

# Enabling Hard Constraints in Differentiable Neural Network and Accelerator Co-Exploration

Deokki Hong<sup>1</sup>, Kanghyun Choi<sup>1</sup>, Hye Yoon Lee<sup>1</sup>, Joonsang Yu<sup>2</sup>, Noseong Park<sup>1</sup>, Youngsok Kim<sup>1</sup>, and Jinho Lee<sup>1\*</sup>

<sup>1</sup>College of Computing, Yonsei University, <sup>2</sup>CLOVA ImageVision, CLOVA AI Lab, NAVER

<sup>1</sup>{dk.hong, kanghyun.choi, hylee817, noseong, youngsok, leejinho}@yonsei.ac.kr <sup>2</sup>joonsang.yu@navercorp.com

## ABSTRACT

Co-exploration of an optimal neural architecture and its hardware accelerator is an approach of rising interest which addresses the computational cost problem, especially in low-profile systems. The large co-exploration space is often handled by adopting the idea of differentiable neural architecture search. However, despite the superior search efficiency of the differentiable co-exploration, it faces a critical challenge of not being able to systematically satisfy hard constraints such as frame rate. To handle the hard constraint problem of differentiable co-exploration, we propose HDX, which searches for hard-constrained solutions without compromising the global design objectives. By manipulating the gradients in the interest of the given hard constraint, high-quality solutions satisfying the constraint can be obtained.

## 1 INTRODUCTION

The primary interest of most *Deep Neural Network* (DNN) researches has been the application performance (i.e., accuracy). However, it also led to the rapid growth in the network size that require immense computational resources for execution. In recent years, numerous schemes have appeared to mitigate the resource problem, mostly belonging to one of these categories – *network-side optimization* and *hardware-side optimization*. Network-side optimization refers to refining the architecture of a neural network to reduce computations while maintaining a comparable accuracy [28, 31]. Hardware-side optimization often involves improving DNN execution efficiency by using optimized hardware, also known as *accelerators* [5, 14]. Unfortunately, effort from one side often hinders the benefits coming from the other. For example, the main advantage of depth-wise separable convolution operation used in MobileNet [28] comes from its structure which uses a single channel for its operation. However, Google’s TPU [14], a renowned accelerator, mainly utilizes channel-level parallelism for gaining speedup. In consequence, MobileNet results in a poor execution time on TPUs [10].

Co-exploration of hardware accelerator and network architecture [1, 6, 9, 11, 19, 22, 30] is therefore a natural direction to fulfill both goals of accuracy and hardware metrics such as latency, energy consumption and silicon chip area. To address the large search space of the co-exploration, *differentiable co-exploration* methods [6, 9, 19] are considered as promising approaches due to their ability to quickly explore the search space compared to its reinforcement learning based counterparts [1, 11, 20, 22, 30].

Unfortunately, differentiable co-exploration has a serious drawback of being unable to deal with hard constraints that are critical in many real-world scenarios. For instance, an important constraint

of object detection system [26] is to meet the frame rate of the camera (e.g., 30 frames per second). In addition, a mobile subsystem running on a limited battery often has a power budget. Because differentiable co-exploration methods rely on a single loss function, they often fail to satisfy the constraints, and have to blindly undergo a several repetitions of hyper-parameter tuning and re-exploration.

In order to address the problem, we present *HDX* (**H**ard-constrained **D**ifferentiable **eX**ploration), which enables hard-constrained differentiable co-exploration of neural architecture and hardware accelerator. The key concept of our proposal is a gradient manipulation method which ensures that the solution does not drift away from meeting the constraints. In addition to the gradients from the global loss, we calculate the gradient from the hardware constraints, which is used to manipulate gradient of the global loss, if any constraint violations, such that i) the dot product of the two are positive (i.e., they point to a similar direction), and ii) the direction can alleviate the violation of the constraints.

To the best of our knowledge, this is the first work that considers hard constraints in a differentiable co-exploration problem. We conduct an extensive amount of evaluation to demonstrate that HDX can 1) satisfy the hardware constraints even under tight constraints and 2) search solutions without compromising the quality.

Our contributions can be summarized as follows.

- We propose HDX, a hard-constrained differentiable co-exploration method for network and accelerator in order to find valid solutions without trial-and-errors.
- We propose using gradient manipulation to gradually move solutions towards the constraint-satisfying region.
- We provide an extensive evaluation for HDX to show its constraint-meeting capability and efficiency.

