} < \gamma$ then
17:      Fine-tuning($W^*$)
18: end if
19: $i+ = 1$
20: end while
21: $\text{subnets}_{\text{pool}} = \text{SubnetsReselection(}\text{subnets}_{\text{pool}}\text{)}$
22: MultitermTraining($\text{subnets}_{\text{pool}}$)

1) Subnets Searching: For CIFAR-10, we randomly choose 5000 images from the validation dataset to perform validation. The accuracy threshold is set to be 50%, and the channel group is four for ResNet20 and MobileNetV1, and eight for VGG11. For ImageNet, we validate the subnet accuracy on 10000 random picked images from the validation dataset. The accuracy threshold is 40%, and the channel group is four for ResNet18, and eight for MobileNetV1 and VGG11-BN.

For all the experiments, we use FLOPS to indicate the computing complexity of subnets. To conduct a fair comparison, for all the competing methods, the subnets generated from the same supernet architecture (e.g., VGG, ResNet, MobileNet) have the same upper bound (i.e., 1.0× channel multiplier) and lower bound (i.e., 0.25× channel multiplier). Thus, the upper bond and lower bond of computing FLOPs of all the methods are the same. The main performance metric is the accuracy comparison between the subnets (with the same or very similar FLOPs) generated from different methods. We also reported the searching time of each method and their measured subnet execution latency in real-world CPU/GPU.

2) Fused Training: The minimum subnet model size is constraint to 0.25× of complete model. For ResNet18 on the ImageNet dataset, we train the network using a momentum stochastic gradient descent (SGD) optimizer, where the initial learning rate is 0.1, and then scaled by 0.1 at epochs 30, 60, and 80, respectively. For AlexNet and VGG11-BN, we use the same configuration as [43], which choose momentum
TABLE I
TRADE-OFF BETWEEN ACCURACY (%) AND FLOPS (M) FOR RESNET20, ALEXNET, AND VGG11-BN ON CIFAR-10

| Network    | US-Net [11] | Ours(L1 Norm) | Ours(Weight Noise) |
|------------|-------------|---------------|--------------------|
|            | Acc | FLOPs | Acc | FLOPs | Acc | FLOPs |
| ResNet20   | 91.29 | 40 | 91.28 | 40 | 91.64 | 39 |
| ALEX Net   | 90.89 | 36 | 91.12 | 36 | 91.39 | 34 |
| VGG11-BN   | 89.51 | 31 | 91.28 | 29 | 91.38 | 28 |
| ResNet18   | 89.99 | 26 | 90.92 | 25 | 90.93 | 23 |
| Average    | 89.21 | 19 | 90.16 | 18 | 89.84 | 17 |
| MobileNetV1| 87.72 | 16 | 88.75 | 14 | 88.44 | 14 |
| Average    | 86.55 | 10 | 88.23 | 9  | 88.00 | 9  |
| VGG11-BN   | 83.83 | 5  | 84.22 | 4  | 84.40 | 3  |
| Average    | 88.76 | 12 | 89.62 | 9  | 89.71 | 7  |

Fig. 2. Trade-off between accuracy and FLOPs for different dynamic networks on CIFAR-10.

B. Main Results

1) Uniform Versus Nonuniform: Tables I and II depict the subnet accuracy and FLOPs trade-off for different networks with dynamic inference on the ImageNet and CIFAR-10 datasets, respectively. Our proposed dynamic network consists of multiple nonuniform subnets sampled through either trained noise ranking or L1-norm-based progressive search. We mainly compare it with the state-of-the-art US-Net [11], which contains uniform subnets. It is clear that the nonuniform subnets sampled based on either L1-norm or our trainable noise ranking provide much better accuracy than the uniform subnets with the same subnet model size, across different networks.

