Towards Best-effort Approximation: Applying NAS to General-purpose Approximate Computing

Weiwei Chen[1,2], Ying Wang[1], Shuang Yang[1,2], Chen Liu[1], Lei Zhang[1]
[1]Institute of Computer Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
Email: {chenweiwei, wangying2009, yangshuang2019, liucheng, zlei}@ict.ac.cn

***Abstract*—The design of neural network architecture for code approximation involves a large number of hyper-parameters to explore, it is a non-trivial task to find an neural-based approximate computing solution that meets the demand of application-specified accuracy and Quality of Service (QoS). Prior works do not address the problem of 'optimal' network architectures design in program approximation, which depends on the user-specified constraints, the complexity of dataset and the hardware configuration. In this paper, we apply Neural Architecture Search (NAS) for *searching* and *selecting* the neural approximate computing and provide an automatic framework that tries to generate the best-effort approximation result while satisfying the user-specified QoS/accuracy constraints. Compared with previous method, this work achieves more than 1.43x speedup and 1.74x energy reduction on average when applied to the AxBench benchmarks.**

***Keywords*—Approximate computing, NAS, Energy efficiency.**

I. Introduction

Neural acceleration has shown great potential of energy-efficiency boost in general-purpose approximate computing systems by replacing the compute-intensive but error-insensitive programs with neural networks (NNs) [1,3,4]. In General, Neural-network-based acceleration [4] is a cross-stack method that takes advantages of both powerful neural networks and specialized hardware for program approximation. However, the choice of neural network architecture has a significant impact on the output quality and the performance of the approximate computing programs, because the changes of network layer and neuron numbers directly determine the computation overhead and the output accuracy of the original workloads.

Conclusively, an ideal neural approximation solution has to customize a balanced NN architecture to the trade-off between accuracy and performance. Customizing such a neural architecture must be aware of several critical aspects. First, the network must be fast enough to meet the QoS constraints. An energy-efficient solution should be able to make the best use of margin offered by application constraints, i.e., searching for the most lightweight neural network that has sufficient accuracy and speed. Second, the selection of the NN must be aware of the target hardware architecture it runs on. However, it is impractical for the analytically model to evaluate the generalization performance of the candidate solutions in the search space. It is also very expensive to evaluate the network performance via repetitive training, validation and simulation. The exhaustive search policy can only search a very limited design space. We need to effectively reduce the overhead of design evaluation.

This paper proposes a NAS framework for automatic generation of the best-effort neural approximate computing solution to save the manual effort. The major contributions of this work are as follows:

- We formalize the problem of 'optimal' neural acceleration architecture design and apply NAS to the implementation of a general-purpose approximate computing. This method tries to generating the best-effort program acceleration architecture to improve the energy efficiency and supports user-specified constraints of QoS and output quality simultaneously.
- We introduce Sequential Model-based Algorithm Configuration (SMAC) into our NAS algorithm and turn it into a zero-order optimization problem. As the solutions in the search space are expensive to evaluate, SMAC emphasizes on being judicious in selecting the NN for evaluation, reducing the chance of unnecessary model estimation.
- We build the simulation system to evaluate the framework on CPU+NPU platform. Including offline SMAC-based neural network search, and the online dynamic NN selector according to the constraints. Our method has achieved more than 1.43x speedup and 1.74x energy reduction on average when compared to the conventional method.

II. Related works and motivation

Approximate computing is a novel paradigm for aggressive performance and energy boost at the cost of output quality [1, 3]. Esmaeilzadeh et al. [4] first propose Neural-network-based acceleration that trains a neural network to replace the original compute-intensive function at software-level and deploys the pre-trained network to NPU for acceleration at hardware-level as in Figure 1. Neural acceleration is useful for its generality with typically workflow of neural acceleration consists of three parts: *observation*, *selection* and *binary generation*. Most neural-acceleration schemes [1, 3] select the 'best' NN approximator in terms of error or quality performance from a small group of candidates, or manually design a neural network for a specified approximate application. A small search space cannot promise high output quality or inference latency when computing on unseen data; while the manual design is inefficient and heavily dependent on the human experience. Besides, previous works are device-agnostic that ignore the hardware characteristics.

