Hardware/Software Co-Exploration of Neural Architectures

Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, *Senior Member, IEEE*, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, *Member, IEEE*, Yiyu Shi, *Senior Member, IEEE*, and Jingtong Hu, *Member, IEEE*

***Abstract*—We propose a novel hardware and software co-exploration framework for efficient neural architecture search (NAS). Different from existing hardware-aware NAS which assumes a fixed hardware design and explores the *NAS space* only, our framework simultaneously explores both the architecture search space and the *hardware design space* to identify the best neural architecture and hardware pairs that maximize both test accuracy and hardware efficiency. Such a practice greatly opens up the design freedom and pushes forward the Pareto frontier between hardware efficiency and test accuracy for better design tradeoffs. The framework iteratively performs a two-level (fast and slow) exploration. Without lengthy training, the fast exploration can effectively fine-tune hyperparameters and prune inferior architectures in terms of hardware specifications, which significantly accelerates the NAS process. Then, the slow exploration trains candidates on a validation set and updates a controller using the reinforcement learning to maximize the expected accuracy together with the hardware efficiency. In this article, we demonstrate that the co-exploration framework can effectively expand the search space to incorporate models with high accuracy, and we theoretically show that the proposed two-level optimization can efficiently prune inferior solutions to better explore the search space. The experimental results on ImageNet show that the co-exploration NAS can find solutions with the same accuracy, 35.24% higher throughput, 54.05% higher energy efficiency, compared with the hardware-aware NAS.**

***Index Terms*—Field-programmable gate array (FPGA), hardware-software co-exploration, multicriteria optimization, neural architecture search (NAS).**

I. Introduction

NEURAL architecture search (NAS) has achieved great success to liberate human labor in the design of neural architectures for various tasks including image classification, image segmentation, and language modeling [1]–[5]. Most recently, targeting a fixed hardware platform, the hardware-aware NAS [6]–[8] has been proposed to take into consideration the estimated timing performance (such as latency or throughput) in addition to accuracy [see Fig. 1(a)].

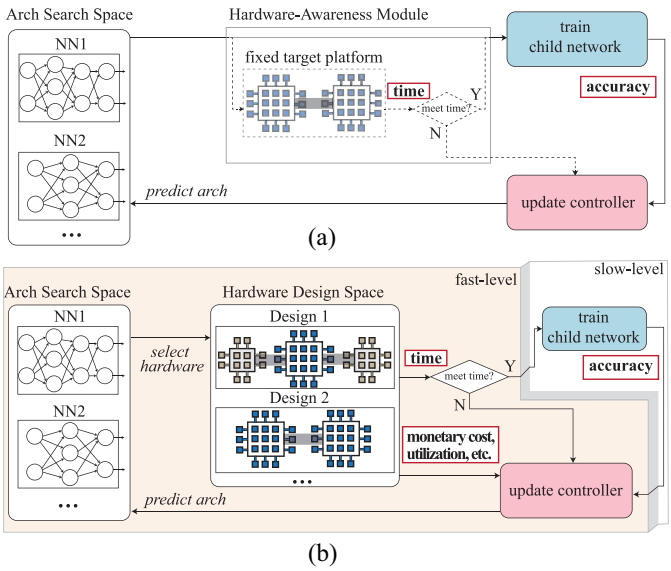

Fig. 1. Comparison between (a) hardware-aware NAS; (b) the proposed hardware/software co-exploration NAS. The red rectangles convey the metrics that can be optimized in the exploration.

All of the existing NAS frameworks explore the *architecture search space* only, without considering the hardware design freedom available in many cloud and edge computing applications. For instance, the cloud platforms (e.g., Amazon AWS [9] and Microsoft Azure [10]) employ field-programmable gate array (FPGA) for neural network acceleration, while the edge computing platforms typically take the programmable FPGAs [11], [12] or application-specific integrated circuit (ASIC) [13], [14]. In addition to neural architecture design, those hardware platforms can also be programmed or even fully customized for the best performance, expanding a *hardware design space*.

Interestingly, the hardware design space is tightly coupled with the architecture search space, i.e., the best neural architecture depends on the hardware (hardware-aware NAS), and the best hardware depends on the neural architecture. It is,

Manuscript received April 26, 2019; revised September 10, 2019 and December 26, 2019; accepted March 9, 2020. Date of publication April 8, 2020; date of current version November 20, 2020. This work was supported in part by the National Science Foundation under Grant CNS-1822099, Grant CCF-1820537, and Grant SPX-1919167, and in part by the National Natural Science Foundation of China under Grant 61972154. This article was recommended by Associate Editor L. P. Carloni. *(Corresponding author: Weiwen Jiang.)*

Weiwen Jiang, Lei Yang, and Yiyu Shi are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: wjiang2@nd.edu; lyang24@nd.edu; yshi4@nd.edu).

Edwin Hsing-Mean Sha, Qingfeng Zhuge, and Shouzhen Gu are with the School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China.

Sakyasingha Dasgupta is with Research and Development Department, Edgecortix Inc., Tokyo 1410031, Japan.

Jingtong Hu is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: jthu@pitt.edu).

Digital Object Identifier 10.1109/TCAD.2020.2986127

TABLE I
ON CIFAR-10 AND XILINX XC7Z015 FPGA: COMPARISONS OF THREE NEURAL ARCHITECTURE AND HARDWARE DESIGN PAIRS IN ACCURACY, THROUGHPUT, AND ENERGY EFFICIENCY (E.-E): A) OPTIMAL ARCHITECTURE ON A FIXED HARDWARE IMPLEMENTATION THROUGH HARDWARE-AWARE NAS; B) THE SAME ARCHITECTURE BUT WITH FURTHER FPGA OPTIMIZATION; AND C) A JOINTLY OPTIMIZED NEURAL ARCHITECTURE AND FPGA IMPLEMENTATION THROUGH OUR CO-EXPLORATION.

ID	Approach	Accuracy	Throughput (FPS)	E.-E (GOPS/W)
A	Hardware-Aware NAS	84.53%	16.2	0.84
B	Sequential Optimization	84.53%	29.7	1.36
C	**Co-Exploration**	**85.19%**	**35.5**	**1.91**

therefore, best to jointly explore both spaces to push forward the Pareto frontier between hardware efficiency and test accuracy for better design tradeoffs. This can be clearly seen from the example in Table I, where three designs on CIFAR-10 and Xilinx XC7Z015 FPGAs are presented: an optimized neural architecture for a fixed FPGA implementation through hardware-aware NAS (design A), the hardware of which is then further optimized through FPGA optimization (design B) [15], and a jointly optimized neural architecture and hardware through our co-exploration (design C). From the table, we can see that further optimizing the hardware for the architecture from hardware-aware NAS can lead to 45.45% higher throughput, 38.24% higher energy efficiency with the same accuracy. On the other hand, compared with such a sequential optimization strategy, our co-exploration approach can identify an architecture with higher accuracy and its tailor-made hardware with 16.33% and 28.80% improvements in throughput and energy efficiency, respectively.

Specifically, our architecture search space and hardware design space co-exploration framework is shown in Fig. 1(b). The proposed co-exploration can be built on any existing NAS framework [8], [16]–[18] by expanding it to delve into the hardware design space, where a two-level (fast and slow) exploration is iteratively conducted. In the fast exploration, the best hardware design is identified for the sampled neural architectures without lengthy training. The architectures with inferior hardware efficiency will be quickly pruned, which significantly accelerates the search process. Thereafter, the superior candidates are trained in the slow exploration (SE) for controller update using policy gradient reinforcement learning to explore the coupled architecture search space. The optimization objectives in the hardware design space can be varied according to the design specifications, such as area, monetary cost, energy efficiency, reliability, resource utilization, etc.

