EH-DNAS: End-to-End Hardware-aware Differentiable Neural Architecture Search

Qian Jiang [*1] **Xiaofan Zhang** [*1 †] **Deming Chen** [1] **Minh N. Do** [1] **Raymond A. Yeh** [2]

Abstract

In hardware-aware Differentiable Neural Architecture Search (DNAS), it is challenging to integrate hardware metrics into network architecture search. To handle hardware metrics, such as inference latency, existing works mainly rely on linear approximations and lack of support for various customized hardware. In this work, we propose End-to-end Hardware-aware DNAS (EH-DNAS), a seamless integration of an end-to-end hardware performance differentiable approximation, and a fully automated DNAS to deliver hardware-efficient deep neural networks on various hardware, including Edge GPUs, Edge TPUs, Mobile CPUs, and customized accelerators. Given a targeted hardware platform, we propose to learn a differentiable model predicting the end-to-end hardware performance of the neural network architectures during DNAS. We also propose E2E-Perf, a benchmarking tool to expand our design to support customized accelerators. Experiments on CIFAR10 and ImageNet show that EH-DNAS improves the hardware performance by an average of $1.5\times$ on customized accelerators and existing hardware processors than the state-of-the-art efficient networks while maintaining highly competitive model inference accuracy.

1. Introduction

The design of architecture plays an increasingly crucial role in deep neural network (deep-net) based methods. Neural architecture search (NAS) has attracted more and more attention with the goal to deliver high-quality deep-nets. Earlier works have focused on improving the model prediction accuracy. Recent approaches have additionally considered hardware metrics, *e.g.*, model inference latency, to further enable practical deep-net designs on mobile and embedded platforms (Hao et al., 2019; Jiang et al., 2020; Stamoulis et al., 2019; Tan et al., 2019; Wu et al., 2019; Zhang et al., 2020).

Existing hardware-aware NAS methods rely on proxies to approximate the end-to-end hardware metrics during the search procedure. Common proxies include the number of multiply-accumulate operations and network parameters (Liu et al., 2019; Real et al., 2019). However, these proxies may fail to align with the realistic hardware performance. To address this gap, prior works (Cai et al., 2019; Dai et al., 2019; Wu et al., 2019) propose to approximate the hardware metrics with look-up-tables (LUTs) which consists of per-layer hardware metrics. Another solution is to directly incorporate the end-to-end hardware metrics, *e.g.*, the latency by running an entire deep-net on targeted devices (Dudziak et al., 2020; Zhang et al., 2020). However, this approach can not be directly adopted by differentiable NAS (DNAS) procedures due to its discrete measurements of each architecture. Additionally, the hardware feedback is hard to obtain in an online fashion during NAS, as it requires additional time and effort to collect.

To address these shortcomings, we propose EH-DNAS, an end-to-end hardware-aware DNAS framework to automatically search for hardware-efficient deep-nets on various hardware. Importantly, we propose to *learn a differentiable function to approximate the end-to-end hardware metrics* of network candidates. This learned function provides hardware feedback that can be directly integrated into *any DNAS workflow*, enabling the search for efficient and accurate architectures. Next, we also introduce E2E-Perf, an efficient tool to benchmark hardware performance on customized hardware accelerators. Fig. 1 presents a visual overview of the proposed design.

To demonstrate, we adopt EH-DNAS to two existing DNAS workflows, including DARTS (Liu et al., 2019) and GDAS (Dong & Yang, 2019), to generate hardware-efficient deep-nets. We evaluate our designs on CIFAR10 and ImageNet using the search space of DARTS (Liu et al., 2019) and HW-Bench-201 (Dong et al., 2021; Li et al., 2021). Results show that the proposed EH-DNAS improves the

[*]Equal contribution [1]Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign. †Now at Google. [2]Department of Computer Science, Purdue University. Correspondence to: Qian Jiang <qianj3@illinois.edu>.

Published at the Differentiable Almost Everything Workshop of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023.

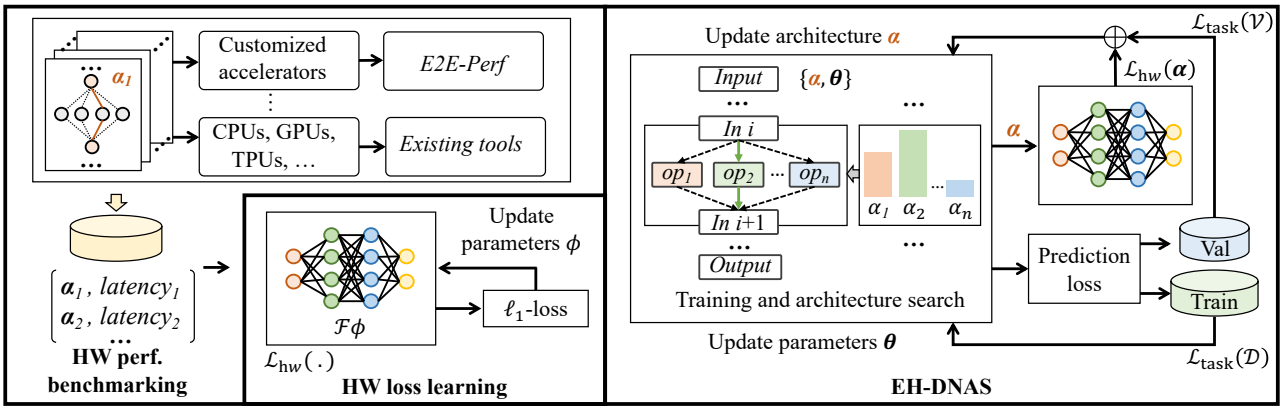


Figure 1. Our proposed EH-DNAS framework. **HW perf. benchmarking:** We measure the hardware (HW) performance of network architectures $\{\boldsymbol{a}_i\}_{i=1}^n$ to collect an HW performance dataset $\{(\boldsymbol{a}_i, latency_i)\}_{i=1}^n$. Specifically, we introduce E2E-Perf for benchmarking customized accelerators. This stage enables diverse hardware support for EH-DNAS. **HW loss learning:** With a collected dataset, we propose a deep-net to learn a differentiable approximation $\mathcal{F}_\phi$ of the hardware performance with respect to any selected network architecture. **EH-DNAS:** We then perform DNAS with learned HW loss $\mathcal{L}_{\text{hw}}(\boldsymbol{a}) \triangleq \mathcal{F}_\phi(\boldsymbol{a})$ providing end-to-end hardware feedback

hardware performance by an average of 1.4× on customized accelerators, and 1.6× on existing hardware processors, including Edge GPUs, Edge TPUs, and Mobile CPUs, while maintaining the model accuracy.

2. Related Work

Proxy-based hardware-aware NAS. Proxy-based hardware estimation is a widely adopted method to provide hardware feedback for NAS designs. However, earlier works have indicated that the popular proxies, *i.e.*, FLOPS and the number of network parameters, are not closely correlated with actual hardware performance on targeted devices (Sandler et al., 2018; Wan et al., 2020; Wu et al., 2019). These proxies are usually non-differentiable as well.

Linear-approximation-based hardware-aware Differentiable NAS. To improve, recent works use LUTs (Dai et al., 2019; Li et al., 2021; Wu et al., 2019) and linear prediction (Cai et al., 2019) for hardware-aware NAS designs. These methods can collect more realistic hardware metrics and are differentiable. However, these linear approximation methods may fail to accurately represent end-to-end hardware metrics. For example, when targeting the same VGG network (Simonyan & Zisserman, 2015) with identical hardware resource budget, the throughput performance of hardware designs (Qiu et al., 2016; Xiao et al., 2017; Zhang et al., 2018) varies from 137 to 262 giga-operation per second (GOPS). The performance variation is mainly caused by different compute and memory access patterns caused by different hardware architectures, which can not be captured by linear approximations.

End-to-end hardware-aware NAS. While end-to-end methods (Dudziak et al., 2020; Li et al., 2021; Zhang et al., 2020) show better consistency for benchmarking the hardware performance, they have some shortcomings, *e.g.*, they need extra time and effort for measurements on actual hardware and cannot be integrated into existing DNAS procedures due to the discrete nature.

