

## MIT Open Access Articles

### *NAAS: Neural Accelerator Architecture Search*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

**Citation:** Lin, Yujun, Yang, Mengtian and Han, Song. 2021. "NAAS: Neural Accelerator Architecture Search." 2021 58th ACM/IEEE Design Automation Conference (DAC).

**As Published:** 10.1109/DAC18074.2021.9586250

**Publisher:** Institute of Electrical and Electronics Engineers (IEEE)

**Persistent URL:** <https://hdl.handle.net/1721.1/143672>

**Version:** Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

**Terms of use:** Creative Commons Attribution-Noncommercial-Share Alike



**Massachusetts Institute of Technology**

# NAAS: Neural Accelerator Architecture Search

Yujun Lin<sup>\*</sup>  
MIT

Mengtian Yang<sup>\*</sup>  
SJTU  
<https://tinyml.mit.edu>

Song Han  
MIT

**Abstract**—Data-driven, automatic design space exploration of neural accelerator architecture is desirable for specialization and productivity. Previous frameworks focus on sizing the numerical architectural hyper-parameters while neglect searching the PE connectivities and compiler mappings. To tackle this challenge, we propose Neural Accelerator Architecture Search (NAAS) that holistically searches the neural network architecture, accelerator architecture and compiler mapping in one optimization loop. NAAS composes highly matched architectures together with efficient mapping. As a data-driven approach, NAAS rivals the human design Eyeriss by  $4.4 \times$  EDP reduction with 2.7% accuracy improvement on ImageNet under the same computation resource, and offers  $1.4 \times$  to  $3.5 \times$  EDP reduction than only sizing the architectural hyper-parameters.

## I. INTRODUCTION

Neural architecture and accelerator architecture co-design is important to enable specialization and acceleration. It covers three aspects: designing the neural network, designing the accelerator, and the compiler that maps the model on the accelerator. The design space of each dimension is listed in Table I, with more than  $10^{800}$  choices for a 50-layer neural network. Given the huge design space, data-driven approach is desirable, where new architecture design evolves as new designs and rewards are collected. Recent work on hardware-aware neural architecture search (NAS) and auto compiler optimization have successfully leverage machine learning algorithms to automatically explore the design space. However, these works only focuses on off-the-shelf hardware [1]–[6], and neglect the freedom in the hardware design space.

The interactions between the neural architecture and the accelerator architecture is illustrated in Table II. The correlations are complicated and vary from hardware to hardware: for instance, tiled input channels should be multiples of #rows of compute array in NVDLA, while #rows is related to the kernel size in Eyeriss. It is important to consider all the correlations and make them fit. A tuple of perfectly matched *neural architecture*, *accelerator architecture*, and *mapping strategy* will improve the utilization of the compute array and on-chip memory, maximizing efficiency and performance.

The potential of exploring both neural and accelerator architecture has been proven on FPGA platforms [7]–[10] where HLS is applied to generate FPGA accelerator. Earlier work on accelerator architecture search [11]–[13] only search

TABLE I: Neural-Accelerator architecture search space.

Accelerator	Compute Array Size (#rows/#columns) (Input/Weight/Output) Buffer Size PE Inter-connection (Dataflow)
Compiler Mapping	Loop Order, Loop Tiling Sizes
Neural Network	#Layers, #Channels, Kernel Size Block Structure, Input Size

TABLE II: The complicated correlation between neural and accelerator design space. It differs from accelerator to accelerator: N is NVDLA and E is Eyeriss.

Accelerator	Neural Architecture Design Space			
	Input Channels	Output Channels	Kernel Size	Feature Map Size
Parameter Space				
Array #rows	N		E	
Array #cols		N		E
IBUF Size	N/E			N/E
WBUF Size	N/E	N/E	N/E	
OBUF Size		N/E		N/E

the architectural sizing while neglecting the PE connectivity (*e.g.*, array shape and parallel dimensions) and compiler mappings, which impact the hardware efficiency.

We push beyond searching only hardware hyper-parameters and propose the Neural Accelerator Architecture Search (NAAS), which fully exploits the hardware design space and compiler mapping strategies at the same time. Unlike prior work [11] which formulate the hardware parameter search as a pure sizing optimization, NAAS models the co-search as a two-level optimization problem, where each level is a combination of indexing, ordering and sizing optimization. To tackle such challenges, we propose an encoding method which is able to encode the *non-numerical* parameters such as loop order and parallel dimension chosen as *numerical* parameters for optimization. As shown in Figure 1, the outer loop of NAAS optimizes the accelerator architecture while the inner loop optimizes the compiler mapping strategies.

Combining both spaces greatly enlarges the optimization space: within the same #PEs and on-chip memory resources as EdgeTPU there are at least  $10^{11}$  hardware candidates and  $10^{17}$  mapping candidates for each layer, which composes  $10^{11+50\cdot17} = 10^{861}$  possible combinations in the joint search space for ResNet-50, while there are only  $10^4$  hardware candidates in NASAIC’s design space. To efficiently search

<sup>\*</sup>Equally contributed to this work.