The recent development of NAS provides one possible solution to reduce the human effort in the neural networks design. Approaches for NAS can mainly be categorized into three branches: Evolutionary algorithm (EA), Reinforcement learning (RL) [6] and Hyper-parameters optimization. EA provides a simple mechanism to explore the space of architectures by making a sequence of changes to networks that are already evaluated, however, they are not ideally suited for optimizing functions that are expensive to evaluate. While RL methods have seen recent success in different tasks, but RL-based approaches need to try a large number of architectures to find the optimum. This is not desirable, especially in computation-power constrained settings.

To automatically deliver the best-effort NN architecture that

This work was supported in part by National Natural Science Foundation of China (No. 61902375).

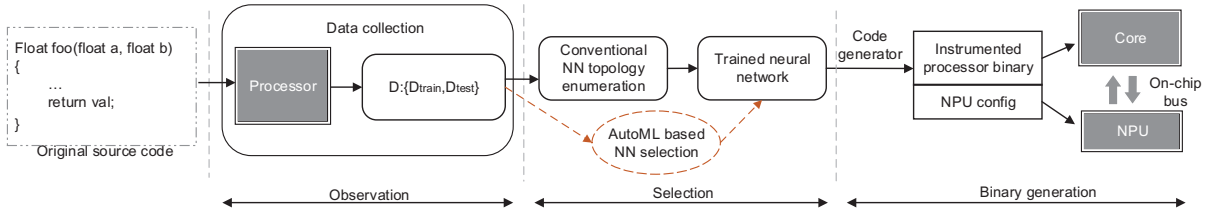


Figure 1. An overview of neural acceleration workflow for general-purpose approximate computing. We replace the conventional NN topology enumeration or manual design method with the SMAC-based NN selection, which frees the human labor for automatically NN design as minimum evaluations attempt as possible.

considers the target hardware characteristics mainly faces two critical challenges: (1) *Huge search space.* For a multi-layer perceptron (MLP) with *L* layers, *P* activation function candidates, and at most *M* neurons in each layer, an exhaustive search has exponential time complexity $O(P*M^L)$; (2) *Expensive evaluation*. We need to perform training, validation, and hardware simulation for every candidate NN. To address the two challenges, we adopt SAMC based NAS that considers the hardware characteristics to improve search efficiency, as shown in Figure 1. The SMAC is one of the Hyper-parameters optimization methods that typically used in setting where a function *f* is expensive to evaluate; it uses Bayesian models for *f* to infer function values at unexplored regions and guide the selection of points for future evaluations [7]. So that it can perform NN selection efficiently in a large search space at affordable evaluations cost. It is usually more complicated to determine future points than other methods, but this pays dividends when the evaluations are very expensive.

III. NAS FOR REAL-TIME PROGRAM APPROXIMATION

A. Problem definition for energy-efficient neural acceleration

Our goal is to automatically select the NN λ that has the maximize performance score f_{perf} in the search space χ while meeting the user-specified constraints $C:\{C_{accuracy}, C_{speedup}, C_{energy}\}$, including accuracy, speed and energy constraint on the target hardware. The NN selection problem can be written as:

$$\lambda^* = \underset{\lambda \in \chi}{\operatorname{argmax}}\ f_{perf}(\lambda, C, D) \tag{1}$$

where *D* is dataset for training and validation, which is application specific. In this work, we also aim at reducing the search cost, in other words, if λ_t is the NN evaluated at time *t*, we want $f_{perf}(\lambda^*) - \max_{t \le n} f_{perf}(\lambda_t)$ to vanish fast as the number of evaluations $n \to \infty$.

Thereby, we introduce our SMAC-based NAS method to achieve this goal, the whole procedure is two-stages: (1) *offline search stage*, aims to collect a set of neural networks candidates for a specific approximate application as "Pareto optimality"; (2) *online selection stage*, selects the most energy-efficient network from the "Pareto optimality" candidates set that meet the dynamically-assigned resource or user-specified constraints.