3. Preliminaries

Differentiable neural architecture search. At a high level, NAS can be formulated as a bi-level optimization problem:

$$\min_{\boldsymbol{a}\in\mathcal{A}} \mathcal{L}(\mathcal{V}, \boldsymbol{\theta}^*(\boldsymbol{a}), \boldsymbol{a}) \;\text{ s.t. }\; \boldsymbol{\theta}^*(\boldsymbol{a}) = \min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{a}), \quad (1)$$

where $\boldsymbol{a}$ denotes an architecture within the search space $\mathcal{A}$, $\mathcal{L}$ denotes a loss function, $\mathcal{D}/\mathcal{V}$ denotes the train/validation set, and θ denotes the model's trainable parameters. A typical search space consists of L network layers where each layer consists of K candidate blocks, *e.g.*, convolution with different filter sizes. More formally, we defined an architecture $\boldsymbol{a}$ and the search space $\mathcal{A}$ as follows:

$$\boldsymbol{a} = [\boldsymbol{a}^{(1)}, \ldots \boldsymbol{a}^{(l)}, \ldots \boldsymbol{a}^{(L)}] \text{ and } \mathcal{A} = \{0,1\}^{K\times L}, \quad (2)$$

where $\boldsymbol{a}^{(l)} = [a_1^{(l)}, \ldots, a_k^{(l)} \ldots, a_K^{(l)}]^T$, $\sum_k a_k^{(l)} = 1$ and $a_k^{(l)} \in \{0,1\}$ indicating the one-hot selection of the k^{th} block at the l^{th} layer, denoted with $f_k^{(l)}$. Brute-force search over the discrete space $\mathcal{A}$ is computationally expensive. Hence, DNAS relaxes $a_k^{(l)} \in [0,1]$ to perform a soft selection. *I.e.*, the selection of a block is formulated as a weighted sum of a block's output:

$$\boldsymbol{x}^{(l+1)} = \sum_k a_k^{(l)} \cdot f_k^{(l)}(\boldsymbol{x}^{(l)}; \theta). \quad (3)$$

With this soft selection, the architecture is now differentiable with respect to (w.r.t.) the loss function. This enables the

use of gradient-based optimization techniques to perform NAS, *i.e.*, solving the bi-level optimization in Eq. 1.

Hardware-aware search. To utilize DNAS for designing hardware-efficient architecture, it is required to choose a loss function of the form

$$\mathcal{L}(\mathcal{V}, \boldsymbol{\theta}, \boldsymbol{a}) = \mathcal{L}_{\texttt{task}}(\mathcal{V}, \boldsymbol{\theta}, \boldsymbol{a}) + \beta \mathcal{L}_{\texttt{hw}}(\boldsymbol{a}), \tag{4}$$

where $\beta \in \mathbb{R}^+$ is a hyperparameter that balances the two losses. Here, $\mathcal{L}_{\texttt{task}}$ captures the task performance, *e.g.*, cross-entropy for classification tasks, and $\mathcal{L}_{\texttt{hw}}$ captures the hardware cost of an architecture, *e.g.*, latency.

4. Method

As reviewed in Sec. 3, the search for a hardware-aware model is based on a hardware loss function $\mathcal{L}_{\texttt{hw}}$. Ideally, $\mathcal{L}_{\texttt{hw}}(\boldsymbol{a})$ should accurately characterize the hardware's metrics, *e.g.*, latency of a given architecture $\boldsymbol{a}$. At the same time, $\mathcal{L}_{\texttt{hw}}$ needs to be computed efficiently and differentiable w.r.t. the architecture $\boldsymbol{a}$. To accomplish these goals, we propose EH-DNAS to: (a) learn a differentiable approximation of the hardware feedback (Sec. 4.1); (b) benchmark realistic hardware performance with our proposed tool E2E-Perf (Sec. C.4). Fig. 1 provides an overview of EH-DNAS.

4.1. Learning a Differentiable Hardware Loss

Problem formulation. A hardware loss, $\mathcal{L}_{\texttt{hw}}(\boldsymbol{a})$ should accurately resemble the desired hardware metrics, *e.g.*, latency, for a given model architecture $\boldsymbol{a}$. To use the hardware loss with DNAS, we also need $\mathcal{L}_{\texttt{hw}}(\boldsymbol{a})$ to be differentiable w.r.t. $\boldsymbol{a}$. To satisfies these requirements, we propose to learn $\mathcal{L}_{\texttt{hw}}$ with a deep-net.

We formulate this learning procedure as a regression task:

$$\min_{\phi} \frac{1}{|\mathcal{A}|} \sum_{\boldsymbol{a} \in A} |\texttt{HWPerf}(\boldsymbol{a}) - \mathcal{L}_{\texttt{hw}}^{\texttt{deep}}(\boldsymbol{a}; \phi)|, \tag{5}$$

to minimize the mean absolute error between the hardware performance $\texttt{HWPerf}$ and the hardware loss $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ over all architectures in $\mathcal{A}$. We defer the description of how to acquire $\texttt{HWPerf}(\boldsymbol{a})$ to Sec. C.4. Here, we choose to model $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ with a deep-net with trainable parameters ϕ, *i.e.*,

$$\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}(\boldsymbol{a}; \phi) \triangleq \mathcal{F}(\boldsymbol{a}; \phi) = g_N \circ g_{N-1} \ldots \circ g_1(\boldsymbol{a}), \tag{6}$$

where ϕ subsumes all the trainable parameters in the layers g_N to g_1.

Model details of hardware loss. We use a standard fully connected network to model $\mathcal{L}_{\texttt{hw}}$. Specifically, $\mathcal{F}$ starts with a linear embedding layer g_1, which represents each candidate block $a_k^{(l)}$ (the k-th candidate block at l-th layer) with a 10-dimensions vector. Formally, an architecture's embedding $\mathcal{E}(\boldsymbol{a})$ is computed as:

$$\mathcal{E}(\boldsymbol{a}) = \text{Concat}([\boldsymbol{W}_1 \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{W}_L \boldsymbol{a}^{(L)}]), \tag{7}$$

where $\boldsymbol{a}^{(l)} = [a_1^{(l)} \ldots, a_K^{(l)}]^T$, and $\boldsymbol{W}_l$ denotes the trainable parameters of the embedding layer. Following the embedding layer are three fully connected layers with ReLU non-linearities. We apply dropout (Srivastava et al., 2014) to the final layer as we observed over-fitting to the training set. We include model design details in Sec. B.

Hardware loss model training. To train this model, we need to compute Eq. 5 and use gradient-based optimization methods as the model involves a deep-net. However, this is not always feasible due to the size of the search space $|\mathcal{A}|$. Hence, we instead optimize an unbiased estimate of Eq. 5 over a feasible number of samples. This is done by sampling architectures from $\mathcal{A}$ and updating the model parameters using mini-batches.

4.2. Architecture Search with EH-DNAS

With $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ trained, we can easily compute an approximation of architecture's hardware performance by running a forward pass through the deep-net. The gradient w.r.t. the architecture can also be computed by running a backward pass through the deep-net, *i.e.*,

$$\frac{\partial \mathcal{L}_{\texttt{hw}}^{\texttt{deep}}(\boldsymbol{a})}{\partial \boldsymbol{a}} = \frac{\partial g_N}{\partial g_{N-1}} \cdot \frac{\partial g_{N-1}}{\partial g_{N-2}} \cdots \frac{\partial g_1}{\partial \boldsymbol{a}}. \tag{8}$$

The overall validation loss function for DNAS becomes

$$\mathcal{L}(\mathcal{V}, \boldsymbol{\theta}, \boldsymbol{a}) = \mathcal{L}_{\texttt{task}}(\mathcal{V}, \boldsymbol{\theta}, \boldsymbol{a}) + \beta \mathcal{L}_{\texttt{hw}}^{\texttt{deep}}(\boldsymbol{a}; \phi), \tag{9}$$

where ϕ is fixed during the architecture search.

We can easily integrate $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ into any DNAS method to perform hardware-aware DNAS and search for hardware efficient models. In our experiments, we demonstrate the effectiveness and generalizability of our hardware loss by incorporating it into two DNAS workflow and search spaces, *e.g.*, DARTS V1 (Liu et al., 2019), GDAS (Dong & Yang, 2019) and NAS-Bench-201 (Dong & Yang, 2020). The benefits of our deep hardware loss $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ are that it does not assume independence nor linearity among the candidate blocks. Hence, it can capture more realistic hardware metrics than the LUT-based approaches.