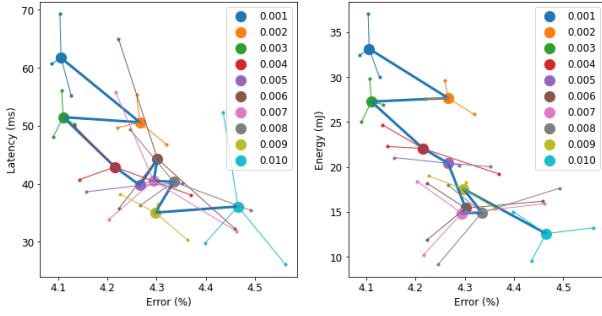
## 2 BACKGROUND AND RELATED WORK

### 2.1 Neural Architecture Search

Neural Architecture Search (NAS) refers to the technique of automating the neural network design process. To mitigate its huge cost (thousands of GPU-hours), Differentiable NAS [21] has been proposed as an efficient alternative, which converts the problem to training a supernet that reduces the time cost by a few digits.

Regardless, optimizing solely on network performance is insufficient as they do not take hardware efficiency into account. Some recent works [4] that address this issue consider hardware costs by adding loss terms, and some attempt to reduce the network size for latency constraints using simple latency models [2, 23]. However, they only consider the network design, and cannot be used for co-exploration since the relation between network accuracy and hardware structure is not reflected in the model.

\* Corresponding author.



**Figure 1: A motivational experiment. In each plot, we swept the value  $\lambda_{Cost}$  0.001 to 0.010. It is clear that the trajectory is not strictly linear to  $\lambda_{Cost}$  with high variations.**

## 2.2 DNN-Accelerator Co-exploration

The early work on the co-exploration utilize variants of reinforcement learning, or evolutionary algorithm to leverage its simplicity [1, 11, 13, 20, 22, 30]. Each candidate network is trained for evaluation, while the accelerator design is analyzed for hardware efficiency. These values create rewards used by the agent to create the next candidate solution. However, they all inherit the same problem from RL-based NAS methods in which they require expensive training to evaluate each candidate solution. To worsen the matter, co-exploration requires even larger network/hardware search space than searching only for networks.

In such regard, differentiable approaches were adopted to co-exploration [19]. Auto-NBA [9] used a differentiable accelerator search engine to build a joint-search pipeline, and DANCE [6] trained auxiliary neural networks for hardware search and cost evaluation. However, none of the above properly addresses the hard constraint problem. In this work, we propose a holistic method of handling hard constraints on differentiable co-exploration.

## 3 MOTIVATIONAL EXPERIMENT

The most straightforward and naive way to handle hard constraints within differentiable co-exploration would be to tune the relative weight to the hardware cost. For example, below is the loss function used in differentiable co-exploration [6, 9].

$$\mathcal{L}_{oss} = \mathcal{L}_{ossCE} + \lambda_{Cost} Cost_{HW}, \quad (1)$$

which is designed co-optimize accuracy and hardware cost simultaneously, and  $\lambda_{Cost}$  balances the two terms<sup>1</sup>. By increasing  $\lambda_{Cost}$ , one can indirectly instruct the search process to consider hardware metrics more. However, giving a larger penalty does not directly lead to reduction in the value of a constrained metric. Figure 1 plots how changing  $\lambda_{Cost}$  in Eq. (1) from 0.001 to 0.010 affects the latency/energy and the classification error for CIFAR-10 dataset. Searches were done three times for each setting and plotted with the same colors with large dots for their averages. Even though some trend is observed that depends on  $\lambda_{Cost}$ , inconsistency in both direction and variance of the trajectory is more dominant.

Consider a scenario where a designer wants to design a neural network-accelerator architecture pair with latency under some constraint (e.g., 33.3 ms), using the conventional co-exploration

<sup>1</sup>It is different from hyperparameters of typical machine learning formulation where the two terms serve toward a single objective.

methods. The designer would try searching with some initial  $\lambda_{Cost}$  and try adjusting the value over the course of multiple searches. However, such inconsistency between  $\lambda_{Cost}$  and the latency makes it extremely difficult to obtain the adequate solution, not to mention the huge time cost of performing the search numerous times. Despite the difficulties that lie in tackling a hard-constrained co-exploration problem, designing an effective strategy is necessary.

## 4 HARD-CONSTRAINED CO-EXPLORATION

### 4.1 Problem Definition

The mathematical formulation of hard-constrained differentiable co-exploration is as below:

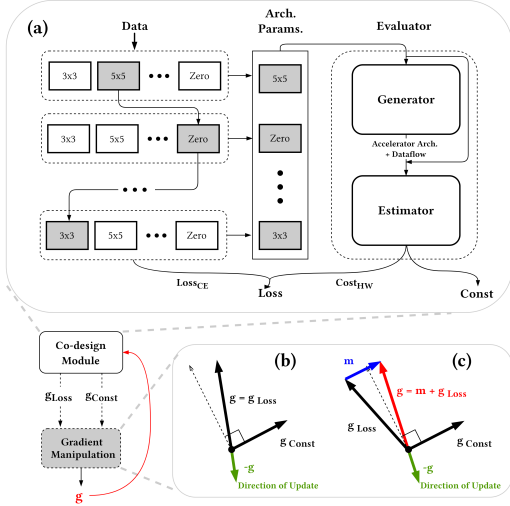
$$\begin{aligned} \arg \min_{\alpha, \beta} (\mathcal{L}_{ossNAS}(w^*, net(\alpha)) + \lambda_{Cost} Cost_{HW}(eval(\alpha, \beta))), \\ \text{s.t. } w^* = \arg \min_w (\mathcal{L}_{ossNAS}(w, net(\alpha))), \quad t \leq T, \quad (2) \end{aligned}$$

