Energy-Efficient and Quality-Assured Approximate Computing Framework Using a Co-Training Method

LI JIANG, ZHUORAN SONG, HAIYUE SONG, and CHENGWEN XU,
Shanghai Jiao Tong University
QIANG XU, Chinese University of Hong Kong
NAIFENG JING, Shanghai Jiao Tong University
WEIFENG ZHANG, Alibaba Group
XIAOYAO LIANG, Shanghai Jiao Tong University

Approximate computing is a promising design paradigm that introduces a new dimension—error—into the original design space. By allowing the inexact computation in error-tolerance applications, approximate computing can gain both performance and energy efficiency. A neural network (NN) is a universal approximator in theory and possesses a high level of parallelism. The emerging deep neural network accelerators deployed with NN-based approximator is thereby a promising candidate for approximate computing. Nevertheless, the approximation result must satisfy the users' requirement, and the approximation result varies across different applications. We normally deploy an NN-based classifier to ensure the approximation quality. Only the inputs predicted to meet the quality requirement can be executed by the approximator. The potential of these two NNs, however, is fully explored; the involving of two NNs in approximate computing imposes critical optimization questions, such as two NNs' distinct views of the input data space, how to train the two correlated NNs, and what are their topologies.

In this article, we propose a novel NN-based approximate computing framework with quality insurance. We advocate a co-training approach that trains the classifier and the approximator alternately to maximize the agreement of the two NNs on the input space. In each iteration, we coordinate the training of the two NNs with a judicious selection of training data. Next, we explore different selection policies and propose to select training data from multiple iterations, which can enhance the invocation of the approximate accelerator. In addition, we optimize the classifier by integrating a dynamic threshold tuning algorithm to improve the invocation of the approximate accelerator further. The increased invocation of accelerator leads to higher energy efficiency under the same quality requirement. We propose two efficient algorithms to explore the smallest topology of the NN-based approximator and the classifier to achieve the quality requirement. The first algorithm straightforward searches the minimum topology using a greedy strategy. However, the

L. Jiang and Z. Song contributed equally to this research.

This research was partially supported by National Natural Science Foundation of China (grant nos. 61332001, 61834006, 61772331, and 61602300); Shanghai Science and Technology Committee (no. 18ZR1421400); National Key Research and Development Program of China (no. 2018YFB1403400); Alibaba Group through the Alibaba Innovative Research (AIR) Program; and State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences.

Authors' addresses: L. Jiang, Z. Song, H. Song, C. Xu, N. Jing, and X. Liang, Shanghai Jiao Tong University, 800 Dongchuan Rd, Minhang District, Shanghai, 200240, China; emails: jiangli@cs.sjtu.edu.cn, {songzhuoran, shyyhs, asd4876, sjtuj}@sjtu.edu.cn, liang-xy@cs.sjtu.edu.cn; Q. Xu, Chinese University of Hong Kong, Horse Material Water, Shatin, New Territories, Hong Kong; email: qxu@cse.cuhk.edu.hk; W. Zhang, Alibaba Group, No. 969, Wenyi West Road, Yuhang District, Hangzhou, China; email: weifeng.z@alibaba-inc.com.

1084-4309/2019/08-ART59 $15.00
https://doi.org/10.1145/3342239

first algorithm incurs too much training overhead. To solve this issue, the second one gradually grows the topology of NNs to match the quality requirement by transferring the learned parameters. Experimental results show significant improvement on the quality and the energy efficiency compared to the existing NN-based approximate computing frameworks.

CCS Concepts: • **Hardware → Power and energy**;

Additional Key Words and Phrases: Approximate computing, error control

ACM Reference format:
Li Jiang, Zhuoran Song, Haiyue Song, Chengwen Xu, Qiang Xu, Naifeng Jing, Weifeng Zhang, and Xiaoyao Liang. 2019. Energy-Efficient and Quality-Assured Approximate Computing Framework Using a Co-Training Method. *ACM Trans. Des. Autom. Electron. Syst.* 24, 6, Article 59 (August 2019), 25 pages.
https://doi.org/10.1145/3342239