B. NAS algorithm for minimizing cost

We use MLP for program approximation, as previous works [3,4]. The λ that present an MLP can be denoted as:

$$\lambda = (L, n_0, A_0, n_1, A_1 \ldots n_L) \tag{2}$$

L is the most number of layers, n_i is the number of neurons and A_i is the activation function at the i^{th} layer.

The NN's performance cannot be directly expressed with the hyper-parameter λ, because either accuracy or speed is decided by many complicated hardware and algorithm factors. We make the NN selection as a zero-order optimization problem.

SMAC is proposed in [7] that is one powerful method for the global optimization of expensive black-box function. It iterates into three phases at time *t*: (1) fit a regression model M_λ to the $< \{\lambda_i\}_{i=1}^{t-1}, \{f_{perf}(\lambda_i)\}_{i=1}^{t-1} >$ pairs collected so far; (2) select a promising candidate λ_t to evaluate next by predicting the function value at arbitrary input scope $\lambda \in \chi$ through a so-called acquisition function $a(\lambda, M_\lambda)$; (3) evaluate the function at λ_t. In this work, we use *random forests* to act as the regression model M_λ, since it trends to perform well with discrete and high-dimensional input data [7]. As the M_λ constructs a conditional probability $p(f_{perf} \mid \lambda)$; we use Gaussian distribution $N(\mu_\lambda, \sigma_\lambda^2)$ to define the acquisition function, where μ_λ is the predictive evaluation means and σ_λ^2 is variance. We then calculate $\lambda_t = \underset{\lambda \in \chi}{\operatorname{argmax}}\ a(\lambda, M_\lambda)$ through expected improvement $E_{M_\lambda}(\lambda)$, which is computed by the closed-form expression:

$$E_{M_\lambda}(\lambda) = \sigma_\lambda \cdot [\mu \cdot \varphi(\mu) + \psi(\mu)] \tag{3}$$

where $\mu = {f_{best} - \mu_\lambda}/{\sigma_\lambda}$, f_{best} is the highest performance score of all NNs measured so far, φ and ψ denote the cumulative distribution function and the probability density of a standard normal distribution, respectively. SMAC greatly improving the sampling efficiency, we can obtain good results with no more than 500 evaluations in our experiment.

As the metric function f_{perf} has a direct impact on the search results, we have designed a mixed metric function. The metric function f_{perf} is defined as follows:

$$f_{perf} = \alpha \times P_{acc} + \beta \times P_{speed} + \varepsilon \times P_{energy} \tag{4}$$

P_{acc}, P_{speed}, P_{energy} are the approximation accuracy, speed, energy consumption, respectively. $\alpha, \beta, \varepsilon$ are the corresponding coefficients. Adjusting the $\alpha, \beta, \varepsilon$ values can guide the search toward different target region. Higher α guides the search towards higher accuracy, higher β prompts higher inference speed, and higher ε leads to higher energy reduction.

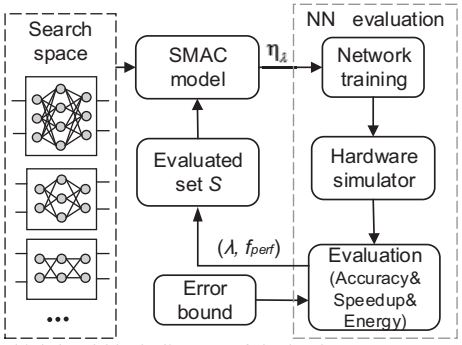


Figure 2. A high-level block diagram of the hardware-aware search pipeline. The SMAC automatically search over the design space with hardware feedback in the exploration loop.

C. Hardware-aware network search pipeline

In our design, we employ a hardware simulator to generate direct feedback of the performance evaluation. Figure 2 shows a high-level block diagram of the hardware-aware search pipeline. As we can see, the NN evaluation procedure is hardware-aware as it simulates the real hardware characteristics. The hardware feedback benefits the NAS algorithm to find the most energy-efficient NN on the target hardware.