where  $t$  denotes the current value of constrained metric such as latency or energy, and  $T$  is the target value (e.g., 33.3 ms for latency).  $\alpha$  and  $\beta$  denote network architecture parameters and hardware accelerator configuration, respectively.  $w$  is the weights of the NAS supernet and  $net(\alpha)$  is the current dominant network architecture selected.  $eval(\alpha, \beta)$  indicates the hardware metrics evaluated for  $\alpha$  and  $\beta$ . The objective of co-exploration is expressed using two distinct evaluation metrics, which are neural architecture loss ( $\mathcal{L}_{ossNAS}$ ) and hardware cost ( $Cost_{HW}$ ) defined from the user.

### 4.2 Differentiable Co-exploration Framework

Although our main contribution is that we enable hard constraints, we explain our framework for the co-exploration since they are closely related. Figure 2 illustrates the overall architecture of the proposed method, being similar to existing methods [6, 9]. Figure 2 (a) is the network search module. This module searches for network architecture by choosing a path from the supernet. The network structure is then fed to the evaluator module.

The evaluator network  $eval()$  is the key to the differentiable co-exploration that enables the gradient to flow into the supernet, considering the relation between the hardware accelerator. It is a composition of two subnetworks: a hardware generator  $gen()$  and an estimator  $est()$ . The hardware generator takes the neural architecture parameters as inputs and uses them to output the optimal hardware implementation ( $\beta$  from Eq. 2). It is jointly trained during the co-exploration so that the generator does not depend on certain cost function, and can adapt to the constraint. The estimator network outputs the hardware-related metrics by taking output of the generator and the network. It is pre-trained according to the network and the accelerator search space. For pre-training the estimator, traditional (non-differentiable) cost estimation frameworks such as MAESTRO [18], Timeloop [25], and Accelergy [29] are used as ground truth. After pre-training, the estimator is frozen during the exploration and is only used to infer the hardware cost given a


**Figure 2: Overall structure of HDX.**

network architecture. With these, we convert Eq. 2 as below:

$$\begin{aligned} \arg \min_{\alpha} (\mathcal{L}_{oss_{NAS}}(w^*, net(\alpha)) + \lambda_{Cost} Cost_{HW}(est(\alpha, gen(v^*, \alpha))), \\ \text{s.t. } w^* = \arg \min_w (\mathcal{L}_{oss_{NAS}}(w, net(\alpha))), \\ v^* = \arg \min_v (Cost_{HW}(est(\alpha, gen(v, \alpha))), \end{aligned} \quad (3)$$

where  $v$  is the weights for the hardware generator.

### 4.3 Enabling Hard-Constraints with Gradient Manipulation

In addition to the differentiable co-exploration methodology, we suggest the novel idea of gradient manipulation as an effective solution to the hard constraint problem. Direct manipulation of gradients is a strategy often used in achieving multiple goals, such as in continual learning [27] or differential equations [15]. In this paper, we present a solution to apply gradient manipulation to the co-exploration problem in the interest of satisfying hard constraints.

The diagrams on Figure 2 (b) and (c) show a high-level abstraction of our gradient manipulation method. The main idea is to artificially generate a force that can push the gradient in the direction that *agrees* with the constraint. The conditions under which the method is applied to compute the new gradient  $g$  are defined as below:

$$g = \begin{cases} g_{Loss} & , \text{ if } t \leq T \\ & \text{ or } t > T \wedge g_{Loss} \cdot g_{Const} \geq 0, \\ m_{\alpha} + g_{Loss} & , \text{ otherwise} \end{cases} \quad (4)$$

$$g_{Const} = \frac{\partial \max(t - T, 0)}{\partial \alpha}. \quad (5)$$

In the above equation,  $g_{Loss}$  is the original gradient from the global loss function defined as

$$\mathcal{L}_{oss} = \mathcal{L}_{oss_{NAS}} + \lambda_{Cost} \cdot Cost_{HW}, \quad (6)$$

as in Eq. 3, and  $g_{Const}$  is the gradient of constraint loss that we define as:  $Const = \max(t - T, 0)$ . Note that  $t$  is a function of  $\alpha$ , and thus can be backpropagated to find the gradient with respect to  $\alpha$ . In an ideal case where the  $t \leq T$ , the constraint is already met so we do nothing. In the unfortunate case when the constraint is still not met,