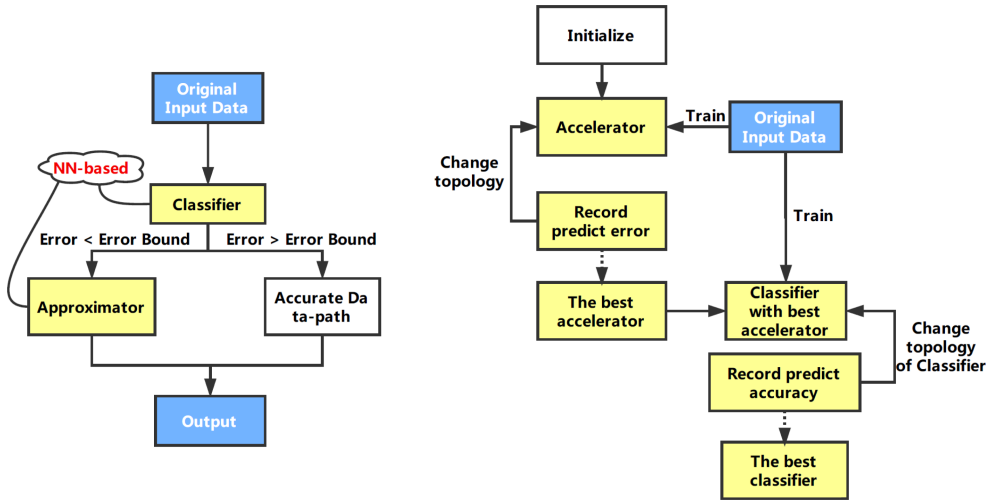
1 INTRODUCTION

Recently, approximate computing has gained significant momentum in both industry and the research community. The inaccurate results by an approximate computation can be inherently tolerated in many applications, especially those referred to recognition, mining, and synthesis (RMS) [1]. The effectiveness of approximate computing is based on the fact that all computations are not equally significant in determining the output quality [20].

The approximation can lie in the circuit/system level without notifying the application. For example, application-specific integrated circuits (ASICs) designed with intentionally overscaled frequency [16], supply voltage [19], and simpler circuitry [17] obtain the extra energy efficiency, whereas the inaccurate results are inherently tolerated in the application level. Besides, the programmer can annotate code segments in the source program if the inaccurate output of the segment can be tolerated by the application. This code segment can be alternated by a so-called *approximator*. The approximator can either be implemented as memorization or a neural network (NN). Given a set of input values, the memorization approach (e.g., look-up table [13]) strives to look up the best output in the precalculated table rather than compute the output, whereas the latter exploits the enormous parallelism in the NN [4, 6, 8].

In this article, we focus on the NN-based approximator because the NN is a universal approximator and the NN can fit any continuous function [11] in theory. A larger scope of applications thus can benefit from neural approximate computing. For example, the higher-order nonlinearity embodied in deep neural networks (DNNs) results in superior approximate precision for those complex applications. Moreover, those specialized accelerators, such as GPU [25, 30], FPGA [24], and ASICs [16] gain massive speedup and energy efficiency when applying an NN-based approximator.

Despite the great gain of speedup and energy efficiency provided by the approximators, quality control is essential to identify the safe-to-approximate input data and provide a tradeoff among the resulting quality and energy efficiency of the computation [3, 4, 7, 26]. The result has been shown that the data with large approximate error generally exist in a small fraction of the data space, but the data can cause most of the quality degradation in applications [15]. To control the approximate error, a classifier is trained to predict whether the input data can be executed by the accelerator [5, 15, 18, 21, 27]. As shown in Figure 1(a), the input data is first sent to the classifier. When unacceptable large errors are predicted, a proactive approach is to schedule the input to exact execution, or a roll-back recovery can be triggered to accurately recompute the data with an accurate data path. Otherwise, the input data is sent to the approximator. Various classifiers [15, 27] are proposed to characterize the error behaviors and make the prediction. The input data is used to train the approximator and is used again in the training of the classifier, except the training data



(a) NN-based approximate computing. (b) Training process.

Fig. 1. Existing NN-based approximate computing and their training processes.

now has the label indicating whether the input data is safe to approximate. These works optimize the predictor for general approximate schemes.

More opportunities emerge when we instantiate both the classifier and approximator with NNs. Existing works, however, separate the training process of the approximator and the classifier. As shown in Figure 1(b), Esmaeilzadeh et al. [4] first train the "best" approximator in terms of the mean relative deviation between the output from the approximator and the ideal output (e.g., mean relative error (MRE)). All of the training input data then have their own predicted error. The approximator can determine whether the input data is safe to approximate by comparing the input data's predicted error to the error bound defined by the user, resulting pairs of input data and "labels." Consequently, Esmaeilzadeh et al. [4] train the "best" classifier with input data and "labels." To find the most cost-effective approximator that can satisfy the quality requirement, Esmaeilzadeh et al. [4] first change the topology of the approximator from the smallest topology until meeting the error bound. After that, Esmaeilzadeh et al. [4] change the topology of the classifier with the fixed "best" approximator from the smallest topology until meeting the accuracy bound. Note that the topology is defined by the number of hidden layers and the number of neurons in each layer [7].

In this work, we argue that the "best" approximator and the "best" classifier stand-alone cannot lead to the best tradeoff of the quality and energy efficiency in whole. What we expect from the training of the approximator should be (i) maximizing the safety-to-approximate portion of the data and (ii) minimizing the MRE of the input data recognized by the classifier rather than minimizing the MRE of the whole input data, because the data causing large error that violates the error bound is not activated in the approximator. The contributions of this article are as follows:

- It is difficult for the two NNs to reach a consensus on the ground-truth safe-to-approximate data. Therefore, we propose a novel iterative co-training method to coalesce the training of the two NNs. The proposed training method enlarges the optimization space of the overall quality and energy efficiency.
- No ideal classifier can perfectly predict whether the training data is suitable for approximate computing, so a part of training data can be misjudged. Thereby, we explore different

policies of selecting training data to the approximator. Based on the exploration, we optimize the proposed training method by selecting training data from multiple iterations.
- In the iterative co-training method, the approximator evolves; the approximation errors change for the same input data. We thereby propose an efficient algorithm that can guide the classifier to evolve itself following the ever-changing approximator during the iterative co-training process. This algorithm can further maximize the invocation of the approximator.
- In addition, existing works strive to minimize the topology of the approximator, whereas all assume and prefer a small classifier. However, a larger classifier can offer a more accurate prediction, which in turn improves the invocation of the approximator and the quality. Thus, we propose two efficient algorithms to explore the smallest topology of the two NNs to achieve the quality requirement. The first algorithm adopts the conventional method by exhaustive search of the minimum topology. This strategy inevitably incurs too much training overhead for the iterative co-training method. To solve this issue, the second algorithm gradually grows the topology of NNs to match the quality requirement by transfer learning.

The rest of this article is organized as follows. In Section 2, we introduce related works and the motivation for our work. Section 3 describes the proposed co-training method of two NNs. The dynamic threshold tuning method for the classifier is introduced in Section 4. Section 5 shows the two topology exploration algorithms. Experiments are presented in Section 6, and Section 7 concludes the article.

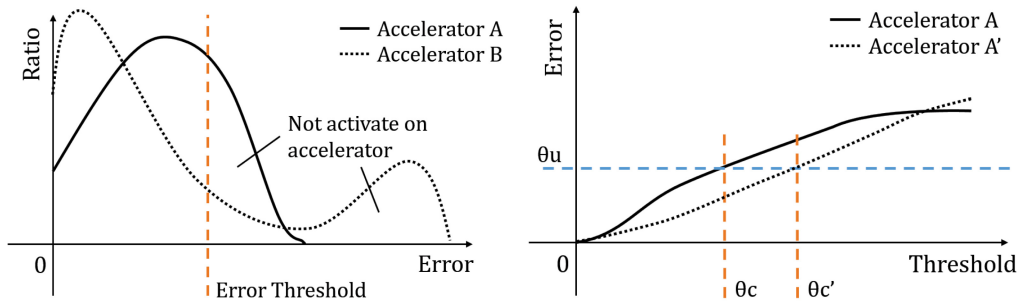
2 PRELIMINARY

This section first introduces the related works on quality control and the motivation for this work.

2.1 Related Works

Approximate computing is an energy-efficient and performance-effective technology. The inherent error resiliency of DNNs has been exploited by introducing approximations to obtain energy savings. Hanif et al. [10] propose a cross-layer approximation methodology that explores the error resiliency of DNNs at the software and hardware level. At the software level, Hanif et al. [10] prune weights, whereas at the hardware level, they utilize approximation of multipliers to perform the weighted sum operation. Shafique et al. [23] employ the configurable approximate modules inside the NN data path. In addition, approximate computing can be utilized to approximate the general-purpose programs [12, 14]. Imani et al. [12] propose a configurable approximate divider to reduce the significant cost and complexity of the floating point divider. Specifically, the approximate divider is implemented by replacing the costly division operation with a subtraction of the input operands mantissa. Imani et al. [14] introduce an iterative method for approximating the product of two operands using fitted linear functions with two inputs.

Quality control is an essential and nontrivial problem in approximate computing. Relative error is a metric to measure the error significance, whereas MRE can further take the error rate into consideration. Baek and Chilimbi [2] and Samadi et al. [22] propose to sample a proportion of data and execute them in both the approximator and the accuracy core. If substantial differences are shown between the two computation units, a more accurate accelerator is used for further approximate computing. These techniques can ensure that the MRE of the accelerator conforms to the user requirement. However, these techniques cannot prevent the approximator from consuming the data that has large relative errors but occur occasionally. These data escape the sampling and result in significant degradation of the computation quality. Thus, simply minimizing the MRE of the accelerator is not good enough.



(a) Data distribution ratio on the approximation error. (b) Approximation error versus classifier threshold.

Fig. 2. A conceptual example.

To alleviate the preceding issue, various classifiers are developed to predict whether the input data, when processed by the approximate accelerator, results in acceptable errors. Khudia et al. [15] propose three prediction-based classifiers, which are based on a linear model, a decision tree, and an exponential moving average technique. If the quality does not satisfy the user requirement, the approximate result will be rolled back, and the input data will be recomputed using an accurate data path. To enhance the prediction accuracy of the classifier, more sophisticated models are utilized, such as a cachelike look-up table and an NN [18]. This work shows the superiority of NN over the look-up table. Sophisticated classifiers, however, result in large performance overhead.

To embrace both the lightweightedness and high prediction accuracy of the predictor, Wang et al. [27] propose three synergistic methods by combining several basic predictors: a voting-based combination, the boosting-based combination, and the stacking-based combination. Even if a specific simple classifier fails in some input data, the combination of these classifiers can still achieve robust prediction accuracy. However, the combination of these classifiers may incur a large cost to explore the specific number and type of basic quality predictors that can give satisfactory prediction accuracy.