It's worth noting that the accuracy indicators for training and evaluation procedure are different. The training procedure uses the MSE to measure the fitting error in model convergence. In the evaluation, we use the error bound mechanism. For the k^{th} test sample x_k with precise output y_k, the error is calculated based on the application-specific quality metric Err. e.g., *image diff* for image processing applications. We label x_k as correctly sample and set $z_k = 1$ if $Err(\eta(x_k) - y_k) < error\ bound$ otherwise $z_k = 0$. The whole evaluation accuracy is the proportion of the correctly samples over the total samples M:

$$P_{acc} = accuracy = \sum_{k}^{M} z_k \Big/ M \tag{5}$$

D. Online network architecture selection

The online stage dynamically selects the neural network from the "Pareto optimality" candidates set S_{pare}. Generally, a higher accuracy solution will need a more complex network; while low-latency and low-energy solution have to accept the drop of accuracy. The online dynamic selection supports three modes, which are shown in Table 1.

TABLE 1. Three modes of the online dynamic selection

Mode	Description
Quality mode	Returns the NN that has the highest accuracy
Energy mode	Returns the NN that has the lowest energy consumption and meets the accuracy threshold
Integrated mode	Chooses the NN that has highest comprehensive score relevant to the user-specified constraints,

In the Integrated mode, we only consider the energy and accuracy to form the comprehensive score. Eq.6 calculates the $\lambda*$ that has the highest comprehensive score.

$$\underset{\boldsymbol{\lambda} \in S_{pare}}{\operatorname{argmax}} [\theta * (\eta_{\text{acc}} - C_{\text{accuracy}}) + (1-\theta) * (\eta_{\text{eer}} - C_{\text{energy}}) / \mathrm{E}_{\text{norm}}] \tag{6}$$

θ is a user-tunable coefficient. η_{acc} is accuracy, η_{eer} is the energy measurement of the candidate NN. E_{norm} is the normalization coefficient of the energy consumption.

IV. Experimental Results

A. Experiment setup

We build a CPU+NPU SoC simulation platform to compare our NAS algorithm with the previous method. We use a modified version of *nn_dataflow* to measure the run time and energy. We use the *Eyeriss* [5] NPU that operates at 1GHz. For the CPU part, we use *MARSSx86* and *Mcpat* for CPU evaluation. The core is based on the *Penryn* microarchitecture and operates at 3.4 GHz. In this paper, we name the previous method as 'fixed topology enumeration' method used in [4]. Same as previous works [3,4], we also choose the AxBench [3] as the benchmarks. Table 2 shows a brief description of these benchmarks. We set two groups of error bounds to evaluate the search performance under different output quality specification. A smaller error bound requires a higher approximation quality, which always needs a more complex network to perform at the corresponding accuracy level.

TABLE 2. The benchmarks used in the evaluation

Benchmark	Domain	Error bound		Error metric
		Group 1	Group 2	
blackscholes	Financial Analysis	0.01	0.015	Relative Error
fft	Signal Processing	0.0001	0.001	Relative Error
inversek2j	Robotics	0.01	0.05	Relative Error
jpeg	Compression	0.002	0.005	Pixel Diff
kmeans	Machine Learning	0.05	0.1	Output Diff
sobel	Image Processing	0.01	0.05	Pixel Diff
jmeint	3D Gaming	-		Classification rate

* jmeint uses the classification rate as the error metric; thus no error bound is needed.

B. Offline search procedure

Previous 'fixed topology enumeration' method restricts the search of NNs architectures with two hidden layers and limits the number of neurons of every layer to powers of two [4]. In our experiments, we have implemented the 'fixed topology enumeration' method and compared with our NAS based method. For fair comparison, we also set the MLP in the search space to have at most two hidden layers. However, the user is free to add deeper the hidden layers if the accuracy constraints are more stringent. We limit the number of neurons per layer at most 128 and select the activation function type from 'Relu', 'Sigmoid', and 'Tanh'. No more than 500 neural networks are selected with every selected neural network is trained for 300 epochs.

