Efficient Accelerator/Network Co-Search with Circular Greedy Reinforcement Learning

Zejian Liu, *Graduate Student Member, IEEE*, Gang Li, *Member, IEEE*, and Jian Cheng, *Member, IEEE*

***Abstract*—Recently, accelerator/network co-search has shown great promise in reducing the complexity of co-design and achieving higher model accuracy. Unlike the manual design methodology, it automatically learns the optimal network architecture and corresponding accelerator under the given constraints. However, prior works do not consider the direct feedback of searched accelerators on the network search in each step, which leads to low converge speed and sub-optimal solutions. To address this issue, we propose DAN, a reinforcement learning-based framework for fast and accurate accelerator/network co-search. The fundamental idea is to model the co-search as interleaved network-aware accelerator search (AS) and accelerator-aware network search (NS) using separate RL agents, which improves the performance of AS and NS, and encourages a tight interaction between AS and NS. Experimental results show that our proposed method consistently outperforms single-agent based method in terms of converge speed and performance.**

***Index Terms*—Accelerator/Network Co-Search, Reinforcement Learning, Performance Estimation, Multi-objective Optimization.**

I. Introduction

In recent years, deep neural networks (DNNs) have made remarkable progress in many artificial intelligence fields [1], [2]. Since most DNNs are computationally intensive, the general-purpose CPUs and GPUs usually can not meet the requirements of different applications in terms of latency and energy consumption, especially for resource-constraint applications. To improve inference efficiency, a spectrum of lightweight models [3], [4] and dedicated hardware accelerators [5], [6] have been proposed. However, a separate methodology that solely optimizes the network architecture or accelerator will degrade the overall performance [7].

Recently, network/accelerator co-search [8]–[11] has been proposed to automatically explore the best architecture of both network and accelerator. Most of the existing co-search frameworks follow the same paradigm, where network architectures and accelerators are sampled from the network and accelerator space respectively and measured with specific metrics periodically, as shown in Algorithm 1. More specifically, the network architecture is first determined according to accuracy-related metrics, then evaluated on a couple of accelerator samples to collect candidates that meet the energy/area/latency requirements. In other words, the co-search process in this paradigm can be viewed as temporal network-aware accelerator search, where the adjustment of the accelerator is explicitly subjected to the network architecture, but the opposite is not. Since

Algorithm 1 Joint Search Method

```
Input: Joint search space $\mathcal{J}$, Search algorithm $f$, Constraint $C$, Maximum search steps $K$
Output: Best Solution $S$
 1: $S$ = {None, None}
 2: for $i$ = 1, 2, ..., $K$ do
 3:     Sample $A$ and $N$ from $\mathcal{J}$ with $f$
 4:     Evaluate $A$ and $N$ to get scores
 5:     Update $f$ and $S$ with scores
 6: end for
 7: return $S$
```

Algorithm 2 Proposed Search Method

```
Input: Joint search space $\mathcal{J}$, Search algorithm $f_N, f_A$, Constraint $C$, Maximum search steps $T, K_N, K_A$
Output: Best Solution $S$
 1: $S$ = {$A$, None}
 2: for t = 1, 2, ..., $T$ do
 3:     // Accelerator-aware network architecture search
 4:     Define subspace $\mathcal{J}_N$ by fixing the accelerator
 5:     Search $N$ in $\mathcal{J}_N$ with $f_N$ for $K_N$ steps
 6:     // Network-aware accelerator architecture search
 7:     Define subspace $\mathcal{J}_A$ by fixing the network
 8:     Search $A$ in $\mathcal{J}_A$ with $f_A$ for $K_A$ steps
 9: end for
10: return S
```

immediate hardware feedback is not explicitly involved in network search in each search step, the above method is prone to cause an unstable and slow converge.

Recall that in the manual co-design, the network architecture and accelerator are essentially refined in an interleaved manner, rather than designing from scratch in every step. For example, if the accelerator tailored for a specific network does not meet the performance requirement, a slight adjustment can be performed by adding more MACs to the compute core. And once the accelerator meets the design goal, it can be used as the target hardware, in turn, to refine the network to achieve higher accuracy. Motivated by this methodology, in this paper, we propose a new co-search method, as illustrated in Algorithm 2. We divides the original problem into two sub-problems: accelerator-aware network search (NS) and network-aware accelerator search (AS), and solve the two sub-problems alternately. This method has two merits. First, the search space of each sub-problem is a subspace of the joint search space. Since the subspace is much smaller

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received April 19, 2021; revised August 16, 2021.

than the joint space, the search complexity is significantly reduced. Second, it can make better use of the search results of intermediate steps, because the search algorithm tries to obtain better solutions by only adjusting the accelerator or network of previously searched.

