Standing on the Shoulders of Giants: Hardware and Neural Architecture Co-Search With Hot Start

Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi

***Abstract*—Hardware and neural architecture co-search that automatically generates artificial intelligence (AI) solutions from a given dataset are promising to promote AI democratization; however, the amount of time that is required by current co-search frameworks is in the order of hundreds of GPU hours for one target hardware. This inhibits the use of such frameworks on commodity hardware. The root cause of the low efficiency in existing co-search frameworks is the fact that they start from a "cold" state (i.e., search from scratch). In this article, we propose a novel framework, namely, HotNAS, that starts from a "hot" state based on a set of existing pretrained models (also known as model zoo) to avoid lengthy training time. As such, the search time can be reduced from 200 GPU hours to less than 3 GPU hours. In HotNAS, in addition to hardware design space and neural architecture search space, we further integrate a compression space to conduct model compressing during the co-search, which creates new opportunities to reduce latency, but also brings challenges. One of the key challenges is that all of the above search spaces are coupled with each other, e.g., compression may not work without hardware design support. To tackle this issue, HotNAS builds a chain of tools to design hardware to support compression, based on which a global optimizer is developed to automatically co-search all the involved search spaces. Experiments on ImageNet dataset and Xilinx FPGA show that, within the timing constraint of 5 ms, neural architectures generated by HotNAS can achieve up to 5.79% Top-1 and 3.97% Top-5 accuracy gain, compared with the existing ones.**

***Index Terms*—FPGA, HW/SW co-design, neural network.**

I. Introduction

THE SUCCESS of deep neural networks (DNNs), has propelled artificial intelligence (AI) in entering every aspect of our lives and is being widely employed for diverse applications on different types of hardware. neural architecture search (NAS), a successful product of automatic machine learning (AutoML), has paved the way from a given dataset to a neural architecture with state-of-the-art accuracy. Moving forward, to be able to use AI for enabling and accelerating different application, we need to be able to design the neural network in a way that the design specifications are met on our target hardware; for instance, real-time constraints for edge devices, low power budgets for IoT devices, etc.

Recently, neural architecture and hardware design (architecture-hardware) co-search frameworks [1]–[9] have been proposed to bridge the gap between neural architecture and hardware design. These frameworks have demonstrated promising results in generating high-accuracy and low-cost systems. However, their search efficiency is low: existing co-search frameworks commonly take hundreds of GPU hours per target hardware. This may become the bottleneck in many emerging applications where fast turn around or short time to market is desired. On the other hand, it has already been shown that the carbon footprint (pounds of CO_2) of NAS for one model is nearly equivalent to five times the lifetime emissions of a car [10]. In this article, we are revisiting the default setting used by existing co-search frameworks, where: the exploration always starts from scratch (i.e., cold start), which results in large search time and low efficiency. However, is a cold start really necessary?

We claim that the architecture-hardware co-search could stand on the shoulders of giants and start the search from a hot state, i.e., using an existing pretrained model in a model zoo. The model zoo can be efficiently created, consisting of the existing neural architectures manually designed by domain experts, identified by NAS, or transferred from models from different datasets. To make full use of the candidates in the model zoo, in this article, we propose a novel co-search framework, namely "HotNAS," to start searching from a hot state. In this way, compared with the cold-start co-search, HotNAS can reduce the search time from hundreds of GPU hours to less than 3 GPU hours for ImageNet and 20 GPU minutes for CIFAR-10 without proxy; while achieving accuracy comparable with the state-of-the-art models.

Fig. 1 shows the results of co-search using a model zoo with 24 models on ImageNet dataset, targeting a system with 5 ms on Xilinx ZCU 102 FPGA. From the top figure, there are only four models that can satisfy the timing constraint and the highest accuracy is 87.50%; however, within the range from 5 to 10 ms, there are a lot of good candidates with accuracy higher than 90%. The existing co-search frameworks ignore these candidates and search from scratch, leading to hundreds of GPU hours. Viewing from the opposite angle, HotNAS takes full use of these pretrained models and customize the models that violate time constraints but have high accuracy to the

Manuscript received April 17, 2020; revised June 12, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported in part by the National Science Foundation under Grant CCF-1919167, Grant CCF-1820537, and Grant CNS-1822099. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2020 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Weiwen Jiang.)*

Weiwen Jiang 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).

Lei Yang is with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131 USA.

Sakyasingha Dasgupta is with Edgecortix Inc., Tokyo 1410031, Japan.

Jingtong Hu is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA.

Digital Object Identifier 10.1109/TCAD.2020.3012863

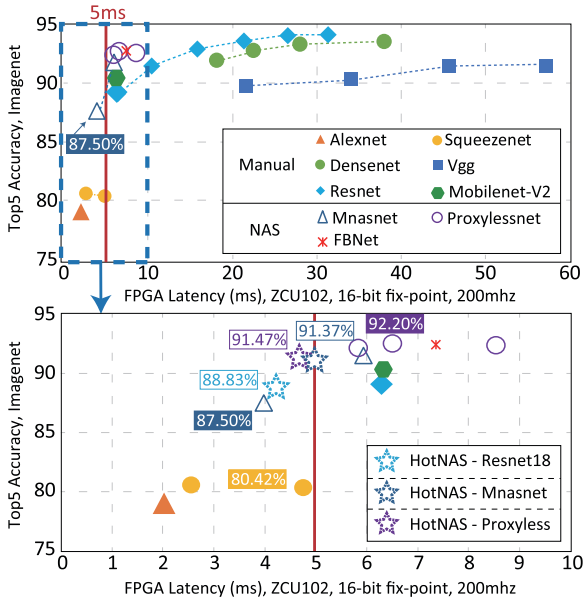


Fig. 1. Architecture-hardware co-search by HotNAS: (top) latency and accuracy of models in the model zoo; (bottom) architectures in model zoo and that identified by HotNASwith 5-ms timing constraint (Best viewed in color).

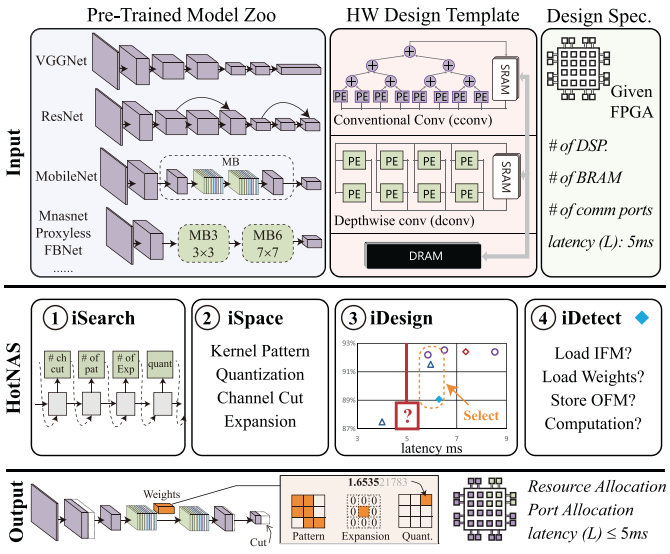


Fig. 2. Implementation from the model zoo to hardware: (top) the given pretrained model and design spec.; (middle) four components in the proposed HotNAS framework; (bottom) outputting the best neural architecture and hardware design.

target hardware. As such, HotNAS can avoid lengthy training procedure to generate the solution in a couple of hours, which is guaranteed to meet timing constraints while greatly improving accuracy to 91.47%, as shown in the bottom figure.

Seemingly straightforward, the architecture-hardware co-search from a hot start is not a simple matter: a fundamental challenge is the discovery of the best search space. Some of the prior co-search works [4]–[6], [11] consider hardware design space of loop tiling and loop order, and NAS space with flexibility across the number of channels, filter size, and model quantization. However, we observe that one of the most efficient techniques, model pruning [12]–[15], has hitherto not been combined in the co-search. Integrating model pruning faces a lot of challenges: First, without the full consideration of hardware design, the model pruning can easily become useless since it introduces overheads. Second, one compression technique does not work for all performance bottlenecks. Finally, the model compression techniques are tightly coupled with hardware design and NAS: As such, a difficult challenge is to simultaneously optimize all these spaces.