we calculate for the dot product of the two gradients to determine the agreement in their directions. If  $g_{Loss} \cdot g_{Const} \geq 0$  (i.e., the angle between two gradients is less than  $90^\circ$ ), it means gradient descent update will contribute towards satisfying the constraint. Thus it is interpreted as an agreement in direction and the same  $g_{Loss}$  is used unmodified. Figure 2 (b) depicts this scenario. However, if they disagree as illustrated in Figure 2 (c) (i.e.,  $g_{Loss} \cdot g_{Const} < 0$ ), we force the gradient to shift its direction by  $m_{\alpha}$ , which is obtained from  $(m_{\alpha} + g_{Loss}) \cdot g_{Const} \geq 0$  to guarantee decrease in target cost after gradient descent. It can be reformulated as  $m_{\alpha} \cdot g_{Const} + g_{Loss} \cdot g_{Const} = \delta$  where  $\delta \geq 0$  is a small value for ensuring gradual movement towards satisfying the constraint.

For updating  $\alpha$  and  $w$ , we solve for optimal  $m_{\alpha}$  with respect to  $\alpha$ , which are the parameters for the network architecture. To minimize the effect of  $m_{\alpha}$  on  $g_{Loss}$ , we use a pseudoinverse-based solution that is known to minimize the size of  $\|m_{\alpha}\|_2^2$  as below:

$$m_{\alpha}^* = \frac{-(g_{Loss} \cdot g_{Const}) + \delta}{\|g_{Const}\|_2^2} g_{Const}. \quad (7)$$

In order to control the magnitude of the pull, we use a small multiplying factor  $p > 0$  on  $\delta$ . The policy for updating  $\delta$  using  $p$  is as follows: Some initial value  $\delta_0$  exists for  $\delta$ . If the target metric fails to meet the constraint,  $\delta$  is multiplied by  $1 + p$  to strengthen the pull ( $\delta' = (1 + p)\delta$ ). In the other case when the constraint is satisfied,  $\delta$  is reset to its initial value ( $\delta' = \delta_0$ ).

Note that we also train  $v$ , weights for the hardware generator using gradient descent. Thus we compute for  $m_v^*$  in the same manner, but use  $g_{Cost_{HW}}$  in place of  $g_{Loss}$  for updating the generator.

Although a single constraint is already a challenging target, our method can be further generalized to accommodate multiple constraints. Now the gradient is modified only in the direction of individual constraints that do not comply. We provide a more generalized formulation:

$$g = \begin{cases} g_{Loss} & , \text{ if } \bigwedge_{i=1}^n (t_i \leq T_i) \\ & \text{ or } \bigvee_{i=1}^n (t_i > T_i) \wedge g_{Loss} \cdot g_{Const} \geq 0, \\ m_{\alpha} + g_{Loss} & , \text{ otherwise} \end{cases} \quad (8)$$

$$g_{Const} = \frac{\partial \sum_{i=1}^n \max(t_i - T_i, 0)}{\partial \alpha}. \quad (9)$$

### 4.4 Implementation Details

**Hardware cost function.** In this work, we choose the inference latency, energy, and the chip area as the widely used hardware metrics. Considering all of them, a commonly used cost function is multiplying them (i.e., EDP, EDAP) as in [6, 9]. However, we found that the energy is usually easier to optimize for, and using such cost function unfairly favors energy-oriented designs. Therefore, we use a balanced weighted sum for the cost function as below.

$$Cost_{HW} = C_E Energy + C_L Latency + C_A Area. \quad (10)$$

**Estimator and Generator Network.** Following [6], we model both the estimator and generator with five-layer Multi-Layer Perceptron (MLP) with residual connections. To train the estimator, we first build a dataset by randomly sampling 10.8M network-accelerator pairs ( $2.95e-9$  % of the total search space) from our search space which are evaluated on hardware metrics using Timeloop [25] and Accelergy [29]. Using this dataset, the estimator

is trained for 200 epochs with the batch size of 256. The weight update is done using Adam optimizer with the learning rate of  $1e-4$ . The accuracy of the estimator was over 99% for all metrics, being powerful enough as an engine for co-exploration. The generator is randomly initialized and jointly trained with the NAS supernet. As the manipulated gradient from the hard-constraint is back-propagated, the generator learns to create accelerators that comply with the constraint on given neural network architecture.

**Search Space.** We use ProxylessNAS [4] as a NAS backbone with path sampling to train  $\alpha$ . It consists of multiple settings of MBConv operation with kernel size {3, 5, 7} and expand ratio {3, 6}. The total number of layers is 18 and 21 for CIFAR-10 [16] and ImageNet [17] dataset, respectively. However, our method is orthogonal to the NAS implementation and has the flexibility to choose from any differentiable NAS algorithms, such as DARTS [21] or OFA [3].