To demonstrate the effectiveness of the proposed method, we design a RL-based co-search method named DAN. It consists of **D**ual agents (RL controller), where the **A**-agent is responsible for optimizing the network architecture for a given accelerator, and the **N**-agent optimizes the accelerator to reduce latency, energy consumption, area, and other metrics. Unlike the single-agent method, the two agents in DAN are designed for different optimization objectives using different reward functions, which has the potential to improve the representation ability.

Experiments show that our approach converges faster and can obtain significantly better search results than the single-agent RL-based method. On the ImageNet, compared with the baseline method, the searched solution of our approach achieves up to 1.8% higher accuracy together with similar or lower EDAP (Energy-Delay-Area-Product). Our method also achieves nearly 2× real speed up.

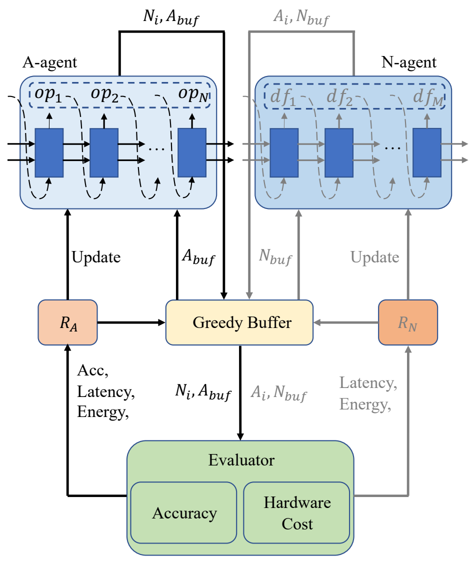


Fig. 1. An overview of our proposed co-search framework. As the A-agent and N-agent will be solved alternately, we use gray to indicate that the N-agent is not currently running. The target accelerator for the A-agent is A_{buf} provided by the greedy buffer, and the A-agent outputs a candidate network N_i. The accuracy and hardware cost are measured by evaluators.

II. Related Work

Neural architecture search (NAS) [12], [13] aims to automatically design models to achieve competitive accuracy. Apart from the accuracy, the deployment of a model also needs to meet the constraints of latency or power consumption. Hardware-aware NAS methods [3], [4] are proposed, which aim to search a network architecture for a target device to achieve a trade-off between accuracy and hardware performance. Though hardware-aware NAS considers the relation between the hardware and model, it does not explore the hardware design space, which can not achieve optimal inference efficiency. Therefore, accelerator/network co-search [8], [10] has been proposed and has shown appealing results. NHAS [14] jointly searches the quantized network architecture and design parameters of a given accelerator. NASAIC [10] is an RL-based method that can simultaneously identify multiple ASIC accelerator designs for one network, which is similar to our method. However, they do not search multiple networks for one accelerator, and their method still uses one reward function and can not be updated independently. In other fields, there are also some works [15], [16] using two agents, but the nature of the problems solved by these works is different from that of co-search, and most of them do not exist interdependence between subproblems.

III. Proposed Method

A. Overview

Generally, the optimization problem of accelerator/network co-search can be formulated as:

$$\max_{(A,N)\in\mathcal{J}} P(A,N) \tag{1}$$

$$s.t. \quad MeetConstrains(A,N) = True \tag{2}$$

where A represents accelerator and N represents network.The joint search space is denoted as $\mathcal{J}$. P is the objective function, such as maximizing accuracy and minimizing latency. $MeetConstrains(A,N) = True$ ensures that the pair of A and N meets the design constraints, such as accelerator area and energy consumption. To solve this problem, most existing works design search algorithms that directly operate on the joint search space. In each search step, the search algorithm first samples design points from the joint space. Then the sampled networks and accelerators are evaluated according to specific measurements to get the accuracy, latency, area and energy consumption. Finally, the search algorithm updates its parameters and continues to perform the following search.