In HotNAS, we address the above challenges by collaboration among four subcomponents: 1) iSearch; 2) iSpace; 3) iDesign; and 4) iDetect. First, iDesign provides hardware design support for different compression techniques. Second, following the observation that different pruning techniques work for different types of bottlenecks; iDetect is developed to identify performance bottleneck for each layer so that we can select the most suitable compression techniques to alleviate performance bottlenecks. According to the detected bottlenecks, iSpace creates a dedicated search space for each layer. Finally, iSearch is devised to jointly search the hardware, neural architecture, and model compression using specification from iSpace.

The main contributions of this article are threefold.

1) We propose a novel NAS mechanism to search from a hot state (i.e., a pretrained model), which allows us to reduce search time from 200 GPU hours to 3 GPU hours; meanwhile, the solution can improve the Top-1 and Top-5 accuracy on the ImageNet dataset by 5.79% and 3.97%, respectively.
2) An automated HotNAS framework is proposed to link the hardware design, NAS, and model compression to automatically generate the architecture and hardware pair, such that the timing constraint can be met with the maximum model accuracy.
3) In HotNAS, dedicated hardware designs to support the existing model compression are proposed, without which the model compression techniques may not achieve any performance gain at all.

The remainder of this article is organized as follows. Section II presents design challenges and motivation. Section III presents the proposed HotNAS. Experimental results are shown in Section IV. Finally, concluding remarks are given in Section V.

II. Challenges and Motivation

This section demonstrates the challenges in the architecture-hardware co-search, and gives the motivation of this article.

Fig. 2 demonstrates the architecture-hardware co-search problem, where we have a set of pretrained models (called model zoo), the hardware design templates and the design specifications (e.g., constraints on the resource, area, and latency) as inputs. The co-search is to optimize neural architectures in the model zoo and hardware design to guarantee all design constraints to be met while maximizing accuracy, as shown in the bottom part in Fig. 2

HotNAS framework is proposed in this article to solve the above problem. As illustrated in the middle part of Fig. 2, it is composed of four subcomponents: ① iSearch, starting

TABLE I
SEARCH COST (GPU HOURS) OF NAS IS TOO HIGH. NOTE THAT THE HARDWARE-AWARE APPROACH NEEDS AN ENTIRE SEARCH FOR EACH SPECIFIC HARDWARE

NAS	NASNet	DARTS	MnasNet	FNAS	FBNet	ProxylessNAS
GPU Hours	48,000	90	40,000	267	216	200

the co-search from hot instead of cold; ② iSpace, building an integrated search space which is in accordance with the performance bottleneck in the implementation; ③ iDesign, providing the design to support compression techniques on FPGAs; and ④ iDetect, detecting the performance bottleneck to guide the creation of iSpace. In the following text, we will show that there exists a couple of challenges in architecture-hardware co-search and all components work collaboratively to address these challenges.

Challenge 1 (How to Efficiently Explore Neural Architectures): The order of hundreds of GPU hours in architecture-hardware co-search cannot satisfy the short time-to-market requirements in many applications; as reported in Table I, the state-of-the-art hardware agnostic NAS techniques (DARTS [16]) requires 90 GPU hours, while the NAS for a specific type of hardware (MnasNet [3], FNAS [5], FBNet [1], and ProxylessNAS [2]) requires over 200 GPU hours. Considering that the current computing system is composed of a large variety of hardware, the search process is simply unacceptable. In addition, the long search time leads to excessive CO_2 emission, which has already been known as a serious problem of existing NAS techniques [10].

We observe that the long search time in the NAS framework is caused by the cold start. This leads to more than 40 000 GPU hours for MnasNet and NASNet to train a large number of potential architectures from scratch, and over 200 GPU hours when the hardware is considered. However, there exists a large set of pretrained neural networks. We revisit the default configuration in the co-search framework: i.e., whether it is necessary to start the exploration from scratch, which results in low efficiency. In ① iSearch, we propose to make full use of the existing models and start the exploration from a hot state (e.g., pretrained models).

Challenge 2 (Meet Real-Time Constraint on Specific Hardware): Arbitrarily picking neural networks from the model zoo and plugging into the given hardware will lead to violations of the design specification, e.g., missing deadline in real-time systems. On the other hand, due to the large variety of hardware (different types of CPU, GPU, FPGA), it is infeasible to conduct co-search for all off-the-shelf hardware in advance.

From results in Fig. 1(a) and (b), we observe that only four models can satisfy the timing constraints with the highest accuracy of 87.50%; while there are a group of networks whose accuracy is much higher, over 92%, with the latency slightly exceeding the timing constraint.

The challenge here is, how can we compress the models to satisfy the timing constraints using its pretrained weights, while a competitive model accuracy can be achieved. ② iSpace is developed to involve model compression in the search space, together with the hardware design space and neural architectures search space.

III. PROPOSED FRAMEWORK: HOTNAS

In response to all the challenges described in the previous section, we propose HotNAS framework in this section. As shown in Fig. 2, HotNAS is composed of four subcomponents, ① iSearch, ② iSpace, ③ iDesign, and ④ iDetect. This section will introduce these subcomponents in detail.

① *iSearch: Search from Hot Start* Fig. 3 illustrates the overview of iSearch, which conducts the NAS in two steps: 1) (top part of figure), it selects backbone architectures to be optimized and then 2) (bottom part of figure), an optimizer tunes hyperparameters of neural architecture and hardware design simultaneously. The goal of iSearch is to find the architecture with the highest accuracy while meeting hardware design specifications. In the following texts, we will formally define the problem, and introduce the optimizer at the end of this section.

Neural Architectures and Model Zoo: A neural architecture is defined as $A = \langle V, E, r, c, ch, o, f, para, \text{acc}\rangle$, composed of a set of nodes V representing the intermediate data [i.e., input and output feature maps (OFMs)], a set of edges $E \subseteq V \times V$ representing the dependency between a pair of nodes. For a node v_i in V, it has three hyperparameters r_i, c_i, and ch_i representing the number of row, column, and channel of v_i. For an edge $e_j \in E$, an operator o_j (e.g., convolution, depthwise convolution, or pooling, etc.) is associated to it. f_j represents the filter (i.e., weights) used in operator o_j, which is composed of a set of kernels. Each filter is associated with two hyperparameters: 1) $s(f_i)$ indicates the size of the filter (e.g., 1×1, 3×3 etc.) and 2) $p(f_i)$ is a pattern applied to prune f_i. Both the size and the pattern of the filter are tunable, which will be introduced in ② iSpace. After all the above hyperparameters are determined and the neural architecture A is identified, it can be trained/fine-tuned on the training datasets (e.g., ImageNet) to obtain the parameters/weights $para(A)$, and finally we can obtain its test accuracy $\text{acc}(A)$ on the test dataset.

A pretrained neural architecture is also called a model, and a model zoo $M = \{A_0, A_1, \ldots, A_{N-1}\}$ is composed of N models. These models can be manually designed by experts, like AlexNet, VGGNet, ResNet, automatically searched via NAS, like MnasNet, ProxylessNas, FBNet, or transferred from models for other datasets, like BiT [17]. In this article, we use the existing model zoo from torchvison, and collect the state-of-the-art pretrained models from github; hence, the cost of building the model zoo can be neglected. Kindly note that, how to create the model zoo is out of the scope of this article; related works can be found in [18] and [19].

FPGA and Accelerator Design: The hardware efficiency is not only related to the neural architecture but also the resource on the FPGA and the accelerator design on it. An FPGA fp has three attributes: 1) mem_{fp}; 2) comp_{fp}; and 3) BW_{fp}, referring to the on-chip memory size, the number of computing resources (e.g., DSPs), and the bandwidth between off-chip and on-chip memories, respectively.

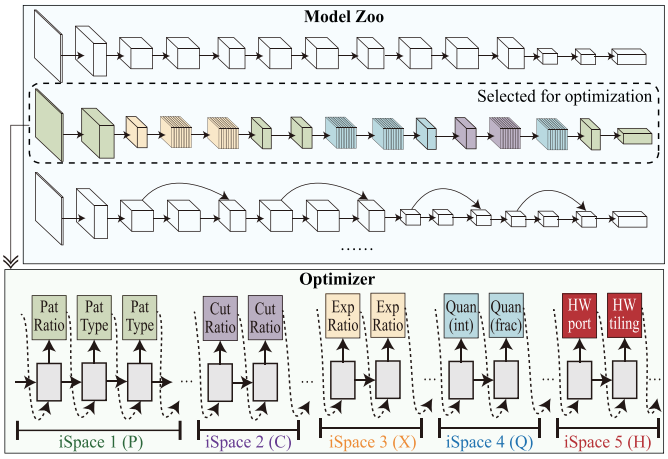


Fig. 3. iSearch, iSpace, iDesign, and iDetect: (top) iSearch selects an architecture from the model zoo for optimization based on the results of iDesign and iDetect; (bottom) iSpace creates the search space in terms of performance bottlenecks, and iSearch determines hyperparameters for each search space.