In order to illustrate our framework, we choose to use FPGA as a vehicle in this article, as it has gradually become one of the most popular platforms to implement deep neural networks (DNNs) due to its programmability, high performance, and energy efficiency, in particular for low-batch inferences [19], [20]. Our co-exploration concept and the general framework, however, can also be easily extended to other hardware platforms such as ASICs. Since timing performance on a single FPGA is limited by its restricted resource, it is prevalent to organize multiple FPGAs in a pipelined fashion [21]–[24] to provide high throughput (frame per second, FPS). In such a system, the pipeline efficiency is one of the most important metrics needing to be maximized, since it determines the hardware utilization, as well as energy efficiency. As such, we use accuracy and pipeline efficiency to guide the exploration of the neural architecture space and hardware design space, respectively, while satisfying a given throughput specifications (e.g., $\geq$30FPS for the ordinary camera). The experimental results show that the co-exploration approach can significantly push forward the Pareto frontier. On ImageNet, the proposed co-exploration framework can identify architecture and hardware pairs to achieve the same accuracy, 35.42% higher throughput, and 54.05% higher energy efficiency with the reduced search time, compared with the hardware-aware NAS.

II. BACKGROUND AND PROBLEM DEFINITION

A. Neural Architecture Search

Although the research on the automatic prediction of neural network architectures can trace back to the 1980s [25], after DNNs have achieved great success in AI domains, there have been growing interests in generating good neural architectures for the interested dataset recently. With the fact that the architectures are growing deeper, the search space expands exponentially, leading to more difficulties in exploring the search space. In the existing work, there are two mainstreams of architecture search: 1) employing reinforcement learning [2], [16], [26] and 2) applying evolutionary algorithms [3], [27], [28]. The basic idea is to iteratively update hyperparameters to generate better "child networks" in terms of accuracy.

Fig. 1(a), without the hardware-aware module, illustrates a typically used reinforcement learning-based NAS [16] framework. As shown in this figure, the RNN controller in NAS iteratively predicts child networks from the architecture search space. These child networks will be trained on a held-out dataset to obtain its accuracy. Then, accuracy will be used as reward to update the RNN controller.

Existing work has demonstrated that the automatically resulting architectures can achieve close or even higher accuracy to the best human-invented architectures [2], [16]. However, there are two important problems in searching architectures. First, the search process is inefficient. Zoph and Le [16] reported that 20 000 networks were trained across 500 P100 GPUs over four days to find the desired network. Second, since the search process is hardware oblivious, neither the time performance nor the hardware efficiency can be guaranteed.

Recently, hardware-aware NAS [6]–[8] has been proposed to search architectures for a target hardware platform, as shown in Fig. 1(a). They always assume a fixed hardware design (e.g., mobile chips) and only explore the architecture search space. However, the hardware design freedom is commonly available in many cloud and edge computing applications, like FPGA in cloud platforms [9], [10] and ASIC in

edge computing platforms [13], [14]. Without the consideration of hardware design space will lead to inferior designs in hardware efficiency, because the hardware design space and the architecture search space are tightly coupled.

Compared with the existing work, the main contribution of this article is to propose a framework to co-explore the architecture search space and the hardware design space, as shown in Fig. 1(b). More specifically, this framework determines the best hardware during the search process, which is tailor-made for the candidate architectures. In this way, the framework can obtain a set of superior architecture and hardware design pairs on the Pareto frontier in terms of accuracy and hardware efficiency tradeoffs. In addition, the search time can be significantly reduced, since we can efficiently prune inferior architectures according to multiple design specifications compared with the hardware-aware NAS.

B. Implementation of DNNs on FPGAs

This article will employ FPGA as a vehicle to study how to co-explore neural architectures and hardware designs. FPGA has demonstrated its excellent ability to achieve high performance and energy efficiency for low-batch real-time inferences [19], [20]. Hence, a large amount of work is made in implementing neural networks on FPGAs, in which tools are developed to automatically design accelerators on FPGAs for a given network architecture. In the early stage, research efforts are mainly focusing on designing accelerators on a single FPGA [29]–[32]. Most recently, implementations on multiple FPGAs has become the mainstream [15], [19]–[21], [23], [24], since limited resource on a single FPGA becomes the performance bottleneck.

To fully utilize the computation power provided by multiple FPGAs, a typical technique is to implement the neural network on multiple FPGAs in a pipelined fashion [15], [21], [23], [24]. Fig. 2 demonstrates one such example, in which a 5-layer network is partitioned into three pipeline stages, and each pipeline stage is mapped to a certain FPGA in an available pool. Finally, those FPGAs are connected as a linear array to function in the pipelined fashion.

Kindly note that this is the first work on the co-exploration of NAS and multiple FPGAs, which is extended from our previous work in [33] for single FPGA. The co-design idea is also verified in [34], which also targets a single FPGA.

C. Definitions and Problem Statement

The goal of the proposed framework is to find both the neural architectures with the highest test accuracy and hardware design with the guaranteed performance (e.g., timing requirement and hardware efficiency). In this article, we will employ the conventional convolutional neural network (CNN) based on the multi-FPGA infrastructure as an example to illustrate such a framework, which is the base for other related problems. In the following, we will first present the relevant definitions. Then, we will formally define the problem. Finally, we will discuss the possible extension.

The child network is the bridge between the architecture search space and the hardware design space. Specifically, in each iteration, the controller RNN will predict child networks from the architecture search space, and then determine their implementations in the hardware design space. We will introduce the hardware design space as follows.

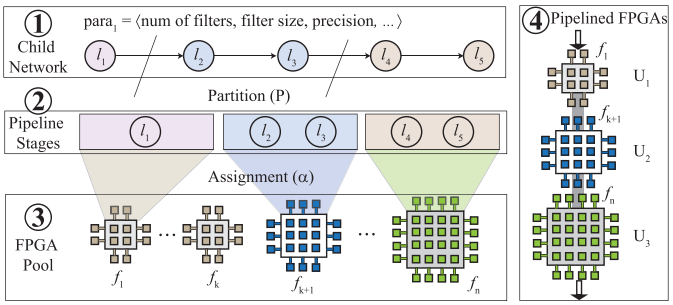

Fig. 2. Overview of implementing a child network onto multiple FPGAs to be organized in the pipelined fashion.

② *Partition Child Network to Pipeline Stages:* Let $P(C)$ be a set of partitions for the child network C. $P(C) = \{P_1, P_2, \ldots, P_M\}$, where P_i is a nonempty subset of set L. We have the following two properties: 1) $\bigcup_{P_i \in P(C)} = L$ and 2) $\forall P_i, P_j \in P(C)$, if $i \neq j$, then $P_i \cap P_j = \emptyset$. After the partitioning, each set in $P(C)$ corresponds to a pipeline stage. For example, in Fig. 2 ②, we partition the given child network into three pipeline stages, $P_1 = \{l_1\}$, $P_2 = \{l_2, l_3\}$, and $P_3 = \{l_4, l_5\}$.