All existing classifier-accelerator hybrid approximate computing architectures are devoted to obtaining an accurate classifier and a "best" approximator by adopting a one-pass training. The data collected from the entry (input) and exit (output) of the code section is used to train the approximator [4]. Afterward, the training data is used to test the derived approximator. If the approximate result contains an error larger than the error bound, the training data will be labeled as "bad" and vice versa. The labeled training data is then used to train the classifier. In the runtime, the classifier makes a binary decision on execution of the selected code segment using either the approximate accelerator or the accurate data path.

2.2 Motivation

The one-pass training process only optimizes the approximator and the classifier separately. Moreover, we know that the classifier and approximator will affect each other especially in the training part, and a better solution exists if the classifier and the approximator are optimized as a whole. For example, Figure 2(a) shows the distribution of the output data resulted from two approximators. The output data from approximator A has a much lower MRE than those from approximator B. Thus, previous works may prefer approximator A to approximator B. However, as the classifier can partition the data according to the error bound, only data with smaller predicted error than the error bound can be executed in the approximator. Therefore, the total error of the hybrid architecture is caused by those data on the left-hand side of the error bound. From this point of

view, approximator B actually performs better than approximator A. And in practice, the error bound is proportional to the final approximating error, as shown in Figure 2(b), and maximizing the error bound under the user requirement error bound can increase the invocation of the approximator [18]. This conceptual example motivates us to propose a holistic training process, which should be able to solve the three problems as follows.

First, we should maximize the safe-to-approximate portion of the data in the training process to maximize the invocation of the approximator; we also should minimize the MRE of the approximator-classifier hybrid architecture rather than the stand-alone MRE of the approximator.

The training process for the approximator and the classifier should be integrated, form a closed loop, and converge to a better solution. We observe that the classifier will prevent a portion of the training data from being executed in the approximator. This portion of data may deteriorate the training of the approximator and thus should be discarded before training the approximator.

Second, the selection of the training data in each iteration is the key to the training. Considering that there is no perfect classifier, some training data will be mispredicted as not suitable for being approximated. Carefully selection of the training data to the approximator is another challenge to us. In short, more data means higher invocation, whereas approximating error and fewer data results in a model with high accuracy but lower invocation. We thus explore the difference using different types of training data and give reasons why some kinds exceed others.

Third, we need an adaptive threshold in the classifier for a tradeoff between the computation quality and the energy efficiency. Larger threshold increases the invocation of the approximator but makes the classifier switch more data to an acceptable part and degrades the computation quality of the approximation results, whereas the smaller threshold accomplishes the opposite. We should find the best threshold that makes the final error meet the user requirement with a maximum acceptable data proportion of the classifier. The existing threshold finding algorithm [18], however, assumes a constant approximator and uses the ground truth labels as a perfect classifier to adjust the threshold, which is impossible in this context.

Last, although a large classifier is not energy efficient, it can improve the confidence of the approximator to take more data into approximation, which in turn improves the energy efficiency. Thus, we should explore the topology of the classifier in coordination with the topology exploration of the approximator.

These problems motivate us to propose a new training framework for the approximate computing based on an accelerator-classifier structure. We design an iterative co-training algorithm to train accelerator and classifier simultaneously and a way to select training data during the proposed iterative co-training process. In addition, a method of exploring the topology of accelerator and classifier is proposed to find the most appropriate topologies with a flexible user requirement. We also propose an efficient threshold finding algorithm to find the best threshold in an iterative co-training process.

3 PROPOSED ITERATIVE CO-TRAINING METHOD

In this section, we first propose an iterative co-training algorithm to coordinate the training process of the accelerator and the classifier comprehensively. In the iterative co-training process, we utilize different kinds of training data to train the approximator to explore the best training data selection policy.

3.1 Iterative Co-Training

The basic idea of coordinating the training of the accelerator and the classifier is a more precise use of training data. The type of NNs we used in the accelerator and the classifier is the multilayer perceptron, because the continuous functions do not need to be fitted by a feature extraction

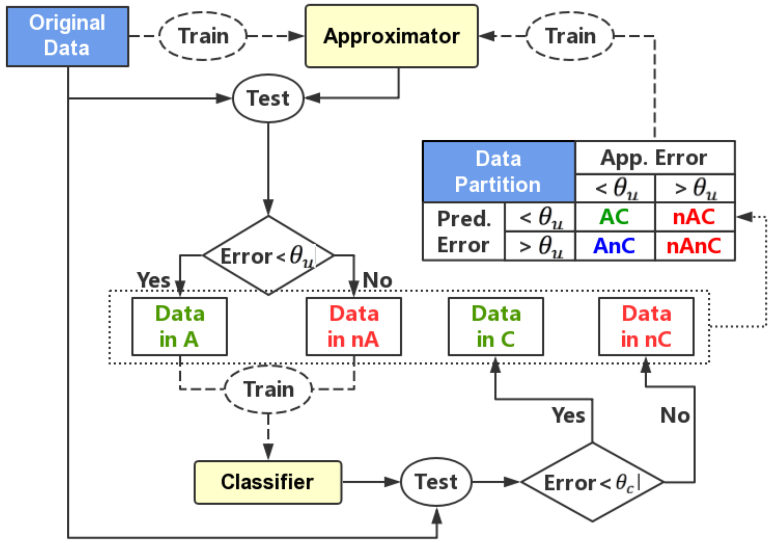


Fig. 3. The process of iterative training.

network, such as a convolutional neural network (CNN). We iteratively train the approximator and the classifier, as shown in Figure 3, and prevent the "garbage" training data filtered by the classifier from training the approximator in future iterations. In the first iteration, the approximator and the classifier are trained beforehand as in previous works [18]. Next, we repeat the following steps in each iteration.