Motivated by the manual co-design methodology, in this paper, we model the accelerator/network co-search as a combinational search process, where the accelerator and network search are conducted in an interleaved style. Specifically, we divide the co-search problem into two sub-problems: hardware-aware network search and network-aware accelerator search. Based on the greedy strategy, the current best solution of one sub-problem serves as the prior of the other sub-problem in the next step. Accordingly, we design a dual-agent RL-based accelerator/network co-search framework, as shown in Figure 1. It consists of two RL agents. The A-agent is used to search networks for a given accelerator, and the N-agent is used to search accelerators for a given network.

B. RL Agent

The RL agent is responsible for finding the optimal solution from the search space. The N-agent and the A-agent has the same architecture. Each agent consists of an actor network and a critic network, both of which are composed of two fully connected layers and one LSTM layer with a hidden

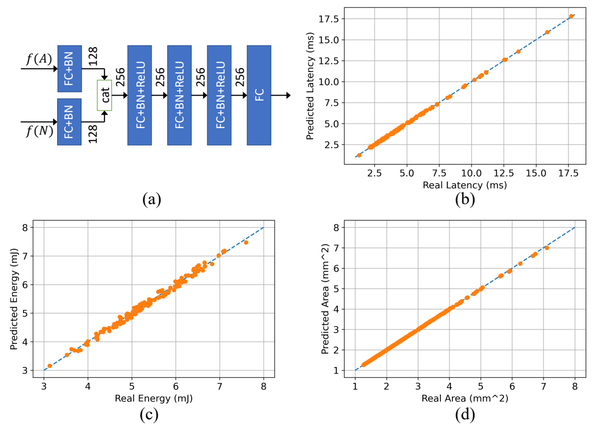


Fig. 2. (a) The architecture of the predictors; (b) MSE for latency prediction is 3.6e-2; (c) MSE for energy prediction is 8.1e-2; (d) MSE for area prediction is 1.4e-4.

size of 64. The actor network generates the configuration of the accelerator or network, and the critic network predicts the expected reward to help the training of the actor network.

C. Reward Function

For each searched solution, the reward function translates its performance metrics (e.g., accuracy, latency) to a reward value, which guides the optimization of the RL agent. The N-agent aims to reduce latency, energy consumption, and chip area by optimizing the accelerator configuration, while the A-agent optimizes the network architecture to maximize the model accuracy and minimize the latency and energy consumption. Note that the N-agent can not influence the model accuracy, and the A-agent can not influence the chip area. As the optimization objective is different, we should design different reward functions for each agent:

$$R = Acc \cdot \left(\frac{Latency}{T_{latency}}\right)^{w_0} \left(\frac{Energy}{T_{energy}}\right)^{w_1} \left(\frac{Area}{T_{area}}\right)^{w_2} \quad (3)$$

$$w_i = \begin{cases} p, & \text{meets the constraint} \\ q, & \text{otherwise} \end{cases} \quad (4)$$

where w_0, w_1, w_2 are three hyperparameters that can be tuned to control the trade-off between different metrics, and R_N (for N-agent) and R_A (for A-agent) have their own w_i. The effect of different p and q are discussed in [17].

D. Performance Estimation

The most accurate method for obtaining the accuracy is training the network from scratch. However, it is impractical for co-search that generally needs to evaluate thousands of networks. In this work, we use the weight-sharing based method [18] to get the accuracy of each network quickly. Specifically, we train a supernet first. Then the derived network will directly inherit the parameters of the supernet and be evaluated on the validating dataset to get the prediction accuracy. In order to avoid the influence of the gap between the prediction accuracy and the actual accuracy on the experimental analysis, we report the prediction accuracy in all experiments.

TABLE I
THE SEARCHED DESIGN FACTORS OF EYERISS.