The accelerator design should meet all resource constraints of a given FPGA. It is composed of two parts: 1) the design of the computing subsystem and 2) the design of the communication subsystem. As the basic operators o in architecture A are conducted in nested loops, the loop optimizations, in particular the loop tiling, are widely studied and used in the design of the computing subsystem in FPGAs [20], [21]. In addition, with the consideration of the large amount of data (i.e., intermediate data and weights), and the limited on-chip buffer in FPGA, it is infeasible to put all data on FPGA. Therefore, data are moved between the off-chip and on-chip memories. As such, the communication bandwidth for moving each type of data needs to be determined in the design phase.

As a whole, the accelerator design is defined as $D = \langle T_m, T_n, T_r, T_c, I_b, W_b, O_b\rangle$, containing the loop tiling design $\langle T_m, T_n, T_r, T_c\rangle$ and bandwidth allocation $\langle I_b, O_b, W_b\rangle$. Specifically, for an operator o_k associated to a pair of nodes $v_i \rightarrow v_j$ in an architecture, T_m, T_n, T_r, T_c are the tiling parameters on OFM channels ch_j, input feature maps (IFMs) channels ch_i, rows r_i, and columns c_i; while $\langle I_b, O_b, W_b\rangle$ are the bandwidth allocated for moving IFM (i.e., v_i), OFM (i.e., v_j), and weights (i.e., f_k). For a design D and an architecture A, the latency of each operator, say o_k, can be determined with ③ iDesign tool. Then, the summation of all operators will be the latency of A, denoted as lat(A).

iSearch: Two-Step Exploration In iSearch, the first step is to select a set of candidate backbone architectures to be optimized. Given a neural architecture and an FPGA, it has already been well studied to obtain the best accelerator design D, as in [20]. Based on the design, HotNAS can generate the search space iSpace (Section III ②). iSearch will select models from the model zoo to be the backbone architecture, which will be the starting point of HotNAS, as shown in the top of Fig. 3. The selection process is based on a Monte Carlo test, where we are given a timing constraint *TC* and the search space iSpace. We can prune the models whose minimum latency in the test fails to meet *TC*. The feasible architectures will be sorted in terms of a weighted reward [will be introduced in (18)] in terms of the minimal latency and original accuracy. Then, Top-K architectures will be selected as a starting point, where K is a user-defined variable.

Now, iSearch gets into the second step to conduct the NAS based on these selected models to make them meet the given timing constraint with high accuracy. iSpace tool will provide search spaces for iSearch, including the filter patterning P, channel cutting C, quantization Q, filter expansion X, and hardware design H. All these search spaces are coupled with each other. In iSearch tool, we develop a reinforcement learning-based optimizer to simultaneously explore all these spaces. Kindly note that other optimization techniques, such as evolutionary algorithms [22] can be easily plugged into the iSearch tool. For better understanding, we will present the details of the optimizer at the end of this section.

Problem Definition: Based on all the above definitions, we formally define the architecture-hardware co-search optimization problem as follows: given a model zoo M, a specific FPGA FP, the accelerator design D_i of model A_i in M on FP, a target timing constraint T, and the baseline accuracy acc_*baseline* we are going to determine.

1) *S:* Selection of architectures from zoo M, denoted as A_0.
2) *P, C, X, Q:* Tuning architecture hyperarameters of A_0.
3) *H:* Tuning hardware design hyperparameters on D_0.

Such that a new architecture A_0' with competitive accuracy over acc_baseline can be identified, while A_0' under hardware design D_0' can meet the timing constraint T.

② *iSpace: An Integrated Search Space* iSpace links the compression technique with the NAS and hardware design. In this article, we consider three model compression techniques: (i) pattern pruning; (ii) channel pruning; and (iii) quantization. In the NAS space, we consider *iv)* filter expansion; for hardware design space, we mainly consider v) communication bandwidth allocation and loop tiling, because FPGA accelerator design has typical templates which provides the above two kinds of hyperparameters in the design phase. Kindly note that, HotNAS is an open framework, with designers having the flexibility to modify or add new search parameters in terms of design needs. As an example, dataflow and loop order can be further integrated into hardware design space when it comes to ASIC design. However, this is out the scope of this article.

i) P: Pattern Pruning The first search space is the pattern pruning space, which prunes the filter in the neural architecture A. A pattern is defined as a mask matrix Mat[x][y]; Mat[x][y] $= 0$ indicating that the weights at position $\langle x, y\rangle$ will be pruned, while Mat[x][y] $= 1$ indicates that the weights will remain. According to the number of 0 in Mat[x][y], we can classify the pattern into different categories, and we use PAT_c to indicate the number of 0 in the pattern, as shown in Fig. 4. Among all patterns, one category will be selected for pruning. Each pattern category is further composed of many patterns; for instance, there are 84 potential patterns in the category of $\text{PAT}_c = 3$, as shown in Fig. 4. For the hardware implementation, it simply cannot apply so many patterns as this will results in a large number of multiplexers in hardware implementation, making the design inefficient. In consequence, we will select a small number of patterns from the selected category, denoted as PAT_n. Fig. 4 gives the example of the pattern

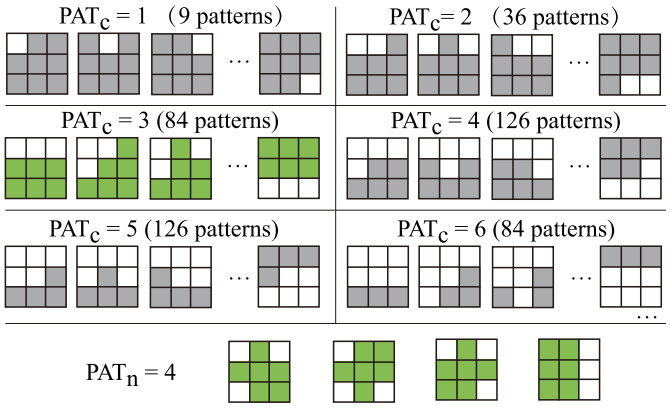


Fig. 4. Patterns pruning for 3 × 3 filters with different pruning categories (PAT_c); each big square represent a pattern, and each blank square in a pattern indicates the weights in the corresponding position to be pruned. The last row shows the number of patterns (PAT_n) and the specific patterns to be selected for pruning filters.

pruning space for 3 × 3 filter, which selects the category of $\text{PAT}_c = 3$ and applies $\text{PAT}_n = 4$ patterns among 84 candidates.

The selected patterns will be applied for a set of filters. As demonstrated in [15], by applying the Euclidean norm, we can specify one pattern for each kernel in a filter, i.e., the determination of $p(f_i)$ (see the definition in ① iSearch). However, when implementing pattern pruning on hardware, the following two questions needing to be answered.

1) How many kernels in a filter will be pruned by each type of pattern.
2) Whether all layers need to be pruned or which layers will be pruned.