Table 3 shows the runtime range of the major components in each search iteration, Training and Validation is the most time-consuming, therefore, although SMAC adds to computation overhead in search, but greatly reduces the times of evaluation and contributes to time saving in total.

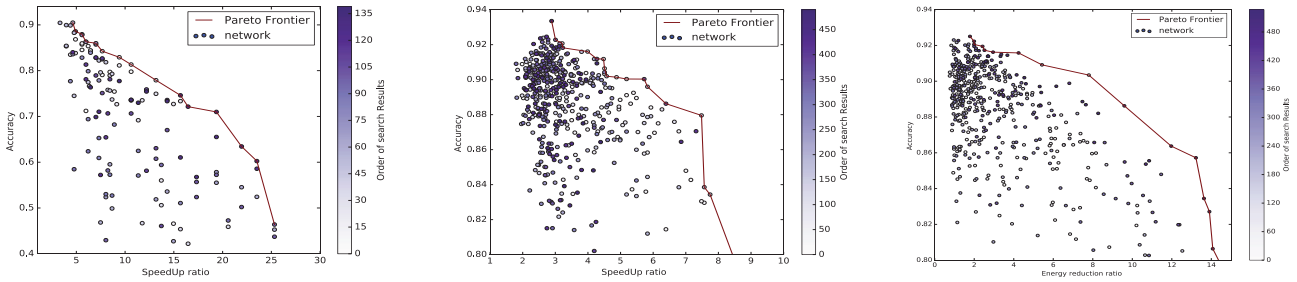
TABLE 3. Runtime range of main computation parts in each iteration

Training & Validation	Hardware Simulation	SMAC Computation
52~400 (seconds)	0.5~6 (seconds)	0.2~5 (seconds)

The offline stage aims at finding the neural networks that have high f_{perf} performance. The user can specify the coefficient $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\varepsilon}$ of f_{perf} to guide the search. For space limit, we only show the experiment results of the *blackscholes* benchmark in Figure 3 as a demonstration. As we can see, compared with the 'fixed topology enumeration', our method can efficiently guide NAS to favor neural networks that have high approximation quality or the trade-off.

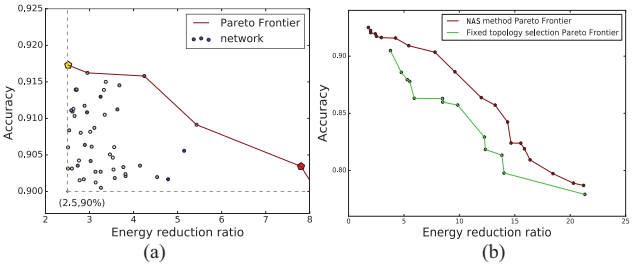
C. Online selection

Figure 4. (a) shows the example of online dynamic selection while meeting the user-specified constraints on the *blacks-choles* benchmark. In the example, we assume the user requires the energy reduction ratio to be at least 2.5x of the bar-value imposed by the vertical dotted line, and the accuracy over 90% indicated by the horizontal dotted line. Online selection will return the corresponding networks denoted by the yellow pentagons and red pentagons for quality mode and energy mode respectively; for integrated mode, the online mode returns one of the NNs of "Pareto Frontier" that satisfy the user constraints based on the user-tunable coefficient.



(a) Previous 'fixed topology enumeration' method (b) Focus on accuracy (c) Trade-off accuracy and energy reduction

Figure 3. Compared with previous 'fixed topology enumeration' method on the *blackscholes* (error bound is 0.01). (a) is the previous 'fixed topology enumeration' search method. (b) is our NAS method that focuses on accuracy (α: 0.9 β: 0.05 ε: 0.01). (c) is our NAS method towards trade-off between accuracy and energy reduction (α: 0.85 β: 0.01 ε: 0.1). The baseline is executing code on the CPU. Average search runtime for one application is near **8 hours** on one Titan-x GPU.