③ *Assign Pipeline Stages to FPGAs:* Then, we can assign each pipeline stage to a specific FPGA in an available FPGA pool, as shown in Fig. 2 ③. An FPGA pool with n FPGAs can be represented by a set $F = \{f_0, f_1, \ldots, f_n\}$. Each FPGA, f_i, has a set of attributes, including memory mem_i, DSP slices dsp_i, etc. These attributes will be utilized to model the timing performance for a child network.

We define the assignment function α from the partition set $P(C)$ to FPGA pool F. We have $\alpha(P_i) = f_j$ to indicate the ith pipeline stage P_i is assigned to the jth FPGA f_j to be implemented. After pipeline stages are assigned to FPGA pool according to α, each FPGA will process one or multiple layers. All FPGAs work together in the pipelined fashion.

④ *Pipelined FPGAs:* The pipelined executions of multiple FPGAs are illustrated in Fig. 2 ④. The system will continuously obtain inputs from the dataset with a fixed rate (frame per second), and generate output data from the last pipeline stage. The input rate of the system reflects the throughput specification TS, which implies that the latency of each pipeline stage should be no more than $1/TS$.

The latency of a pipeline stage under an assignment function can be easily captured with a performance model [29]. For FPGA f_i, its latency is denoted as Lat_i. After obtaining the latency of each FPGA, we introduce pipeline efficiency, which is composed of the hardware utilization in each pipeline stage (corresponding to an FPGA). The utilization of FPGA f_i is equal to $\text{Lat}_i \times TS$. Higher utilization of an FPGA indicates the less idle time in processing and higher energy efficiency. Therefore, high average utilization of all FPGAs is always desired.

Problem Statement: Based on the above definitions, we formally define the problem of "hardware/software co-exploration

of neural architectures" as: Given a dataset, a pool of FPGAs F, and a throughput specification TS, we are going to co-explore architecture search space and hardware design space to find a child network C.

1) *para:* Parameters of all layers in the child network.
2) *P:* The partition of layer set L in the child network.
3) α*:* The assignment of pipeline stages to set F.

Such that the accuracy of child network C is maximized, the pipeline FPGA system can meet the required throughput TS, and the average utilization of all FPGAs is maximized.

Extensions: The targeting problem is the basis for more general problems. Therefore, the proposed framework in the next section can be applied to different scenarios with little or no modifications. In the following, we will discuss different extensions from both hardware and software perspectives.

From the hardware perspective, the fundamental problem of mapping child network onto multiple FPGAs is equivalent to that of mapping child network onto multiple processing elements (PEs) in one FPGA, where each PE indicates a processor for one data tile (also known as layer processor in [30]). Splitting one FPGA to multiple PEs [30] is a promising solution when the single FPGA is large enough or the size of neural architecture is relatively small. In this scenario, a PE can be regarded as an FPGA in the hardware pool in Fig. 2. To apply the proposed technique, we only need to iteratively generate a PE pool (i.e., the number of PEs and the size of each PE) according to the FPGA resource, and conduct co-exploration to identify the best solution for each PE pool.

From the software perspective, first, the proposed framework can handle neural networks with residual connections by integrating techniques in [35] to partition DAG-based child network; second, it can explore different operations (e.g., group convolutions, depthwise separable convolution, etc.) for each node in a child network by adding one additional parameter in $para_i$ to determine a specific operation for the node.

Finally, throughput (frame per second, FPS) in the above problem is set as a constraint. But we can wrap a binary search procedure to maximize throughput together with the pipeline utilization. Kindly note that by replacing the metrics of FPS to operation per seconds (OPSs), the proposed framework can also be applied to optimize other efficiency metrics, like OPS/LUT or OPS/DSP.

In the following of this article, we will focus on determining the best neural architectures and hardware implementations with the conventional CNN structure and multi-FPGA scenario, using the throughput as a constraint and maximizing the hardware utilization.

III. HW/SW Co-Exploration Framework

In this section, we will present the proposed framework. We will use the NAS discussed in [16] as the backbone framework and FPGA as the hardware platform to demonstrate our concept. It can be integrated with any existing NAS techniques [8], [16]–[18] or extended to incorporate other hardware platforms.

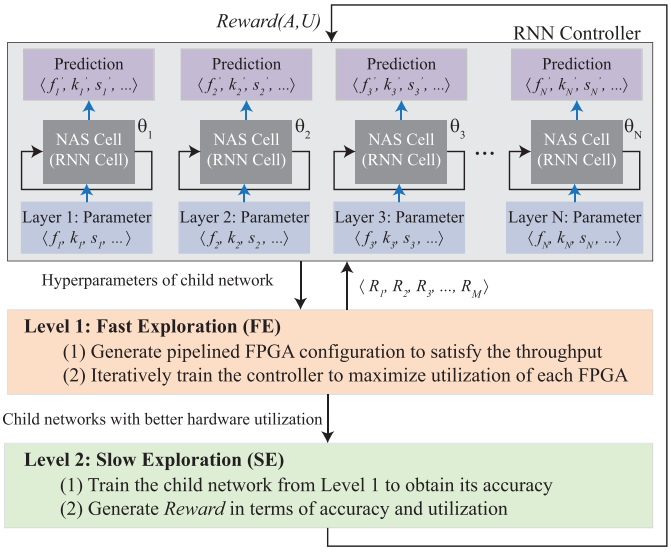

Fig. 3. Overview of HW/SW co-exploration framework: The controller contains multiple reconfigurable RNN cells and predicts the hyperparameters in a child network; the fast exploration level prunes child networks with inferior hardware utilization; the SE level updates controller using hardware utilization and accuracy obtained by training child networks.

A. Framework Overview

Fig. 3 shows the HW/SW co-exploration framework. The framework contains an RNN-based controller and two levels of explorations. Unlike that in [16], the controller has multiple RNN cells instead of one. More specifically, each layer in a child network has a corresponding RNN cell. During the exploration, cells will be reorganized to support different optimization goals.

In the first level, a fast exploration is carried out in four steps: 1) it first predicts an architecture with probability p; 2) then, it explores the design space to generate a pipelined FPGA system to meet the throughput requirement; 3) according to the pipeline structure, it then reorganizes RNN cells in the controller; and 4) it updates the controller using reinforcement learning to maximize the pipeline efficiency. This level explores the hardware design space without training child networks, therefore, it performs efficiently.

In the second level, we train the child network obtained from the first level on the held-out validation set. After that, we generate a reward based on both the yielded accuracy and pipeline efficiency, which is used to update the RNN controller. In case that no child network can meet the required throughput specification in the first level, we generate a negative reward to update the controller. After this level, the controller will predict a new child network from the architecture search space for the fast exploration level.