Parameter Name	Potential Values
PE_x	6, 8, 10, 12, 14, 16, 18, 20, 24
PE_y	6, 8, 10, 12, 14, 16, 18, 20, 24
Ifmap_spad	5, 8, 12, 16, 20, 24, 28, 32, 36
Weight_spad	16, 32, 64, 96, 128, 160, 192, 224, 256
Psum_spad	2, 4, 8, 12, 16, 20, 24, 28, 32

For hardware performance, TimeLoop [19] and Accelergy [20] are commonly used state-of-the-art evaluators. They can model different accelerators and evaluate the performance of different workloads. However, the simulation process often takes a long time. We design and train multi-layer perceptron (MLP) based predictors to speed up the estimation. The network architecture is shown in Figure 2(a). $f(A)$ is the design factors of the accelerator, and $f(N)$ is the feature vectors of the network architecture, which consists of the size of the input feature map, the kernel size, and other features. Because the latency and energy of different layers running on different accelerators are very different, the training of predictors that directly predict the latency and energy is extremely unstable. Considering that the FLOPs (floating point operations) of the target layer and the number of computing units of the target accelerator are known, we predict the energy consumption per FLOP and the utilization of computing units and then convert them into the actual latency and energy consumption. We experimentally use MAPE (Mean Absolute Percentage Error) loss for training evaluator networks. Compared with TimeLoop+Accelergy, MLP-based predictors achieve nearly 4,000$\times$ speed improvement (0.05s vs. 207s).

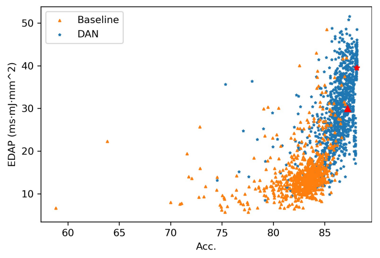
IV. EXPERIMENTS

A. Evaluation Environment

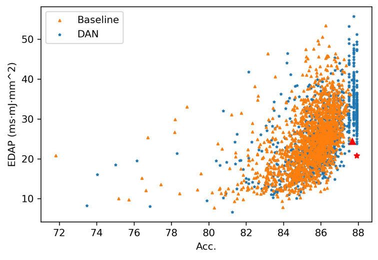
a) ***Search Space.:*** For co-search, the search space is a combination of network space and accelerator space. Follows [11], we adopt the search space of FBNet as this space is more hardware friendly [4].The network has 22 layers, and each layer has seven candidate blocks: k3_e1, k3_e3, k3_e6, k5_e1, k5_e3, k5_e6, and skip. We drop two candidate blocks (k3_e1_g2 and k5_e1_g2) from the original search space because these blocks contain the channel shuffle operation, which is hardware inefficiency. For accelerator architecture search, we choose the Eyeriss [5] as the template. The searched design factors of the accelerator are shown in Table I.

b) ***Settings for Evaluators:*** For the training of the supernet, we follow the method in [18] that uniformly samples a child network from the supernet and trains it in each step. We train the supernet for 300 epochs with an SGD optimizer and a cosine learning rate scheduler. The batch size is 128, and the initial learning rate is 0.1. We use a weight decay of 0.00004 and a momentum of 0.9.

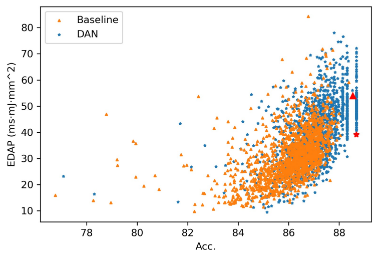
To train the MLP-based predictors, a dataset of 100k samples is built by using TimeLoop and Accelergy. CACTI7 [21] and Aladdin [22] at 45nm technology are used as energy estimation plug-ins. We train the predictors using an Adam optimizer with a learning rate of 0.001 for 600 epochs, and the batch size is 256.



(a) Latency-constrained



(b) Energy-constrained



(c) Balance-constrained

Fig. 3. Sample points of DAN and the baseline method under different constraints. Samples that do not meet the constraints are removed.

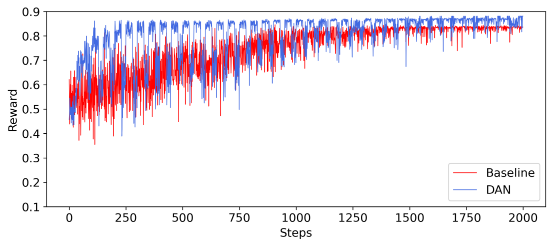


Fig. 4. Search convergence of DAN and baseline method.

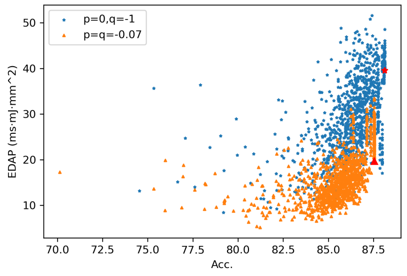


Fig. 5. Effect of w. The constraint is latency-constrained.