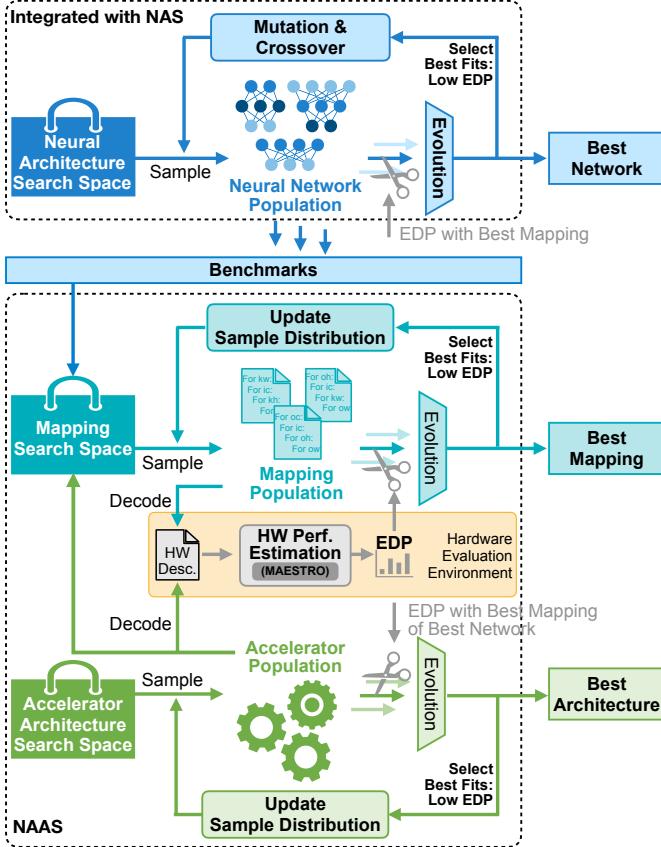


Fig. 1: Neural Accelerator Architecture Search.

over the large design space, NAAS leverages the biologically-inspired evolution-based algorithm rather than meta-controller-based algorithm to improve the sample efficiency. It keeps improving the quality of the candidate population by ruling out the inferior and generating from the fittest. Thanks to the low search cost, NAAS can be easily integrated with hardware-aware NAS algorithm by adding another optimization level (Figure 1), achieving the joint search.

Extensive experiments verify the effectiveness of our framework. Under the same #PE and on-chip memory constraints, the NAAS is able to deliver  $2.6\times$ ,  $4.4\times$  speedup and  $2.1\times$ ,  $1.4\times$  energy savings on average compared to Eyeriss [14], NVDLA [15] design respectively. Integrated with Once-For-All NAS algorithm [4], NAAS further improves the top-1 accuracy on ImageNet by 2.7% without hurting the hardware performance. Using the similar compute resources, NAAS achieves  $3.0\times$ ,  $1.9\times$  EDP improvements compared to Neural-Hardware Architecture Search [12], and NASAIC [11] respectively.

## II. NEURAL ACCELERATOR ARCHITECTURE SEARCH

Figure 1 shows the optimization flow of Neural Accelerator Architecture Search (NAAS). NAAS explores the design space of accelerators, and compiler’s mappings simultaneously.

### A. Accelerator Architecture Search

*a) Design Space:* The accelerator design knobs can be categorized into two classes:

- 1) Architectural Sizing: the number of processing elements (#PEs), private scratch pad size (L1 size), global buffer size (L2 size), and memory bandwidth.
- 2) Connectivity Parameters: the number of array dimensions (1D, 2D or 3D array), array size at each dimension, and the inter-PE connections.

Most state-of-art searching frameworks only contains architectural sizing parameters in their design space. These sizing parameters are numerical and can be easily embedded into vectors during search. On the other hand, PE connectivity is difficult to encode as vectors since they are not numerical numbers. Moreover, changing the connectivity requires redesigning the compiler mapping strategies, which extremely increase the searching cost. In NAAS, besides the architectural sizing parameters which are common in other frameworks, we introduce the connectivity parameters into our search space, making it possible to search among 1D, 2D and 3D array as well, and thus our design space includes almost the entire accelerator design space for neural network accelerators.

*b) Encoding:* We first model the PE connectivity as the choices of parallel dimensions. For example, parallelism in input channels (C) means a reduction connection of the partial sum register inside each PE. Parallelism in output channels means a broadcast to input feature register inside each PE. The most straight-forward method to encode the parallel dimension choice is to enumerate all possible parallelism situations and choose the index of the enumeration as the encoding value. However, since the increment or decrement of indexes does not convey any physical information, it is hard to be optimized.