2) Trainable Noise Ranking Versus L1-Norm Ranking: As discussed in the related work section, Lp-norm ranking is a popular metric used in searching nonuniform subnets, e.g., in OFA Net [31]. To demonstrate the efficacy of our proposed trainable noise ranking, we also compare it with L1-norm ranking, as depicted in Table I, Fig. 2 and Table II. It is noteworthy that from the detailed quantitative results, we observe that trainable noise ranking outperforms L1-norm ranking and US-Net in almost all the subnets. It clearly shows that our trainable noise-ranking-based searching mechanism could lead to higher quality of subnets than both the uniform and L1 norm-based ranking counterparts. Note that the model size M for different networks can be formalized as: M(ResNet20) < M(MobileNetV1) < M(VGG11) on CIFAR-10, and M(AlexNet) < M(ResNet18) < M(VGG11-BN) on ImageNet. It is also intriguing to find that our trainable noise ranking method works much better in larger networks, such as VGG, compared with the smaller counterparts as shown in Fig. 2. In addition, due to the extreme large searching cost for the traditional progressive searching method [32], we compare its performance with other methods.
TABLE III
SEARCHING COST OF SIX NETWORKS WITH DIFFERENT PARAMETERS

| Network  | GPU-hours | Group setting/No. sub-nets |
|----------|-----------|---------------------------|
|          | Traditional progressive | $L^1$-norm ranking | Noise ranking |
| ResNet20 | 35.8      | 0.23                      | 0.26          | Group 4 / 57 |
| MobileNetv1 | 652.0 | 0.30                      | 0.31          | Group 4 / 41 |
| VGG11    | 41.9      | 0.09                      | 0.09          | Group 8 / 59 |
| ResNet18 | 2.5 x 10^4 | 7.1                     | 7.0           | Group 4 / 30 |
| AlexNet  | 1.2 x 10^5 | 7.2                     | 7.2           | Group 3 / 41 |
| VGG11-BN | 9 x 10^5  | 10.9                     | 10.7          | Group 8 / 60 |

TABLE IV
COMPARING TRAINABLE NOISE RANKING AS A PRUNING METHOD WITH OTHER POPULAR METHODS IN THE IMAGENET DATASET

| Model   | Method  | Top-1 Prune Acc | Acc Drop | PLOPs | Prune Ratio |
|---------|---------|----------------|----------|-------|-------------|
| ResNet-18 | LCCL [44] | 66.33% | 3.65% | 1.19E9 | 34.6% |
|         | SEP [55] | 67.10% | 3.18% | 1.06E9 | 41.8% |
|         | PFPM [5]  | 68.41% | 2.87% | 1.06E9 | 41.8% |
|         | TAS [46] | 69.15% | 2.50% | 1.21E9 | 33.3% |
|         | Ours     | 69.05% | 2.69% | 1.21E9 | 33.3% |

using ResNet20 on CIFAR-10 as illustrated in Fig. 2(a). It shows that our proposed method could achieve similar accuracy for different subnets. But the searching time of the traditional progressive searching is 35.8 GPU hours, and our framework only takes 0.26 GPU hours, which is around 137× speedup for the subnet searching process.

3) Searching Speed Comparison: Our another objective is to reduce the searching complexity, thus to speed-up the searching process, especially compared with the traditional progressive searching based on brute-force pruning. Note that US-Net does not have this searching step, since it predefines the uniform subnets before fused training as introduced in Section II. Different from that, we aim to search the nonuniform subnets that could generate higher quality subnet pool and further reveal the layer sensitivity of different models (Section IV-C.) Table III lists the time required for the traditional progressive searching, trainable noise ranking, and $L^1$-norm ranking on CIFAR-10 and ImageNet, respectively, on four-way NVIDIA Titan-Xp GPUs. It is noteworthy that other proposed techniques, such as channel-group sampling and selective fine-tuning, are applied to trainable noise ranking and $L^1$-norm ranking. So they have almost the same searching cost. It can be easily seen that the traditional progressive searching method takes significantly more time compared with the ranking-based searching methods based on either trainable noise ranking or $L^1$-norm.

4) Trainable Noise Ranking Used in Static Network Pruning: To further show that our proposed trainable noise ranking method can lead to high-quality subnet structure, we randomly select one sampled subnet from our method and retrain it as a new conventional static pruned model to compare with other popular static channel pruning methods. As shown in Table IV, we could achieve state-of-the-art performance compared with other recent network pruning methods.

