SCORCH: Neural Architecture Search and Hardware Accelerator Co-design with Reinforcement Learning

Siqin Liu and Avinash Karanth
School of Electrical Engineering and Computer Science
Ohio University Athens, OH, 45701
Email:ls847719@ohio.edu and karanth@ohio.edu

Abstract—**The ability to automatically generate a neural network architecture and the corresponding hardware implementation to optimize both accuracy and performance characteristics (latency, power) simultaneously for edge-based Artificial Intelligence (AI) applications is becoming prevalent. As both neural architecture search (NAS) and hardware implementation have ample design space, it is very challenging to integrate with resource-constrained edge computing hardware since the current co-search frameworks take several hundreds of GPU hours to converge. In this paper, we propose SCORCH, a novel neural architecture search and hardware accelerator co-design framework with reinforcement learning to maximize accuracy, and increase energy efficiency and throughput while converging faster. By predicting hyperparameters of neural networks together with hardware resources, we use a reinforcement-based multi-phased controller to explore neural architecture to achieve higher accuracy and hardware performance simultaneously by applying customized dataflows, voltage/frequency scaling, and tunable Network-on-Chip (NoC) hardware parameters. Our simulation results on the CIFAR-10/100 and ImageNet datasets show that SCORCH achieves identical neural network accuracy while achieving 2.6% higher accuracy, and 35.6%, 26.2%, and 65.8% reductions in latency, energy, and area compared with state-of-art co-search frameworks such as DANCE, NANDS, and NASAIC.**

Index Terms—**hardware and software codesign, neural architecture search, reinforcement learning**

I. Introduction

Artificial Intelligence (AI) algorithms have achieved remarkable improvements in prediction accuracy for a wide range of deep neural network (DNN) applications [12], [18], [24], [26], [31], [44]. Ever since AlexNet [18] debuted as the first successful deep CNN architecture, neural networks have undergone significant optimizations, such as deeper layers [29], inception modules [32], residual connections [12], super nets [36], and more. Such hand-crafted modifications, including structural reformulation, regularization, and parameter optimization is labor-intensive and requires domain knowledge. In response, a recent approach called neural architecture search (NAS) [3], [21], [30], [43], [44] has shown great success in automatically developing DNNs that outperform human-crafted designs. Apart from finding improved neural network architectures, there is tremendous interest to optimize their implementation on hardware accelerators. The goal of studying hardware implementation is to improve performance characteristics, such as latency, throughput, and energy efficiency. When deploying architectures explored by NAS to real-world platforms, the Hardware-aware Neural Architecture Search (HNAS) has therefore been drawing a lot of attention to explore neural architectures by considering hardware efficiency on a fixed-target hardware platform [2], [9], [14], [36].

Despite the achievements of HNAS and hardware/software co-design approaches, there are several optimization opportunities that have not been fully explored. Hardware design search for general purpose computing (CPUs and GPUs) will involve optimizing DNN implementations such as kernel fusion and memory access optimizations whereas for spatial accelerators, it will involve implementing techniques such as quantization, loop tiling and parallelization [40]. Hardware configuration search not only provides more accurate performance evaluation of latency and throughput, but more importantly, provides instant guidance to hardware-aware DNN design during NAS. In HNAS design, the search space (hyperparameters) needs to be reduced for quick convergence while searching through several fine-grain hardware parameters (loop tiling, dataflow, link bandwidth, quantization, and others). Such a co-exploration of NAS search with hardware implementation that converges quickly is desirable especially in resource constrained edge devices that typically implement different DNN models.

In this paper, we propose **SCORCH**, a novel neural architecture search and hardware accelerator co-design framework using reinforcement learning algorithm to explore the optimal co-design configuration while converging faster as illustrated in Figure 1. SCORCH aims to identify the best neural network architecture and efficient hardware accelerator design candidate among the large design space, such that the network accuracy is maximized while hardware performance satisfies the constraints of throughput and power budget. Specifically, we design a reinforcement learning-based controller that simultaneously predicts hyperparameters of DNN architectures as well as hardware implementation. We then integrate the reinforcement-based controller with a multi-phase manager, which guides the exploration by simultaneously changing the hyperparameters to improve network accuracy and hardware configuration to improve throughput and energy efficiency. We explore hardware design space (HDS) by applying efficient

Network-on-Chip (NoC) based accelerator methodologies such as loop tiling, dataflow, link bandwidth, DVFS, and quantization. A reward function that incorporates accuracy and accelerator performance characteristics updates the controller until convergence. The main contributions of this work are:

- We propose SCORCH, a framework that co-explores NAS and HDS simultaneously. We devise a multi-phase manager to guide exploration to gradually converge to optimal solutions with the best accuracy-efficiency trade-off.
- We propose a fast convergence search algorithm to better explore HDS space without sacrificing accuracy in neural network architectures. The faster convergence generalizes in saving training costs among a wide variety of image datasets while meeting the rigid timing and power constraints in edge computing.
- Our simulation results on CIFAR-10, CIFAR-100, and ImageNet datasets show that SCORCH achieves comparable network accuracy while achieving 32.4%, 25.7%, and 63.2% reductions on latency, energy and area compared with state-of-art co-search frameworks such as DANCE [6], NANDS [38] and NASAIC [39]. With the same NAS and HDS, SCORCH converges fastest among other frameworks and generates the optimal configuration within two GPU hours given a single hardware specification constraint.