The proposed controller integrated with multiple RNNs, operated in two levels of optimizations as shown in Fig. 3, can make a better tradeoff between efficiency and accuracy. First, in Level 1, RNNs operate independently to optimize a given architecture for each pipeline stage. As a result, it can explore the search space more efficiently. On the other hand, RNNs will work together in Level 2 to determine the backbone architecture and pipeline structure. Specifically, let $D_i = 10^3$

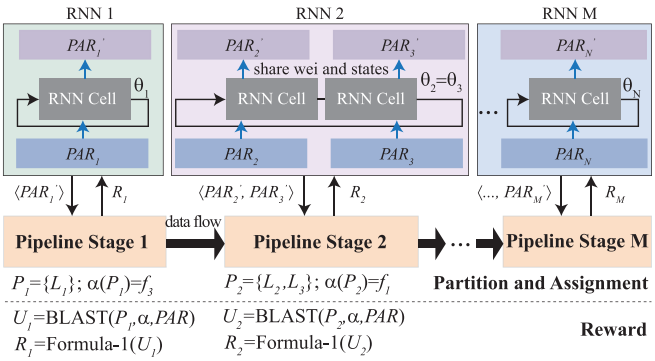

Fig. 4. FE: organize RNN cells in the controller according to the partition for pipeline stages; independently update multiple RNNs in the controller to predict parameters of layers assigned to each pipeline stage.

be the size of search space for pipeline stage p_i. The proposed controller with multiple RNN can optimize each pipeline stage independently, and, therefore, the design space is $O(\sum_i\{D_i\})$ [i.e., $O(10^3)$ in the example]. On the contrary, for the controller with only one RNN, it will jointly determine substructure for all pipeline stages, leading the search space to be $O(\prod_i D_i)$ [i.e., $O(10^9)$]. Kindly note that a huge design space will not only significantly prolong the exploration time but also make it difficult to find the best solution. The advantages of the proposed framework in both efficiency and effectiveness will be verified in the experimental results.

B. Fast Exploration for High Resource Utilization

In the first level, namely fast exploration (FE), the objective is to maximize pipeline efficiency under the throughput specification *TS*. FE takes three types of inputs: 1) a set of available FPGAs F; 2) hyperparameters of a child network H; and 3) a throughput specification *TS*. It will generate a new child network, whose throughput at the inference phase can meet *TS* using a subset of FPGAs in F. In addition, the average hardware utilization of FPGAs can be maximized. In FE, there are two challenges needing to be addressed: first, how to partition a given child network and assign each partition to a specific FPGA (Partition and Assignment); second, how to reorganize the RNN cells in the controller and then update them to generate child networks with higher pipeline efficiency (Reorganize and Update Controller).

Partition and Assignment: In the search process, a number of candidate child networks need to go through the partition and assignment process. Consequently, an efficient automatic tool should be employed to avoid performance degradation on the search process. In this article, we employ the BLAST algorithm in [21]. BLAST takes child network H, FPGAs F, the throughput specification *TS*, and the attributes of each FPGA as inputs. It outputs a serial of FPGAs, each of which will implement one or multiple layers in a pipeline stage. The resultant system will satisfy *TS* with the maximum pipeline efficiency. As shown in Fig. 4, layers in a child network are divided into M partitions, and each partition is assigned to one specific type of FPGA under function α.

Reorganize and Update Controller: According to the generated pipeline structure, we then reorganize the controller and iteratively update the controller to generate child networks with higher hardware utilization. Our goal is to maximize the average hardware utilization, which is equivalent to maximize the utilization of each hardware. However, the design space of maximizing the average hardware utilization is exponentially larger than that of maximizing the utilization of each hardware. To efficiently explore the design space, we choose to maximize the hardware utilization of different pipeline stage independently. Therefore, we reorganize RNN cells in the controller according to the determined pipeline structure. More specifically, for multiple layers in one pipeline stage, their corresponding RNN cells will be configured to form one RNN and their weights and states are shared (e.g., RNN 2 in Fig. 4). In consequence, there will be N RNNs for N pipeline stages. In this way, each RNN can be trained to maximize the hardware utilization for each FPGA pipeline stage.

After we form the RNNs, we apply reinforcement learning to update the parameters in those N RNNs, and use these RNNs to predict the hyperparameters of child networks. In each iteration, we will predict T child networks, which can be viewed as a list of actions $a_{1:T}$. Correspondingly, notation $a^i_{1:T}$ represents the hyperparameters of the ith pipeline stage in these child networks. For each child network predicted by the controller, we can obtain the utilization of the ith pipeline stage (corresponding to one FPGA) using BLAST, denoted as U_i. Then, for RNN i, we utilize U_i to generate a reward R_i to update its parameters θ_i. The reward R_i can be calculated using the following formula:

$$R_i = \begin{cases} U_i & U_i \le 1 \\ 1 - U_i & 1 < U_i \le 2 \\ -1 & U_i > 2 \end{cases} \tag{1}$$

where $U_i > 1$ indicates that the required throughput cannot be satisfied, and we give the negative reward. For each RNN, our objective is to maximize the expected reward for actions from time 1 to T, represented by $J(\theta_i) = E_{P(a^i_{1:T};\theta_i)}[R_i]$. Since the reward is nondifferentiable, we apply the policy of gradient method to update θ_i. Specifically, the method of REINFORCE rule [36] has been employed as in [8] and [16].

C. Slow Exploration for High Accuracy

After obtaining a child network meeting the timing specification through the fast exploration level, we now move to the second level. In this level, we aim to update the controller RNN to generate new child networks with higher accuracy and pipeline efficiency. We will train the child network on the held-out validate set, and, therefore, the exploration speed is much slower than that of the first one. We call it SE.

As shown in Fig. 5, SE takes the generated child network, the partition and the assignment from FE as the inputs. The child network is first trained to obtain accuracy A. Then, the average pipeline efficiency U of the child network under the partition and assignment will be calculated. Finally, we compute the reward to update the controller using the

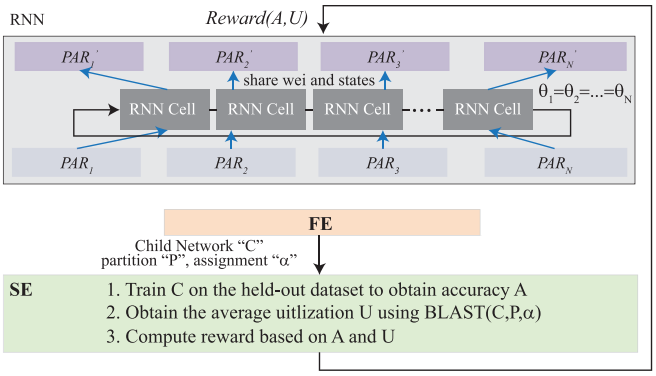

Fig. 5. SE: configure RNN cells in the controller to be one RNN; generate reward based on accuracy and pipeline efficiency to update the controller RNN.

following formula:

$$\text{Reward}(A, U) = \beta \times A + (1 - \beta) \times U \tag{2}$$

where β is an adjustment parameter, which reflects the bias on test accuracy and hardware utilization. The value of β ranges from 0 to 1. We will discuss how to scale β in Section V. After that, we update the controller using the reward by applying the policy gradient reinforcement learning, which is the same as that in FE level. As shown in Fig. 5, all RNN cells share the same weights and states in this level, since we have only one reward.

D. Interface Between Fast-Slow Explorations

Before updating the RNN cells in the controller in the fast exploration level, we take a snapshot *Snap* of all RNN cells. During the fast exploration level, we obtain the hardware design (i.e., pipeline configuration) for the input child network. Based on the determined pipeline structure, RNN cells are reorganized as introduced in Section III-B. Reorganized cells will be trained to generate better child networks for the previously obtained hardware design (i.e., pipeline configuration). Finally, a child network with maximum hardware efficiency on the determined pipeline will be sent to the SE level.

After entering the SE level, the RNN cells in the controller will be recovered using the previously saved snapshot *Snap*. Then, SE will train the child network to obtain the accuracy, which will be used to calculate the reward. Using this reward, we will update the recovered RNN. Then, the updated RNN will be used to generate new child networks for the next iteration. In this way, the SE process will always keep improving the RNN accuracy while the FE process will always generate the best hardware design for each iteration.