We use Eyeriss [5] as the accelerator’s backbone architecture. It is composed of a two-dimensional Processing Element (PE) array where each PEs has a Multiply-Accumulate (MAC) unit attached to a register file. Therefore, hardware accelerator design space comprises PE array size from  $12 \times 8$  to  $20 \times 24$ , register file size per PE from 16B to 256B. In addition, the search space includes dataflow of Weight-Stationary (WS) similar to [14], Output-Stationary (OS) similar to [8] and Row-Stationary (RS) similar to [5].

## 5 EXPERIMENTS

### 5.1 Experimental Environment

We have conducted experiments on HDX using CIFAR-10 [16] and ImageNet [17] dataset. For all the hardware metrics (latency, energy, and chip area) reported, we have used the direct evaluation on the designed hardware from Timeloop [25] and Accelergy [29] instead of the values outputted by the estimator to avoid any possible error in the learned model. Evaluation of network accuracy is done by training the final network architecture, which we train from scratch for 300 epochs using the batch size of 64. We use SGD optimizer with Nesterov momentum [24], and cosine learning rate scheduling with 0.008 as its initial value, while weight decay term and momentum is  $1e-3$  and 0.9 respectively. Augmentation for the train data is adopted from AutoAugment [7], on both dataset.

### 5.2 Comparison with Existing Methods

Table 1 lists the existing differentiable co-exploration methods in comparison with HDX. We compare the following methods:

- **NAS  $\rightarrow$  HW:** A plain differentiable NAS [4] only, followed by an exhaustive hardware search using Timeloop [25].
- **Auto-NBA [9]:** A differentiable co-exploration method that directly searches for hardware parameters using gradient descent, where the relation between hardware and the DNN is expressed as a lookup table.
- **DANCE [6]:** State-of-the-art differentiable co-exploration method without hard constraints where the generator and estimator are modeled as neural networks.
- **DANCE + Soft Constraint:** To represent a reasonable approach for considering constraints with existing solutions, we have added a soft-constraint term  $\lambda_{Soft} \cdot \max(t/T - 1, 0)$  as in [12].
- **HDX:** The proposed method with  $p = 1e-2$ .

Method	Hard Constraint	NN-HW Relation	Search with 60 FPS Constraint		
			#Searches	Cost*	Avg. Err. (%)
NAS [4] $\rightarrow$ HW search	✗	✗	5.0	10.9h	7.26
Auto-NBA [9]	✗	✓	6.8	10.2h	5.67
DANCE [6]	✗	✓	6.6	12.2h	5.32
DANCE + Soft const. [12]	✗	✓	4.9	9.1h	5.36
<b>HDX (Proposed)</b>	✓	✓	<b>1</b>	<b>2.0h</b>	<b>4.65</b>

\*GPU-hours

**Table 1: Comparison of Differentiable Co-explorations**

As displayed in Table 1, HDX is the only method that considers hard constraints with differentiable co-exploration.

For a quantitative comparison, we devised an algorithm for finding constrained solution using co-exploration without hard-constraints. Because a constrained metric (e.g., latency) being too low often means an inefficient solution (in terms of other metrics (e.g., accuracy and energy), we set the criteria of having a solution of 50%~100% of the target constraint. A rough sketch of the algorithm similar to binary search is given below:

- (1) Choose a control parameter ( $\lambda_{Soft}$  and  $\lambda_{Cost}$ ) that indirectly affects the metric under constraint (e.g., latency).
- (2) Perform a search using the default control parameter value.
- (3) Perform searches by doubling the control parameter each step until the metric is under the constraint.
- (4) If the metric is under 50% of the target value, shrink the control parameter in a binary search manner.

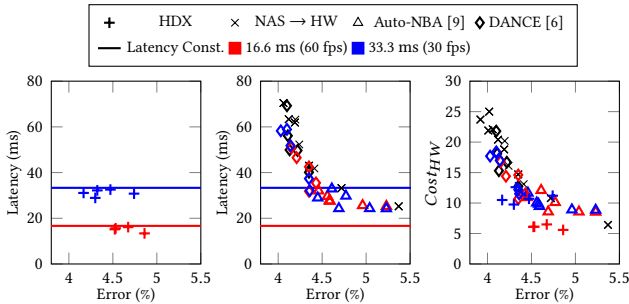
Note that the problem cannot exactly be solved by binary search, because of huge per-search variations and non-linear relations as demonstrated in Section 3. In the actual implementation, more features are added to deal with corner cases and to avoid falling into wrong parameter region (which can happen from per-search variations). We ran the algorithm 100 times and report the average. We also report the average error as a proxy to the solution quality.