5. Experiments

We evaluate the proposed approach in three folds. First, we evaluate the performance of our framework on DARTS search space, where we focus on customized hardware accelerator performance using our proposed tool E2E-Perf.

Table 1. Quantitative results on ImageNet on DARTS search space. We report customized hardware accelerator latency using our proposed hardware performance benchmarking tool E2E-Perf (Appendix E).

Approach	Latency ↓ (ms)	Top1 Error↓ (%)	Params↓ (M)	#GFLOP↓	Search Cost↓ (GPU Days)
DARTS (Liu et al., 2019)	7.46	30.8	4.7	1.03	1
DARTS + FBNet LUT (Wu et al., 2019)	7.40	30.0	4.1	0.94	1
DARTS + EH-DNAS	**5.81**	30.4	**3.7**	**0.84**	1

Table 2. Quantitative results on CIFAR10 on NAS-Bench-201 (Dong & Yang, 2020) search space. We show results based on two DNAS algorithms. For each row (representing an approach), we search for four architectures, each for a type of hardware. We report hardware latency using HW-NAS-Bench (Li et al., 2021)

Approach		Edge GPU		Edge TPU		Raspi 4		Pixel 3	
Search Algorithm	Hardware Feedback	Latency ↓ (ms)	Top1 ↓ (%)	Latency ↓ (ms)	Top1↓ (%)	Latency ↓ (ms)	Top1↓ (%)	Latency ↓ (ms)	Top1↓ (%)
DARTS (Liu et al., 2019)	-	3.74	45.7	0.60	45.7	3.84	45.7	**1.61**	45.7
	FBNet LUT (Wu et al., 2019)	2.45	15.8	0.60	45.7	2.96	29.1	**1.61**	45.7
	EH-DNAS	**2.41**	**15.7**	**0.51**	**29.1**	**2.96**	**29.1**	1.66	**29.1**
GDAS (Dong & Yang, 2019)	-	6.58	**6.6**	1.26	**6.6**	69.19	6.6	27.70	6.6
	FBNet LUT (Wu et al., 2019)	4.05	24.4	1.27	6.6	0.01	90.0	0.01	90.0
	EH-DNAS	**1.88**	8.1	**1.10**	6.9	**56.89**	**6.4**	**16.97**	**6.4**

We adopt EH-DNAS to DARTS and evaluate searched architectures on CIFAR10 and their performance when transferring to ImageNet. Next, we evaluate on NAS-Bench-201 search space, where we focus on existing hardware performance using HW-NAS-Bench for benchmarking hardware performance. We adopt EH-DNAS to two DNAS workflows including DARTS and GDAS(Dong & Yang, 2019), and evaluate the performance of searched architectures on CIFAR10. We aim to search for neural architectures with optimized hardware performance without sacrificing classification accuracy. We provide comprehensive results, analysis, ablation study, and implementation details in the Appendix.

5.1. Results on Customized Hardware Accelerators

Experiment setup. We estimate the customized hardware performance of architectures on DARTS (Liu et al., 2019) search space. We sampled 10,000K architectures to train the hardware loss and then integrate it into the DARTS training pipeline. We search for cells on CIFAR-10 and evaluate them on ImageNet. All hardware performance metrics are measured using our proposed tool E2E-Perf configured with a large hardware budget (4800 DSPs 141Mb on-chip memory, comparable to a mid-range cloud processor). Note that $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ can generalize on different hardware (Sec. B).

Results on ImageNet. Following DARTS, we conduct experiments transferring the cells searched on CIFAR10 to ImageNet. In Tab. 5 we report the quantitative results on ImageNet. Compared to DARTS, we improve the classification accuracy by 0.4% while improving hardware performance by $1.3\times$, with a 1M reduction in the number of parameters. Our approach also achieves the lowest latency, number of parameters, and FLOPs among all baselines.

5.2. Results on Existing Hardware Processors

Experiment setup. We use HW-NAS-Bench to acquire existing hardware processors (Edge GPU, Edge TPU, Raspi 4, and Pixel 3) performance on NAS-Bench-201 search space. To further demonstrate that EH-DNAS as a general approach can be applied to any DNAS algorithm, we adapt it to two differentiable neural architecture search algorithms (DARTS and GDAS). We resemble LUT from HW-NAS-Bench for FBNet to make fair comparison.

Results on CIFAR10. In Tab. 6, we report quantitative results on CIFAR10 regarding two search algorithms (DARTS and GDAS) on four hardware processors. With DARTS, our approach improves the hardware performance by an average of $1.3\times$ while improving the classification accuracy by an average of 20% for all hardware processors. With GDAS, our approach features an average of $1.9\times$ hardware performance improvement while maintaining high classification accuracy. We note that FBNet LUT does not find meaningful architectures on Raspi 4 and Pixel 3 possibly due to the mismatch between real latency and the approximated LUT.

6. Conclusion

We present EH-DNAS, an end-to-end hardware-aware DNAS framework to deliver hardware-efficient deep-nets. We integrate hardware performance benchmarking, differentiable hardware loss approximation, and DNAS to search for efficient and accurate networks. On CIFAR10 and ImageNet, we demonstrated that EH-DNAS can improve the hardware performance by an average of $1.5\times$ on customized accelerators and existing hardware processors while maintaining highly competitive model inference accuracy.

References

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *Proc. ICLR*, 2017.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proc. ICLR*, 2019.

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE JSSC*, 2016.

Chen, Y.-H., Emer, J., and Sze, V. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro*, 2017.

Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., and Jha, N. K. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proc. CVPR*, 2019.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *Proc. CVPR*, 2019.

Dong, X. and Yang, Y. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proc. ICLR*, 2020.

Dong, X., Liu, L., Musial, K., and Gabrys, B. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE TPAMI*, 2021.

Dudziak, L., Chau, T., Abdelfattah, M., Lee, R., Kim, H., and Lane, N. BRP-NAS: Prediction-based NAS using GCNs. In *Proc. NeurIPS*, 2020.

Hao, C., Zhang, X., Li, Y., Huang, S., Xiong, J., Rupnow, K., Hwu, W.-m., and Chen, D. FPGA/DNN co-design: An efficient design methodology for iot intelligence on the edge. In *Proc. DAC*, 2019.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. In *Proc. ICCV*, 2019.

Jiang, W., Yang, L., Dasgupta, S., Hu, J., and Shi, Y. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE TCAD*, 2020.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. ISCA*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., and Lin, Y. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *Proc. ICLR*, 2021.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proc. ECCV*, 2018.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *Proc. ICLR*, 2019.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *Proc. ICML*, 2018.

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., et al. Going deeper with embedded FPGA platform for convolutional neural network. In *Proc. FPGA*, 2016.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proc. AAAI*, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. CVPR*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y. (eds.), *Proc. ICLR*, 2015.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path NAS: Designing hardware-efficient convnets in less than 4 hours. In *Proc. ECML-PKDD*, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proc. CVPR*, 2019.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proc. CVPR*, 2020.

Wei, X., Liang, Y., Li, X., Yu, C. H., Zhang, P., and Cong, J. TGPA: tile-grained pipeline architecture for low latency CNN inference. In *Proc. ICCAD*, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proc. CVPR*, 2019.

Xiao, Q., Liang, Y., Lu, L., Yan, S., and Tai, Y.-W. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *Proc. DAC*, 2017.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *Proc. ICLR*, 2019.

Xu, P., Zhang, X., Hao, C., Zhao, Y., Zhang, Y., Wang, Y., Li, C., Guan, Z., Chen, D., and Lin, Y. AutoDNNchip: An Automated DNN Chip Predictor and Builder for Both FPGAs and ASICs. In *Proc. FPGA*, 2020.

Ye, H., Zhang, X., Huang, Z., Chen, G., and Chen, D. HybridDNN: A framework for high-performance hybrid dnn accelerator design and implementation. In *Proc. DAC*, 2020.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. In *Proc. ICML*, 2019.