We first apply the original data to the accelerator and the classifier. The output data from the accelerator are compared to the error bound. After the comparison, the original data is partitioned by the error bound into two parts: "good" data that satisfies the error bound and "bad" data violating the error bound; we label them as "A" and "nA," respectively. Similarly, the classifier divides the original data into "good" data and "bad" data, predicting whether the data is safe to approximate or not; we label them as "C" and "nC," respectively. These labels divide the original data into four categories, which will be explained in detail later.

Next, we judiciously select the training data from these categories to train the approximator and classifier in the next iteration. The i^{th} training begins with the approximator and the classifier derived in $(i-1)^{th}$ training—for example, the weights of the NN-based approximator and classifier are inherited from those trained in the previous iteration. As the training data changes, the threshold of the classifier should also change; thus, we integrate the threshold tuning algorithm with the iterative co-training (see details in Section 4). When the threshold tuning algorithm terminates, the iterative co-training process also terminates. At the end of the iterative co-training, the topologies of the approximator and the classifier are explored by our proposed topology exploration algorithm, which is discussed in Section 5.

The judicious selection of the training data in each iteration is the key for the iterative co-training. We use a conceptual figure (Figure 4) to illustrate how we partition the original data. The X axis and the Y axis are the same as those in Figure 2(a). The error distribution of the data executed by the accelerator is shown in the solid curve, which is divided by the error bound into two parts, labeled as "A" and "nA." An ideal classifier, although impossible, will identify all of the "A" category data and label them as safe-to-approximate data (i.e.,"C"). In practice, only part of the "A" category data will be labeled as the "C" category and further be executed on the accelerator.

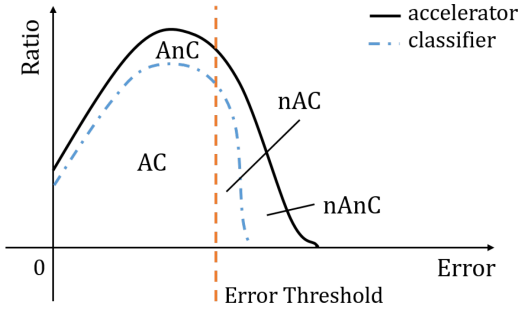


Fig. 4. Distribution of the four categories of the original data derived from the accelerator and the classifier.

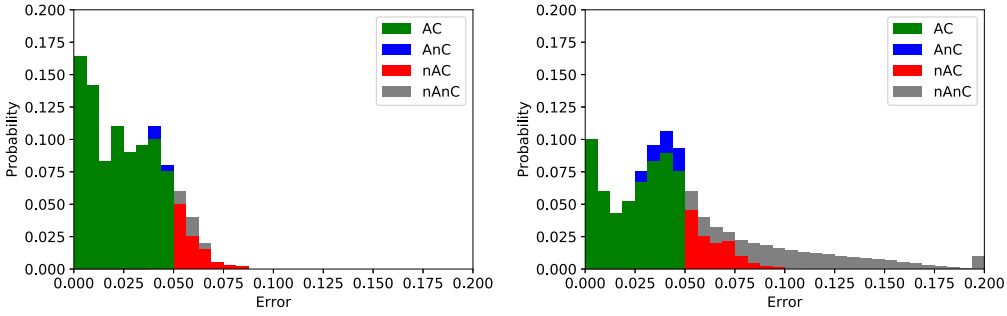
Similarly, the classifier will mistakenly regard part of the "nA" data as safe-to-approximate data and label them as "C." Therefore, we use a dashed line curve to represent the distribution of data derived from the classifier. The area below the dashed line is labeled as "C" and vice versa. Thus, the four categories of the original data are labeled as "AC," "nAC," "AnC," and "nAnC". In other words, the precision of the prediction by the classifier is $\frac{AC}{AC+nAC}$ and the recall is $\frac{AC}{AC+AnC}$.

Obviously, the "nAnC" category is redundant and must be discarded in the next training process. Different combination of the remaining data categories, as the training data, causes different training results. We can choose to keep the "AC" part as the training data; this results in high accuracy of the classifier, but this combination also shrinks the portion of data executed on the accelerator and thus degrades the energy efficiency of the hybrid architecture. We can also keep "AC," "AnC," and "nAC" categories, on the contrary, to approximate more data for performance boost; but this may limit the enhancement of the classifier. In Section 3.2, we explore these different strategies of training data and will show them in the experiment in a very intuitionistic way.

It should be noted that in each iteration, we always select the training data from the original data; thus, the size of the training data will not suffer the continuous reduction in the iterative co-training process. Although the iterative co-training method costs more training time compared to the original one-pass training process, the iterative co-training method is a one-time effort when a new program is about to be approximated in the hybrid architecture, and the method will not affect the approximator's runtime performance.