Fig. 3. Run-time tuning between latency and accuracy in Titan GPU and Xeon CPU, for (top) ResNet-20, (middle) MobileNetv1, and (bottom) VGG11 on CIFAR-10.

Fig. 4. Measured run-time tuning between latency and accuracy in Titan GPU and Xeon CPU, for (top) ResNet-18, (middle) AlexNet, and (bottom) VGG11 on ImageNet.

5) Executing Constructed Dynamic DNN in Real CPU and GPU Hardware: To demonstrate that our constructed dynamic DNN could achieve run-time tuning of computing latency in real-word computing hardware, such as CPU and GPU, we deploy the constructed dynamic inference model based on different network structures into Nvidia Titan-Xp GPU and Intel Xeon CPU for the CIFAR-10 and ImageNet datasets. Its run-time latency and accuracy measured in real hardware are shown in Figs. 3 and 4. It can be seen that our proposed
trainable noise ranking method outperforms the state-of-the-art US-Net by a large degree in both CPU and GPU execution, showing better accuracy with the same latency or smaller latency with the same accuracy. In addition, compared with $L^1$-norm ranking, our trainable noise ranking could also achieve better accuracy under the same latency in almost all the cases.

C. Discussion and Ablation Study

1) Impact of Channel-Group Sampling: Here, we explore how channel-group sampling will influence the performance using ResNet20 on the CIFAR-10 dataset. Four different channel-group settings with the corresponding searching costs and subnet accuracy are shown in Fig. 5. We define group $G$ as the number of groups in each layer. For example, Group 4 represents the output channel of the layer is divided into four groups. We observe that configuring the group size as 4, 8, or 16 achieves almost similar accuracy, in contrast to Group 2 with much worse performance. We believe that the accuracy of the sampled subnet is robust to channel granularity due to channel redundancy. For example, the number of channels is 4096 and 1024 in the last several layers for Vgg and MobileNetv1, respectively. In the extreme case, if we set the group size to the number of channels for each layer (i.e., layerwise pruning), it will greatly reduce the accuracy of each subnet. Based on this, we show that there should exist a sweet point for the group size setting that could achieve good accuracy with minimum searching cost. However, it is obvious that smaller group size leads to a faster searching speed. For example, Group 2 requires around 0.17 h of searching time, while Group 16 requires 0.88 h of searching time for this particular example. To get a balance between the searching speed and accuracy performance, the group size is set as four or eight in our work as defined earlier for different experiments.

2) Threshold of Selective Fine-Tuning: During the progressive subnet search, we use a selective fine-tuning method that only fine-tunes the network if its accuracy is lower than a preset threshold. To demonstrate its effect on overall performance, we vary such accuracy threshold to be 80%, 50%, and 20% on the CIFAR-10 dataset, as depicted in Fig. 6. Considering the searching time cost, it is obvious that a higher threshold will require more fine-tuning and thus higher time cost during searching. From example, the searching time increases from 0.28 to 0.78 h if increasing the tuning threshold from 20% to 80%. It is interesting to see that the tuning accuracy thresholds of 80% and 50% lead to almost similar accuracy versus FLOPS trade-off, compared with the threshold of 20%. It indicates that searching with intermittent subnets’ fine-tuning is beneficial to identify better subnets, at the cost of extra computations.

3) Subnet Filtering: As discussed earlier, we also adopt a technique called subnet filtering to guarantee that a larger subnet always has higher accuracy than a smaller subnet in the pool. To demonstrate its effect on the overall performance, we conduct experiments to construct dynamic inference with and without subnet filtering, as shown in Fig. 7. It clearly shows that with subnet filtering, we could get better accuracy for all the subnets with different FLOPS, as well as eliminating the cases where larger network has smaller accuracy. We believe that this technique is critical since weights are partially shared between subnets during fused training. If a nonoptimal subnet exists, it will influence the overall performance.