Zhang, X., Wang, J., Zhu, C., Lin, Y., Xiong, J., Hwu, W.-m., and Chen, D. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *Proc. ICCAD*, 2018.

Zhang, X., Lu, H., Hao, C., Li, J., Cheng, B., Li, Y., Rupnow, K., Xiong, J., Huang, T., Shi, H., Hwu, W.-m., and Chen, D. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. In *Proc. MLSys*, 2020.

Zhao, Y., Li, C., Wang, Y., Xu, P., Zhang, Y., and Lin, Y. DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures. In *Proc. ICASSP*, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *Proc. ICLR*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proc. CVPR*, 2018.

The appendix contains the following:

- In Sec. A, we provide complete experimental results comparing with more baselines.
- In Sec. B, we provide analysis and ablation study.
- In Sec. C, we provide additional experimental results.
- In Sec. D, we provide the design details of E2E-Perf.
- In Sec. E, we provide implementation and experimental details.
- In Sec. F, we discuss limitations and potential negative societal impact.
- In Sec. G, we provide an overview of the attached code.

A. Complete comparison with baselines

A.1. Results on Customized Hardware Accelerators

Experiment setup. We estimate customized hardware accelerators performance of architectures on DARTS (Liu et al., 2019) search space. Since the original search space is infeasible ($8^{28} \approx 10^{25}$ architectures), we sampled 1,000K, 200K, and 200K architectures to form the training, validation, and test sets. Each architecture is evaluated with E2E-Perf under a relatively large hardware budget (4800 DSPs 141Mb on-chip memory, comparable to a mid-range cloud processor) for corresponding hardware performance. The dataset collection costs 12 hours in total for each paradigm.

We train our deep hardware loss $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ using the collected dataset and integrate it into DARTS training pipeline. We consider two paradigms of customized hardware accelerators, namely, pipeline paradigm (PP) and generic paradigm (GP). For each paradigm, we train a separate hardware loss. The training time of hardware loss is 2 hours per paradigm. Due to the compact design of the hardware loss model, see Sec. 4.1, the inference time is minimal.

DARTS architecture search consists of two stages. In the first stage, we search for the best cell choices based on the validation performance of both classification accuracy and hardware performance. In the second stage, the final searched cells are selected and cells are stacked to construct the final architecture, where the number of cells is 8 for CIFAR10 and 14 for ImageNet. Lastly, this final architecture is trained from scratch to evaluate classification performance.

For hardware performance (latency), we evaluate using E2E-Perf with the same hardware budget as collecting the hardware performance dataset. We show the generalizability of the trained $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ on different hardwares in Sec. B.

We consider recent baselines including popular NAS and DNAS methods, *e.g.*, NASNET (Zoph et al., 2018), AmoebaNet (Real et al., 2019), ProxylessNAS (Cai et al., 2019), MobileNet V3 (Howard et al., 2019; Sandler et al., 2018), SNAS (Xie et al., 2019), DARTS (Liu et al., 2019), and FBNet (Wu et al., 2019). Note that for a fair comparison, we adjust FBNet baseline by acquiring a latency look-up-table (LUT) using E2E-Perf with generic paradigm. Note that the LUT-based method can only be applied to the generic paradigm (GP) as all layers are required to be executed by the same hardware components.

Results on CIFAR10. We report the quantitative results on CIFAR10 regarding two paradigms, the pipeline paradigm (PP) in Tab. 3 and the generic paradigm (GP) in Tab. 4. Each row represents the performance of the architecture searched using the corresponding approach.

In Tab. 3, observe that compared to DARTS, we improve the hardware performance by $1.8\times$, as well as improve the classification accuracy by 0.16%. Our approach reaches the lowest hardware latency among all baselines and comparable classification accuracy. Our approach also features the lowest number of parameters and FLOPS.

In Tab. 4, when compared to DARTS, we improve the hardware performance by $1.3\times$, and improve the classification accuracy by 0.08%. Our approach has the lowest latency, number of parameters, and FLOPS among all baselines, with comparable classification accuracy.

Results on ImageNet. Following DARTS, we conduct experiments transferring the cells searched on CIFAR10 to ImageNet. The generic paradigm allows constructing generic reusable hardware compute units to recurrently process all deep-net

Table 3. Quantitative results on CIFAR-10 on DARTS search space for customized hardware accelerators with the pipeline paradigm (PP). We report the latency of the searched architecture evaluated with E2E-Perf. Note that LUT can not be obtained with PP

Approach	Latency ↓ (ms)	Test Error↓ (%)	Params↓ (M)	#GFLOP↓	Search Cost↓ (GPU Days)
NASNet-A (Tan et al., 2019)	1.38	**2.83**	3.1	0.12	2000
AmoebaNet-A (Real et al., 2019)	0.93	3.12	3.1	0.11	3150
SNAS-mild (Xie et al., 2019)	**0.31**	2.98	2.9	0.08	1.5
DARTS (Liu et al., 2019)	0.55	3.00	3.3	0.09	**1**
DARTS + EH-DNAS	**0.31**	2.84	**2.5**	**0.07**	**1**

Table 4. Quantitative results on CIFAR-10 on DARTS search space for customized hardware accelerators with the generic paradigm (GP). Latency is evaluated using E2E-Perf

Approach	Latency ↓ (ms)	Test Error↓ (%)	Params↓ (M)	#GFLOP↓	Search Cost↓ (GPU Days)
NASNet-A (Tan et al., 2019)	5.49	2.83	3.1	0.12	2000
AmoebaNet-A (Real et al., 2019)	4.39	3.12	3.1	0.11	3150
SNAS-mild (Xie et al., 2019)	2.48	2.98	2.9	0.08	1.5
DARTS (Liu et al., 2019)	2.99	3.00	3.3	0.09	**1**
DARTS + FBNet LUT (Wu et al., 2019)	2.93	**2.82**	2.9	0.08	**1**
DARTS + EH-DNAS	**2.31**	2.92	**2.5**	**0.07**	**1**

layers, which is more widely used in newly developed customized accelerators. Thus we use the cell searched with the generic paradigm and stack 14 layers of cells to construct the final model for ImageNet. Our training details are provided in the Appendix.

In Tab. 5 we report the quantitative results on ImageNet. Compared to DARTS, we improve the classification accuracy by 0.4% while improving hardware performance by 1.3 times, with 1M reduction in the number of parameters. Our approach also achieves the lowest latency, number of parameters, and FLOPs among all baselines.

A.2. Results on Existing Hardware Processors

Experiment setup. We use HW-NAS-Bench to acquire existing hardware processors performance on NAS-Bench-201 search space. From HW-NAS-Bench, we can obtain the hardware performance of any architecture candidate from NAS-Bench-201 search space. In total, we train four hardware losses, one for each hardware processor. The training time of hardware loss is one hour per hardware. Due to the compact design of hardware loss models, see Sec. 4.1, the inference time is minimal.

During the architecture search, we search for the best cell choices based on validation performance on both classification accuracy and hardware performance. We follow all settings in NAS-Bench-201. To further demonstrate that EH-DNAS as a general approach can be applied to any DNAS algorithm, we adapt it to two differentiable neural architecture search algorithms (DARTS and GDAS (Dong & Yang, 2019)). We consider FBNet (Wu et al., 2019) LUT approach as a baseline and for a fair comparison we resemble their latency look-up table from HW-NAS-Bench.

Results on CIFAR10. In Tab. 6, we report quantitative results on CIFAR10 regarding two search algorithms (DARTS and GDAS) on four hardware processors. With DARTS, our approach improves the hardware performance by an average of $1.3\times$ while improving the classification accuracy by an average of 20% for all hardware processors. With GDAS, our approach features an average of $1.9\times$ hardware performance improvement while maintaining the classification accuracy. We note that FBNet LUT does not find meaningful architectures on Raspi 4 and Pixel 3. This is mainly due to the mismatch between real latency and the approximated LUT which assumes additive loss between blocks. We provide more analysis in the next section.