IV. Experiments

Datasets: We use CIFAR-10 and ImageNet datasets to study the efficacy of our approach and compare it with the state-of-the-art. During the exploration of child networks, we only use the training images in these datasets, while the test images are used to test the accuracy of the resultant architectures. To evaluate the accuracy in the search process, we randomly select 10% of the samples from the training set as a validation set. All the images undergo the data preprocessing and augmentation procedure, including whitening, upsampling, random cropping, and random horizontal flip, which are common among the related work.

Architecture Search Space: For CIFAR-10, we use convolutional architectures as the backbone. For every convolutional layer, we first determine the filter size in [24, 36, 48, 64], the kernel size in [1, 3, 5, 7], and the strides. Two sets of experiments are carried out to determine the strides: 1) by exploring the child networks with a fixed stride of 1 and 2) by allowing the controller to predict the strides in [1, 2]. After each layer, the rectified linear units [37] and the batch normalization [38] are appended.

For ImageNet, the architecture repeats mobile inverted bottleneck convolution layers instead of ordinary convolutional ones, same as that in [8]. The controller explores the architectures with various kernel sizes [3, 5, 7], strides [1, 2], and expansion ratios [3, 6].

Hardware Design Space: The hardware design space is composed of up to three Xilinx FPGAs (XC7Z015), each of which contains 74 K logic cells, 4.9-Mb on-chip memory, and 150 DSP Slices. One reason for our selection is that such an FPGA provides high-speed serial communication (up to 16.8 Gb/s of bandwidth), so that a high-speed hardware pipeline can be formed by multiple FPGAs. In the implementation, the child network is partitioned into pipeline stages, and each stage is mapped to one FPGA. Kindly note that our hardware exploration may not end up using all three FPGAs; it is possible to use fewer for higher hardware efficiency.

In the experiments, we use pipeline efficiency as the metrics to measure the hardware efficiency. As stated in Section I, the pipeline efficiency is one of the most important metrics, since it is related to the hardware utilization, energy efficiency, and timing performance. Then, the timing specifications are set according to the desired processing speed of the data at the inference phase, which are commonly decided by the data collector (e.g., camera). For CIFAR-10, we set the throughput specification to 35FPS, which can satisfy most cameras; whereas for ImageNet, due to the more complicated architectures and the limited resource, we set the specification to 10FPS. Finally, for both data and weights, we apply the commonly used 16-bit fixed-point data, as that in [21], [29], [30], and [39].

Training Details: For CIFAR-10, the training settings for both the RNN controller and the child networks are the same as [16]. For the controller RNN, in both slow and fast explorations, it is trained by using the calculated rewards with the ADAM optimizer [40] with a learning rate of 0.0006. Parameter β in (2) is set to 0.5 to equally optimize test accuracy and pipeline efficiency. For the child networks, we apply Momentum Optimizer with a learning rate of 0.1, weight decay of 10^{-4} and momentum of 0.9. Each child network is trained for 50 epochs.

For ImageNet, we build the distributed GPU training environment on top of Uber Horovod [41]. Training settings are similar to those for CIFAR-10, with the exceptions that we set the initial learning rate to 0.0125, decay 10× at selected

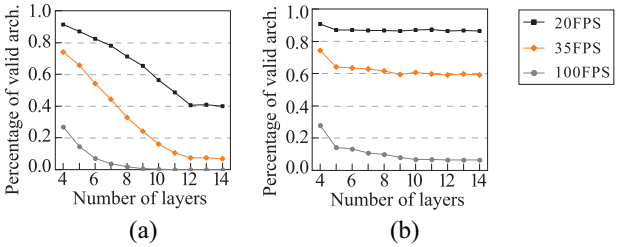

Fig. 6. Percentages of valid architectures for different timing specifications. (a) Fixed stride of 1. (b) Predictable strides.

epochs, and for the momentum optimizer the weight decay is 5×10^{-5} and the momentum is 0.9.

V. Results

This section will report comparison results in four sets of experiments: 1) we compare the proposed framework with different configurations; 2) we compare the proposed framework with the existing NAS frameworks; 3) we compare the identified architectures with the existing ones; and 4) we show the design space exploration in terms of model size and hardware efficiency to demonstrate the importance of hardware/software co-exploration.

A. Comparison Results With Different Configurations

Before reporting the results, we first introduce the setting for the proposed framework, namely "Co-Exploration." First, the search spaces and training settings can be found in Section IV. Second, the controller will iteratively search child networks for 10 000 episodes through the 2-level exploration. Third, in each episode, the SE phase will obtain accuracy of 16 child networks (train from scratch if one has never been trained or obtain accuracy from a history table); these child networks are identified by the fast exploration phase, where 100 trails will be taken for each child network to optimize the hardware efficiency. Since the proposed framework has multiple optimization goals on both software (e.g., accuracy) and hardware (e.g., pipeline efficiency), we record a set of superior architecture and hardware design pairs during the exploration, which forms the Pareto frontier. On the frontier, we denote the solution with the maximum accuracy as "OptSW" and the solution with the maximum pipeline efficiency as "OptHW."

Impact of Timing Specifications: Fig. 6 reports the impact of timing specifications for the Co-Exploration framework. We randomly sample 10 000 architectures for the layer size ranged from 4 to 14, and obtain the percentage of valid architectures that can meet the timing specification on the CIFAR-10 dataset. In Fig. 6, it is obvious that if the constraint is tight (e.g., FPS = 100), only a few architectures can satisfy the specification, indicating that the number of architectures with high accuracy is reduced compared with the one without timing constraints. In this case, we can scale up the parameter β in (2) to pursue higher accuracy. On the other hand, if the constraint is loose (e.g., FPS = 20), there are a large number of valid architectures. Correspondingly, we can scale down β to find more hardware efficient designs with high accuracy.

TABLE II
Co-Exploration With Predictable Stride Performs Better Than That With Fixed Stride Under 35FPS Timing Specification

Models	Depth	Accuracy	Pipeline Eff.
Co-Exploration fixed stride (OptSW)	13	81.50%	91.92%
Co-Exploration fixed stride (OptHW)	10	78.57%	98.56%
Co-Exploration pred. stride (OptSW)	14	**85.19%**	92.15%
Co-Exploration pred. stride (OptHW)	6	80.18%	**99.69%**

Comparison Between Fixed Stride and Predictable Stride: Table II reports the comparison between the exploration with the fixed stride and that with the predictable stride on CIFAR-10.[1] In the table, column "depth" indicates the number of layers in the resulting architecture. As shown in this table, for the exploration with the fixed stride, OptSW achieves 2.93% higher accuracy but 6.64% loss in pipeline efficiency than OptHW. These figures are 5.01% and 7.54% for the exploration with the predictable strides. In addition, it is obvious that compared with fixed stride, the stride prediction can help controller to find better results in both accuracy and pipeline efficiency. As such, in the following experiments we will use predictable stride as the default setting for Co-Exploration.