For the first question, it is related to the tiling design parameters. In a tile, if multiple types of patterns are applied, it will break the execution pipeline and pattern pruning cannot improve performance at all. This will be shown in ③ iDesign. For the second question, applying patterns for the layers whose performance bottleneck is at communication, it will not help in improving performance but may reduce accuracy. Details will be illustrated in ④ iDetect.

ii) C: Channel Pruning Unlike pattern pruning that changes the structure, the neural architecture will not be changed, with the channel pruning modifying the number of channels for a node $v_i \in V$ in architecture A. The left figure in Fig. 5 shows the channel pruning, where CUT_n represents the number of channels to be cut off. We take $\text{CUT}_n = 2$ in this example. There are three consecutive nodes $v_i \rightarrow v_j \rightarrow v_k$, and we perform the channel pruning on v_j. In this figure, the gray channels in v_j indicate the ones to be cut off. A ripple effect is taken to both filters of $f_{i\rightarrow j}$ and $f_{j\rightarrow k}$. However, as the channel pruning may easily result in the accuracy drop since features are directly removed, we carefully formulate its search space for channel pruning only if the performance bottlenecks cannot be alleviated by other techniques (details in ④ iDetect).

iii) Q: Quantization Quantization is another model compression technique. In general, the original model applies the data type of 32-b floating point, and we can convert it to the 16-b fixed point without accuracy loss. Such a fixed point representation is composed of two parts, the integer and fraction

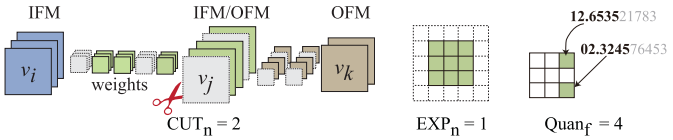


Fig. 5. Three architecture search spaces: (left) channel cutting with ratio parameter CUT_r, (middle) kernel expansion with size parameter EXP_s, and (right) weight quantization with fraction parameter Quan_f.

parts represented by $\langle I, F\rangle$. For a given pretrained architecture A, we can get the maximum and minimum parameters of one operator. Then, we can analyze the number of bits required by integer part I. Since the integer part is the most-significant bits, we will keep its bit width, and further squeeze the fraction part F only, denoted as Quan_f as shown in the right part of Fig. 5. As will show in ④ iDetect, not all layers need to perform quantization, since it cannot alleviate specific types of performance bottlenecks.

iv) E: Filter Expansion The previous three search spaces belong to model compression; while filter expansion belongs to NAS space. This is motivated by the following two aspects: 1) many state-of-the-art neural architectures identified by NAS contains larger sized filters and 2) for specific layers, the increase of filter sizes will not add latency overhead. This will be shown in ④ iDetect. We define EXP_n as the expansion factor on a filter, as shown in the middle part of Fig. 5.

Furthermore, we have the following theorem to guarantee that the accuracy will not be reduced by expanding the kernel.

Theorem 1: Given a pretrained model $A = \langle V, E, r, c, ch, o, f, \text{para}, \text{acc}\rangle$, for any operator o_i on edge e_i, the expansion on filter f_i with factor EXP_n will not decrease the accuracy, if the initial weights of the newly added weights on f_i are set to 0, and o_i is padded by EXP_n.

The proof of the above property is straightforward, since all computations remain the same when we increase the kernel size and padding with extra 0s. With the guarantee of no accuracy loss, the expanded kernel makes it possible to increase accuracy after a fine-tuned process.

v) H: Hardware Design Space Finally, after the modifications to architectures, the original hardware design identified by the optimization algorithms may not be the optimal one. In iSpace, we also provide flexibility to modify the hardware design and build the hardware design space. In particular, according to the existing performance bottleneck, we create a search space to adjust bandwidth-related design hyperparameters: $\langle I_b, O_b, W_b\rangle$, and computation-related design hyperparameters, $\langle T_m, T_n, T_r, T_c\rangle$.

③ *iDesign: Compression-Aware Performance Model*

Fig. 6 demonstrates the overview of system design, where the left-hand part is the off-chip memory to hold IFM, OFM, and weight; while the right-hand part is the on-chip accelerator design that implements both conventional convolution and depthwise convolution using on-chip computing resource (e.g., DSPs). In the accelerator design, say conventional convolution, a set of multiplication-and-accumulation are computed in parallel. Such a design has been using in many research works [20], [23]; however, it still lacks a systematic model to efficiently support depthwise convolution and different

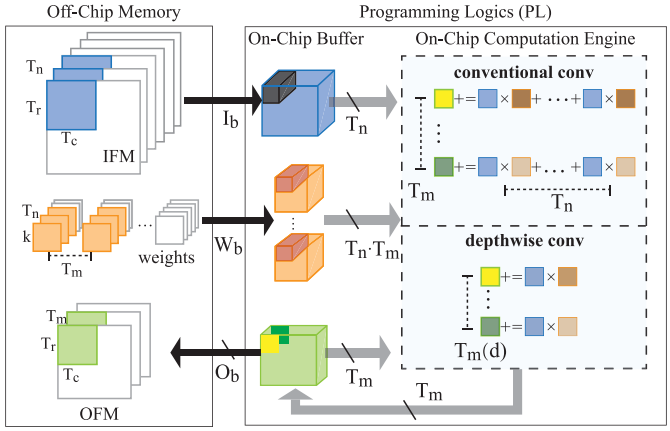


Fig. 6. Illustration of accelerator architecture and design parameters: (left) off-chip memory to hold intermediate data and weights; (right) on-chip buffer and accelerator design.

compression techniques. In the following text, we will first overview the performance model of the conventional convolution [23], and then we revise the performance model to support depthwise convolution and compression.

First, we introduce the computing accelerator part. Let $\mathbb{D}$ be the number of DSPs in the given FPGA, and K be the size of the filter. As shown in the right-hand part in Fig. 6, the conventional convolution involves $T_m \times T_n$ multiplication and additions (MACs). For 16-b data, each MAC needs one DSP. In addition, to consume all data in on-chip buffers, it needs to repeat $K \cdot K \cdot Tr \cdot Tc$ times for computation; and the pipeline initial interval (II) is optimized to 1 cycle. Therefore, we have the following constraints on computing resources and latency:

$$T_m \times T_n \leq \mathbb{D} \tag{1}$$

$$t\text{Comp} = K \cdot K \cdot \text{Tr} \cdot Tc \times 1 \tag{2}$$

where tComp is the latency of computation for all data provided by the on-chip buffer.

Second, the size of the on-chip buffer is limited by $\mathbb{B}$. There are three types of data in communication: 1) IFM; 2) OFM; and 3) weights. We need to determine the on-chip buffer size for each type of data, denoted as bI, bO, bW, which can be easily obtained from the left part in Fig. 6. Kindly note that the size of one on-chip buffer (BRAM) is limited, say 18Kb for ZCU102. For the dimension of data that needs to be accessed in parallel (e.g., channels of IFM, i.e., T_n), they need to be placed in different BRAMs. Hence, the amount of data without a parallel requirement (e.g., T_r and T_c in IFM) is divided by 18Kb. Finally, the size of the buffer is equal to two times tile size, where 2 indicates the double buffer utilized to hide communication by computation. We have the following constraints:

$$bI = 2 \times T_n \times \lceil T_r \cdot T_c \cdot \text{bit}_I / 18Kb \rceil \tag{3}$$

$$bO = 2 \times T_m \times \lceil T_r \cdot T_c \cdot \text{bit}_O / 18Kb \rceil \tag{4}$$

$$bW = 2 \times T_m \times T_n \times \lceil K \cdot K \cdot \text{bit}_W / 18Kb \rceil \tag{5}$$

$$bI + bO + bW \leq \mathbb{B} \tag{6}$$

where bit_I, bit_W, and bit_O are the bit width of the data type used for IFM, weights, and OFM.

Third, based on the buffer size and the bandwidth (I_b, W_b, O_b) allocated for each type of data buffer, we can get the communication latency (tI_{mem}, tW_{mem}, tO_{mem}) as follows:

$$tI_{\text{mem}} = \lceil T_n \cdot T_r \cdot T_c \cdot \text{bit}_I / I_b \rceil \tag{7}$$

$$tW_{\text{mem}} = \lceil T_m \cdot T_n \cdot K \cdot K \cdot \text{bit}_W / W_b \rceil \tag{8}$$

$$tO_{\text{mem}} = \lceil T_m \cdot T_r \cdot \text{bit}_O \cdot T_c / O_b \rceil \tag{9}$$

$$(I_b + W_b + O_b) \leq \mathbb{W} \tag{10}$$

where $\mathbb{W}$ is the maximum bandwidth between off-chip memory and on-chip memory.

Finally, based on the above formulations, we can derive the latency model. Let M, N, R, C be the number of OFM channels, IFM channels, rows, and columns of the convolution layer. We have the following models:

$$\text{Lat}_1 = \max\{t\text{Comp}, tI_{\text{mem}}, tW_{\text{mem}}\} \tag{11}$$

$$\text{Lat}_2 = \max\left\{\left\lceil \frac{N}{T_n} \right\rceil \cdot \text{Lat}_1, tO_{\text{mem}}\right\} \tag{12}$$

$$\text{Lat} = \left\lceil \frac{R}{T_r} \right\rceil \times \left\lceil \frac{C}{T_c} \right\rceil \times \left\lceil \frac{M}{T_m} \right\rceil \times \text{Lat}_2 + (tO_{\text{mem}} + \text{Lat}_1). \tag{13}$$

Since OFM is reused and stay in on-chip, it will be flushed to off-chip memory when IFM and weights are loaded for $\lceil N/T_n \rceil$ times. Lat_1 indicates the latency of computation, loading IFM, loading weights to be fired once, and Lat_2 indicates the latency of OFM to be flushed to off-chip memory. Finally, for one layer, OFM is flushed to off-chip memory for $B \times \lceil R/T_r \rceil \times \lceil C/T_c \rceil \times \lceil M/T_m \rceil$ times, and we have the total latency Lat.