To solve this problem, we proposed the “importance-based” encoding method for choosing parallelism dimensions in the dataflow and convert the indexing optimization into the sizing optimization. For each dimension, our optimizer will generate an importance value. To get the corresponding parallel dimensions, we first collect all the importance value, then sort them in decreasing order, and select the first  $k$  dimensions as the parallel dimensions of a  $k$ -D compute array. As shown in the left of Figure 3, the generated candidate is a 2D array with size  $16 \times 16$ . To find the parallel dimension for this 2D array candidate, The importance values are first generated for 6 dimensions in the same way as other numerical parameters in the encoding vector. We then sort the value in decreasing order and determine the new order of the dimensions. Since the importance value of “C” and “K” are the largest two value, we finally select “C” and “K” as the parallel dimensions of this 2D array. The importance value of the dimension represents the priority of the parallelism: a larger value indicates a higher priority and a higher possibility to be paralleled in the computation loop nest, which contains higher relativity with accelerator design compared to indexes of enumerations.

For other numerical parameters, we use the straight-forward encoding method. The whole hardware encoding vector is

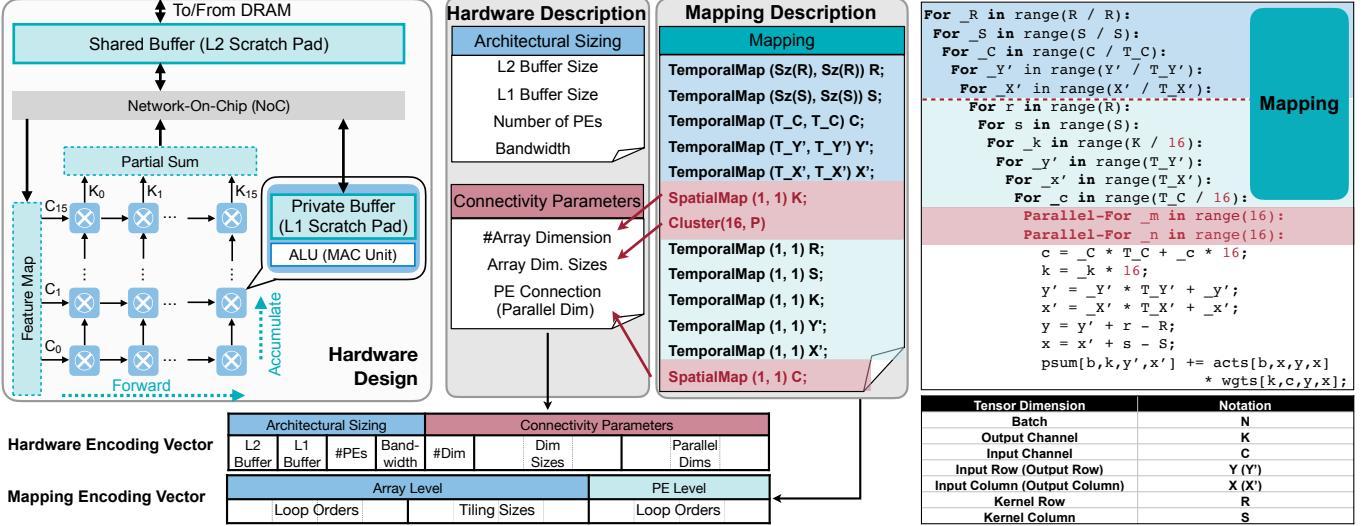


Fig. 2: Encoding the accelerator design and compiler mapping into vectors. An accelerator design is described by architectural sizing and connectivity parameters, where PE inter-connection is presented by parallel dimension. A mapping strategy is represented by loop order and corresponding tiling at each dimension of array (the innermost level is PE level). The mapping description in the figure is in MAESTRO [16] format which fuses the array parameters and mapping strategy.

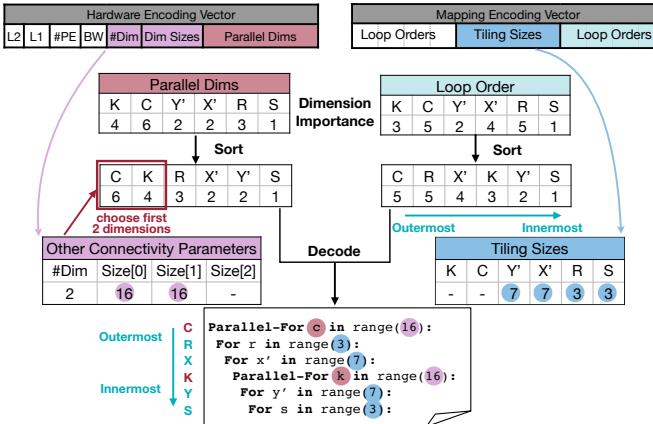


Fig. 3: We convert the non-numerical ordering optimization into numerical optimization for parallel dimensions and loop order searching. Both accelerator optimizer and mapping optimizer will generate an importance value for each dimension. The left part shows the parallel dimensions of this 2D array candidate are the dimensions with the largest two importance value. The right part shows the order of each dimension in for-loop is in the decreasing order of its importance value.