B. Comparison Results With the Existing NAS Frameworks

Next, we compare the proposed Co-Exploration framework with the existing NAS frameworks. To be fair, we use the same setting as the Co-Exploration: exploring 10 000 episodes and getting accuracy of 16 child networks in each episode. Because the existing Hardware-Aware NAS frameworks [6]–[8] target fixed hardware (e.g., GPU) instead of programmable FPGAs, and they use various settings; for fair evaluation, we use the NAS discussed in [16] as the backbone to implement a Hardware-Aware NAS for FPGA with the same search spaces and training settings as described in Section IV. Unlike the Co-Exploration framework, the Hardware-Aware NAS assumes fixed accelerator designs (i.e., optimization parameters) in FPGAs. As shown in Fig. 1(a), in the search loop, the controller will first predict a neural architecture; second, the framework tests the hardware efficiency of the predicted architecture on FPGAs; third, it trains architecture to get its accuracy; finally, it utilizes hardware efficiency and accuracy to update the controller. This framework is denoted as Hardware-Aware NAS in the results.

In addition, for the final architectures obtained by the Hardware-Aware NAS, we further optimize their hardware implementation to achieve a better design in terms of hardware efficiency. Such a heuristic approach is denoted as "Sequential Optimization" in the results.

Impact of Different Exploration Frameworks on Pareto Frontier: Fig. 7 reports the design space exploration assuming the hardware design space contains up to (a) two FPGAs or (b) three FPGAs. The x-axis and y-axis represent the accuracy and pipeline efficiency, respectively. For clear demonstration, we only include the architectures whose pipeline efficiency is no less than 85% for two FPGAs in Fig. 7(a) and no less than

[1]Models accessed at: https://github.com/PITT-JZ-COOP/Co-Explore-NAS.

TABLE III
COMPARISON AMONG CO-EXPLORATION, HARDWARE-AWARE NAS, AND SEQUENTIAL OPTIMIZATION ON CIFAR-10 AND IMAGENET DATASETS

Dataset	Models	Depth	Parameters	Accuracy (Top1)	Accuracy (Top5)	Pipeline Eff.	FPS	Energy Eff. GOPS/W
CIFAR-10	Hardware-Aware NAS	13	0.53M	84.53%	-	73.27%	16.2	0.84
	Sequential Optimization	13	0.53M	84.53%	-	92.20%	29.7	1.36
	Co-Exploration (OptHW)	10	0.29M	80.18%	-	**99.69%**	**35.5**	**2.55**
	Co-Exploration (OptSW)	14	0.61M	**85.19%**	-	92.15%	**35.5**	1.91
ImageNet	Hardware-Aware NAS	15	0.44M	68.40%	89.84%	81.07%	6.8	0.34
	Sequential Optimization	15	0.44M	68.40%	89.84%	86.75%	10.4	0.46
	Co-Exploration (OptHW)	17	0.54M	68.00%	89.60%	**96.15%**	**12.1**	**1.01**
	Co-Exploration (OptSW)	15	0.48M	**70.24%**	**90.53%**	93.89%	10.5	0.74

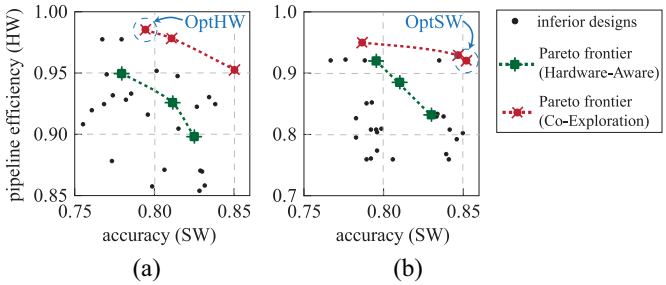

Fig. 7. Pareto frontiers between accuracy and pipeline efficiency for Hardware-Aware NAS and Co-Exploration, both of which are designed under the timing specification of 35FPS. (a) Designs with two FPGAs. (b) Designs with three FPGAs.

TABLE IV
CO-EXPLORATION USES MUCH FEWER GPU HOURS THAN THAT OF HARDWARE-AWARE NAS, BENEFITING FROM THE EARLY STAGE PRUNING

Dataset	Approach	Arch for Training	GPU Hours	Impr.
CIFAR-10	Hardware-Aware NAS	108,000	16,586	1
	Co-Exploration	308	102+1.9=103.9	159×
ImageNet	Hardware-Aware NAS	7,263	36,315	1
	Co-Exploration	53	256+1.8=266.8	136×

75% for three FPGAs in Fig. 7(b). In these figures, the circled design points correspond to those in Table II. The red lines represent the Pareto frontiers explored by Co-Exploration. The green lines, on the other hand, represent the frontier obtained by Hardware-Aware NAS (by examining the top architectures identified). These figures clearly show that by exploring hardware design space, our Co-Exploration can significantly push forward the Pareto frontiers in the accuracy and efficiency tradeoffs. It effectively identifies better designs not available through architecture search space only, i.e., those between the two frontiers.

Comparing the two exploration results in Fig. 7(a) and (b), we can also see that the solution with the highest pipeline efficiency is located in Fig. 7(a), while the one with the highest accuracy is located in Fig. 7(b). In general, we can always observe that the average accuracy on three FPGAs is higher than that on two FPGAs, yet the pipeline efficiency is lower. This is because more FPGAs can accommodate deeper architecture in layers for higher accuracy. On the other hand, more layers will easily result in unbalanced pipeline stages, which in turn reduces the pipeline efficiency.

Comparison Between Co-Exploration and Existing Frameworks: Table III reports the comparison results on accuracy, pipeline efficiency, throughput, and energy efficiency on CIFAR-10 and ImageNet. All the architectures identified have fewer than 1M parameters mainly due to the hardware capacity. This inevitably leads to accuracy loss; however, as we can see, the architecture explored by OptSW can still achieve 85.19% test accuracy on CIFAR-10, and 70.24% top-1 accuracy on ImageNet. These results demonstrate the effectiveness of the Co-Exploration approach in resource-limited scenarios. In addition, OptSW outperforms Hardware-Aware NAS by achieving 54.37% and 35.24% higher throughput, and 56.02% and 54.05% higher energy efficiency on CIFAR-10 and ImageNet, respectively. Compared with Sequential Optimization, OptSW achieves 16.34% and 28.79% improvements on CIFAR-10 in throughput and energy efficiency, respectively; and on ImageNet, it can also slightly improve throughput, and achieve 37.84% improvements on energy efficiency.

Finally, Table IV reports the comparison results on normalized search time between the Hardware-Aware NAS and the Co-Exploration. Results in this table show that the Co-Exploration can significantly accelerate the search process, achieving 159× and 136× fewer GPU hours on CIFAR-10 and ImageNet, respectively. The speedup is achieved from the efficient early stage pruning in the fast exploration level. As discussed in Section III-A, compared with the conventional Hardware-Aware NAS with a single RNN in the controller, the proposed Co-Exploration framework with multiple RNNs can dramatically shrink the design space from $O(\prod_i D_i)$ to $O(\sum_i D_i)$, where D_i is the size of design space for the ith pipeline stage. Since the number of architecture to be trained is proportional to the size of design space, from column "arch for training" in Table IV, we can see that Co-Exploration trains much fewer architectures compared with the Hardware-Aware NAS. Therefore, our Co-Exploration achieves significant speedup over the Hardware-Aware NAS. From the table, we have another observation that the training process takes much longer time than the hardware exploration process, where the hardware exploration only occupies less

TABLE V
COMPARISON WITH THE EXISTING ARCHITECTURES ON IMAGENET WITH THE TIMING SPECIFICATION OF 10FPS

Models	Depth	Accuracy (Top-1)	Accuracy (Top-5)	FPS	Energy Eff.
MobileNetV2 [42]	18	71.80%	91.00%	4.5	0.47
ProxylessNet [8]	21	74.60%	92.50%	3.1	0.41
Co-Exploration (OptHW)	17	68.14%	89.60%	**12.1**	**1.01**
Co-Exploration (OptSW)	15	**70.24%**	**90.53%**	10.5	0.74