II. BACKGROUND AND CHALLENGES

As the rapid deployment of various DNN models on edge devices has gained traction, NAS is becoming more prevalent. Since the very first work for NAS with reinforcement learning [44], there has been a tremendous interest to study efficient neural architecture search [13], [30]. Integrating hardware awareness into the search loop provides one more dimension to the research and has attracted efforts on hardware-aware NAS [36], [2]. Taking one step further, most recently, co-exploration of neural architecture and hardware design has been proposed [14]. Unlike the original NAS with a mono-objective on maximizing accuracy, those hardware-aware NAS frameworks take inference latency into consideration, and push forward the deployment of DNNs onto edge devices. Below we further elaborate on some of the challenges:

It is challenging to effectively co-explore neural architectures and ASIC accelerator designs. The large design space of ASIC accelerators (dataflows, PE arrays, NoC, etc) makes it challenging to map NAS designs to ASIC accelerators. Unlike GPUs or FPGAs with well-structured hardware, ASIC designs have the potential to provide maximum flexibility to designers to organize the hardware. While ASIC accelerators provide maximum flexibility and thereby enable efficient designs, it also results in significantly enlarging the search space. Fortunately, there exists prior research in designing ASIC-based AI accelerators [27], [1], [23], making it possible to shrink the design space on top of existing designs.

The architectures automatically explored by NAS have more irregular and complicated structures than the human-invented ones. Since the neural networks designed with NAS contain various types of kernels, uniform hardware accelerator designs may not be efficient solutions. Moreover, all ASIC accelerators such as Shidiannao [10], NVDLA, and Eyeriss [5], have a specific dataflow, which determines how input feature maps and weight kernels move across the PE array from the global buffer. For instance, NVDLA involves an adder tree to calculate the partial sum of output feature maps. Dataflows have a direct impact on the overall hardware efficiency and therefore should be considered as the first search parameter.

Developing AI algorithms and deploying accelerators on hardware are two crucial but complex tasks. These challenges are heightened when targeting resource-constrained edge devices, such as mobile and portable devices, due to their strict resource, power constraints, and the need for high performance. Prior approaches, including hardware-aware Neural Architecture Search (NAS), compact model design, model compression, and model-aware accelerator design, have sought to reconcile the demands of hardware limitations with the goals of high model accuracy and fast inference speeds. However, these methods face significant challenges. Firstly, it's difficult to find a balance between algorithmic metrics (such as accuracy, complexity, and robustness) and hardware performance metrics (including throughput, latency, resource usage, and power consumption). This necessitates a back-and-forth between algorithm and hardware designers to meet their respective goals, leading to a time-consuming, error-prone process without assured outcomes. Secondly, the lack of comprehensive co-design between tasks can result in suboptimal AI algorithms that are not hardware-friendly and accelerators that do not effectively support the models.

To overcome these challenges, algorithms and their hardware accelerators should be designed together, unlocking significant optimization possibilities. Co-designing ensures the development of hardware-oriented or specific AI algorithms, enabling the selection of hardware devices that best match the algorithms' needs in terms of computational capability, memory, and adaptability. This method not only aligns algorithms with the specific features of hardware, enhancing computational efficiency but also ensures that designs meet the resource and performance constraints of the hardware. Additionally, co-design can significantly reduce design cycles. Unlike traditional iterative methods that require extensive effort to achieve satisfactory results, an automated co-design process can simultaneously optimize an AI algorithm and its deployment on hardware, ensuring efficient and effective solutions.

Deploying applications on edge devices introduces another layer of complexity due to the need for multiple tasks to operate in tandem, often involving several Deep Neural Networks (DNNs) running at the same time. These DNNs must be executed on the accelerator under a single set of design criteria, including latency, energy use, and space efficiency. Therefore, optimizing each DNN one after the other using hardware-aware Neural Architecture Search (NAS) does not yield the best overall performance. A more efficient method involves