B. Analysis and Abalation Studies

Generalizability and transferability of trained hardware loss. Beyond the quantitative results, we further study how our $\mathcal{L}_{\texttt{hw}}^{\texttt{deep}}$ learned on one hardware performance dataset can generalize for other hardware. With support of E2E-Perf, we

Table 5. Quantitative results on ImageNet on DARTS search space. We evaluate latency on customized hardware accelerators with the generic paradigm (GP) using E2E-Perf

Approach	Latency ↓ (ms)	Top1 Error↓ (%)	Params↓ (M)	#GFLOP↓	Search Cost↓ (GPU Days)
NASNet-A (Tan et al., 2019)	11.90	26.0	5.3	1.23	2000
AmoebaNet-A (Real et al., 2019)	9.17	25.5	5.1	1.07	3150
ProxylessNAS (Cai et al., 2019)	20.41	**24.9**	4.1	0.66	200
MobileNet-V3 (Howard et al., 2019)	9.90	26.0	5.5	0.12	-
FBNet-A (Wu et al., 2019)	11.63	27.0	4.3	0.48	9
SNAS-mild (Xie et al., 2019)	6.02	27.3	4.3	0.89	1.5
DARTS (Liu et al., 2019)	7.46	30.8	4.7	1.03	1
DARTS + FBNet LUT (Wu et al., 2019)	7.40	30.0	4.1	0.94	1
DARTS + EH-DNAS	**5.81**	30.4	**3.7**	**0.84**	1

Table 6. Quantitative results on CIFAR10 on NAS-Bench-201 (Dong & Yang, 2020) search space. We show results based on two DNAS algorithms. For each row (representing an approach), we search for four architectures, each for a type of hardware. We report hardware latency using HW-NAS-Bench (Li et al., 2021)

Approach		Edge GPU		Edge TPU		Raspi 4		Pixel 3	
Search Algorithm	Hardware Feedback	Latency ↓ (ms)	Top1 ↓ (%)	Latency ↓ (ms)	Top1↓ (%)	Latency ↓ (ms)	Top1↓ (%)	Latency ↓ (ms)	Top1↓ (%)
DARTS (Liu et al., 2019)	-	3.74	45.7	0.60	45.7	3.84	45.7	**1.61**	45.7
	FBNet LUT (Wu et al., 2019)	2.45	15.8	0.60	45.7	2.96	29.1	**1.61**	45.7
	EH-DNAS	**2.41**	**15.7**	**0.51**	**29.1**	**2.96**	**29.1**	1.66	**29.1**
GDAS (Dong & Yang, 2019)	-	6.58	**6.6**	1.26	**6.6**	69.19	6.6	27.70	6.6
	FBNet LUT (Wu et al., 2019)	4.05	24.4	1.27	6.6	0.01	90.0	0.01	90.0
	EH-DNAS	**1.88**	8.1	**1.10**	6.9	**56.89**	**6.4**	**16.97**	**6.4**

can measure the latency of hardware with any budget. We evaluate the searched architecture from Sec. A.1 on two smaller hardware budgets (medium budget: 2400DSPs with 70Mb on-chip memory; and small budget:1400DSPs with 46Mb on-chip memory), which cover from cloud to edge computing hardware budgets.

In Tab. 7, we show that for each paradigm, the searched architecture on CIFAR-10 using $\mathcal{L}_{\text{hw}}^{\texttt{deep}}$ learned on large budget HW dataset leads to latency improvement on every hardware with different budgets. On the other hand, other approach (FBNet LUT) does not provide such benefits, *e.g.*, the latency on small and medium budget hardware even increases compared to DARTS without hardware optimization. It is also notable that DARTS searched architecture exceeds the small HW budget, indicating it is unable to be deployed on such edge device with limited hardware resources. This as well signify the superiority of our approach on optimizing hardware performance.

In other words, for any hardware under the same design paradigm/the same type of hardware, *e.g.*, generic paradigm or pipeline paradigm, it is sufficient to reuse the learned hardware loss. This significantly reduces the effort of re-collecting dataset and retraining and shows the generalizability of our learned hardware loss. For more results please refer to the Appendix.

Quality of hardware performance prediction. To better understand the mechanism behind our approach, we further analyze the learned hardware loss. In Tab. 8, we report the relative average error rate (%) of hardware performance prediction. The error rate is defined as the absolute difference between predicted and true hardware performance, divided by true hardware performance. The true hardware performance are measured by E2E-Perf for customized accelerators and HW-NAS-Bench for existing hardware processors. We report mean and standard deviation over three runs with different random initialization seeds. Note that the LUT-based approach is deterministic where we obtain the latency LUT for different hardware per layer. The overall network latency is calculated by summing up the latency from each layer.

Notably, observe that a LUT leads to more significant prediction errors for Raspi 4 and Pixel 3, where mobile CPUs are involved. This could lead to the failure cases of the LUT approach in Tab. 6. The major reason for this phenomenon is that mobile CPUs perform not only deep-net inference but also run other tasks, such as the operating systems. In addition, the limited memory access bandwidth in mobile CPU is likely to be the bottleneck that significantly slows down the overall performance. In comparison, our approach reaches much lower average error rates for all hardware platforms, as well as

Table 7. Results on the generalizability of our approach. The searched architecture using $\mathcal{L}_{\text{hw}}^{\texttt{deep}}$ trained on the large budget hardware(HW) performance datasets are measured under HW of different budgets. Small, medium and large indicate the latency performance on HW with respective budgets. ✗ indicates the architecture exceeds the HW budget. Note that LUT can not be obtained with PP

Approach	Pipeline Paradigm (PP) Latency (ms)↓			Generic Paradigm (GP) Latency (ms)↓		
	Small	Medium	Large	Small	Medium	Large
DARTS (Liu et al., 2019)	✗	1.66	0.55	5.59	3.60	2.99
DARTS + FBNet LUT (Wu et al., 2019)	-	-	-	5.99	3.72	2.93
DARTS + EH-DNAS	**2.49**	**0.83**	**0.31**	**4.41**	**2.82**	**2.31**

Table 8. Average error rate of approximated hardware performance on different hardware. PP denotes the pipeline paradigm and GP denotes the generic paradigm for customized hardware accelerators

Approach	Average hardware estimation error rate (%)↓					
	Accelerator (GP)	Accelerator (PP)	Edge GPU	Edge TPU	Raspi 4	Pixel 3
FBNet LUT	32.5±0.0	-	65.3±0.0	78.5±0.0	879.7±0.0	890.5±0.0
EH-DNAS	**3.6±0.5**	**7.9±0.1**	**1.9±0.2**	**4.6±1.6**	**38.0±0.6**	**48.4±7.3**

accommodate both pipeline and generic paradigms.

Model design of hardware loss. We examine how the model complexity of our deep hardware loss $\mathcal{L}^{\texttt{deep}}$ influences the hardware performance prediction. We aim for a model with minimum complexity that is sufficient for accurate hardware performance estimation.

A few hyperparameters we considered are (a) embedding size, (b) number of layers, and (c) Type of non-linearity. These hyperparameters are selected following a standard grid search based on the validation set performance. We observe that the estimation error rate increases by 2% with embedding size 50 and increases by 9% with embedding size 2. Increased complexity does not necessarily lead to better estimation. For number of layers, we search over (1,3,5) layers, and found 1 hidden layer to underfit and 5 hidden layers to overfit the training data with error rate of (9.6%, 7.9%, 16.3%) respectively. For non-linearity, RELU is commonly used and empirically, we found it to achieve the lowest error rate (7.9%) compared to Sigmoid (26.7%) and Tanh (73.1%).

Dataset size for training hardware loss. Recall, to train the hardware loss we perform subsampling over all possible architectures. In Fig. 2, we visualize how the size of random samples relates the hardware loss's accuracy. We choose 100K as it is the number of maximum samples that can be collected within a day and achieve adequate prediction performance. As expected, we observe that larger training set size leads to more accurate hardware loss prediction. Importantly, our approach is consistently more accurate than LUT based methods. Our datasaet of size 100K is sufficient to provide an efficient feedback for hardware-aware DNAS.

Hardware and classification performance trade-off. In Fig. 3, we show the trade-off between classification and hardware performance. We obtain different searched architectures by adjusting the hyperparameter β that controls the scale of the hardware loss term. Larger β generally leads to better hardware performance. The experiments are conducted on customized accelerators with the generic paradigm. Note that with a slight compromise of classification accuracy, *i.e.*, less than 1%, we can improve hardware performance by almost twice. This shows the benefit of optimizing hardware-aware metrics. With a proper β, we can find the architecture that meets the classification accuracy requirement with optimized hardware performance.