As presented in Table 1, all baselines require multiple searches to find constraint-satisfying solutions. For DANCE, it takes 6.6 searches with 12.2 GPU-hours on average and 4.9 searches and 9.1 GPU-hours on even with soft constraints. On the other hand, HDX always searches for constraint-satisfying solutions in a single search. Furthermore, the number of searches for baselines strongly depend on the choice of the default value of control parameter. Guessing a right initial parameter is difficult, and could result in even larger repetition cost. This advocates the need for HDX.

### 5.3 Co-exploration Results

Figure 3 plots the co-exploration experimental results from multiple techniques on CIFAR-10 dataset. In the experiments, we have set two different constraints for the latency: 16.6ms (60 fps) and 33.3ms (30 fps). For all co-exploration methods, we obtain five solutions by varying  $\lambda_{Cost}$  from 0.001 to 0.005 to fairly compare the various design points obtained by each approach. For *NAS  $\rightarrow$  HW*, we obtain 10 solutions of various range for reference, because directly applying  $\lambda_{Cost}$  is infeasible. For DANCE and Auto-NBA, the black markers are unconstrained, and colored markers are obtained with soft constraint of the corresponding colors.

In all experiments, we have used  $C_E = 2.9$ ,  $C_L = 6.2$ , and  $C_A = 1.0$  from Eq. 10, which makes the difference scale of each metric approximately the same to make a fair comparison. For the NAS only method,  $Cost_{HW}$  was used only for the hardware design phase.



**Figure 3: Co-exploration results. (left) and (mid) represent the latency and (right) represent the hardware cost. Colored marks are methods with constraints of the same color.**

Figure 3 (left) and (mid) show the relation between error and latency. The colored horizontal bars represent the two latency targets we applied. It can be easily seen that all solutions found by HDX satisfy the given hard constraints regardless of the value of  $\lambda_{Cost}$ . Furthermore, all solutions have the latency right below the constraint, showing that the solutions did not over-optimize for the constrained metric (latency). DANCE [6] and Auto-NBA [9] were able to exploit the trade-off between hardware metrics and accuracy, but has no control over meeting the constraint. Even with soft-constraint terms, they mostly failed to obtain in-constraint solutions. Auto-NBA at a glance seems to be slightly better at meeting the constraints, but it is because its baseline method favors hardware-efficient solutions over high-accuracy ones, not because of its ability to meet the constraint, exemplified by the fact that there is no solution with high accuracy, or latency under 16.6 ms.

#### 5.4 Solution Quality Found by HDX

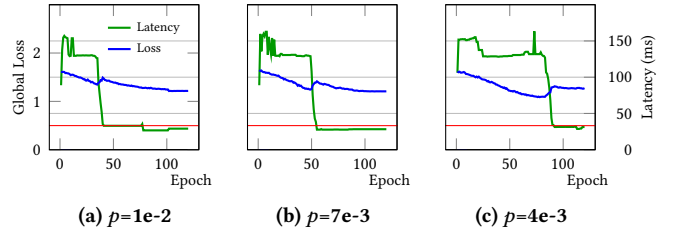
In this subsection, we demonstrate that HDX can 1) handle constraints from all three metrics (latency, energy, and chip area), 2) handle multiple constraints, and 3) obtain solutions of good overall quality. Figure 3 (right) plots  $Cost_{HW}$  and error together, which allows evaluating quality of the solutions in terms of Pareto-optimality. Because Figure 3 (left) and (mid) overlook the other metrics, comparing the  $Cost_{HW}$  together is required to be fair.

From the plot, it is clear that quality of solutions from HDX is better than the NAS→HW method, and has no degradation from the existing co-exploration methods. In fact, the tightly constrained (16.6ms) solutions even find better solution than those of the existing solutions in terms of Pareto-optimality.

**Table 2: Results Showing the Quality of Solutions**

Index	Constrained	Lat. (ms)	E (mJ)	Area (mm <sup>2</sup> )	Error (%)	$Cost_{HW}$	Loss
A	Anchor	<b>69.23</b>	<b>37.00</b>	<b>2.53</b>	4.10 ± 0.16	21.84	0.632
	Latency	<b>43.99</b>	21.79	2.10	4.20 ± 0.07	13.87	0.624
	Energy	51.98	<b>29.18</b>	2.53	4.38 ± 0.17	17.44	0.630
	Chip Area	64.00	34.82	<b>2.53</b>	4.05 ± 0.06	20.56	0.629
	All	<b>63.72</b>	<b>12.09</b>	<b>1.86</b>	4.12 ± 0.18	13.29	0.623
B	Anchor	<b>49.65</b>	<b>27.53</b>	<b>2.53</b>	4.22 ± 0.06	16.67	0.638
	Latency	<b>48.02</b>	27.33	2.53	4.27 ± 0.09	16.41	0.644
	Energy	95.02	<b>24.45</b>	1.89	4.05 ± 0.10	20.76	0.648
	Chip Area	54.74	29.81	<b>2.53</b>	4.11 ± 0.13	17.96	0.645
	All	<b>41.32</b>	<b>8.59</b>	<b>1.86</b>	4.35 ± 0.05	9.50	0.629

\*Bold colored numbers indicate that they are under constraint of the same colored non-bold numbers.