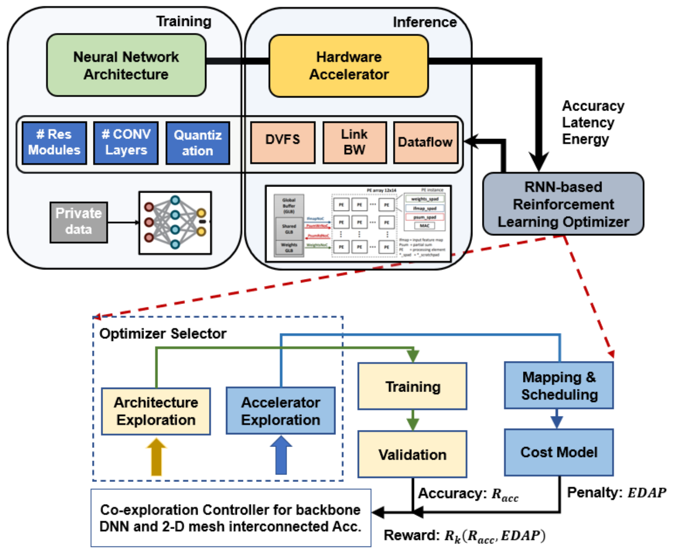

Fig. 1. Our proposed SCORCH co-design methodology with three kernel components, i.e., neural architecture training, hardware accelerator inference, and RNN-based reinforcement learning optimizer. Parameters for neural architecture and accelerator are generated by the co-exploration controller; then the identified architecture and accelerator will be evaluated; eventually, a reward combining neural network accuracy with hardware performance will be computed to update the controller via feedback.

optimizing multiple neural architectures simultaneously, ensuring they all meet these shared design specifications.

In this work, we will address the above challenges.

III. PRIOR WORK

From the first work that proposed NAS with reinforcement learning [43], the idea has been to mitigate human interference to design more efficient neural architectures [3], [21], [30], [44]. Integrating hardware awareness into the search loop provides one more dimension to co-optimize the hardware implementation and has attracted significant efforts on hardware-aware NAS architectures [2], [9], [13], [14], [19], [33], [34], [36], [39]. Unlike the original NAS with a single objective function on maximizing accuracy, these NAS and HDS co-exploration frameworks take hardware performance into consideration, and push forward the deployment of DNNs onto edge devices.

One challenge of the co-exploration is the extensive training of the massive network architecture candidates and corresponding evaluation. It is laborious to search for each synaptic weight of the network over case-by-case redesigns since the search spaces are of extremely high dimension [3]. Existing NAS methods either formulate the search space as a super-network ("supernet") and train only one-shot to approximate the performance of every architecture in the search space via weight-sharing [21], or treat the selection as a sequential decision-making process or utilize evolutionary algorithms [30]. However, these NAS algorithms suffer from increased time to converge and consume excessive hardware resources.

On the other hand, ASIC accelerator designs have the maximum flexibility to determine the hardware to most efficiently implement the target workloads. Hardware accelerators are generally spatial in nature, where a set of PEs are integrated to provide high throughput and parallelism. Different ways of arranging the PEs can constitute numerous topologies and dataflows, thereby increasing the design space. Existing research works endeavor to avoid the large design space by starting from a "hot" state based on a set of existing accelerator templates [13], [19], [39], or restrain the search space within sub-components of accelerators, such as the NoC design [38], arithmetic precision [34], power management [33], dataflows [6], and many more [25], [41]. No prior work can efficiently address all these critical design factors of ASIC accelerators in the co-exploration framework.

Compared to prior neural network hardware/software co-design frameworks, SCORCH uses the novelty of RNN-based Reinforcement Learning Optimizer to search for the coarse-grained hyperparameters of neural network architecture, such as the number of Inception modules, the type of the modules, and others which highly reduces the search space for direct weights and leads to a fast convergence without sacrificing the accuracy. We conduct an extensive and fine-grained exploration of hardware implementation, such as loop tiling, dataflow, link bandwidth, DVFS, and quantization, while prior works mostly focus on one or a few of them. Instead of using existing sub-accelerator templates, such as NVADIA style [42], Shi-diannao [10], Eyeriss [5], we explore the hardware design search from scratch, which can achieve the optimal design in terms of throughput and energy consumption with slightly increase in overall convergence time. Overall, SCORCH uniquely combines coarse-grain hyperparameter selection along with fine-grain hardware exploration to improve accuracy and performance over prior work.

IV. SCORCH FRAMEWORK AND IMPLEMENTATION

Figure 1 demonstrates the overview of SCORCH. It contains three components, including neural network training, hardware accelerator inference, and RNN-based reinforcement learning optimizer. In general, the controller samples neural architectures and hardware configurations in each episode (or iteration). The predicted sample goes through training and inference steps to generate the accuracy and hardware cost. Finally, a reward is generated to update the controller through a reinforcement learning optimizer. Note that since the hardware constraints are non-differentiable, differentiable neural architecture search [6], [20] is not applied.