Searched cells. In Fig. **??**, we visualize the cells found on DARTS search space. Observe that for normal cells, FBNet LUT approach tends to simply reduce the layer complexity to achieve better hardware performance. This potentially explains that LUT-based approach could benefit hardware efficiency yet the simplicity of its linear approximation limits the effectiveness. Note that the cell searched with EH-DNAS has much lower latency (5.81 ms) on ImageNet than FBNet LUT (7.40 ms), even the searched cell has greater layer complexity.

E2E-perf performance. To validate E2E-Perf's accuracy in capturing hardware performance, we compare the estimated hardware performance of the customized accelerators to their measured results from FPGA board-level implementation. As

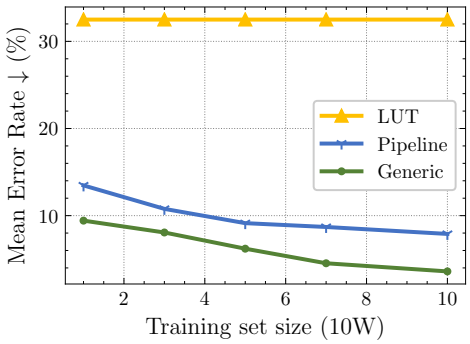


Figure 2. The influence of dataset sample size on hardware performance prediction error rate. We report the error rate for both generic paradigm and pipeline paradigm as comparison to LUT approach

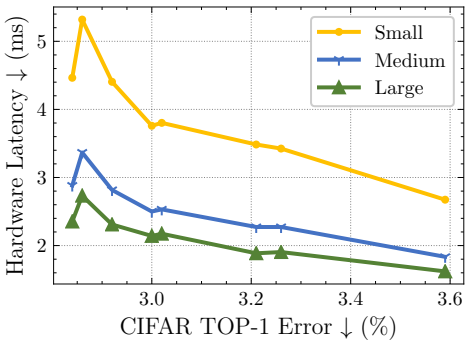


Figure 3. Trade-off between hardware performance and classification accuracy. We report EH-DNAS results on three different hardware budgets on customized accelerator of generic paradigm

Table 9. Estimation errors between the estimated and the board-level performance of customized deep-net accelerators

Estimator / Error	Avg.	Min	Max
AutoDNNchip (Xu et al., 2020)	5.20%	2.12%	7.67%
HybridDNN (Ye et al., 2020)	4.03%	-	-
DNN-chip (Zhao et al., 2020)	-	-	16.84%
E2E-Perf (PP)	1.15%	0.18%	2.26%
E2E-Perf (GP)	2.17%	0.29%	8.65%

shown in Tab. 9, the estimation error introduced by E2E-Perf is 1.15% on average (range 0.18% to 2.26%) for the pipeline paradigm and 2.17% on average (range 0.29% to 8.65%) for the generic paradigm. Compared to the recently published tools (Xu et al., 2020; Ye et al., 2020; Zhao et al., 2020), E2E-Perf provides more accurate performance estimation and significantly improves the hardware feedback quality for guiding the network architecture search in EH-DNAS.

C. Additional results

C.1. Inference with quantization

Approach	Quantized Test Error↓ (%)	Test Error↓ (%)
DARTS + EH-DNAS (PP)	2.90	2.84
DARTS + EH-DNAS (GP)	2.96	2.92

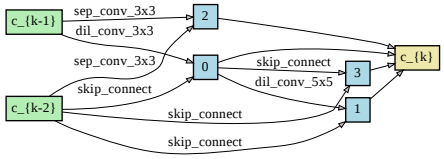
Table 10. Comparison of test error on CIFAR10 with quantization.

Real world mobile and edge applications on hardware usually apply data quantization to ensure hardware efficiency. Fixed-point data format causes less memory footprints and faster computation than the floating-point format. Here, we study the effect of quantization during inference. We quantize all weights, bias, and intermediate outputs in the trained neural network to 16 bits.

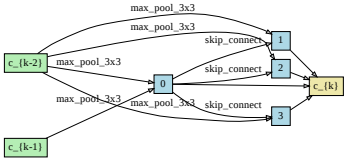
As shown in Tab. 10, quantiziation minimally increases the test error for DARTS + EH-DNAS, by 0.06% for PP and 0.04% for GP, on the CIFAR10 dataset. This indicates that our approach can be successfully applied to real world mobile applications while maintaining the original classification performance.

C.2. Dataset size for training hardware loss

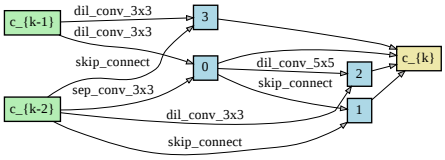
In section Sec. B, we show the results on the influence of dataset size on the average hardware performance prediction error rate. To further show that our chosen dataset size is sufficient, we provide additional results on the maximum prediction error rate, *i.e.*, the maximum error rate on the testing set. In Fig. 2, we observe that our maximum error rate is around 30% when using Dataset of size 100K. Also our approach is consistently lower compared to LUT which leads to 125% maximum error rate.



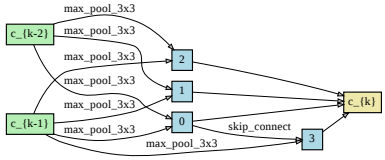
EH-DNAS (GP) searched normal cell



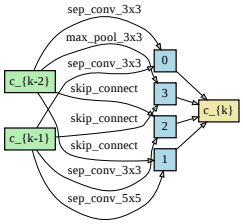
EH-DNAS (GP) searched reduction cell



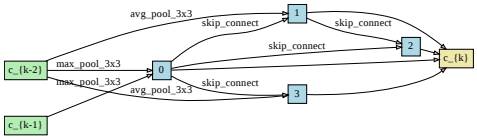
EH-DNAS (PP) searched normal cell



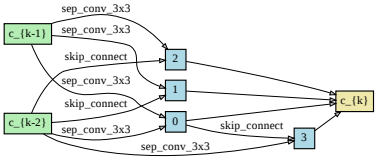
EH-DNAS (PP) searched reduction cell



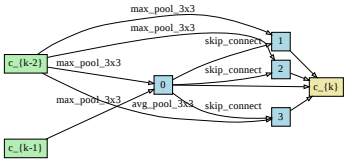
FBNet LUT searched normal cell



FBNet LUT searched reduction cell



DARTS searched normal cell



DARTS searched reduction cell

Figure 4. Cells found on CIFAR10.

C.3. Visualization of Searched Cells

In Fig. 4, we show the final cells found on CIFAR10 by EH-DNAS, FBNet LUT and DARTS. During the searching, we also observe that 1) GP starts to reduce the number of layers in cells earlier than PP, and converges faster for the reduction cell; 2) The searched normal cells are similar, and PP favours more dilated convolution compare to GP; 3) The searched reduction cells tend to be more different in terms of overall architecture, and both paradigms only keep max pooling and skip connect; 4) By increasing the scale of hardware loss term, GP tends to reduce the number of layers in normal cells more aggressively than PP. One reasonable explanation is that NAS is more inclined to directly reduce network layers for hardware performance improvement when targeting accelerators following GP due to its iterative execution nature. Such a trend is not obvious when targeting PP as the design space configuration is individual from each stage.

C.4. Benchmarking Hardware Performance

In order to train the hardware performance prediction loss, we need to acquire hardware performance datasets. Note that EH-DNAS is a general approach that can support any DNAS algorithm and any hardware benchmarking tool. To demonstrate this, we experiment on two types of hardware: customized deep-net hardware accelerators and existing hardware.

Customized deep-net accelerators. We propose E2E-Perf to perform accurate end-to-end benchmarking for customized deep-net accelerators. Compared to existing tools, E2E-Perf is fully automated with direct support to most deep-nets under any hardware budget. It can generate end-to-end hardware metrics instantly after taking an architecture and arbitrary hardware budget, so there is no need to perform manual data collection. It supports customized architecture paradigms. It also comes with a design space exploration engine to optimize hardware configuration following various paradigm-specific

optimization strategies.