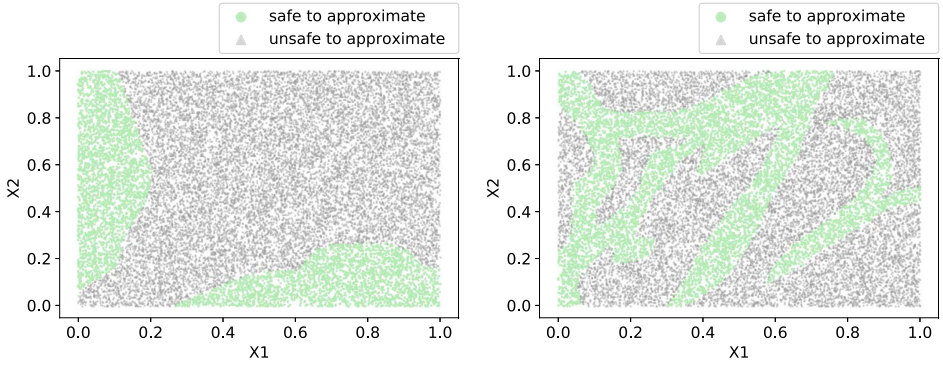
3.2 Exploring Training Data Selection Policy

Using different categories of training data will result in different accuracy, recall, and invocation of the approximator and the classifier. To investigate the underlying reason, we show the error distribution of iterative co-training by using "C" category data and "A" category data, as shown in Figure 5. In the figure, the green part represents the input data that is accepted by bot the approximator and classifier. The blue part stands for the input data whose error is lower than the error bound but unfortunately misclassified by the classifier. Moreover, the red part represents the input data with an error higher than the error bound but is labeled as the "C" category. Similarly we have the grey part, which is discarded by both the approximator and classifier. Our target is to reduce the blue part and the red part that present the difference between the approximator and the classifier, and reduce the grey part and increase the green part, if possible, which means that there are more data to be calculated by the approximator. From Figure 5, we can see that using category "C" performs better compared to category "A" because of the higher recall, calculated by $\frac{AC}{AC+AnC}$. In the meantime, the mean error of using category "C" is lower, as we can see that most of its predicted error is closer to zero.



(a) Training with category "C". (b) Training with category "A".

Fig. 5. Error distribution using different data selection policies in benchmark blackscholes.



(a) Training with category "C". (b) Training with category "A".

Fig. 6. Training data distribution in benchmark bessel.

To further investigate why category "C" has a better recall than category "A," we need more information on the iteration training process. In benchmark bessel, the input is 2-dimension, and thus we can illustrate the input dimension easily and intuitively. Figure 6 shows the distribution of training data when we select "C" category data and "A" category data separately. The green input samples present the training data that will be used to train the approximator in the next iteration. From Figure 6(a), we can see that by iterative co-training with category "C," the safe-to-approximate data will cluster together rather than spread across in the whole input space, just like Figure 6(b). As the size of the classifier is normally small for energy efficiency, it is difficult for us to train the classifier by using training data with such complicated distribution.

Figures 7 and 8 show the training data distribution in three continuous training iterations. When we choose category "C" as the training data, we can get a similar training dataset in each iteration, which means that the training process of the approximator is relatively stable. But when choosing category "A," the approximator has to adjust itself to learn a different dataset in each iteration, which means that the training process is unstable and hard to reach a consensus.

We also find that category "AC" is similar to "C" and that category "AorC" is similar to "A." As a result, when training the model, it is better to use category "AC" or "C" rather than "A" or "AorC" due to the better distribution continuity of data. In our training, we always use category "AC" to get a higher recall and accuracy of the approximator.

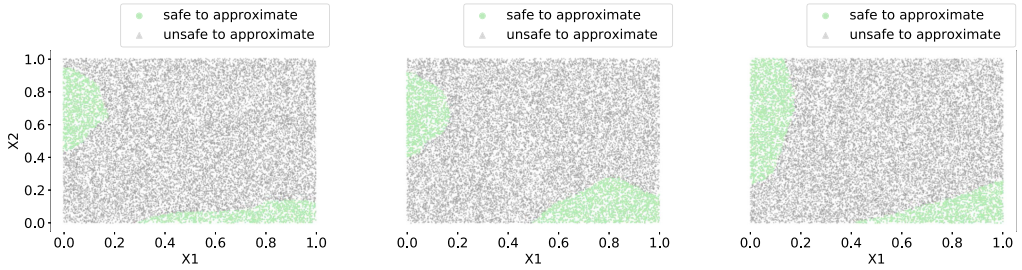


Fig. 7. Training data distribution with category "C" in three continuous iterations.