Co-Exploration Controller: Driven by the requirement of multitask in one application workload, we propose a novel reinforcement learning-based Recurrent Neural Network (RNN) controller to simultaneously predict multiple neural architectures and the corresponding hardware implementations. Figure 2 demonstrates the proposed controller. It consists of six segments, which represent the hyperparameters in DNN architectures and hardware implementations. For the segment associated with a DNN, its outputs determine the neural network's hyperparameters (NAS function). The remaining

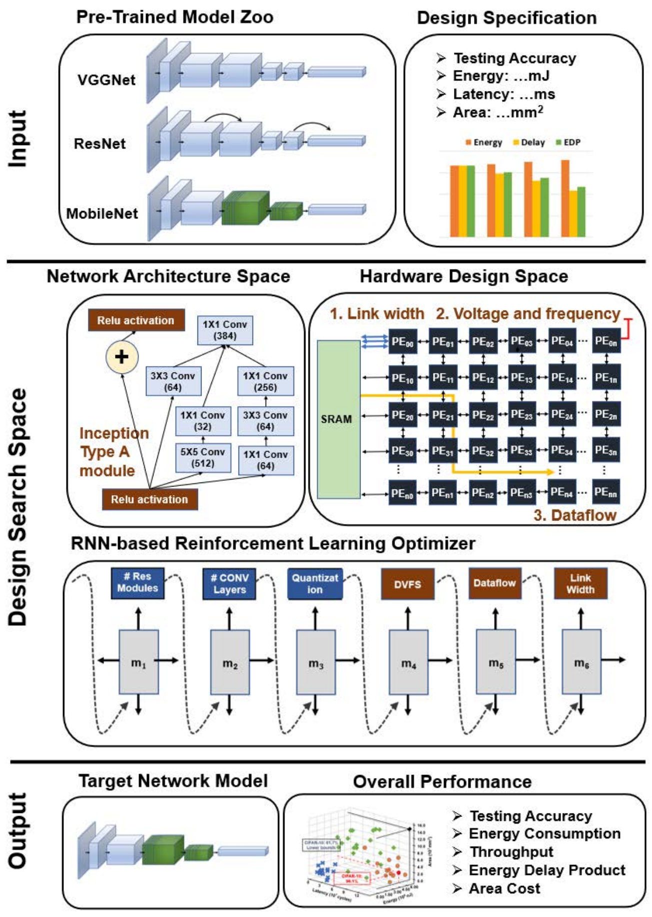

Fig. 2. Implementation from the model zoo to hardware: (top) the given pre-trained model and design specification; (middle) the proposed design search space and RL-based parameter controller for both network and hardware architecture; (bottom) outputting the best network candidate and hardware design.

segments for the hardware accelerator determine the hardware design parameters. Specifically, in each episode, the controller first predicts a design sample and gets its reward R based on the evaluation results for both algorithm (accuracy) and hardware performance (energy, latency, and area) as formulated in Eq.1. Then, we employ the Monte Carlo policy gradient algorithm [35] to update the controller in Eq.2:

$$R_k(acc, EDAP) = \alpha \times R_{acc} + (1-\alpha) \times (R_e \times R_l \div R_a) \quad (1)$$

$$\nabla J_\theta = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\gamma^{T-t}\nabla_\theta log\pi_\theta(a_t|a_{(t-1):1})(R_k - b) \quad (2)$$

where α balances the tradeoff between accuracy and hardware efficiency. 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 and the baseline b is the average exponential moving of rewards.

Network Architecture Space: We consider one neural network layer as composed of multiple Inception modules as adopted in ResNet [12]. For the convolutional operation, the parameterized exploration space includes the number of filters, filter height, filter width, stride height, and stride width, which are grouped into three module types - Inception A, B, and C. Each of the layers has 7 candidate operations in addition to a skip connection. Beyond this, the hyperparameters include the number of filters and the number of skip layers for each residual block. Quantization gives the trade-off between accuracy and computation cost, co-optimized in hardware as arithmetic precision.

TABLE I
ACCELERATOR HARDWARE SEARCH SPACE WITH 10^8 POSSIBLE VALUES. WHEN COMBINED WITH MAPPING AND LOOP TILING SEARCH SPACES, THE SCORCH TOTAL SEARCH SPACE REACHES 10^{10}.

Parameter Name	Type	Potential Values
PEs_array_size_x	int	1 to 256, powers of 2
PEs_array_size_y	int	1 to 256, powers of 2
NoC_lin_width	enum	1,2,4
DVFS_config	enum	mode 1 to 5
dataflow_config	enum	WS,OS,IS,RS
Scratchpad_config	enum	Private, Shared
Scratchpad_size	int	128B to 128KB, powers of 2
Input_buffer_size	int	1KB to 1MB, powers of 2
Weight_buffer_size	int	1KB to 1MB, powers of 2
Output_buffer_size	int	1KB to 1MB, powers of 2
DDR3_channels	int	1 to 8, powers of 2

Hardware Design Space: As shown in Figure 2. hardware design space, we target a highly-parameterized and general ML accelerator template capable of modeling a wide range of previously proposed architectural designs based on grids of processing elements (PEs). Given the enormous design space of hardware accelerators, we explore parameters such as dataflow, link bandwidth, DVFS, PE array size, buffer size, etc., as described in Table I. We represent tiling parameters to exploit spatial and temporal locality of data accesses in the DNN loop nests. Loop tiling factors determine how to store data within each memory hierarchy (e.g., DRAM, global buffer, NoC, and Scrachpad in PE) to effectively accommodate the data reuse patterns, and can be derived from all possible choices under the resource budget (e.g., memory and computation resource budgets).