shown in Figure 2, which contains all of the necessary parameters to represent an accelerator design paradigm.

c) ***Evolution Search:*** We leverage the evolution strategy [17] to find the best solution during the exploration. In order to take both latency and energy into consideration, we choose the widely used metric Energy-Delay Product (EDP) to evaluate a given accelerator configuration on a specific neural network workload. At each evolution iteration, we first sample a set of candidates according to a multivariate normal

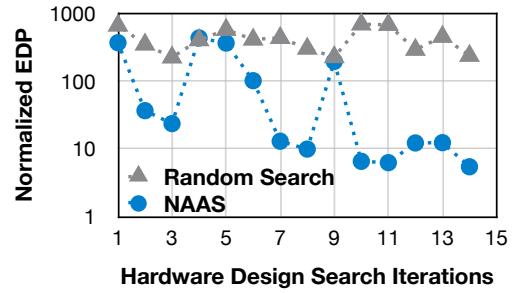


Fig. 4: The average of EDP decreases as NAAS is learning.

distribution in  $[0, 1]^{\theta}$ . Each candidate is represented as a  $|\theta|$ -dimension vector. These candidates are then projected to hardware encoding vectors and decoded into accelerator design. We rule out the invalid accelerator samples and keep sampling until the candidate set reaches a predefined size (population size) in our experiments. To evaluate the candidate performance, we need to perform mapping strategy search in Section II-B on each benchmark and adopt the best searched result as the EDP reward of this candidate. After evaluating all the candidates on the benchmarks, we update the sampling distribution based on the relative ordering of their EDP. Specifically, we select the top solutions as the “parents” of the next generation and use their center to generate the new mean of the sampling distribution. We update the covariance matrix of the distribution to increase the likelihood of generating samples near the parents [17]. We then repeat such sampling and updating process.

Figure 4 shows the statistics of energy-delay products of hardware candidates in the population. As the optimization continues, the EDP mean of NAAS candidates decreases while that of random search remains high, which indicates that NAAS

gradually improves the range of hardware selections.

### B. Compiler Mapping Strategy Search

The performance and energy efficiency of deep learning accelerators also depend on how to *map* the neural network task on the accelerator. The search space of compiler mapping strategy is much larger than accelerator design, since different convolution layers may not share the same optimal mapping strategy. Hence we optimize the mapping for each layer independently using the similar evolution-based search algorithm to accelerator design search in Section II-A0c.

The compiler mapping strategy consists of two components: the execution order and the tiling size of each for-loop dimension. Similar to the accelerator design search, the order of for-loop dimensions is non-trivial. Rather than enumerating all of the possible execution orders and using indexes as encoding, we use the similar “importance-based” encoding methods in Section II-A0b. For each dimension of the array (corresponding to each level of for-loop nests), the mapping optimizer will assign each convolution dimension with an importance value. The dimension with the highest importance will become the outermost loop while the one with the lowest importance will be placed at the innermost in the loop nests. The right of Figure 3 gives an example. The optimizer firstly generates the importance values for 6 dimensions, then sort the value in decreasing order and determine the corresponding order of the dimensions. Since “C” and “R” dimension have largest value 5, they will become the outermost loops. “S” dimension has the smallest value 1, so it is the innermost dimension in the loops. This strategy is interpretable, since the importance value represents the data locality of the dimension: the dimension labeled as most important has the best data locality since it is the outermost loop, while the dimension labeled as least important has the poorest data locality therefore it is the innermost loop.

As for tiling sizes, since they are highly related to the network parameters, we use the scaling ratio rather than the absolute tiling value. Hence, the tiling sizes are still numerical parameters and able to adapt to different networks. The right part of figure 3 illustrates the composition of the mapping encoding vector. Note that for PE level we need to ensure that there is only one MAC in a PE, so we only search the loop order at PE level. For each array level, the encoding vector contains both the execution order and tiling size for each for-loop dimension.

### C. Integrated with Neural Architecture Search

Thanks to the low search cost, we can integrate our framework with neural architecture search to achieve neural-accelerator-compiler co-design. Figure 1 illustrates integrating NAAS with NAS. The joint design space is huge, and in order to improve the search efficiency, we choose and adapt Once-For-All NAS algorithm for NAAS. First, NAAS generates a pool of accelerator candidates. For each accelerator candidate, we sample a network candidate from NAS framework which satisfies the pre-defined accuracy requirement. Since each subnet of Once-For-All network is well trained, the accuracy

evaluation is fast. We then apply the compiler mapping strategy search for the network candidate on the corresponding accelerator candidate. NAS optimizer will update using the searched EDP as a reward. Until NAS optimizer reaches its iteration limitations, and feedback the EDP of the best network candidate to accelerator design optimizer. We repeated the process until the best-fitted design is found. In the end, we obtain a tuple of matched accelerator, neural network, and its mapping strategy with guaranteed accuracy and lowest EDP.