4) Nonuniform Structure of Sampled Subnets: Our experiments have shown that our proposed nonuniform subnet sampling provides better accuracy than the uniform counterpart with identical subnet model size. It reveals that the layerwise sensitivities over accuracy are indeed different, which aligns with many prior static network pruning works using different methods. However, there is no standard golden metric to define what kind of subnet structure is optimal. We select two sampled nonuniform subnets learned by our proposed method and compare with the uniform ones with the same model size, as shown in Fig. 8. It provides some heuristic thinking for pruning and neural architecture search (NAS) exploration. Our sampled nonuniform structure has better accuracy compared with uniform subnets with the same model size. We summarize the main properties of the sampled nonuniform structures across different network typologies as below.
1) Compared with uniform structures, we observe that all these three DNNs, i.e., ResNet18, AlexNet, and VGG11-BN, have larger number of channels in the first and last layers. It aligns with the general conclusion from many prior works that these two layers are very important in the performance of the overall network, which typically needs more channels to extract sufficient features or accurately classify into correct groups.

2) ResNet consists of one single convolutional layer followed by several convolutional blocks and a fully connected layer sequentially. Each block includes two convolutional layers and an identity shortcut connection for ResNet18. For the blocks starting from the 2nd layer as shown in Fig. 8(a), we observe that the sampled first convolutional layer is smaller and the second one is larger than the uniform structures with the same model size. It might be because the second layer receives features from both the previous layer and the skip connection, thus requiring large #channels to avoid information bottleneck.

3) AlexNet is a single-path structure, which includes several convolutional layers and two fully connected layers. We observe that the sampled subnets have less number of channels in the last convolutional layer [i.e., the 5th layer in Fig. 8(b)]. This is because the input channel of the first fully connected layer of the full-size AlexNet is extremely large (i.e., $512 \times 6 \times 6$), indicating very high redundancy. A similar phenomena can also be observed in VGG.

In addition, we also study how the group channel setting will influence the subnet structures as shown in Fig. 9. Larger group setting means only a single channel is pruned in each iteration, which creates the best fine-grained structure.

Fig. 9. Subnet structures with three model sizes for different channel group settings on ResNet20. (Top) No group. (Middle) Group 16. (Bottom) Group 4.

From the experimental results, we observe that Group four setting, which has the most coarse-grained structure, but with smallest searching cost, still keeps similar characteristics of nonuniform structure, namely, larger channel numbers in the layers where the inputs are from both the previous layer and skip connection layer, while the next connected layer has smaller channel number. For example, 11th always has much larger channel numbers than layer 12th for all the settings. Considering such consistent property in our subnet searching, it explains why Group four setting could still achieve very similar performance with other larger group settings as discussed in Fig. 5, while requiring least searching time. It also supports our claim that the proposed trainable noise ranking is a fast and accurate method to indicate the sensitivity or importance of channels/groups, and thus could be used to quickly sample nonuniform subnets.

V. CONCLUSION

In this work, we target to construct a dynamic DNN structure that is able to adjust its inference structure on-the-fly within a group of subnets, through a novel proposed trainable noise-ranking-based subnet progressive searching method. Extensive experiments on the CIFAR-10 and ImageNet datasets indicate that our method could outperform the state-of-the-art counterparts. Beyond that, the constructed dynamic networks are deployed to Nvidia Titan GPU and Intel Xeon CPU to demonstrate its dynamic trade-off between accuracy and latency.