For the model of depthwise convolution, we only need to modify T_m in the above formulas to be $T_m(d)$ and Tn to be 1. Kindly note that we consider the real-time scenario where the batch size is 1, and therefore, the communication subsystem [including on-chip buffer model (3)–(6), and off-chip memory access model (7)–(9)] of two types of convolutions are shared. However, the accelerators are independent; therefore, we revise (1) as follows:

$$T_m \times T_n + T_m(d) \leq \mathbb{D}. \tag{14}$$

④ *iDetect: Performance Bottleneck Detector*

Based on the iDesign, we have several observations for the techniques introduced in ② iSpace, and we propose iDetect tool to analyze these search spaces in turn. Kindly note that all operators in a neural architecture will be implemented on one board and reuse these resources. Before discussing each search space, we first present the following corollary to detect the performance bottleneck of a layer based on iDesign.

Property 1: Given a layer and design parameters, we can detect the performance bottlenecks by considering Lat_1 and Lat_2 as follows.

1) *O:* If Lat_2 is dominated by tO_{mem}, the performance bottleneck is on transmitting OFM data, otherwise.
2) *I:* If Lat_1 is dominated by tI_{mem}, the performance bottleneck is on transmitting IFM data.
3) *W:* If Lat_1 is dominated by tW_{mem}, the performance bottleneck is on transmitting weights.

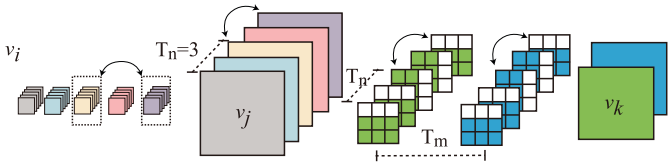


Fig. 7. Reorder the IFMs to make the pattern pruning take effects in reducing the computation latency.

4) *C:* If Lat_1 is dominated by *t*Comp, we have fully utilized the involved computation resource.

Pattern Pruning can Reduce Computation Time: Now, we are ready to answer the 1) question left in ② i): the number of kernels pruned by each type of pattern is coupled with the titling factor T_m and T_n. As we can see from Fig. 6, the data movement from on-chip weight buffer to the accelerator is conducted in a pixelwise way. As a result, it requires $K \times K$ iterations to traverse the whole filter. To enable the effect of, we need to make sure that all patterns in one data tile are the same, as such we can skip these pruned weights in the outer loop to reduce the computation time. In this way, we can modify (2) as follows:

$$t\text{Comp} = (K \cdot K - \text{PAT}_n) \cdot Tr \cdot Tc \tag{15}$$

where PAT_n is the number of 0s in the pattern mask.

Next, since the pattern selection for kernels is based on the Euclidean norm, it cannot guarantee all patterns for same type of data tiles. We propose the IFM reorder method to solve this problem. As shown in Fig. 7, we can change the third and fifth channels for the filter used in the operator $o_{j,k}$. Correspondingly, we need to switch the feature map in node v_j. It will also affect the operator from v_i to v_j, where we need to switch the third and fourth filters. In this way, we can make the pattern pruning take effects and reduce the computation latency.

From (5) and 8, it may appear that pattern pruning can also reduce the on-chip buffer size and latency of loading weights. However, for buffer size, all layers reuse this buffer, and it cannot be specialized for one layer; while for loading weights, the pattern pruning will lead the loading procedure from sequential memory access to random access, as a result the latency maybe even increased. Hence, we will keep the sequential memory access to guarantee performance.

Property 2: By applying the proposed reorder technique, pattern pruning can be employed to reduce the computing latency, but cannot reduce the latency of loading weights.

Channel Pruning can Conditionally Reduce Latency: Channel pruning directly reduces the number of channels of feature maps in a node, and it can potentially reduce the latency. Let Cut_n be the number of channels cut on the feature maps of node v_i. When v_i acts as the IFMs for an operator, we need to modify (12) as follows:

$$\text{Lat}_2 = \max\left\{\left\lceil \frac{N - \text{Cut}_n}{T_n} \right\rceil \cdot \text{Lat}_1, tO_{\text{mem}}\right\}. \tag{16}$$

Then, when v_i acts as the OFMs for an operator, we revise (13) as follows:

$$\text{Lat} = \left\lceil \frac{R}{T_r} \right\rceil \times \left\lceil \frac{C}{T_c} \right\rceil \times \left\lceil \frac{M - \text{Cut}_n}{T_m} \right\rceil \times \text{Lat}_2 + (tO_{\text{mem}} + \text{Lat}_1). \tag{17}$$

Property 3: Channel pruning can reduce the latency of a layer if and only if (1) $\lceil (M - \text{Cut}_n)/(T_m) \rceil \leq \lceil M/T_m \rceil$ or (2) $\lceil (N - \text{Cut}_n)/T_n \rceil < \lceil N/T_n \rceil$ and Lat_2 is not dominated by storing OFM data.

The above property indicates that pruning a small number of channels may not make an impact. As such, it guides the iSpace of channel pruning to take T_m or T_n as the step.

Quantization can Reduce the Latency of Loading Weights: Quantization is widely used in the neural network-based FPGA implementations. It is demonstrated hybrid quantization can achieve good performance [24], where weights in different layers have different bit widths. When we adopt such a hybrid approach, what benefits can be achieved? From (8), we can see that the quantization can take effects in reducing the latency of loading weights. This can be implemented by composing multiple weights into one package. As with computing latency, since the initial interval is already optimized to 1 cycle as shown in (2), the lower bit-width operations cannot further reduce clock cycles. Lower bit width can reduce the number of computing resources and have the potential to achieve high clock frequency. However, when we consider an end-to-end implementation, the computing engine is shared by all layers. Therefore, the layer with the largest bit width dominates the design performance.

Property 4: Quantization on a single layer can reduce the latency of loading weights, but it may not reduce the computation latency if there exists another layer with larger bit width.

⑤ *Optimizer: Search Space Exploration*

Finally, we introduce the RNN-based reinforcement learning optimizer employed in iSearch. As shown in the bottom part of Fig. 3, an RNN controller is designed based on the created design space by the ② iSpace tool. Specifically, the controller is composed of a softmax classifier to predict hyperparameters for each search space in iSpace (e.g., quantization $Quan_f$ for a layer). The predicted hyperparameters will identify a specific neural network and hardware design, which can derive a reward in terms of accuracy and latency. The search process will optimize the controller by tuning its parameters θ_c to maximize the expectation of the reward. A policy gradient method will be employed to update parameters θ_c, aiming to predict better architectures over a series of episodes.

In each episode, the predicted hyperparameters can be regarded as actions. Based on the actions, the optimized neural architecture A and hardware design D can be derived. In order to update the controller for the next episode, we need to compute the reward according to the following procedures.

1) Calculate latency lat of architecture A on design D by using the performance models proposed in ③ iDesign and ④ iDetect.

2) Verify whether timing constraint T can be satisfied; if lat $>$ T, we will directly calculate the reward without fine-tuning the model, otherwise, the reward is calculated based on accuracy and latency in the next step.
3) Fine-tune architecture A to obtain accuracy acc on a hold-out dataset; since the model is pretrained, we do not need to train the model from scratch; instead, we fine-tune the model for a small number of data batches (not epochs), say $\beta = 10$, to obtain acc.

Finally, the calculation of reward is based on the following formula:

$$R(\text{acc}, \text{lat}) = \alpha \times r\_\text{acc} + (1 - \alpha) \times r\_\text{lat} \tag{18}$$

where α is a scaling parameter to control with the search is for higher accuracy (i.e., larger α) or lower latency (i.e., smaller α). If lat $>$ T indicating that the timing constraint cannot be satisfied, we have $r\_\text{acc} = -1$ and $r\_\text{lat} = T - \text{lat}$; otherwise, we normalize $r_$acc and $r_$lat to the range from -1 to 1, as follows: $r\_\text{acc} = [(\text{acc} - A\_\text{min})/(A\_ori - A\_\text{min})] \times 2 - 1$ and $r\_\text{lat} = [(T - \text{lat})/(T - T\_\text{min})] \times 2 - 1$, where A_ori is the original accuracy of backbone architecture; T is the timing constraint; $A_$min and $T_$min are the lower bounds on accuracy and latency, which are involved for a better normalization.