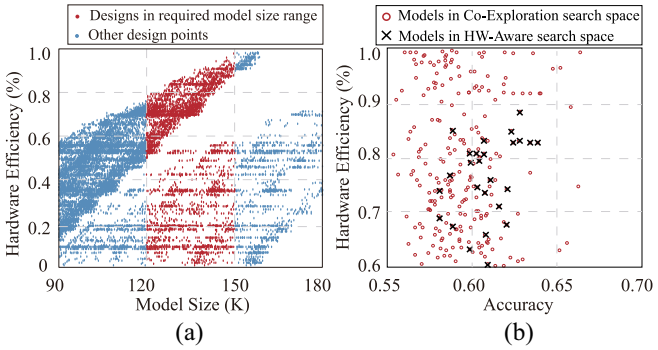

Fig. 8. Design space of architectures with the depth of 4: (a) model size versus hardware efficiency and (b) accuracy versus hardware efficiency using co-exploration and hardware-aware NAS approaches.

than 1% GPU hours in the whole search process (1.9 GPU hours for CIFAR-10 and 1.8 GPU hours for ImageNet).

C. Comparison Results With the Existing Architectures

In this section, we compare the neural architectures identified by the proposed Co-Exploration framework with the existing architectures: ProxylessNet [8] and MobileNetV2 [42]. We set the throughput constraint as 10FPS for Co-Exploration framework as a baseline. To obtain the hardware efficiency (throughput, energy efficiency, etc.) of these architectures, we apply the BLAST approach [21] to partition them onto multiple FPGAs. For a fair comparison, all models involve 3 FPGAs.

Table V reports the results. As we can see, both MobileNetV2 and ProxylessNet cannot meet the timing specification of 10 FPS, while ours can. In comparison with the manually designed MobileNetV2 [42], OptSW with top-5 accuracy loss of 0.47% can achieve 2.33× and 1.57× improvement on throughput and energy efficiency, respectively. On the other hand, in comparison with ProxylessNet [8], whose throughput is 3× lower than the specifications, OptSW can find architectures that meet the specs with 90.53% top-5 accuracy against 92.50% from ProxylessNet. Results show that the proposed framework can make a better tradeoff between hardware efficiency and architecture accuracy. In addition, it can guarantee that the final architecture identified can meet the timing specification, which is important in real-time AI systems.

D. Importance of Co-Exploration

Finally, we show the importance of co-exploration on NAS and hardware design spaces, instead of: 1) using a heuristic on restricting the size of models for only NAS exploration or 2) applying hardware-aware NAS exploration. Fig. 8 shows the results of the design space exploration of architectures with four layers.

In Fig. 8(a), the *x*-axis and *y*-axis represent the model size and the hardware efficiency (i.e., pipeline efficiency). Each point in this figure is a design, which is optimized using the algorithm in [21]. We have marked the design points whose model size ranges from 120 K to 150 K. From this figure, we can see that, for the designs whose model size ranges from 120 K to 150 K, the optimized hardware efficiency ranges from 1.29% to 98.35%. Moreover, for a much narrower range from 149 K to 150 K, the efficiency still ranges from 7.02% to 98.35%. All the above results reflect that we cannot guarantee the hardware efficiency by restricting the model size only. This is mainly because there are a large number of designs with similar model size, but their structures are quite different, leading to different hardware efficiency. Therefore, it verifies the NAS space and hardware design space are tightly coupled and emphasizes the importance of conducting hardware and software co-exploration.

In Fig. 8(b), we unveil the fundamental difference between co-exploration and hardware-aware architecture search. In this figure, the black crosses and red circles represent the valid design points in HW-aware NAS and co-exploration search spaces, respectively. We can observe that the HW-aware NAS has a much narrower search space than the proposed co-exploration approach. Basically, HW-aware NAS will prune the architectures that violates hardware specifications on a fixed hardware design. However, by opening the hardware design space, it is possible to find a tailor-made hardware design for the pruned architectures to make them meet the hardware specifications. Therefore, compared with the HW-aware NAS, the co-exploration approach enlarges the search space. As a result, it can make better tradeoffs between accuracy and hardware efficiency.

VI. CONCLUSION

We proposed the co-exploration framework to open up the hardware design freedom in NAS. This is driven by the trend that the hardware platform can be programmed or even fully customized for the best performance in cloud and edge computing applications. This article took the FPGA as a vehicle to show that through jointly exploring architecture search space and hardware design space, the design Pareto frontier on accuracy and hardware efficiency tradeoffs can be significantly pushed forward.

The framework proposed in this article will be the base for neural architecture and hardware co-exploration. Based on the proposed co-exploration framework, we list two promising future directions as follows. First, mixed-precision was recently proposed [43] for a fixed architecture; in the future, we plan to further co-explore neural architectures, quantizations and hardware designs. Second, innovations on computing architecture achieves great success for executing inference

phase of neural networks [44], we plan to apply the proposed framework to co-explore neural architectures with the novel computing architectures (e.g., computing-in-memory).

REFERENCES

[1] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd Conf. Assoc. Advan. Artif. Intell. (AAAI)*, 2018, pp. 2787–2794.

[2] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

[3] E. Real *et al.*, "Large-scale evolution of image classifiers," 2017. [Online]. Available: arXiv:1703.01041.

[4] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017. [Online]. Available: arXiv:1711.00436.

[5] V. Nekrasov, H. Chen, C. Shen, and I. Reid, "Architecture search of dynamic cells for semantic video segmentation," 2019. [Online]. Available: arXiv:1904.02371.

[6] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," 2018. [Online]. Available: arXiv:1812.03443.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," 2018. [Online]. Available: arXiv:1807.11626.

[8] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018. [Online]. Available: arXiv:1812.00332.

[9] *EC2 F1 Instances*, Amazon, Seattle, WA, USA, 2017. Accessed: Jan. 20, 2019. [Online]. Available: https://aws.amazon.com/ ec2/instance-types/f1

[10] *Real-Time AI: Microsoft Announces Preview of Project Brainwave*, Microsoft, Redmond, WA, USA, 2018. Accessed: Jan. 20, 2019. [Online]. Available: https://blogs.microsoft.com/ai/build-2018-project-brainwave/

[11] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, "Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA," in *Proc. IEEE 28th Int. Conf. Field Program. Logic Appl. (FPL)*, Dublin, Ireland, 2018, pp. 163–1636.

[12] F. Shafiq, T. Yamada, A. T. Vilchez, and S. Dasgupta, "Automated flow for compressing convolution neural networks for efficient edge-computation with FPGA," 2017. [Online]. Available: arXiv:1712.06272.

[13] S. Venkataramani *et al.*, "SCALEDEEP: A scalable compute architecture for learning and evaluating deep networks," *ACM SIGARCH Comput. Architect. News*, vol. 45, no. 2, pp. 13–26. 2017.

[14] P. Whatmough, S. Lee, N. Mulholland, P. Hansen, S. Kodali, D. Brooks, and G. Wei, "DNN ENGINE: A 16nm sub-uj deep neural network inference accelerator for the embedded masses," in *Proc. IEEE Hot Chips 29 Symp.*, 2017.