We consider the design space of dataflow in compliance with the conventional dataflow taxonomy to take advantage of the temporal data reuse [31]. For instance, a weight being reused in the same PE for multiple time window instances is called weight stationary (WS), which takes advantage of temporal multicasting. Determine the data reuse patterns. Here we search from four patterns: row stationary (RS), input stationary (IS), weight stationary (WS), and output stationary (OS) for each chunk, and thus have a total of 64 (4*4*4) choices for reuse pattern of the three sub-accelerators in our accelerator.

The selection of DVFS configurations is based on the observed workload in pre-training. Each DVFS model consists of one inactive state (power-gated) and four active states.

In an inactive state, the voltage supply to the specific PE and its outgoing interconnection is reduced to 0 V with no clock applied to the PE. PE in an active state can operate in any one of the four different voltage levels. The V/F pairs used in the DVFS models are $\{$0.8V/2.75ns, 1.0V/2.25ns, 1.1V/2ns, 1.2V/1.8ns$\}$ which are numbered as V/F modes 2-5 with power-gated state as mode 1. These voltage levels are commonly configured in accelerators and selected based on the principle that when they operate in different modes, the voltage and frequency are proportionally decreased/increased.

Mapping and scheduling. For the hardware accelerator, a set of hyperparameters identifying the design choice are given by the controller. We then need to get the hardware metrics including latency r_l, energy r_e, and area r_a. SCORCH incorporates the state-of-the-art cost model Timeloop [22], Accelergy [37], and a mapping and scheduling algorithm to obtain the above metrics. For area r_a, we can directly obtain it from Accelergy with the given hardware accelerator parameters. The latency r_l and energy r_e are determined by the mapping and scheduling combined with Timeloop. We integrate the two hardware evaluation tools in our framework and parameterize the inputs using the six outputs of the RNN controller. To simplify the algorithm for mapping and scheduling, we only explore representative dataflows in a limited space (output, weight, and row stationary [4]), each dataflow with a fixed loop execution order. Any existing mapping and scheduling algorithm can be embedded into the framework.

Quantization Selector: The quantization in DNNs determines the overall performance and computational complexity of a network. Therefore, it is imperative to automate the design of quantization together with the design of the architecture. The implementation of architecture quantization joint search may vary by different settings and discretion. In this paper, we focus on the single-controller method and extend the RNN-based controller. As displayed in the framework, we insert an additional step into the controller as the quantization parameter.

Training and evaluation. For neural network architecture, the hyperparameters A_i for DNN architecture are obtained from the RNN controller. For each DNN in the candidate population, we train it from backbone networks and get its accuracy acc_i on a held-out validation dataset. Based on the accuracy, we compute the weighted accuracy as a batch process in DNN computation for multiple DNNs evaluation at a time as follows:

$$weighted(A) = \sum_{i=1,2,\ldots,N} (\beta_i \times acc_i) \tag{3}$$

where N is the total number of DNNs evaluated at a time, and β_i is a weight ranging from 0 to 1.

Hardware Performance Estimation: For the hardware accelerator, a set of hyperparameters identifying the design choice are given by the RNN-based controller. We need to get the hardware metrics including latency R_l, energy R_e, and area R_a. SCORCH incorporates the state-of-the-art cost estimators, Timeloop [22], Accelergy [37], and mapping algorithm [15] to obtain the above metrics. For area R_a, we can directly obtain it from Accelergy with the given hardware accelerator parameters. The latency R_l and energy R_e are determined by the mapping and scheduling combined with Timeloop.

Recurrent Neural Network based Controller: In SCORCH, we use a controller to generate architectural hyperparameters of neural networks and hardware accelerators. To be flexible, the controller is implemented as a recurrent neural network. Given the network accuracy and hardware constraints, we can then use the controller to generate their hyperparameters as a sequence of tokens.

In our experiments, the process of generating an architecture stops if the number of layers exceeds the threshold or the cost of the hardware exceeds the specification. Once the controller RNN finishes generating an architecture, a neural network with this architecture is built and trained. A hardware configuration is generated simultaneously and evaluated with the network inference task to obtain the hardware cost, i.e., energy, latency, and area. At convergence, the accuracy of the network on a held-out validation set is recorded. The parameters of the RNN-based controller, θ, are then optimized in order to maximize the expected accuracy of the proposed architectures and the hardware performance. In the next section, we will describe a policy gradient method that we use to update parameters θ so that the RNN-based controller generates better architectures over time.