## III. EVALUATION

We evaluate Neural Accelerator Architecture Search’s performance improvement step by step: 1) the improvement from applying NAAS given the same hardware resource; 2) performance of NAAS which integrates the Once-For-All to achieve the neural-accelerator-compiler co-design.

### A. Evaluation Environment

*a) Design Space of NAAS:* We select four different resource constraints based on EdgeTPU, NVDLA [15], Eyeriss [14] and Shidiannao [18]. When comparing to each baseline architecture, NAAS is conducted within corresponding computation resource constraint including the maximum #PEs, the maximum total on-chip memory size, and the NoC bandwidth. NAAS searches #PEs at stride of 8, buffer sizes at stride of 16B, array sizes at stride of 2.

*b) CNN Benchmarks:* We select 6 widely-used CNN models as our benchmarks. The benchmarks are divided into two sets: classic large-scale networks (VGG16, ResNet50, UNet) and light-weight efficient mobile networks (MobileNetV2, SqueezeNet, MNasNet). Five deployment scenarios are divided accordingly: we conduct NAAS for large models with more hardware resources (EdgeTPU, NVDLA with 1024 PEs), and for small models with limited hardware resources (ShiDianNao, Eyeriss, and NVDLA with 256 PEs).

*c) Design Space in Once-For-All NAS:* When integrating with Once-For-All NAS, the neural architecture space is modified from ResNet-50 design space following the open-sourced library [4]. There are 3 width multiplier choices (0.65, 0.8, 1.0) and 18 residual blocks at maximum, where each block consists of three convolutions and has 3 reduction ratios (0.2, 0.25, 0.35). Input image size ranges from 128 to 256 at stride of 16. In total there are  $10^{13}$  possible neural architectures.

### B. Improvement from NAAS

Figure 5 shows the speedup and energy savings of NAAS using the same hardware resources compared to the baseline architectures. When running large-scale models, NAAS delivers  $2.6\times$ ,  $2.2\times$  speedup and  $1.1\times$ ,  $1.1\times$  energy savings on average compared to EdgeTPU and NVDLA-1024. Though NAAS tries to provide a balanced performance on all benchmarks by using geomean EDP as reward, VGG16 workload sees the highest gains from NAAS. When inferencing light-weight models, NAAS achieves  $4.4\times$ ,  $1.7\times$ ,  $4.4\times$  speedup and  $2.1\times$ ,  $1.4\times$ ,  $4.9\times$  energy savings on average compared to Eyeriss, NVDLA-256, and ShiDianNao. Similar to searching for large

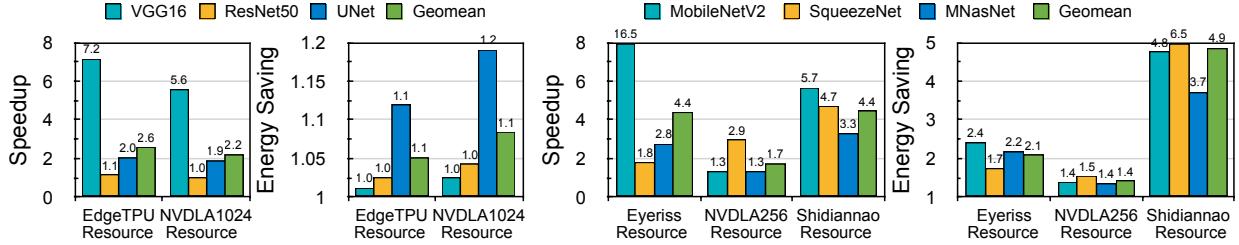


Fig. 5: Speedup and energy savings when NAAS searches the accelerator and mapping for a set of network benchmarks within the baseline hardware resources. For instance, using the same #PEs and SRAM size as EdgeTPU, the accelerator designed by NAAS runs VGG16, ResNet50 and UNet 2.6× faster than EdgeTPU on average.

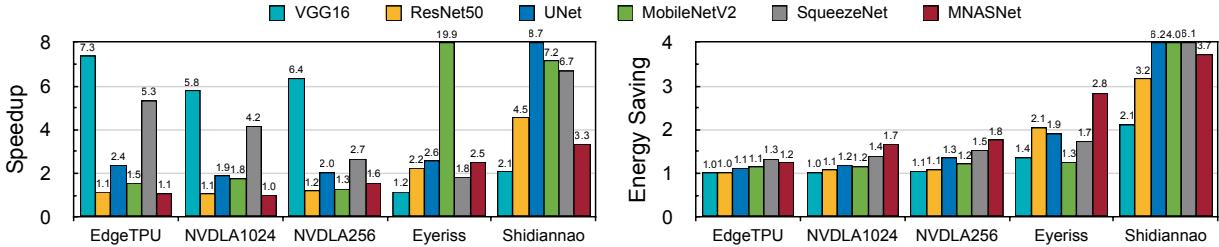


Fig. 6: Speedup and energy savings when NAAS searches the accelerator and mapping for single network benchmark within the baseline hardware resources.