*c) **Settings for RL:*** We use PPO [23] as the reinforcement algorithm. All agents are trained by an Adam optimizer, and the initial learning rate is 0.0003 for the actor network and 0.001 for the critic network. For DAN, the search process consists of 40 loops, and the A-agent and N-agent will search 30 samples and 20 samples in each loop, respectively. The total number of search steps is 2000. Similar to the settings in [7], we set different constraints to search different solutions: (3ms, 9mJ, 3mm^2) for latency-constrained, (10ms, 4mJ, 3mm^2) for energy-constrained, and (5ms, 6mJ, 3mm^2) for balance-constrained. All experiments run on a server with an Intel Xeon GOLD 5220 CPU and one RTX2080Ti GPU.

*d) **Baseline Method:*** The baseline method is also an RL-based co-search framework. However, it only has one agent, which directly predict the network and accelerator in each step. The architecture of the agent is the same as DAN's. It also only has one reward function, and we use $p = 0$ and $q = -1$.

B. Results on CIFAR-10

*a) **Search Convergence:*** To demonstrate that DAN can converge, we plot the curve of the reward of the baseline method and our proposed method in Figure 4. Due to the instability of RL training, training runs may have large performance differences, so we run each experiment 5 times with different seeds and report the average result. In order to ensure the fairness of the comparison with the baseline method, we also use $p = 0$ and $q = -1$ for R_N and R_A. We see that both methods gradually find solutions with higher reward scores. DAN converges faster than the baseline method, and the final reward score is higher. Since the time required to obtain the accuracy is much longer than that to obtain the hardware performance, the DAN's time cost is about half of the baseline method's, because the number of models that DAN needs to evaluate is less than that of the baseline method (1200 vs. 2000).

TABLE II
EFFECT OF INTERLEAVING HYPER-PARAMETERS: THE NUMBER OF LOOPS (#L), A-AGENT'S STEPS (#A), AND N-AGENT'S STEPS (#N).

#L,#A,#N	Acc. (%)	Lat. (ms)	Ene. (mJ)	Area (mm^2)	EDAP (ms·mJ·mm^2)
1,1200,800	86.5	3.0	2.6	2.0	**15.7**
	87.8	4.5	3.8	1.8	29.8
	88.2	2.8	5.1	2.8	40.2
100,10,10	87.8	2.8	5.0	2.6	35.9
	87.7	4.4	4.0	2.1	36.7
	88.5	3.3	5.5	2.6	46.2
40,30,20	**88.1**	2.9	5.3	2.5	39.5
	87.9	3.0	3.8	1.8	**20.8**
	88.7	4.2	4.9	1.9	**39.0**

*b) **Pareto Frontier:*** Figure 3 shows the distribution of searched samples. Points located towards the bottom right are Pareto-optimal. We see that both methods gradually converge to the region with a better trade-off between accuracy and EDAP. The optimization objective is to find the solution with the highest accuracy while meeting given constraints, and DAN is able to find many points with higher accuracy than the baseline. What's more, some searched points of DAN form several vertical lines, which corresponds to the optimization of the N-agent that only optimizes accelerator architecture and does not affect the network accuracy. This optimization process helps the framework avoid missing potential solutions.

*c) **Effect of** w:* Figure 5 shows the effect of w in the reward function. When $p = 0$ and $q = -1$, the RL agents will focus on improving the accuracy if the solution has met the performance constraints. On the contrary, when $p = q = -0.07$, the RL agents will still try to improve the hardware

TABLE III
COMPARISON OF EXISTING CO-EXPLORATION ALGORITHMS.

Alg.	Method	Space	Acc.	EDAP	GPU
[7]	Gradient	$4.8e10$	93.23%	7.4e6	3h
			94.00%	6.2e6	
			93.73%	8.3e6	
ours	RL	$2.3e23$	93.96%	7.9e6	2.5h
			93.46%	4.2e6	
			94.21%	7.8e6	

TABLE IV
COMPARISON OF DIFFERENT SEARCH METHODS ON THE IMAGENET.