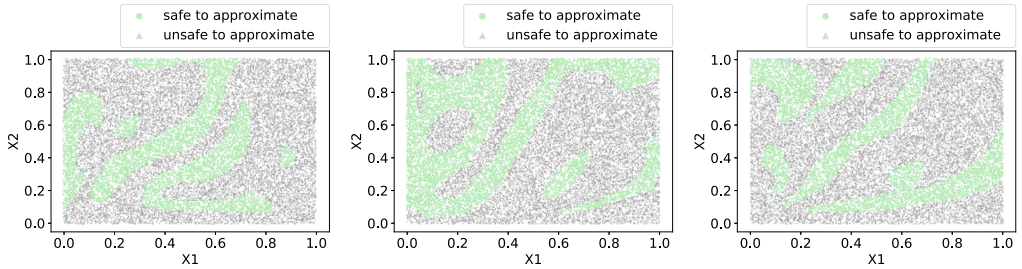


Fig. 8. Training data distribution with category "A" in three continuous iterations.

The proposed iterative co-training method can make better use of training data with higher accuracy and higher invocation in most cases. We provide a math proof in Appendix A to demonstrate the effectiveness of the method theoretically.

4 DYNAMIC THRESHOLD TUNING FOR THE CLASSIFIER

The threshold of the classifier affects the relative error of the approximate accelerator; the threshold also affects the proportion of the data computed on the accelerator (denoted by $Invocation$). Finding an appropriate threshold is essential for the hybrid architecture. Unfortunately, the threshold finding algorithm in Mahajan et al. [18] only works for a fixed accelerator. Therefore, we propose a new threshold tuning algorithm in Algorithm 1.

The key idea is to find the threshold φ by binary search method with a scaling parameter δ—the tuning granularity—so that the relative error derived from the accelerator-classifier hybrid architecture $Error(A, C, D)$ can satisfy the user requirement q. Here, A and C denote the models of the accelerator and the classifier, and D refers to the original data. To facilitate the description, we denote the i^{th} iteration of the training process as $IterativeTrain(A, C, \varphi, i)$.

We first initialize the parameters (lines 1–3). The error of the pretrained accelerator in the first iteration also satisfies the error bound (line 4). We iteratively train the accelerator-classifier hybrid architecture (lines 6 and 7) and tune the threshold (lines 8–20) until the tuning granularity is sufficiently small (line 5). Then, we adjust the threshold φ with the tuning granularity δ as follows. If the current approximate error satisfies the error bound (line 8), we increase φ by δ (line 12); otherwise (line 14), we decrease φ by δ (line 18). When we tune the threshold, the approximate error may fluctuate around the error bound. We use the indicator satisfy to record the change of the satisfaction of the error (lines 13 and 19). If the satisfaction indicator changes, the approximating error jumps over the error bound; consequently, we need to shrink the tuning granularity δ by half and tune the threshold in a reverse direction to regress the approximating error and make the error approach to the error bound. When δ is smaller than a specific value Δ, the algorithm

ALGORITHM 1: Iterative co-training with threshold tuning

```
1  φ ← Φ //Φ: initial threshold;
2  δ ← 2^MaxDepth * Δ //Δ: minimum tuning granularity;
3  i ← 0 //the iterative co-training begins with the 0^th iteration;
4  satisfy ← True //An satisfied indicator;
5  while δ > Δ do
6      IterativeTrain(A, C, φ, i++);
7      Record(A, C, φ);
8      if Error(A, C, D) < q then
9          if not satisfy then
10             δ ← δ/2;
11         end
12         φ ← φ + δ;
13         satisfy ← True;
14     else
15         if satisfy then
16             δ ← δ/2;
17         end
18         φ ← φ − δ;
19         satisfy ← False;
20     end
21 end
```

terminates and we can choose the best solution from the saved records. Due to the randomness of the training process, there are several empirical parameters, which are commonly acceptable in an NN training method.

5 TOPOLOGY EXPLORATION FOR THE HYBRID ARCHITECTURE

We emphasize that the topology of the accelerator and the topology of the classifier influence the MRE, the energy efficiency differently. The accelerator and classifier with deeper layers and more neurons improve the MRE but degrade the energy efficiency; moreover, a large classifier can increase the invocation of the accelerator, which in turn improves the energy efficiency. We can explore the combination of topologies of the accelerator and the classifier with a brute force search. However, the search algorithm requires quadratic time complexity, and each solution results in a time-consuming training process. To ease the preceding challenge, we propose an efficient heuristic, as shown in Algorithm 2, to find a sweet point.

5.1 A Linear Search with NN Retraining

The key idea of our heuristic is to provide a large enough search space by intentionally enlarging the topology of the classifier and accelerator, and then search the sweet point by gradually and alternatively shrinking these two NNs. Once a new topology is derived, in the beginning, we continuously increase the topology of the accelerator until the approximation error can meet the error bound (q) or the network topology reaches the max size (lines 1–4). Then, we continuously increase the topology of the classifier until its size is as same as the accelerator (lines 5–9). Now, we have a sufficient large topology of the accelerator and classifier that contains "sweet points." Next, we keep decreasing the topology of the accelerator and the classifier alternatively (lines 10–21) by synaptic pruning and NN retraining [9] without countable quality loss (we name it the