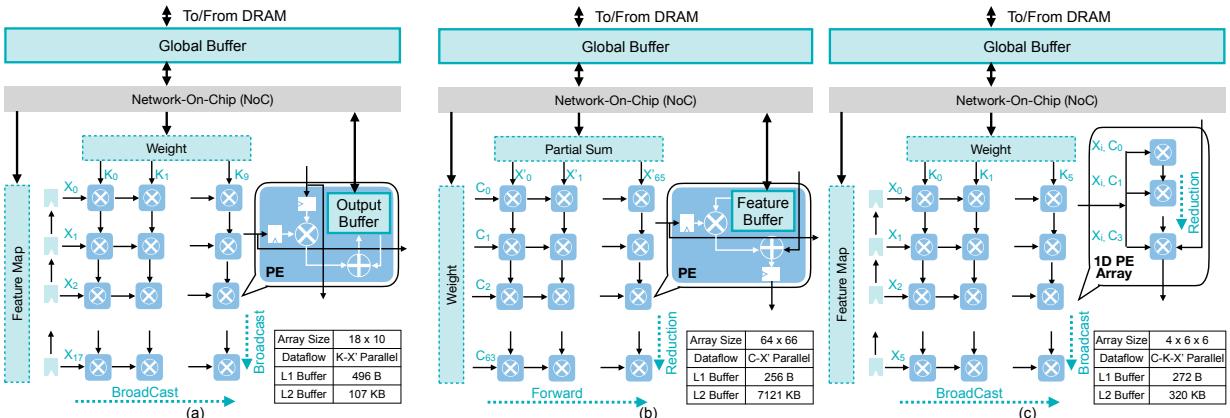


Fig. 7: NAAS offers different architectures for different networks when given different computation resources: (a) 2D array with K-X' parallelism for ResNet using Eyeriss resource; (b) 2D array with C-X' parallelism for VGG16 using EdgeTPU resource; (c) 3D array with C-K-X' parallelism for VGG16 using Shidiannao resource.

models, different models obtain different benefits from NAAS under different resource constraints.

Figure 7 demonstrates three examples of searched architectures. When given different computation resources, for different NN models, NAAS provides different solutions beyond numerical design parameter tuning. Different dataflow parallelisms determine the different PE micro-architecture and thus PE connectivities and even feature/weight/partial-sum buffer placement.

Figure 8 illustrates the benefits of searching connectivity parameters and mapping strategy compared to searching architectural sizing only. NAAS outperforms architectural

sizing search by 3.52×, 1.42×EDP reduction on VGG and MobileNetV2 within EdgeTPU resources, as well as 2.61×, 1.62×improvement under NVDLA-1024 resources.

Figure 9 further shows the EDP reduction using different encoding methods for non-numerical parameters in hardware and mapping encoding vectors. Our proposed importance-based encoding method significantly improves the performance of optimization by reducing EDP from 1.4×to 7.4×.

### C. More Improvement from Integrating NAS

Different from accelerator architectures, neural architectures have much more knobs to tune (*e.g.*, network depths, channel

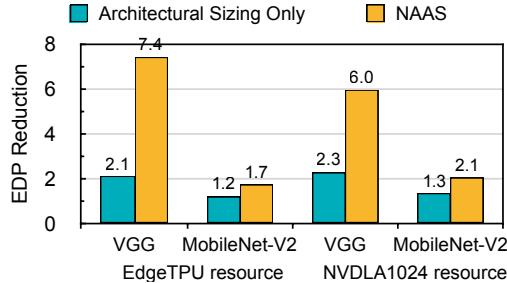


Fig. 8: Compared to searching the architectural sizing only [11], [12], searching the connectivity parameters and mapping strategies as well achieves considerable EDP reduction.

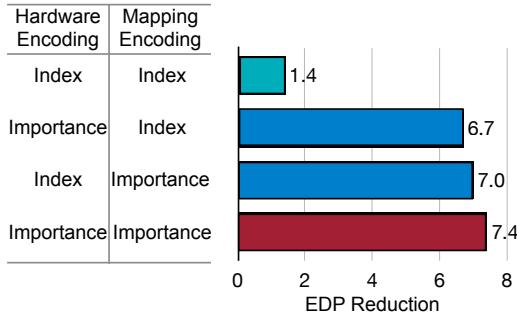


Fig. 9: Compared to index-based encoding, importance-based encoding achieves better EDP reduction.

numbers, input image sizes), providing us with more room to optimize. To illustrate the benefit of NAAS with NAS, we evaluate on ResNet50 with hardware resources similar to Eyeriss. Figure 10 shows that NAAS (accelerator only) outperforms Neural-Hardware Architecture Search (NHAS) [12] (which only searches the neural architecture and the accelerator architectural sizing) by  $3.01\times$  EDP improvement. By integrating with neural architecture search, NAAS achieves  $4.88\times$  EDP improvement in total as well as 2.7% top-1 accuracy improvement on ImageNet dataset than Eyeriss running ResNet50.