(a) (b)

Figure 4. (a)Example of online network selection with on the *blackscholes* benchmark (error bound is 0.01). The vertical dotted line presents the energy reduction ratio threshold, while the horizontal dotted line presents the accuracy threshold. (b) Search results comparison

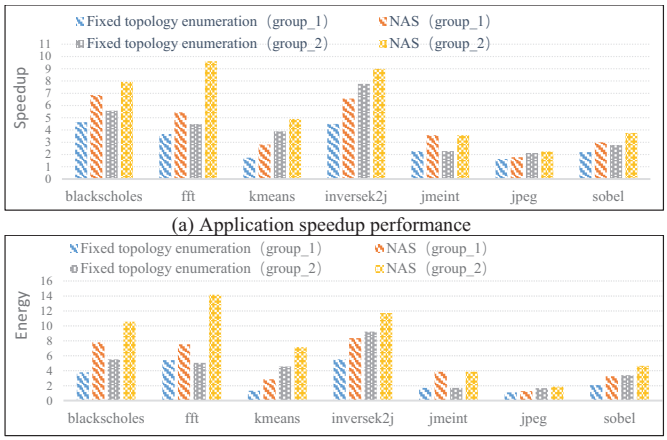
D. Search results comparison

Figure.4 (b) provides the analysis of Pareto frontiers achieved by the 'fixed topology enumeration' and NAS method on the *blackscholes* benchmark, showing the NAS method is able to find a better neural network in terms of both energy reduction and accuracy under the same condition. We provide the result about the Pareto-optimal candidates found during the search process by comparing the energy reduction and accuracy. From the result, the highest accuracy of NAS method is 93.7% while 'fixed topology enumeration' method is 90.3%. Under the same accuracy level, NAS method has at most near 2x more energy reduction compared to the 'fixed topology enumeration'.

E. Speedup and energy improvement

Figure 5 compares the two methods for improving the speedup and energy efficiency. Following the previous work [4], which limit the quality loss to 10%, we also set the minimum accuracy threshold to be 90% except for the jmeint benchmark has the highest accuracy of 84.5% in our experiments due to inhomogeneous distribution of input data. The baseline is executing the original compute-intensive and approximable code on the CPU.

In experiments, we have tested NAS on group_1 and group_2, divided by the error bound. In group_1 that denotes the constraint of higher output accuracy, the 'fixed topology enumeration' method offers 2.95x speedup and 3.0x energy reduction on average compared to the baseline, while our NAS method offers 4.29x speedup and 5.0x energy reduction. In group_2 with lower accuracy demand, the 'fixed topology enumeration' method offers 4.13x speedup and 4.5x energy reduction on average, while the NAS method offers 5.86x speedup and 7.7x energy reduction. Across all benchmarks, the NAS method can achieve 1.43x more speedup improvement and 1.74x more energy reduction. Overall, our NAS method can be more effective in selecting the most energy efficient neural network for approximate computing while satisfying the user-specified constraints.



(a) Application speedup performance

(b) Application energy reduction

Figure 5. Speedup and energy improvements.

V. Conclusion

In this paper, we apply NAS to general-purpose approximate computing. This method makes the best use of the quality/performance margin in the approximate computing workloads to improve system energy efficiency. The whole procedure is performed automatically to avoid unqualified network design and saves the manual effort while considering the hardware characteristics and user-specified constraints. Compared with the previous method, our method relatively achieves 1.43x more speedup improvement and 1.74x more energy reduction on average.

References

[1] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In ETS, 2013.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in ASPLOS, 2012.

[3] A. Yazdanbakhsh, et al., "Axbench: A multiplatform benchmark suite for approximate computing," IEEE Design & Test, vol. 34, no. 2, pp. 60–68, 2017.

[4] H. Esmaeilzadeh, A. Sampson, L. Ceze et al., "Neural Acceleration for General-Purpose Approximate Programs," in MICRO, 2012.

[5] Chen, Emer, and Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in ISCA, 2016.

[6] Zoph, Barret and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning," In ICLR'17, 2017.

[7] Frank Hutter, et al., "Sequential model-based optimization for general algorithm configuration," In LION. Rome, Italy, 507–523, 2017.