Training with reinforce: The list of tokens that the controller predicts can be viewed as a list of actions $a_{1:T}$ to design an architecture for a child network and the hardware. At convergence, this child network will achieve an accuracy R_{acc} on a held-out dataset and the hardware latency, energy, and area cost R_l, R_e, and R_a. We combine these scores as the joint reward R_k and use reinforcement learning to train the controller. More concretely, to find the optimal design point, we ask our controller to maximize its expected reward, represented by $J(\theta_c)$ in Eq. 2. Since the reward signal R is non-differentiable, we need to use a policy gradient method to iteratively update θ_c. As formulated in Eq. 2, the conjugated validation accuracy and hardware metrics that the k-th design point achieves after being trained on a training dataset is R_k. In order to reduce the variance of this estimate we employ a baseline function b. As long as the baseline function b does not depend on the current action, then this is still an unbiased gradient estimate.

V. Simulation Evaluation

A. Simulation Setup

Three image classification datasets are employed (CIFAR-10, CIFAR-100 [17], and ImageNet [7]) to verify the efficacy of SCORCH. During the search process, we only use the training images, and randomly select 10% of them to build a validated set. Test images are used to test the accuracy of the resultant architectures. All images undergo data augmentation, including upsampling, random cropping, and random horizontal flip.

B. Simulation Results

Figure 3 demonstrates the exploration results of SCORCH on three application workloads. In this figure, the x-axis, y-axis, and z-axis represent latency, energy, and area, respectively. The black diamond indicates the design specs (upper bound); each green diamond is a solution (neural architecture-ASIC design pair) explored by SCORCH; each blue cross is a solution based on the smallest neural network in the search space combined with different ASIC designs (lower bound); and the red star refers to the best solution in terms of the average accuracy explored by SCORCH. The numbers in the rectangles with blue, green, and red colors represent the accuracy of the smallest network, the inferior solutions, and our best solutions, respectively.

Table II shows the performance of SCORCH on CIFAR-10 dataset. For the baseline, we have performed the architecture search using ProxylessNAS [2] (w/ or w/o Flops penalty term), and conducted hardware generation on the searched network using the exhaustive-search tool. It represents the typical separate design performed in practice. Using SCORCH, we have performed co-exploration with the cost functions described in Section III. Overall, SCORCH was able to obtain a neural network accelerator design superior to the baseline. For SCORCH, we report two designs, one with high accuracy (-A) and the other towards efficient hardware design (-B). For the high accuracy design (-A), SCORCH achieves almost the same accuracy as the baseline network architecture that was searched without any hardware cost penalty on the loss function. On the other hand, with the associated optimal hardware accelerators, SCORCH yields much better cost metrics than the baseline (w/ or w/o Flops penalty). For the efficient hardware design (-B), we select the design which shows the best cost function within at most 1% accuracy drop. It shows that SCORCH achieves up to 4× better EDAP, or almost 4× better latency by performing an efficient co-exploration. In addition, SCORCH supports flexible hardware exploration by redefining the hardware cost as a weighted linear combination of latency, energy consumption, and area as shown in the two major rows.

Comparisons with state-of-the-art frameworks: Table III reports the experimental results of SCORCH against three state-of-art neural architecture and hardware co-exploring frameworks, DANCE [6], NANDS [38] and NASAIC [39] over three datasets, including CIFAR-10, CIFAR-100, and ImageNet. We report solutions by NANDS with the maximum throughput ("Opt TP") and the maximum accuracy ("OptAcc."). For NAS [43] and HW-aware NAS [2], we report the finally identified architectures. SCORCH can make better accuracy-throughput tradeoffs against state-of-the-art NAS frameworks. Specifically, NAS generates solutions beyond the real-time constraint, while SCORCH can find valid solutions with marginal loss. In addition, SCORCH achieves 3.09×, and 3.14× speedup over NAS, respectively, on two different datasets. Compared with HW-aware NAS, SCORCH takes less time in searching, even though HW-aware NAS does not thoroughly explore the hardware design space. On CIFAR-10 datasets, SCORCH(-A) can achieve 42.99% higher throughput and 1.58% improvement in accuracy compared with NANDS(Opt TP).

Since the CIFAR dataset only covers limited image classes (10) with 32x32 resolution, we further expand the evaluation of SCORCH on ImageNet dataset, which contains over 10,000,000 labeled images of 1000 object categories for classification. To accelerate the search convergence of our framework on this larger dataset, we use critical sampling [16] to reduce the multi-dimensional correlation. We used 5000 samples for training and 1000 samples for evaluation. The model was trained with 50 iterations using an Adam optimizer until the target throughput was achieved. As seen in Fig. 3(c), the best design point for latency and energy models are only 0.06 ms and 0.018 W. Therefore, the proposed SCORCH can converge to the optimal design point over a larger dataset.