*a) Comparison to NASAIC:* We also compare our NAAS with previous work NASAIC [11] in Table III. NASAIC adopts DLA [15], ShiDianNao [18] as source architectures for its heterogeneous accelerator, and only searches the allocation of #PEs and NoC bandwidth. In contrast, we explore more possible improvements by adapting both accelerator design searching and mapping strategy searching. Inferencing the same network searched by NASAIC, NAAS outperforms NASAIC by  $1.88\times$  EDP improvement ( $3.75\times$  latency improvement with double energy cost) using the same design constraints.

*b) Search Cost:* Table IV reports the search cost of our NAAS compared to NASAIC and NHAS when developing accelerator and network for  $N$  development scenarios, where “AWS Cost” is calculated based on the price of on-demand P3.16xlarge instances, and “CO<sub>2</sub> Emission” is calculated based on Strubell *et al.* [19]. NASAIC relies on a meta-controller-based search algorithm which requires training every neural

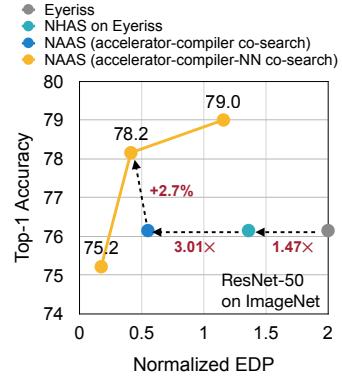


Fig. 10: Accuracy vs. Normalized EDP on ImageNet (batch = 1) under Eyeriss hardware resource. Integrating NAAS with OFA NAS further improves model accuracy.

TABLE III: NAAS (accelerator only) achieves better EDP.

Search Approach	Arch	Cifar-10 Accuracy	Latency (cycles)	Energy (nJ)	EDP (cycles·nJ)
NASAIC [11]	DLA [15] Shi [18]	93.2 91.1	3e5	1e9	3e14
NAAS	DLA [15]	93.2	8e4	2e9	2e14

architecture candidates from scratch. NHAS [12] decouples training and searching in NAS but it also requires retraining the searched network for each deployment which costs  $16N$  GPU days. The high sample efficiency of NAAS makes it possible to integrate the Once-For-All NAS in one search loop. As a conservative estimation, NAAS saves more than  $120\times$  search cost compared to NASAIC on ImageNet.

#### IV. RELATED WORKS

*a) Accelerator Design-Space Exploration:* Earlier work focuses on fine-grained hardware resource assignment for deployment on FPGAs [7], [10], [13], [20]–[22]. Several work focuses on co-designing neural architectures and ASIC accelerators. Yang *et al.* [11] (NASAIC) devise a controller that simultaneously predicts neural architectures as well as the selection policy of various IPs in a heterogeneous accelerator. Lin *et al.* [12] focuses on optimizing the micro architecture parameters such as the array size and buffer size for given accelerator design while searching the quantized neural architecture. Besides, some work focuses on optimizing compiler mapping strategy on fixed architectures. Chen *et al.* [23] (TVM) designed a searching framework for optimizing for-loop execution on CPU/GPU or other fixed platforms. Mu *et al.* [24] proposed a new searching algorithm for tuning mapping strategy on fixed GPU architecture. On the contrary, our work explores not only the sizing parameters but also the connectivity parameters and the compiler mapping strategy. We also explore the neural architecture space to further improve the performance. Plenty of work provides the modeling platform for design space exploration [25]–[28]. We choose MAESTRO [28] as the accelerator evaluation backend.

TABLE IV: We achieve much lower search cost on ImageNet (Gds: GPU days. N: number of development scenarios.)

Approach	Co-Search Cost (Gds)	NN Training Cost (Gds)	Total Cost (Gds)	AWS Cost	CO <sub>2</sub> Emission
NASAIC	$500 \times 12N = 6000N$	$16N$	$6000N$	\$441,000N	41,000N lbs
NHAS	$12 + 4N$	$16N$	$12 + 20N$	\$1,500N	150N lbs
Ours	$< 0.25N$	50	$< 50 + 0.25N$	$< \$18N$	$< 2N$ lbs

\* NASAIC's search cost is an optimistic projection from Cifar10.

\* AWS cost \$75/Gd and CO<sub>2</sub> Emission is 7.5 lbs/Gd.

*b) AutoML and Hardware-Aware NAS:* Researchers have looked to automate the neural network design using AutoML. Grid search [28], [29] and reinforcement learning with meta-controller [2], [11], [30] both suffer from prohibitive search cost. One-shot-network-based frameworks [1], [3], [31] achieved high performance at a relatively low search cost. These NAS algorithms require retraining the searched networks while Cai *et al.* [4] proposed Once-For-All network of which the subnets are well trained and can be directly extracted for deployment. Recent neural architecture search (NAS) frameworks started to incorporate the hardware into the search feedback loop [1]–[4], [8], [9], though they have not explored hardware accelerator optimization.