[15] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2016, pp. 326–331.

[16] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[17] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018. [Online]. Available: arXiv:1806.09055.

[18] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholm, Sweden, 2018, pp. 549–558.

[19] E. Chung *et al.*, "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar./Apr. 2018.

[20] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. 45th Int. Symp. Comput. Architect. (ISCA)*, Los Angeles, CA, USA, 2018, pp. 1–14.

[21] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, "Heterogeneous FPGA-based cost-optimal design for timing-constrained CNNs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2542–2554, Nov. 2018.

[22] W. Zhang, J. Zhang, M. Shen, G. Luo, and N. Xiao, "An efficient mapping approach to large-scale DNNs on multi-FPGA architectures," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Florence, Italy, 2019, pp. 1–4.

[23] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, "A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing," in *Proc. 28th Int. Conf. Field Program. Logic Appl. (FPL)*, Dublin, Ireland, 2018, pp. 394–3944.

[24] T. Geng *et al.*, "FPDeep: Acceleration and load balancing of CNN training on FPGA clusters," in *Proc. 26th Int. Symp. Field Program. Custom Comput. Mach. (FCCM)*, Boulder, CO, USA, 2018, pp. 81–84.

[25] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. Int. Workshop Comb. Genet. Algorithms Neural Netw. (COGANN)*, Baltimore, MD, USA, 1992, pp. 1–37.

[26] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016. [Online]. Available: arXiv:1611.02167.

[27] L. Xie and A. Yuille, "Genetic CNN," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, 2017, pp. 1388–1397.

[28] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. ICML AutoML Workshop*, 2017, pp. 1–8.

[29] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. Int. Symp. Field Program. Gate Arrays (FPGA)*, 2015, pp. 161–170.

[30] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. Int. Symp. Comput. Architect. (ISCA)*, 2017, pp. 535–547.

[31] X. Zhang *et al.*, "DNNbuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAS," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Diego, CA, USA, 2018, p. 56.

[32] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "TGPA: Tile-grained pipeline architecture for low latency CNN inference," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Diego, CA, USA, 2018, pp. 1–8.

[33] W. Jiang *et al.*, "Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[34] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[35] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, H. Dong, and X. Chen, "On the design of minimal-cost pipeline systems satisfying hard/soft real-time constraints," *IEEE Trans. Emerg. Topics Comput.*, early access, Jan. 1, 2018, doi: 10.1109/TETC.2017.2788800.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: arXiv:1502.03167.

[39] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 843–856, Apr. 2020.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.

[41] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018. [Online]. Available: arXiv:1802.05799.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2018. [Online]. Available: arXiv:1801.04381.

[43] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 8612–8620.

[44] W.-H. Chen *et al.*, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 494–496.

Weiwen Jiang received the Ph.D. degree from the Department of Computer Science, Chongqing University, Chongqing, China.

He was a Research Scholar with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA, from October 2017 to June 2019. He is currently a Postdoctoral Associate with the University of Notre Dame, Notre Dame, IN, USA. His current research interests include neural architecture search, FPGAs, non-volatile memories, and HW/SW co-optimization.

Dr. Jiang was a recipient of Best Paper Awards in ICCD'17, and Best Paper Nominations in DAC'19, CODES+ISSS'19, and ASP-DAC'20.

Lei Yang received the B.E. and Ph.D. degrees from Chongqing University, Chongqing, China, in 2019 and 2013, respectively.

She was a Research Scholar with the University of California at Irvine, Irvine, CA, USA, from October 2017 to February 2019, and a Research Scholar with the University of Pittsburgh, Pittsburgh, PA, USA, from February 2019 to August 2019. She is currently a Postdoctoral Research Associate with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA. Her research interests are in automated machine learning, embedded systems, and high-performance computing architectures.

Edwin Hsing-Mean Sha (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science, Princeton University, USA, in 1992.

From August 1992 to August 2000, he was with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA. Since 2000, he has been a Tenured Full Professor with the University of Texas at Dallas, Dallas, TX, USA. Since 2012, he has been served as the Dean of the College of Computer Science, Chongqing University, Chongqing, China. He is currently a Distinguished Professor with East China Normal University, Shanghai, China. He has published more than 390 research papers in refereed conferences and journals.

Dr. Sha received the Teaching Award, the Microsoft Trustworthy Computing Curriculum Award, the NSF CAREER Award, and the NSFC Overseas Distinguished Young Scholar Award, the Chang-Jiang Honorary Chair Professorship, and China Thousand-Talent Program.

Qingfeng Zhuge received the B.S. and M.S. degrees in electronics engineering from Fudan University, Shanghai, China, and the Ph.D. degree from the Department of Computer Science, University of Texas at Dallas, Dallas, TX, USA, in 2003.

She is currently a Professor with East China Normal University, Shanghai, China. She has published more than 90 research articles in premier journals and conferences. Her research interests include parallel architectures, embedded systems, supply chain management, real-time systems, optimization algorithms, compilers, and scheduling.

Dr. Zhuge received the Best Ph.D. Dissertation Award in 2003.

Shouzhen Gu received the Ph.D. degree from the Department of Computer Science, Chongqing University, Chongqing, China. He is currently an Assistant Professor with the School of Software Engineering, East China Normal University, Shanghai, China. Her research interests are scheduling and data allocation on MPSoC, optimization and management of nonvolatile memory, optimization for embedded systems.

Sakyasingha Dasgupta (Member, IEEE) received the Dr.rer.nat doctoral degree from the Max Planck Institute for Dynamics and Self Organization, University of Göttingen, Göttingen, Germany, and the master's degree in artificial intelligence from the University of Edinburgh, Edinburgh, U.K., and the MIT Sloan School of Management, Cambridge, MA, USA.

He was a Senior Research Scientist and a Lead with IBM Research, Yorktown Heights, NY, USA. He is the CEO and Founder with Edgecortix Inc., a deep tech startup based in Tokyo, Japan, and Singapore, automating machine learning driven AI hardware and software co-design for an intelligent distributed edge ecosystem. He has held senior research and development positions at organizations like RIKEN Center for Brain Science, Saitama, Japan, Microsoft, Redmond, WA, USA, and IBM Research with over a decade of experience in AI, robotics and brain-inspired computing. He has filed over 15 patents and published widely in the areas of machine learning and neural computation.

Yiyu Shi (Senior Member, IEEE) received the B.S. degree (Hons.) in electronic engineering from Tsinghua University, Beijing, China, in 2005, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2007 and 2009, respectively.

He is currently an Associate Professor with the Departments of Computer Science and Engineering and Electrical Engineering, University of Notre Dame, Notre Dame, IN, USA. His current research interests include 3-D integrated circuits, hardware security, and renewable energy applications.

Prof. Shi was a recipient of several best paper nominations in top conferences, the IBM Invention Achievement Award in 2009, the Japan Society for the Promotion of Science Faculty Invitation Fellowship, the Humboldt Research Fellowship for Experienced Researchers, the National Science Foundation CAREER Award, the IEEE Region 5 Outstanding Individual Achievement Award, and the Air Force Summer Faculty Fellowship.

Jingtong Hu (Member, IEEE) received the B.E. degree from the School of Computer Science and Technology, Shandong University, Jinan, China, in 2007, and the Ph.D. degree in computer science from the University of Texas at Dallas, Dallas, TX, USA, in 2013.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include embedded systems, nonvolatile memory, wireless sensor network, and cyber-physical systems.