REFERENCES

[1] J. M. Alvarez et al., “Learning the number of neurons in deep networks,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 2270–2278.
[2] W. Wen et al., “Learning structured sparsity in deep neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 2074–2082.
[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient ConvNets,” 2016, arXiv:1608.08710.
[4] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 2736–2744.
[5] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 4340–4349.
[6] L. Liu and J. Deng, “Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 1–8.
[7] Z. Wu et al., “BlockDrop: Dynamic inference paths in residual networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 8817–8826.
[8] X. Wang et al., “SkipNet: Learning dynamic routing in convolutional networks,” in Proc. ECCV, 2018, pp. 409–424.
[9] G. Huang et al., “Multi-scale dense networks for resource efficient image classification,” 2017, arXiv:1703.09844.
[10] J. Yu, L. Yang, N. Xu, B. Yang, and T. Huang, “Slimmable neural networks,” 2018, arXiv:1812.08928.
[11] J. Yu and T. Huang, “Universally slimmable networks and improved training techniques,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1803–1811.
[12] D. Molchanov et al., “Variational dropout sparsifies deep neural networks,” in Proc. 54th Int. Conf. Mach. Learn., vol. 2017, pp. 2498–2506.
[13] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” 2016, arXiv:1611.01578.
[14] C. Liu et al., “Progressive neural architecture search,” in Proc. ECCV, 2018, pp. 19–34.
[15] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” 2018, arXiv:1806.09655.
[16] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C.-Z. Xu, “Dynamic channel pruning: Feature boosting and suppression,” 2018, arXiv:1810.05331.
[17] Y. Tang et al., “Manifold regularized dynamic network pruning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2021, pp. 5018–5028.
[18] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, “CondConv: Conditionally parameterized convolutions for efficient inference,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 1–12.
[19] L. Yang, S. Angizi, and D. Fan, “A flexible Processing-in-memory accelerator for dynamic channel-adaptive deep neural networks,” in Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC), Jan. 2020, pp. 313–318.
[20] L. Yang, Z. He, S. Angizi, and D. Fan, “Processing-in-memory accelerator for dynamic neural network with run-time tuning of accuracy, power and latency,” in Proc. IEEE 33rd Int. Symp.- Chip Conf (SOCC), Sep. 2020, pp. 117–122.
[21] L. Yang, Z. He, Y. Cao, and D. Fan, “Non-uniform DNN structured subnets sampling for dynamic inference,” in Proc. 57th ACM/IEEE Design Autom. Conf. (DAC), Jul. 2020, pp. 1–6.
[22] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, “Dynamic slimmable network,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2021, pp. 8607–8617.
[23] S. Han et al., “Learning both weights and connections for efficient neural network,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 1135–1143.
[24] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, arXiv:1510.00149.
[25] X. Ma et al., “Non-structured DNN weight pruning—Is it beneficial in any platform?” IEEE Trans. Neural Netw. Learn. Syst., early access, Mar. 18, 2021, doi: 10.1109/TNNLS.2021.3063265.
[26] Z. He, L. Yang, S. Angizi, A. S. Rakin, and D. Fan, “Sparse BD-Net: A multiplication-less DNN with sparse binarized depth-wise separable convolution,” ACM J. Emerg. Technol. Comput. Syst. (JETC), vol. 16, no. 2, pp. 1–24, 2020.
[27] L. Yang, Z. He, and D. Fan, “Harmonious coexistence of structured weight pruning and ternarization for deep neural networks,” in Proc. 2019 IEEE IJCNN Conf. Artif. Intell. (AAAI), 2020, pp. 6623–6630.
[28] M. Lin et al., “HRank: Filter pruning using high-rank feature map,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 1529–1538.
[29] Y. Sui, M. Yin, Y. Xie, H. Phan, S. Aliari Zonouz, and B. Yuan, “Chip: Channel independence-based pruning for compact neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 24604–24616.
[30] T. Yang et al., “NetAdapt: Platform-adaptive neural network adaptation for mobile applications,” in Proc. ECCV, 2018, pp. 285–300.
[31] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” 2019, arXiv:1908.09791.
[32] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” 2016, arXiv:1611.06444.
[33] X. Liu et al., “Towards robust neural networks via random self-ensemble,” in Proc. ECCV, 2018, pp. 369–385.
[34] Z. He, A. S. Rakin, and D. Fan, “Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 588–597.

Li Yang (Graduate Student Member, IEEE) received the B.S. degree in control engineering from Northeastern University, Qinhuangdao, China, in 2014, and the M.S. degree in electrical engineering from the University of Central Florida, Orlando, FL, USA, in 2018. He is currently pursuing the Ph.D. degree with Arizona State University, Tempe, AZ, USA, advised by Dr. Deliang Fan.