**Figure 4: Sensitivity to  $p$  on HDX. The red lines represents latency constraint at 33.3 ms.**

To further study the quality of the solutions found by HDX, we have conducted another set of experiments. We selected a few solutions found from DANCE method as ‘Anchor’ solutions and listed them in Table 2. From those, we chose either one or all three of the hardware metrics to be fixed as the hard constraint, and performed co-explorations using HDX. Because it is guaranteed that such solution exists, a good method should be able to find a solution meeting the constraint, of at least a similar quality. As in the Section 5.3, all of the 8 cases we have examined succeeded in finding a valid solution. Furthermore, all the solutions show similar global loss values from the anchor solutions as shown in the rightmost column.

#### 5.5 Results from ImageNet Dataset

Table 3 shows the co-exploration results from ImageNet dataset [17], under 125 ms constraint. As displayed in the table, HDX always succeeded in finding a solution within constraint where the others often failed to satisfy. Furthermore, the Top-1 error and the global loss shows that the quality of the solution found by HDX is not compromised at all, compared to DANCE or its variant.

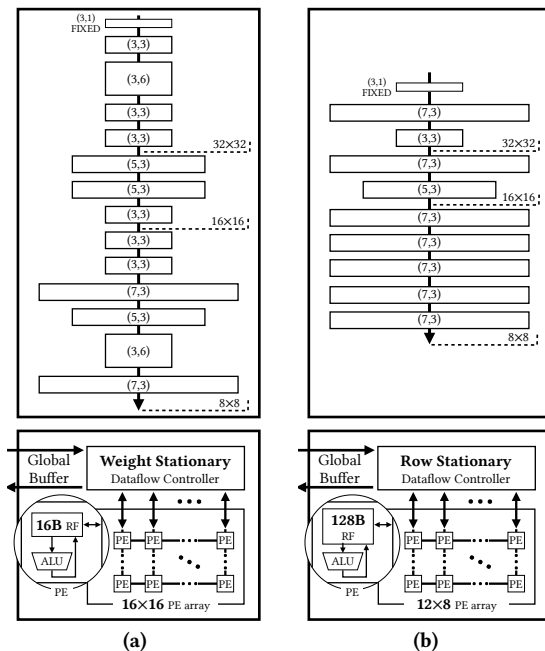
#### 5.6 Sensitivity Study on Pulling Magnitude

In HDX, the only hyperparameter is  $p$  that controls the pulling magnitude. Figure 4 illustrates how the global loss and latency changes over latency-constrained (33.3 ms) explorations, with varying  $p$  of 1e-2, 7e-3, and 4e-3.

Regardless of the value of  $p$ , the curve for the constrained value shows a similar trend. At the beginning, the global loss becomes mainly optimized, while the latency stays steady. During this phase the pulling magnitude  $\delta$  (See Eq. 7) is still growing, and is not strong enough to make meaningful changes. At certain point,  $\delta$  becomes strong enough to pull the solution towards lowering latency. When the latency satisfies the constraint, global loss starts to decrease while maintaining the latency. There is no significant discrepancy between the final solution in the global loss and the latency, which shows that HDX is insensitive to the hyperparameter  $p$ .

**Table 3: Experimental Results for ImageNet**

Method	in-const?	Lat. (ms)	Error (%)	CostHW	Loss
NAS→HW	✗	242.92	24.84	46.29	1.99
	✗	135.39	28.83	24.26	2.17
DANCE	✗	165.98	25.46	29.37	2.04
	✓	121.58	25.28	25.92	2.09
DANCE+Soft Const.	✗	188.69	25.09	33.14	1.99
	✓	105.65	26.37	25.58	2.08
HDX (Proposed)	✓	92.06	25.01	24.48	1.98
	✓	112.11	25.20	22.63	2.00



**Figure 5: Searched network and accelerator for 60 fps and 30 fps constraints.  $(m,n)$  refers MBConv block with  $m \times m$  kernel and  $n$  expand ratio.**

### 5.7 Analysis on the Searched Solutions

Fig. 5 visualizes the network and accelerator searched for 60 fps (a) and 30 fps (b) constraints. For the found design pair (a), the design contains relatively smaller kernels, more layers, and a powerful accelerator. To meet a tight constraint while maintaining accuracy, the network has small kernels, mainly of  $3 \times 3$ . Using smaller kernels quadratically reduces the number of multiplications. Therefore, decreasing the kernel size and increasing number of layers is a good choice for reducing inference latency. Looking at the accelerator design, it has relatively large PE array ( $16 \times 16$ ) to achieve low latency. It takes weight stationary (WS) dataflow, which is known to have low latency. In addition, there are some kernels with high channel expand ratio in the network. WS exploits channel parallelism for fast execution, and thus has advantage over the found network.