Based on the reward function, the optimizer will iteratively work in two steps. First, the controller predicts a sample, and gets its reward R. Then, the Monte Carlo policy gradient algorithm [25] is employed to update the controller

$$\nabla J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \gamma^{T-t} \nabla_\theta \log \pi_\theta \left(a_t | a_{(t-1):1}\right) (R_k - b) \tag{19}$$

where m is the batch size and T is the number of steps in each episode. Rewards are discounted at every step by an exponential factor γ and baseline b is the average exponential moving of rewards.

IV. Experimental Results

The proposed HotNAS is evaluated on commonly used datasets, ImageNet [26] and CIFAR-10 with Xilinx ZCU102 board. In the following texts, we will first introduce the experimental setup. Then, we will compare HotNAS with the state-of-the-art models to show that HotNAS can achieve up to 5.79% top-1 accuracy gain with the same timing constraint. Next, we will visualize the results explored by HotNAS, followed by the design space exploration results to demonstrate the importance of co-exploring all design spaces in iSpace. Finally, we report results and detailed analysis on CIFAR-10, showing that HotNAS can achieve consistent improvement on different datasets.

A. Experimental Setup

Model Zoo: For ImageNet dataset, we apply all models in torchvision, including AlexNet, VGGNet, ResNet, MobileNet-v2, Mnasnet, etc., as shown in Fig. 1. We also include the FBNet [1] and ProxylessNAS [2] for comparison. In the experiments, we select a set of models to be optimized. According to iDesign and iDetect, we run Monte Carlo Tests to get statistic latency for 100 solutions in iSpace, as shown in Table II. We prune the models whose minimum latency cannot satisfy the timing constraints, say $T \leq 5$ in our settings. Kindly note that the maximum latency can be larger than the original model latency, because we change the hardware configuration during the search, which may reduce bandwidth for the data movement which is the performance bottleneck. Based on the results in Table II, we select ProxylessNAS (mobile), MnasNet 1.0 (depth multiplier of 1.0), and Resnet-18 (with 18 layers) [27] for optimization. We denote them as ProxylessNAS, Mnasnet, and Resnet, respectively.

TABLE II
Model Selection With 100 Monte Carlo Tests Using iSpace, With the Timing Constraint of $T \leq 5$ ms

Models	Original Latency (ms)	Latency (ms) in Monte Carlo Tests			
		Min	Sat.	Max	Ave.
ProxylessNAS	5.83	4.47	✓	5.95	5.08
MnasNet	5.94	4.68	✓	7.88	5.21
ResNet	6.27	4.31	✓	5.9	5.05
MobileNet	6.29	**5.07**	×	7.88	6.2

For CIFAR-10 dataset, we collect the four sets of pretrained models, including ResNet-18 [27], DenseNet-121 [28], MobileNet-v2 [29], and BiT [17], among which BiT achieves the state-of-the-art accuracy on CIFAR-10 dataset, which is built on top of existing neural networks. In our experiments, with the hardware performance consideration, we select the ResNet-50-based version for BiT, which provides a baseline with the accuracy of 97.07% and latency of 6.88 ms. For a better presentation, we denote the above models as ResNet, DenseNet, MobileNet, and BiTNet, respectively.

iSearch: In iSearch component, we first need to determine parameters α and β. We set $\alpha = 0.7$ to generate the reward as shown in (18), and set the number of batch size $\beta = 10$ to be used in the fine-tune phase. Furthermore, we will investigate the effects on performance made by different configurations of α and β on CIFAR-10. Second, we need to set the number of episodes for reinforcement learning; here, we set the maximum episode to be 2000 which can guarantee the convergence of the controller. After running iSearch, we can obtain a set of architectures, and we will select the best architectures, i.e., the architecture with the highest accuracy under the given timing constraints.

iSpace: A new module that supports pattern pruning, channel pruning, filter expansion, and quantization is implemented in Pytorch by overriding the existing Conv2d module. During the iSearch process, the module can be customized for each layer in terms of the searched parameters, and automatically integrated into the model with the original weights.

iDesign: We apply Xilinx ZCU102 board with XCZU9EG chip as the implementation hardware, which is composed of 600k logic cells, 32.1-Mb on-chip buffers, 2520 DSPs. For data movement between on-chip and off-chip memory, there are four HP ports with the bandwidth of 128 b for each.

TABLE III
ON IMAGENET, COMPARISON OF THE STATE-OF-THE-ART NEURAL ARCHITECTURES WITH TIMING CONSTRAINTS OF 5 MS

Model	Type	Latency	Sat.	Param. (#)	Param. (S)	Top-1	Top-5	Top-1 Imp.	Top-5 Imp.	GPU Time
AlexNet	manually	2.02	✓	61.1M	122.20MB	56.52%	79.07%	-	-	-
*MnasNet 0.5 **	*auto*	*3.99*	✓	*2.22M*	*4.44MB*	*67.60%*	*87.50%*	-	-	*40,000H*
SqueezeNet 1.0	manually	4.76	✓	1.25M	2.50MB	58.09%	80.42%	-	-	-
ProxylessNAS	auto	5.83	×	4.08M	8.16MB	74.59%	92.20%	-	-	200H
MnasNet	auto	5.94	×	4.38M	8.77MB	73.46%	91.51%	-	-	40,000H
Resnet	manually	6.27	×	11.69M	23.38MB	69.76%	89.08%	-	-	-
Co-Exploration [6]	auto	-	-	-	-	70.42%	90.53%	-	-	266H
HotNAS-Resnet(4ms)	**auto**	**4.00**	✓	**10.99M**	**17.49MB**	**68.27%**	**88.21%**	**0.67%**	**0.71%**	**2H22M**
HotNAS-Resnet	**auto**	**4.22**	✓	**11.19M**	**17.90MB**	**69.14%**	**88.83%**	**1.54%**	**1.33%**	**2H01M**
HotNAS-ProxylessNAS	**auto**	**4.86**	✓	**4.38M**	**8.31MB**	**73.39%**	**91.47%**	**5.79%**	**3.97%**	**2H37M**
HotNAS-Mnasnet	**auto**	**4.99**	✓	**4.07M**	**6.56MB**	**73.24%**	**91.37%**	**5.64%**	**3.87%**	**1H50M**

"*": baseline; "*auto & manually*": the model identified by NAS or human experts; "× & ✓": violate or meet timing constraints.

Results of Co-Exploration is derived from [6], since the pre-trained model is not provided; The latency is not reported since it uses different hardware.

B. Results on ImageNet

1) Comparison with HotNAS: Table III reports the comparison results of HotNAS with the existing state-of-the-art models. In the table, the column "Type" shows whether the model is identified by NAS or manually designed. The column "Sat." shows whether the model satisfies the timing constraint of 5 ms. Columns "Param. (#)" and "Param. (S)" reports the number of parameters and the size of parameters, respectively. Columns "Top-1," "Top5," "Top-1 Imp.," and "Top-5 Imp." are model accuracy and accuracy gain to the baseline model on ImageNet. Column "GPU Hours" shows the cost to identify the model for all models identified by NAS. Finally, the rows marked as bold are models identified by the proposed HotNAS.

From the results in Table III, we have three important observations as follows.

1) Directly plugging the existing models onto the target FPGA board will easily result in the latency to be violated; while the proposed HotNAS can guarantee to find the architectures to meet the latency constraints, meanwhile achieving high accuracy.
2) For the existing models that can directly satisfy the timing constraints, the highest top-1 accuracy and top-5 accuracy are merely 67.60% and 87.50%. In comparison, HotNAS can achieve 5.79% and 3.97% accuracy gain with 73.39% for top-1 and 91.47% for top-5.
3) The cost of the existing NAS is extremely high, which is at least 200 GPU hours. In comparison, the proposed HotNASonly takes less than 3 GPU hours to identify the model.

Furthermore, compared with the existing co-exploration method, the search time can be significantly reduced from 266 GPU hours with 2.97% Top-1 accuracy gain. All these observations clearly demonstrate the superiority of HotNAS to obtain solutions with high accuracy and low search cost.