His research interests primarily focus on hardware-aware deep learning networks’ compression and optimization, and field programmable gate arrays (FPGA)-based deep neural network accelerator design. During his Ph.D. study, he has been the author of 26 publications on IEEE/Association for Computing Machinery (ACM) top-tier journal and conference (e.g., Computer Vision and Pattern Recognition (CVPR), International Conference on Learning Representation (ICLR), Association for the Advancement of Artificial Intelligence (AAAI), Design Automation Conference (DAC), ASP-DAC, ISVLSI, GLSVLSI) in the aforementioned areas.

Zhezi He (Member, IEEE) received the B.S. degree in information engineering from Southeast University, Nanjing, China, in 2012, the M.Eng. degree in electrical and computer engineering from Oregon State University, Corvallis, OR, USA, in 2016, and the Ph.D. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2020.

He is currently an Assistant Professor with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China. His research interests are in the area of intelligent system, neuromorphic computing, secure and efficient deep learning, and post-CMOS technologies. He has authored and coauthored over 80 publications on top-tier journal and conference in the aforementioned areas.

Dr. He’s works also resulted in the receipt of the Best Paper Awards from DATE 2022, ISVLSI 2017, and ISVLSI 2018.
Yu Cao (Fellow, IEEE) received the B.S. degree in physics from Peking University, Beijing, China, in 1996, and the M.A. degree in biophysics and the Ph.D. degree in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1999 and 2002, respectively.

He was a Summer Intern with Hewlett-Packard Labs, Palo Alto, CA, USA, in 2000, and also with the IBM Micro-Electronics Division, East Fishkill, NY, USA, in 2001. He worked as a Post-Doctoral Researcher with the Berkeley Wireless Research Center (BWRC), Berkeley. He is currently a Professor of electrical engineering with Arizona State University, Tempe, AZ, USA. He has published numerous articles and two books on nano-CMOS modeling and physical design. His research interests include brain-inspired computing, hardware design for on-chip learning, and reliable integration of nanoelectronics.

Dr. Cao was a recipient of the 2000 Beatrice Winner Award at International Solid-State Circuits Conference, the 2004 Best Paper Award at International Symposium on Quality Electronic Design, the 2006 NSF CAREER Award, the 2006 and 2007 IBM Faculty Award, the 2007 Best Paper Award at International Symposium on Low Power Electronics and Design, the 2008 Chunhui Award for Outstanding Oversea Chinese Scholars, the 2009 Association for Computing Machinery (ACM) SIGDA Outstanding New Faculty Award, the 2009 Promotion and Tenure Faculty Exemplar, Arizona State University, the 2009 Distinguished Lecturer of IEEE Circuits and Systems Society, the 2012 Best Paper Award at the IEEE Computer Society Annual Symposium on VLSI, and the 2010, 2012, 2013, 2015, and 2016 Top 5% Teaching Award, Schools of Engineering, Arizona State University. He served as an Associate Editor of the IEEE Transactions on CAD and on the Technical Program Committee of many conferences.

Deliang Fan (Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively.

He is currently an Associate Professor with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. His primary research interests include energy-efficient and high-performance big data processing-in-memory circuit, architecture, and algorithm, with applications in deep neural network, data encryption, graph processing, and bioinformatics acceleration-in-memory system; brain-inspired (neuromorphic) computing; machine learning system hardware–software codesign; and adversarial AI security. He has authored and coauthored around 140+ peer-reviewed international journal/conference papers in those areas.

Dr. Fan was a recipient of the NSF Career Award, the Best IP Paper Award of DATE 2022, and the Best Paper Award of GLSVLSI 2019, ISVLSI 2018, and ISVLSI 2017. His research works were also nominated as the Best Paper Candidate of ASPDAC 2019 and ISQED 2019. He is also the TPC Co-Chair of ISQED 2023, the Technical Area Chair of DAC 2021, GLSVLSI 2019–2022, ISQED 2019–2022, and the Financial Chair of ISVLSI 2019. Refer to https://dfan.engineering.asu.edu/ for more details.