VI. Conclusions

In this work, we have proposed a framework, namely SCORCH, to co-explore the neural architecture and hardware accelerator designs targeting search efficiency and superior performance on edge devices. SCORCH was able to co-design the search space with efficient hardware implementation and marginal accuracy loss. The search space can be extended with the same framework to scale up for higher accuracy and more restricted hardware specification. In addition, an extensive and fine-grained exploration of hardware implementation has been developed to simultaneously determine the neural architectures under preset design specification. The efficacy of SCORCH is finally verified through a set of comparisons with state-of-the-art network and hardware co-design frameworks [8], [11], [28].

Acknowledgment

This research was partially supported by NSF grants CCF-1703013, CCF-1901192, CCF-1936794, CCF-2324645 and CCF-2311544. We sincerely thank the anonymous reviewers for their excellent feedback.

References

[1] B. Asgari, R. Hadidi, T. Krishna, H. Kim, and S. Yalamanchili, "Alrescha: A lightweight reconfigurable sparse-computation accelerator," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 249–260.

[2] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[3] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective," *arXiv preprint arXiv:2102.11535*, 2021.

[4] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

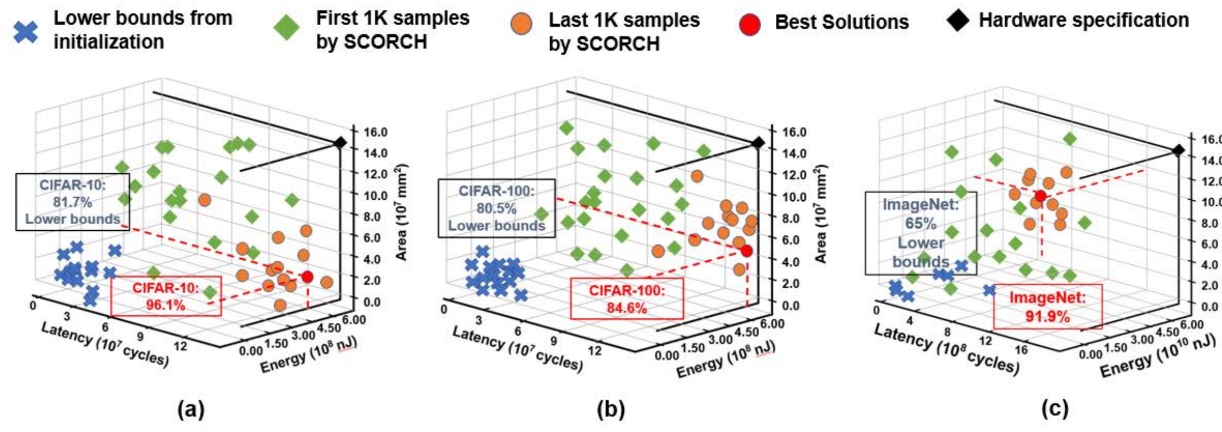

Fig. 3. Visualization of the search process by SCORCH for two different datasets: (a) CIFAR-10 (b) CIFAR-100 (c) ImageNet. Green diamonds represent the intermediate results of the search, randomly distributing among the bounds (blue cross and black diamond). The red diamond shows the final optimal solution with the highest score considering architecture accuracy and hardware performance.

TABLE II
PERFORMANCE OF SCORCH ON CIFAR-10. HARDWARE COST IS MEASURED IN TWO METHODS, I.E., EDAP AND LINEAR COMBINATION OF LATENCY, ENERGY, AND AREA AS PROPOSED IN [6]. THE HYPERPARAMETER λ IS FINE-TUNED TO ACHIEVE THE OPTIMAL EDAP SOLUTION FOR SCORCH.

HW$_{Cost}$	Hyperparam.			Method	Acc (%)	Latency (ms)	Energy (mJ)	EDAP (norm.)
CostHW$_{EDAP}$	λL	λE	λA	Baseline + HW	93.4	9.1	3.5	133.1
	NA	NA	NA	SCORCH (scratch)	91.9	2.8	6.8	93.8
				SCORCH (hot start)-A	93.2	2.6	6.5	74.4
				SCORCH (hot start)-B	92.3	1.4	3.6	19.7
CostHW$_{Linear}$	3.8	4.8	1.2	Baseline + HW	93.2	3.5	9.8	162.2
				SCORCH (scratch)	92.2	2.8	1.1	21.8
				SCORCH (hot start)-A	93.1	1	1.2	15.7
				SCORCH (hot start)-B	92.3	0.9	1.2	13.2

[6] K. Choi, D. Hong, H. Yoon, J. Yu, Y. Kim, and J. Lee, "Dance: Differentiable accelerator/network co-exploration," *arXiv preprint arXiv:2009.06237*, 2020.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[8] D. F. DiTomaso, "Reactive and proactive fault-tolerant network-on-chip architectures using machine learning," Ph.D. dissertation, Ohio University, 2015.