Besides, from the results, we can see that HotNASperforms good at reducing the latency while maintaining high accuracy.

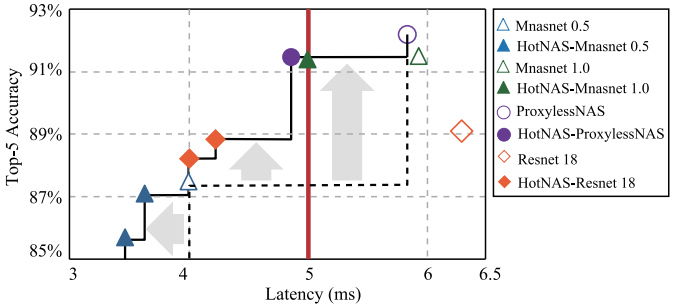
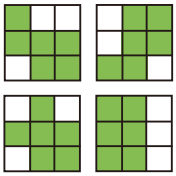


Fig. 8. Pushing forward the Pareto frontier between latency and top-5 accuracy from the one built by the existing models to that by HotNASunder the timing constraint of 5 ms.

For Resnet18, HotNAScan reduce the latency from 6.27ms to 4.22ms with 32.70% reduction, while the top-5 accuracy loss is merely 0.25%; for ProxylessNAS, the latency reduction is 16.64% with only 0.53% top-5 accuracy loss; for Resnet18, these figures are 15.9% and 0.14%. We will have a detailed and visualized analysis in the latency reduction later in this section. A further observation from the above result is that the manually designed Resnet18 can achieve larger reductions in latency than the automatically identified ones. This is reasonable since the automatically designed architectures have already be used for optimizing for other platforms, while manually designed architectures may have more redundant parameters. This can also be observed by the reduction in both the number of parameters and the size of parameters.

Fig. 8 further shows the comparison of Pareto frontiers built by the existing models and HotNAS. In this figure, the x-axis and y-axis represent the latency and accuracy, respectively. The red line stands for the timing constraints. The solid points are solutions identified by HotNAS, while the hollow ones are the existing models. The arrows in this figure clearly demonstrate that HotNAS can significantly push forward the Pareto frontier between accuracy and latency in two directions: 1) vertical

TABLE IV
SOLUTION VISUALIZATION, USING THE MODIFIED LAYERS AND LATENCY REDUCTION IN HOTNAS-RESNET18 AS AN EXAMPLE

Layers/HW	iDetect	iSpace	Exploration Results	Red. (ms)
layer1[0].conv1 layer1[0].conv2 layer1[1].conv1 layer1[1].conv2 layer2[0].conv2 layer2[1].conv1 layer2[1].conv2	C	Pattern	PATr=3, PATn=4 	0.57
layer4[0].conv1 layer4[1].conv1	I	Channel	$512 \rightarrow 480$ $512 \rightarrow 496$	0.15
layer4[0].conv1 layer4[1].conv1 layer4[0].conv2 layer4[1].conv2	- - W	Quant.	$[1, 15] \rightarrow [1, 7]$	1.01
I_b W_b	-	HW	$18 \rightarrow 20$ $6 \rightarrow 5$	0.32
		Total		**2.05**

direction: improving accuracy and 2) horizon direction: reducing latency. The results in this figure again demonstrate the efficiency and effectiveness of the proposed HotNAS.

2) Results Visualization: Table IV shows the visualization results of HotNAS-Resnet18. For other resultant architectures, they have similar results, but the model is too large to demonstrate. In this table, column "iDetect" shows the performance bottleneck with the original design detected by HotNAS, and column "iSpace" shows the built search spaces for these the corresponding layers. The column "exploration results" show the detailed changes from the original architecture to the resultant model. Finally, the column "Red." shows the latency reduction contributed by each search space.

It is clearly shown in this table that the proposed HotNAS can identify different types of performance bottleneck in the architecture, and apply the matched techniques to alleviate the performance bottlenecks. Specifically, pattern pruning identifies four patterns in pattern category $\text{PAT}_r = 3$, and achieves 0.57-ms latency reduction. Channel pruning, quantization, and hardware modifications achieve a reduction of 0.15, 1.01, and 0.32 ms, respectively. As a whole, the reduction is 2.05 ms, from 6.27 to 4.22 ms, as shown in Table III. Kindly note that since the latency of loading IFM and loading weights are quite close for layer 4, iSpace creates search spaces for both channel pruning and quantization.

3) No Space in iSpace can be Dispensed: There are a lot of existing techniques that focus on devising a specific technique for model compression. We compare with the two most effective methods using pattern pruning only [15], denoted by PatternOnly; and hybrid quantization [15], denoted by QuantOnly. However, as discussed in this iDetect, no technique can cover all kinds of performance bottlenecks. Results in Fig. 9 verify this claim. Kindly note that the hardware space

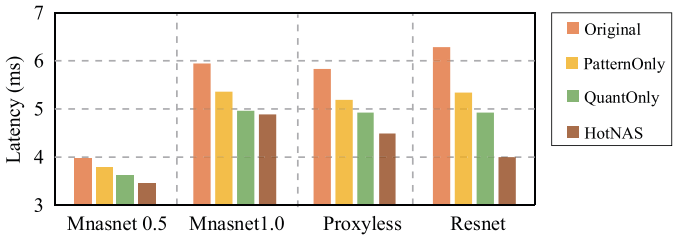


Fig. 9. Comparison results in the latency reduction among three techniques using the original architecture as baseline: 1) PatternOnly: apply pattern pruning [15]; 2) QuantOnly: apply hybrid quantization [24]; and 3) HotNAS.

is kept for all techniques for a fair comparison. In this figure, the *x*-axis is the backbone architecture, and the *y*-axis is the latency that can be achieved with the same accuracy constraint. The baseline is the original neural architecture without optimization.

Results in Fig. 9 clearly demonstrate that without fully considering the performance bottleneck and apply only one technique for optimization will lead to inferior solutions. Taking Resnet as an example, PatternOnly can reduce the time from 6.27 to 5.34 ms, and QuantOnly can further reduce it to 4.92 ms. By a full consideration of all kinds of bottlenecks, HotNAS can achieve the architecture with 4 ms, which achieves 25.09% and 18.71% latency reductions compared with PatternOnly and QuantOnly, respectively. Results from this group of experiments emphasizes the needs of an automatic tool to analyze the model, detect the performance bottlenecks, and alleviate each kind of bottlenecks using the correct technique.

C. Results on CIFAR-10

1) Pushing Forward Accuracy-Latency Pareto Frontier: HotNAS can consistently push forward the accuracy-latency Pareto frontier for different datasets. On CIFAR-10 dataset, we achieve similar results as ImageNet dataset. Table V reports a detailed comparison of the best architectures identified by HotNAS over the baseline model. The best architecture is selected based on the architectures with the highest accuracy while satisfying the timing constraint. Then, we fine-tune the selected architecture for 10 epochs to obtain the final accuracy. The accuracy and latency for the original model and the one identified by HotNAS are reported in Columns "baseline" and "HotNAS" under Columns "Accuracy" and "Latency."

From Table V, it is clear to see that HotNAS can efficiently reduce the latency which achieving accuracy gain on the CIFAR-10 dataset. Specifically, for ResNet, HotNAS identifies the solution with 43.90% latency reduction and 0.03% accuracy gain; these figures are 28.55% and 0.05% for DenseNet; 16.74% and 0.10% for MobileNet; and 48.26% and 0.06% for BitNet. The above results demonstrate the efficiency and effectiveness of HotNAS.

2) Exploration With Different Configurations: There are two hyperparameters in the RNN-based optimizer: 1) β for the batch size of fine-tuning in the search process and 2) α for the weights in the reward formulation. In the following, we will test different settings on both.