On the other hand, in the design for 30 fps (b), the design settles at a solution that can optimize the energy consumption while satisfying the constraint. The design uses larger kernels in the network and row stationary (RS) dataflow in the accelerator. RS is known to have good energy efficiency [5], and exploits parallelism from spatial dimensions of kernel and the activation. Thus, having larger kernels have advantages on RS dataflow. To reduce the energy consumption, the design has fewer PEs ( $12 \times 8$ ), larger RFs to save off-chip access energy, and fewer layers in the network.

## 6 CONCLUSION

In this paper, we proposed HDX, a hard-constrained differentiable co-exploration method. By conditionally applying gradient manipulation that moves the solution towards meeting the constraints, hard constraints can be reliably satisfied with high-quality solutions. We believe this proposal would ease the development of DNN based systems by a significant amount.

## ACKNOWLEDGMENTS

This work has been supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1C1C1008131, 2022R1C1C1011307), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)).

## REFERENCES

- [1] M. S. Abdelfattah *et al.* 2020. Best of Both Worlds: AutoML Codesign of a CNN and its Hardware Accelerator. In *DAC*.
- [2] M. Berman *et al.* 2020. AOWS: Adaptive and Optimal Network Width Search with Latency Constraints. In *CVPR*.
- [3] H. Cai *et al.* 2019. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.
- [4] H. Cai, L. Zhu, and S. Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.
- [5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE JSSC* (2016).
- [6] K. Choi *et al.* 2021. DANCE: Differentiable Accelerator/Network Co-Exploration. In *DAC*.
- [7] E. D. Cubuk *et al.* 2019. AutoAugment: Learning Augmentation Strategies From Data. In *CVPR*.
- [8] Z. Du *et al.* 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ISCA*.
- [9] Y. Fu *et al.* 2021. Auto-NBA: Efficient and Effective Search Over the Joint Space of Networks, Bitwidths, and Accelerators. In *ICML*.
- [10] S. Gupta and B. Akin. 2020. Accelerator-aware Neural Network Design using AutoML. *arXiv preprint arXiv:2003.02838* (2020).
- [11] C. Hao *et al.* 2019. FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge. In *DAC*.
- [12] Y. Hu, X. Wu, and R. He. 2020. TF-NAS: Rethinking Three Search Freedoms of Latency-Constrained Differentiable Neural Architecture Search. In *ECCV*.
- [13] W. Jiang *et al.* 2019. Accuracy vs. Efficiency: Achieving Both Through Fpga-Implementation Aware Neural Architecture Search. In *DAC*.
- [14] N. P. Jouppi *et al.* 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*.
- [15] J. Kim *et al.* 2021. DPM: A Novel Training Method for Physics-Informed Neural Networks in Extrapolation. In *AAAI*.
- [16] A. Krizhevsky, G. Hinton, et al. Learning Multiple Layers of Features from Tiny Images. <http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf>
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.
- [18] H. Kwon *et al.* 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* (2020).
- [19] Y. Li *et al.* 2020. EDD: Efficient Differentiable DNN Architecture and Implementation Co-Search for Embedded AI Solutions. In *DAC*.
- [20] Y. Lin, M. Yang, and S. Han. 2021. NAAS: Neural Accelerator Architecture Search. In *DAC*.
- [21] H. Liu, K. Simonyan, and Y. Yang. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.
- [22] Q. Lu *et al.* 2019. On Neural Architecture Search for Resource-Constrained Hardware Platforms. In *ICCAD*.
- [23] N. Nayman, Y. Aflalo, A. Noy, and L. Zelnik-Manor. 2021. HardCoRe-NAS: Hard Constrained differentiable Neural Architecture Search. *arXiv preprint arXiv:2102.11646* (2021).
- [24] Y. E. Nesterov. 1983. A Method for Solving the Convex Programming Problem with Convergence Rate  $O(1/k^2)$ . *Dokl. Akad. Nauk SSSR* (1983).
- [25] A. Parashar *et al.* 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *ISPASS*.
- [26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-time Object Detection. In *CVPR*.
- [27] G. Saha, I. Garg, and K. Roy. 2021. Gradient Projection Memory for Continual Learning. In *ICLR*.
- [28] M. Sandler *et al.* 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*.
- [29] Y. N. Wu, J. S. Emer, and V. Sze. 2019. Accelerger: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *ICCAD*.
- [30] L. Yang *et al.* 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. In *DAC*.
- [31] S. Zhou *et al.* 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint arXiv:1606.06160* (2016).