[9] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. So, and K. Keutzer, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 50–59.

[10] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[11] R. Guirado Liñan, "Wireless chip-scale communications for neural network accelerators," B.S. thesis, Universitat Politècnica de Catalunya, 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4154–4165, 2020.

[14] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[15] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[16] Y.-K. Kim and K.-S. Na, "Application of machine learning classification for structural brain mri in mood disorders: Critical review from a clinical perspective," *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, vol. 80, pp. 71–80, 2018.

[17] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[20] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[21] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[22] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

TABLE III
COMPARISON OF PERFORMANCE BETWEEN NAS, HW-AWARE NAS, NANDS, NASAIC, AND DANCE. THE CONFIGURATION AND PERFORMANCE OF NAS AND HW-AWARE NAS ARE BOTH INHERITED FROM [38]

Dataset	Spec.(Gbps)	Models	Arch. Property			Accuracy	Throughput	(Gbps)	HW EDP	Elapsed Time	impr.
			Depth	Para. (x106)	MACs(GOP)	(%)	degr.		(norm.)	(minute)	
CIFAR-10	0.5	NAS	9	0.89	0.7	94.72	0	0.22	133.1	1115	1x
		HW-Aware NAS	8	0.19	0.05	90.95	-3.46	0.66	65.9	164	6.8x
		NANDS (Opt TP)	8	0.2	0.06	91.58	-2.83	2.4	52.5	361	3.1x
		NANDS (Opt Acc.)	10	0.4	0.21	93.59	-0.82	0.9	55.1	361	3.1x
		NASAIC	10	0.6	0.35	93.14	-1.58	0.62	159.8	960	1.2x
		DANCE	9	0.77	1.21	94.41	-0.74	1.1	71.8	827	1.4x
		SCORCH-A	8	0.91	0.75	93.02	-1.31	1.3	34.5	126	8.9x
		SCORCH-B	9	1.13	0.82	93.33	-0.91	0.85	36.9	148	7.5x
CIFAR-100	0.45	NAS	12	1.04	1.02	76.58	0	0.45	238.5	2928	1x
		HW-Aware NAS	8	0.35	0.07	71.43	-5.15	0.28	\	246	11x
		NANDS (Opt TP)	8	0.25	0.15	72.22	-4.36	0.9	\	594	4.9x
		NANDS (Opt Acc.)	12	0.63	0.46	75.58	-1	0.45	55.1	361	9.1x
		NASAIC	9	0.94	0.83	71.1	-3.51	1.1	65.5	641	4.6x
		DANCE	9	0.81	1.75	75.1	-1.48	0.53	167.3	980	3.0x
		SCORCH-A	9	1.25	0.88	74.58	-2	0.97	38.3	155	19x
		SCORCH-B	10	1.47	0.96	75.44	-1.14	1.45	42.9	171	17x
ImageNet	0.5	NAS	9	1.56	0.97	93.51	0	0.18	198	1728	1
		NANDS (retrain)	10	0.85	0.85	90.55	-2.96	0.55	59.7	242	7.1x
		NASAIC (retrain)	11	1.13	1.33	90.13	-3.38	1.9	49.3	320	5.4x
		SCORCH-A	10	0.95	0.83	91.1	-2.41	1.95	39.5	165	10.5x
		SCORCH-B	11	1.32	1.21	91.9	-1.61	0.91	41.4	180	9.6x

[23] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, 2015, pp. 234–241.

[25] A. Samajdar, P. Mannan, K. Garg, and T. Krishna, "Genesys: Enabling continuous learning through neural network evolution in hardware," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 855–866.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[27] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[28] K. D. Shiflett, "Photonic deep neural network accelerators for scaling to the next generation of high-performance processing," Ph.D. dissertation, Ohio University, 2022.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[30] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[31] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[33] F. Tu, W. Wu, Y. Wang, H. Chen, F. Xiong, M. Shi, N. Li, J. Deng, T. Chen, L. Liu *et al.*, "Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 658–673, 2020.

[34] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[36] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[37] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[38] 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 *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 85–90.

[39] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[40] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 27–42.

[41] X. Zhang, W. Jiang, Y. Shi, and J. Hu, "When neural architecture search meets hardware implementation: from hardware awareness to co-design," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 25–30.

[42] G. Zhou, J. Zhou, and H. Lin, "Research on nvidia deep learning accelerator," in *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. IEEE, 2018, pp. 192–195.

[43] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[44] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.