Search Approach	Acc. (%)	Lat. (ms)	Ene. (mJ)	Area (mm^2)	EDAP ($\text{ms}\cdot\text{mJ}\cdot\text{mm}^2$)
Baseline$_L$	53.9	9.0	16.6	2.9	**430.2**
DAN$_L$	**55.7**	9.8	17.3	2.9	497.6
Baseline$_E$	54.2	13.2	12.8	2.0	343.5
DAN$_E$	**54.5**	8.5	12.1	2.2	**224.4**
Baseline$_B$	55.2	9.6	14.1	2.5	336.0
DAN$_B$	**55.6**	8.5	15.0	2.4	**300.8**

performance after meeting the constraints because it is still possible to increase the reward, even if it may result in an accuracy drop. As a result, the former has higher accuracy, while the latter has a lower EDAP.

*d) **Effect of interleaving hyper-parameters**:* The interleaving hyper-parameters include the number of loops, the A-agent's search steps, and the N-agent's search steps. Table II shows the results with different hyper-parameters. The performance of (1,1200,800) is worst because it only searches one loop and cannot search the network for different accelerators. The accuracy of (100,10,10) is similar to that of (40,30,20). However, the hardware performance of the former is relatively poor. The main reason is that the searched accelerator architecture is more likely suboptimal due to the limitation of search steps, but the accelerator architecture significantly impacts hardware performance.

*e) **Compared with SOTA**:* Compared with DANCE [7], which is a differential-based method and also uses Eyeriss as the template accelerator, our method achieves similar or better performance under different constraints. In terms of search efficiency, our approach costs 2.5 hours, which is slightly less than that of DANCE. However, our method's search space is much larger than DANCE's. The larger the search space, the more difficult it is to find the best solution.

C. Results on ImageNet

Finally, we provide the experimental results on the ImageNet in Table IV. Our method still discovers better solutions than the baseline method, proving that DAN is suitable for different datasets. Compared with the baseline method, our method consistently achieves higher accuracy. Apart from latency-constrained, DAN achieves both higher accuracy and lower EDAP under energy-constrained or balanced-constrained. For latency-constrained, the EDAP of our method's solution is higher than the baseline, but it also improves accuracy by 1.8%.

V. CONCLUSION

Most previous works follow the same paradigm that directly samples a pair of accelerator and network from a pre-defined joint search space in each step. However, this paradigm suffers from the problem of low converge speed because it does not consider the direct feedback of searched accelerators on the network search in each step. In this work, we model the co-search as interleaved network-aware accelerator search and accelerator-aware network search. We propose a dual-agent RL-based co-search method to demonstrate the efficiency of the proposed method. Compared to the single-agent method, our method achieves better results under the same search steps.

REFERENCES

[1] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[2] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.
[3] H. Cai *et al.*, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.
[4] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *CVPR*, 2019, pp. 10 734–10 742,.
[5] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.
[6] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1–12.
[7] K. Choi *et al.*, "Dance: Differentiable accelerator/network co-exploration," in *DAC*, 2021, pp. 337–342.
[8] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *TCAD*, vol. 39, no. 12, pp. 4805–4815, 2020.
[9] Y. Lin *et al.*, "NAAS: neural accelerator architecture search," in *DAC*, 2021, pp. 1051–1056.
[10] L. Yang *et al.*, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *DAC*, 2020, pp. 1–6.
[11] Y. Fu *et al.*, "Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators," in *ICML*, 2021, pp. 3505–3517.
[12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2016.
[13] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018, pp. 8697–8710.
[14] Y. Lin *et al.*, "Neural-hardware architecture search," *NeurIPS WS*, 2019.
[15] Y. Wu *et al.*, "Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices," in *DAC*, 2020, pp. 1–6.
[16] R. Lowe *et al.*, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *NeurIPS*, vol. 30, 2017.
[17] Y. Zhou *et al.*, "Towards the co-design of neural networks and accelerators," in *MLSys*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 141–152. [Online].
[18] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," in *ECCV*, 2020, pp. 544–560.
[19] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *ISPASS*, 2019, pp. 304–315.
[20] Y. N. Wu *et al.*, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *ICCAD*, 2019, pp. 1–8.
[21] R. Balasubramonian *et al.*, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *TACO*, vol. 14, no. 2, pp. 1–25, 2017.
[22] Y. S. Shao *et al.*, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ISCA*, 2014, pp. 97–108.
[23] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.