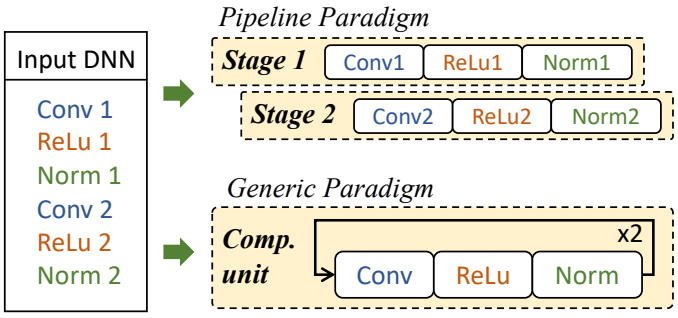


Figure 5. Different paradigms cause different hardware execution patterns even for running the same network

In the first stage, the network definition files and available hardware budgets are passed to E2E-Perf for network model analysis and resource boundary setup. Next, a particular architecture paradigm is selected to continue benchmarking. E2E-Perf supports two popular customized accelerator paradigms, including the pipeline paradigm (Wei et al., 2018; Zhang et al., 2018) and the generic paradigm (Chen et al., 2016; Jouppi et al., 2017; Ye et al., 2020) (Fig. 5). These two paradigms come with their unique optimization opportunities that lead to significantly different hardware design spaces, compute and memory access patterns, resulting in various achievable hardware performance and costs. Those two paradigms are representative of typical customized deep-net accelerators.

In the last stage, E2E-Perf explores the accelerator design spaces and provides paradigm-specific optimization strategies given the input constraints. More details of E2E-Perf are included in the Appendix.

Existing hardware processors. We also adapt the existing hardware processors benchmarking tools for obtaining the hardware performance dataset. These tools, *e.g.*, HW-NAS-Bench (Li et al., 2021), collect the measured hardware performance of all the candidate networks in the search spaces of NAS-Bench-201 (Dong & Yang, 2020) on multiple hardware devices. Such benchmarking tools provide us with desired hardware performance datasets for training differentiable hardware losses.

In this paper, we adopt HW-NAS-Bench (Li et al., 2021) to show the effectiveness of EH-DNAS for four hardware devices, including Edge GPU (NVIDIA TX2), Edge TPU, and two Mobile CPUs (on Raspberry Pi 4 and Pixel 3 mobile phone).

D. E2E-Perf details

E2E-Perf is proposed to perform accurate end-to-end benchmarking for customized deep-net accelerators. Its overall flow is illustrated in Fig. 6. E2E-Perf directly takes a selected network candidate and customized hardware budgets as inputs and automatically generates end-to-end hardware metrics through three major steps: 1) network model analysis, 2) customized architecture modeling, and 3) paradigm-specific optimization.

D.1. Network model analysis

E2E-Perf aims to aid the NAS process with reliable end-to-end hardware performance by considering input workloads, hardware budgets, architecture paradigms, and hardware optimizations. First, network definition files of the selected network candidate are passed to E2E-Perf. Next, a parser captures network layer information (*e.g.*, layer types and configurations) while the profiler calculates the compute and memory demands. Layer fusion and reorganization are then performed to aggregate layers for higher hardware efficiency. For example, lightweight activation layers are aggregated to their neighbouring major layers, which dominate the compute and memory consumption.

In addition to the network model, hardware budgets specified by designers are captured by E2E-Perf to establish resource boundaries when optimizing the customized accelerators. Major hardware budgets include computation resources, on-chip memory, external memory access bandwidth, and working frequency. These are essential factors that affect various metrics in hardware implementation, *i.e.*, customized accelerators' performance, energy consumption, and area overhead.

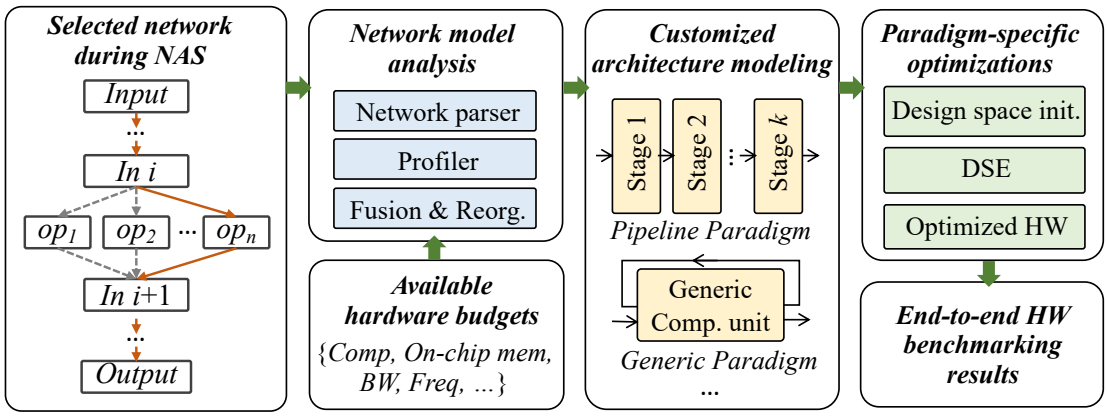


Figure 6. E2E-Perf flow for end-to-end customized accelerator benchmarking

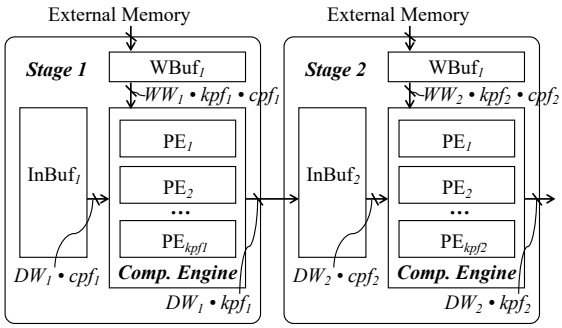


Figure 7. The pipeline paradigm

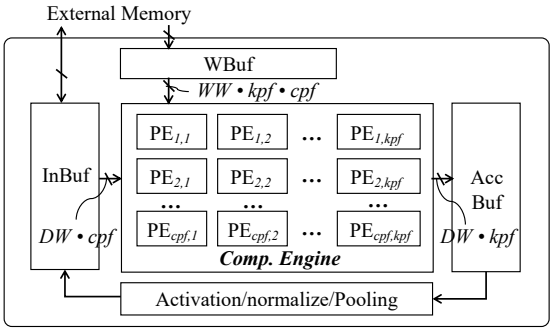


Figure 8. The generic paradigm

D.2. Customized architecture modeling

After network model analysis, a particular architecture paradigm is selected to continue benchmarking. E2E-Perf features multiple paradigm support to enable more precise accelerator architecture modeling as different paradigms combining with their unique optimizations can lead to significantly different hardware design spaces, compute and memory access patterns, resulting in various achievable hardware performance and costs. By investigating recent accelerator designs, we can see two most popular paradigms: 1) a pipeline paradigm with dedicated pipeline stages instantiated on hardware for executing different layers (Wei et al., 2018; Zhang et al., 2018) and 2) a generic paradigm with a shareable compute unit for handling different layers (Chen et al., 2016; Jouppi et al., 2017; Ye et al., 2020).

For the pipeline paradigm in Fig. 7, hardware resources are distributed to each pipeline stage according to the optimization guidelines. Each pipeline stage integrates three types of resources as the compute resources for compute engine, the on-chip memory for input and weight buffers (InBuf, WBuf), and the external memory access bandwidth for loading network parameters from off-chip memory. Customization is reflected in a variety of stage-wise configurable parameters, such as the parallel factors along with the input (cpf) and output (kpf) channel, the data bit-width of feature maps (DW) and parameters (WW). As shown in Fig. 7, kpf process elements (PEs) are instantiated and each PE performs cpf multiply-accumulations (MACs) in parallel with the total parallel factors as $cpf \times kpf$.

Similarly, the generic paradigm also has a rich list of configurable as illustrated in Fig. 8. They include a compute engine with $cpf \times kpf$ PEs, an auxiliary compute unit for activation and pooling layers, and a set of on-chip buffers for keeping part of the input feature maps, network parameters, and intermediate results. All these components are first instantiated on hardware and then shared by all layers of the targeted network, so layers need to be processed in a recurrent manner. According to different data partitioning and scheduling strategies, the customized accelerators can adapt to various dataflows (*e.g.*, input and weight stationary) where hardware metrics can be different.