TABLE V
ON CIFAR-10, COMPARISON OF THE BASELINE MODELS AND THE BEST SOLUTIONS EXPLORED BY HOTNAS AFTER FINE-TUNING

Model	Accuracy			Latency (ms)		
	baseline	**HotNAS**	**comp.**	baseline	**HotNAS**	**impr.**
ResNet	93.33%	**93.36%**	**+0.03%**	3.44	**1.93**	**43.90%**
DenseNet	94.14%	**94.19%**	**+0.05%**	4.01	**2.87**	**28.55%**
MobileNet	94.17%	**94.27%**	**+0.10%**	2.14	**1.79**	**16.74%**
BiTNet	97.07%	**97.13%**	**+0.06%**	6.88	**3.56**	**48.26%**

TABLE VI
ON CIFAR-10, COMPARISON OF DIFFERENT SETTING IN HOTNAS ON FINE-TINE BATCH SIZE β DURING THE SEARCH PROCESS; $\beta = 195$ FOR 1-EPOCH SEARCH AND $\beta = 10$ FOR FAST-SEARCH

Model	1-epoch-search			fast-search		
	Accuracy	Latency	GPU Time	Accuracy	Latency	GPU Time
ResNet	93.36%	1.93	7M21S	92.74%	1.84	**3M26S**
DenseNet	94.08%	2.79	55M26S	**94.19%**	2.87	**12M04S**
MobileNet	94.27%	1.79	20M15S	94.21%	1.79	**4M26S**
BiTNet	97.13%	3.56	2H20M	97.04%	3.84	**18M44S**

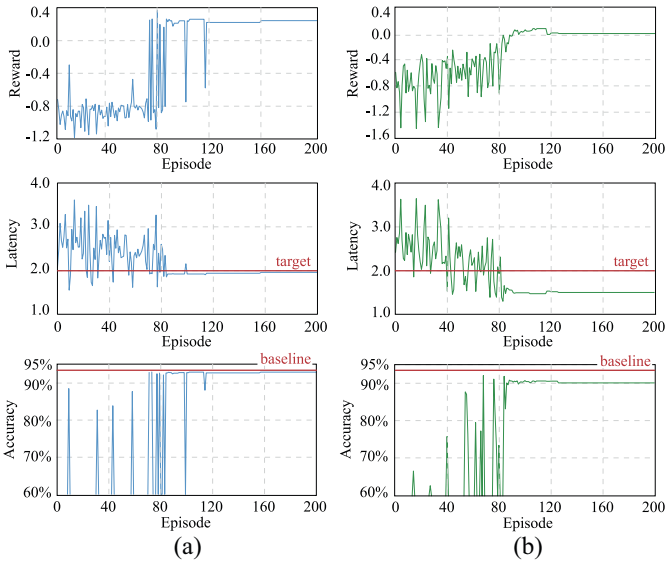


Fig. 10. On CIFAR-10, comparison of different settings on scaling parameters α in optimizing ResNet.

First, we apply two settings on β: 1) $\beta = 195$ for "1-epoch search" which will fine-tune the identified architecture using the whole training set and 2) $\beta = 10$ for "fast-search" which only use a portion of dataset as in ImageNet experiments; Table VI reports the results. We can see that fast-search can achieve competitive accuracy compare with 1-epoch search; in particular, for DenseNet, fast-search achieves 0.11% higher accuracy. In addition, for all models, fast-search can find solutions in 20 min. These results demonstrate the efficiency of HotNAS.

Second, from Fig. 10, we can see that the search processes are converged after 160 and 120 episodes for the high-accuracy setting and the low-latency setting, respectively. At the convergence, the latency of solutions identified by low-latency setting is lower than the high-accuracy setting; more interesting, the high-accuracy setting finds solutions with latency near to the threshold 2 ms. For accuracy, we can see that high-accuracy setting finds solutions with higher accuracy, which is almost the same with the baseline accuracy. As shown in the results in Table V, after a fine-tuned process, the accuracy can even higher than the baseline. One more thing noted by the accuracy results is that there are several episodes having no accuracy. This is because the latency cannot be satisfied, and we terminate the training procedure to accelerate the search process.

V. CONCLUSION

In this article, we identify the last mile problem in current NAS and hardware accelerator design and propose the HotNAS toolset to solve the problem. Instead of search architectures from scratch, we propose to stand on the shoulders of the existing models to conduct an incremental hardware-aware NAS. In HotNAS, four components work collaboratively to 1) identify the hardware performance bottleneck by iDetect; 2) build search spaces iSpace in terms of results from iDetect; and 3) co-design the neural architecture and hardware accelerator by iSearch with the performance model provided by iDesign. Experimental results on ImageNet dataset demonstrate that HotNAS can guarantee the resultant system to meet timing specifications, while achieving over 5.6% top-1 and over 3.8% top-5 accuracy gain, compared with the state-of-the-art models.

REFERENCES

[1] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10734–10742.
[2] H. Cai, L. Zhu, and S. Han. (2018). *ProxylessNAS: Direct Neural Architecture Search on Target Task And Hardware*. [Online]. Available: https://arxiv.org/abs/1812.00332
[3] M. Tan *et al.*, "MNASNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.
[4] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. IEEE 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
[5] 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.
[6] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Apr. 8, 2020, doi: 10.1109/TCAD.2020.2986127.
[7] W. Jiang *et al.*, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Trans. Comput.*, early access, Apr. 30, 2020, doi: 10.1109/TC.2020.2991575.
[8] L. Yang *et al.* (2020). *Co-Exploration of Neural Architectures and Heterogeneous Asic Accelerator Designs Targeting Multiple Tasks*. [Online]. Available: https://arxiv.org/abs/2002.04116
[9] L. Yang, W. Jiang, W. Liu, H. Edwin, Y. Shi, and J. Hu, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *Proc. IEEE 25th Asia–South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 85–90.
[10] E. Strubell, A. Ganesh, and A. McCallum. (2019). *Energy and Policy Considerations for Deep Learning in NLP*. [Online]. Available: https://arxiv.org/abs/1906.02243

[11] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu. (2019). *On Neural Architecture Search for Resource-Constrained Hardware Platforms*. [Online]. Available: https://arxiv.org/abs/1911.00105
[12] S. Han, H. Mao, and W. J. Dally. (2015). *Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding*. [Online]. Available: https://arxiv.org/abs/1510.00149
[13] H. Mao *et al.* (2017). *Exploring the Regularity of Sparse Structure in Convolutional Neural Networks*. [Online]. Available: https://arxiv.org/abs/1705.08922
[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[15] X. Ma *et al.*, "PCONV: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," in *Proc. AAAI*, 2020, pp. 5117–5124.
[16] H. Liu, K. Simonyan, and Y. Yang. (2018). *Darts: Differentiable Architecture Search*. [Online]. Available: https://arxiv.org/abs/1806.09055
[17] A. Kolesnikov *et al.* (2019). *Big Transfer (Bit): General Visual Representation Learning*. [Online]. Available: https://arxiv.org/abs/1912.11370
[18] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[19] X. Dong and Y. Yang. (2020). *NAS-Bench-102: Extending the Scope of Reproducible Neural Architecture Search*. [Online]. Available: https://arxiv.org/abs/2001.00326
[20] 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. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2015, pp. 161–170.
[21] 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*, 2016, pp. 326–331.
[22] E. Real *et al.* (2017). *Large-Scale Evolution of Image Classifiers*. [Online]. Available: https://arxiv.org/abs/1703.01041
[23] W. Jiang *et al.*, "Achieving super-linear speedup across multi-FPGA for real-time dnn inference," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5S, pp. 1–23, 2019.
[24] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 8612–8620.
[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*, 2009, pp. 248–255.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

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, nonvolatile memories, and HW/SW co-optimization.

Dr. Jiang was a recipient of the Best Paper Awards in ICCD'17 and NVMSA'15, and the 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, a Research Scholar with the University of Pittsburgh, Pittsburgh, PA, USA, from February 2019 to August 2019, and a Postdoctoral Research Associate with the University of Notre Dame, Notre Dame, IN, USA, from October 2019 to August 2020. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, USA. Her research interests are in automated machine learning, embedded systems, and high-performance computing architectures.

Sakyasingha Dasgupta received the master's degree in artificial intelligence from the University of Edinburgh, Edinburgh, U.K., in 2010, the Ph.D. degree in 2014, the degree from the MIT Sloan School of Management, Cambridge, MA, USA, in 2017, and the Dr.rer.nat doctoral degree from the Max Planck Institute for Dynamics and Self Organization, University of Göttingen, Göttingen, Germany.

He was a Senior Research Scientist and lead at IBM Research. He is the CEO and Founder of Edgecortix Inc. a deep tech startup-based in Tokyo and Singapore, automating machine learning driven AI hardware and software co-design for an intelligent distributed edge ecosystem. He has held a Senior Research and Development positions at organizations like RIKEN Center for Brain Science; Microsoft 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 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.

Dr. 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 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.