## V. CONCLUSION

We propose an evolution-based accelerator-compiler co-search framework, NAAS. It not only searches the architecture sizing parameters but also the PE connectivity and compiler mapping strategy. Integrated with the Once-for-All NAS algorithm, it explores the search spaces of neural architectures, accelerator architectures, and mapping strategies together while reducing the search cost by 120× compared with previous work. Extensive experiments verify the effectiveness of NAAS. Within the same computation resources as Eyeriss [14], NAAS provides 4.4× energy-delay-product reduction with 2.7% top-1 accuracy improvement on ImageNet dataset compared to directly running ResNet-50 on Eyeriss. Using the similar computation resources, NAAS integrated with NAS achieves 3.0×, 1.9× EDP improvements compared to NHAS [12], and NASAIC [11] respectively.

**Acknowledgements.** This work was supported by NSF CAREER Award #1943349 and SRC GRC program under task 2944.001. We also thank AWS Machine Learning Research Awards for the computational resource.

## REFERENCES

- [1] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *ICLR*, 2019.
- [2] Q. Lu *et al.*, “On neural architecture search for resource-constrained hardware platforms,” 2019.
- [3] K. Wang *et al.*, “HAQ: Hardware-aware automated quantization with mixed precision,” in *CVPR*, 2019, pp. 8612–8620.
- [4] H. Cai *et al.*, “Once for all: Train one network and specialize it for efficient deployment,” in *ICLR*, 2020.
- [5] T. Chen *et al.*, “Learning to optimize tensor programs,” in *NeurIPS*, 2018, pp. 3389–3400.
- [6] S.-C. Kao and T. Krishna, “Gamma: Mapping space exploration via genetic algorithm,” in *ICCAD’20*, 2020.
- [7] C. Hao *et al.*, “FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge,” in *DAC*, 2019, pp. 1–6.
- [8] Y. Li *et al.*, “EDD: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions,” *DAC*, 2020.
- [9] X. Zhang *et al.*, “Skynet: a hardware-efficient method for object detection and tracking on embedded systems,” 2020.
- [10] S.-C. Kao *et al.*, “Confucius: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning,” in *MICRO*, 2020.
- [11] L. Yang *et al.*, “Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks,” 2020.
- [12] Y. Lin *et al.*, “Neural hardware architecture search,” *NeurIPS WS*, 2019.
- [13] W. Jiang *et al.*, “Hardware/software co-exploration of neural architectures,” *TCAD*, 2020.
- [14] Y.-H. Chen *et al.*, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IJSSC*, vol. 52, no. 1, 2016.
- [15] NVIDIA, “NVIDIA deep learning accelerator,” 2017. [Online]. Available: <http://nvidia.org>
- [16] J. Albericio *et al.*, “Cnvlutin: Ineffectual-neuron-free deep neural network computing,” *ACM SIGARCH*, vol. 44, no. 3, pp. 1–13, 2016.
- [17] N. Hansen, “The CMA evolution strategy: a comparing review,” in *Towards a new evolutionary computation*. Springer, 2006, pp. 75–102.
- [18] Z. Du *et al.*, “Shidiannao: Shifting vision processing closer to the sensor,” in *ISCA*, 2015, pp. 92–104.
- [19] E. Strubell *et al.*, “Energy and policy considerations for deep learning in nlp,” in *ACL*, 2019, pp. 3645–3650.
- [20] M. Motamedi *et al.*, “Design space exploration of fpga-based deep convolutional neural networks,” in *ASP-DAC*. IEEE, 2016, pp. 575–580.
- [21] G. Zhong *et al.*, “Design space exploration of fpga-based accelerators with multi-level parallelism,” in *DATE*. IEEE, 2017, pp. 1141–1146.
- [22] Y. Chen *et al.*, “Cloud-dnn: An open framework for mapping dnn models to cloud fpgas,” in *FPGA*, 2019, pp. 73–82.
- [23] T. Chen *et al.*, “{TVM}: An automated end-to-end optimizing compiler for deep learning,” in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.
- [24] J. Mu *et al.*, “A history-based auto-tuning framework for fast and high-performance dnn design on gpu,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [25] Y. S. Shao *et al.*, “The aladdin approach to accelerator design and modeling,” *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.
- [26] Y. N. Wu and V. Sze, “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” in *TCAD*, 2019.
- [27] A. Parashar *et al.*, “Timeloop: A systematic approach to dnn accelerator evaluation,” in *ISPASS*. IEEE, 2019, pp. 304–315.
- [28] H. Kwon *et al.*, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *ISCA*, 2019, pp. 754–768.
- [29] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *ICML*, 2019, pp. 6105–6114.
- [30] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *ICLR*, 2017.
- [31] Z. Guo *et al.*, “Single path one-shot neural architecture search with uniform sampling,” *arXiv preprint*, 2019.