E2E-Perf supports both above-mentioned paradigms and can be extended to support more special paradigms. According to the selected paradigm, E2E-Perf adopts dedicated analytical models to outline the hardware design space, apply optimization

strategies, and estimate end-to-end hardware performance and costs.

D.3. Paradigm-specific optimizations

Optimizations are essential to construct customized hardware accelerators as they help deliver improved performance given limited hardware resources. To effectively model these optimizations, E2E-Perf features paradigm-specific design spaces to outline all possible accelerator design combinations.

	Pipeline paradigm	**Generic paradigm**
Comp. Eng. config.	$CPF = \{cpf_1, \cdots cpf_i\}$ $KPF = \{kpf_1, \cdots kpf_i\}$ $WW = \{WW_1, \cdots WW_i\}$ $DW = \{DW_1, \cdots DW_i\}$	$CPF = \{cpf\}$ $KPF = \{kpf\}$ WW DW
Arch. config.	*Inter-stage optimization*	*Dataflow*

Table 11. Paradigm-specific hardware design space

In Tab. 11, we list the design spaces for pipeline and generic paradigm, where parameters are divided into two categories for configuring the compute engine and the overall architecture. Regarding the pipeline paradigm, its design space consists of individual configuration from each pipeline stage. With more pipeline stages, we have a higher dimensional design space. Assuming an accelerator with i pipeline stages, we define four i-dim vectors to outline its design space. In addition, architecture-level configurations can also be included to provide inter-stage optimizations, such as the column-based cache published by Zhang *et al*. (Zhang et al., 2018). For the generic paradigm, E2E-Perf adopts a fixed set of parameters as there is only one compute engine instantiated. Due to its iterative execution nature, we also explore different data reuse strategies, such as input and weight stationary (Chen et al., 2017).

E2E-Perf then starts design space exploration (DSE) to explore optimized configurations by considering the targeted architecture paradigm and customized hardware budgets. For a pipeline architecture, E2E-Perf aims to optimize the accelerator's throughput performance by load-balancing the pipeline stages. On the other hand, E2E-Perf optimizes the end-to-end latency of the generic architecture following the main idea to fully utilize available resources. Detailed optimization algorithms for both paradigms are shown in the supplementary material.

E. Implementation details

E.1. For customized hardware accelerators

Hardware loss learning. We uniformly sampled 1,000K, 200K, 200K architectures to form the training, validation, and test sets. Each architecture is evaluated with E2E-Perf under a relatively large hardware budget (4800 DSPs 141Mb on-chip memory, comparable to a mid-range cloud processor) for corresponding hardware performance. We use the embedding model with an embedding size of 10. We train the hardware loss model for 30 epochs with batch size 128 and learning rate 0.0001. We use Adam (Kingma & Ba, 2015) as optimizer and apply weight decay of 0.01. We train four hardware loss in total one for each hardware accelerator design paradigm.

Architecture Search on CIFAR10. We follow DARTS' setup to search for the best cells. We use 8 cells (layers), among which the third and sixth cells are reduction cells, and the rest are normal cells. We use initial channels of 16. We train for 50 epochs with batch size 64. We use SGD with momentum to optimize the model parameter weights with initial learning rate 0.025. We then decay the learning rate with a cosine scheduler, along with momentum of 0.9 and weight decay of 3e-4. For architecture parameters, we use zero initialization and Adam as optimizer with learning rate 3e-4 and weight decay 1e-2.

For our hardware loss $\mathcal{L}_{\texttt{hw}}$, we use β to balance between the hardware performance and task performance. Grid search is performed on β over the range $\{1, 0.1, 0.01, 0.001, 0.005, 0.0001\}$. During the search of 50 epochs, we early stop when hardware performance stops improving. We follow DARTS to use weighted sum of eight operations. To comply with that, embedding of operation choice on one edge is also the weighted sum of eight embedding.

Evaluation on CIFAR10. We follow DARTS' protocol to select and drop edges by keeping the top-2 strongest operations for each node among all non-zero operations from all previous nodes. We then evaluate the hardware performance of the final architecture using E2E-Perf, which has 8 cells and 36 initial channels. Next, we construct a large network of 20 cells and 36 initial channels to train from scratch following DARTS setup. We train 600 epochs with batch size of 96, use

auxiliary head with weight 0.4 and dropout with probability 0.2. The training takes 2 days on a single GPU.

Evaluation on ImageNet. We take the searched cells on CIFAR10 to construct a larger model of 14 cells and 48 initial channels. We evaluate the hardware performance of the constructed model using E2E-Perf. Next, we train the model from scratch for 150 epochs with batch size 512. We use SGD with weight decay 4e-5, and initial learning rate 0.1. Since we use larger batch size compared to DARTS, we increase the learning rate to 0.4 within the first four epochs. We perform grid search over the learning rate schedule; either decaying the learning rate by 0.97 or 0.94. The training takes three days on four NVIDIA V100 GPUs.

E.2. For existing hardware processors

Hardware loss training. We use HW-NAS-Bench (Li et al., 2021) which records the hardware performance of every architecture in NAS-Bench-201 search space (Dong & Yang, 2020; Dong et al., 2021), where there are 5 operations for each of six edges, constructing a search space of $5^6 = 15625$. We use all the architecture for training. We use the embedding model with an embedding size of 20. We train the hardware loss model for 500 epochs with batch size 128 and learning rate 0.001. We use Adam (Kingma & Ba, 2015) as optimizer and apply weight decay of 0.01. We train four hardware loss in total one for each hardware (Edge GPU, Edge TPU, Raspi 4, Pixel 3).

Architecture Search on CIFAR10. We follow the setup in NAS-Bench-201. The architecture includes three stacks of cells, connected by a residual block. Each cell is stacked 5 times, with the number of output channels as 16, 32 and 64 for each stage. GDAS is trained 5 times longer (250 epochs) than DARTS (50 epochs). It is claimed that because at every iteration, GDAS only optimize one fifth of the parameters compared to DARTS.

Evaluation on CIFAR10. NAS-Bench-201 trained every architecture under its search space and collect their performance. The original training set of CIFAR10 is split by half to formulate a validation set. Each architecture is trained via Nesterov momentum SGD, using the cross-entropy loss for 200 epochs in total. The weight decay is 0.0005 and the learning rate decay is cosine annealing from 0.1 to 0. After searching for the cell architecture, we directly report the classification performance on test set provided by NAS-Bench-201.

F. Discussion

Potential negative social impact. We study neural architecture search which is a generic algorithm, hence we do not foresee a direct negative social impact of our work.

Limitations and future work. While our approach demonstrates effective End-to-end HW-aware DNAS, we note that there are still exploring space. For example, alleviating current limitations to cover more hardware metrics as optimization targets, *e.g.*, energy/power consumption, area overhead, and leverage more emerging hardware paradigms. We believe the proposed EH-DNAS opens opportunities for future research in these directions.

G. Code Release

We will public our EH-DNAS implementation as the supplementary material. The repository is organized as follows:

```
e2e_dnas/
├── perf/
    ├── layers/
    ├── conf/
    ├── chip_define.py
    └── ...
├── our_nn/
    ├── main.py
    ├── networks.py
    ├── dataset.py
    └── ...
├── model_zoo/
```

```
    ├── darts.py
    └── DARTS/
        ├── train_search.py
        ├── train.py
        ├── train_imagenet.py
        └── ...
├── util/
├── e2eperf.py
├── e2eperf_loop.py
└── README.md
```

Descriptions:

- `perf/` contains scripts and hardware settings for E2E-Perf.
- `our_nn/` contains scripts to train the neural network to approximate E2E-Perf.
- `model_zoo/` contains scripts to run DARTS with or without our hardware loss.
- `e2eperf.py` contains script to evaluate a single architecture.
- `e2eperf_loop.py` contains script to evaluate a set of architectures.
- `README.md` contains instructions to reproduce our results either from scratch or from our searched architectures in the paper.

We will provide in `README.md` details on installation and reproducing our experiments.