ALGORITHM 2: Topology exploration

```
1  while Error(A, D) > q do
2  |   Enlarge(A);
3  |   Train(A, D);
4  end
5  SetMinSize(C);
6  while Size(C) < Size(A) do
7  |   Enlarge(C);
8  |   IterativeTrain(A, C, D);
9  end
10 while Error(A, C) < q do
11 |   A', C' ← Decrease(A), Decrease(C);
12 |   IterativeTrain(A', C, D);
13 |   IterativeTrain(A, C', D);
14 |   if Error(A, C) > q then
15 |   |   break;
16 |   end
17 |   if Invo(A', C, D, q) > Invo(A, C', D, q) then
18 |   |   A, C = A', C;
19 |   else
20 |   |   A, C = A, C';
21 |   end
22 end
```

Linear Decrease function). Note that we shrink the classifier and the accelerator (line 11), followed by retraining (lines 12 and 13) independently. Two possible solutions (i.e., A' and C') are derived. We choose the one that invokes the accelerator more often under the quality requirement (lines 17–21). Here, $Invo(A, C, D, q)$ calculates the invocation of the accelerator with quality requirement q. Notice that the retraining of the NN relies on the proposed iterative co-training process (lines 8, 12, and 13). Whenever the decreasing of the topology causes the violation of the error bound, the algorithm terminates (lines 10 and 14–16). The new derived topology stored in A' or C' is discarded.

5.2 Exponential Enlarge and Linear Decrease with NN Fine Tuning

The linear topology search heuristic with iterative co-training is too time consuming. To cope with this challenge, we propose an exponential enlarge network algorithm, as shown in Algorithm 3. At first, we set the "enlarge step" $\eta = 1$, and set the minimum topology of the approximator and classifier (lines 1–3). Then, we simultaneously enlarge the approximator and the classifier until their error is less than the error bound q (lines 4–17). Finally, we use the *Linear Decrease* function defined in Algorithm 2 to decrease the topology of the approximator and the classifier (lines 19 and 20).

When enlarging the topology, we use a greedy algorithm to choose to add neurons or to add a layer containing a specific number of neurons. On the one hand, we add η neurons to the existing layers and obtain the approximator A^1 (lines 5 and 6), wherein η increases exponentially in the power of two after each iteration. On the other hand, we add a layer with θ neurons to the existing structure and obtain the approximator A^2 (lines 7 and 8). Next, we directly make the topology of classifier C^1 (C^2) the same as approximator A^1 (A^2). Note that the last layers of the approximator

ALGORITHM 3: Exponential enlarge network algorithm

```
1  Initially set η = 1
2  SetMinSize(A)
3  SetMinSize(C)
4  while Error(A, C) > q do
5  |   A^1 ← AddNeuron(A, η)
6  |   S = ComputeSynapsis(A^1)
7  |   θ = ComputeNewLayerNeuron(A, S)
8  |   A^2 ← AddLayer(A, θ)
9  |   C^1 ← RebuildFrom(A^1)
10 |   C^2 ← RebuildFrom(A^2)
11 |   if Error(A^1, C^1, D) < Error(A^2, C^2, D) then
12 |   |   A, C ← A^1, C^1
13 |   |   η = η * 2
14 |   else
15 |   |   A, C ← A^2, C^2
16 |   |   η = θ
17 |   end
18 end
19 while Error(A, C) < q do
20 |   A, C ← LinearDecrease(A, C)
21 end
```

and the classifier have different neurons because the two NNs learn different tasks. To determine which pair of NN should be chosen, we compare the error of A^1 and C^1 and the error of A^2 and C^2 and choose the pair with the lower error (lines 11–17). Note that the synapses added by the preceding two options are equal to each other for a fair comparison. Only log_n solutions are searched and verified before the topology of the two NNs reaches a layer with n neurons instead of n solutions in the liner search method.

When enlarging the topology of the two NNs, we keep the existing synaptic weights unchanged and fine tune the new added synaptic weights. This strategy makes the training process more continuous and avoids training the NNs from scratch. Consequently, the whole training process converges faster. Besides, we use an early-stop technique [28], which terminates the training process earlier. The preceding techniques significantly accelerate the topology search process.

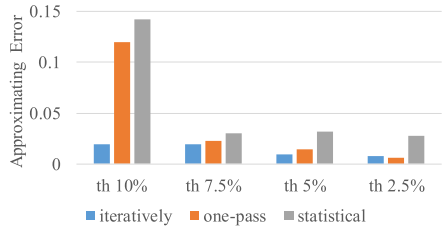
6 EXPERIMENTS AND RESULTS

6.1 Experimental Setup

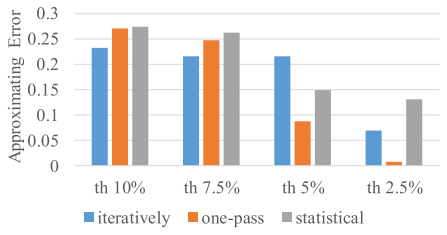
To obtain a sufficiently trained NN, the RMSprop optimizer is used and epoch is set as 10,000. Considering that this work focuses on the training process, we compare the proposed iterative co-training method (denoted as "iteratively") to the one-pass training method [4] (denoted as "one-pass") and the statistical-control method [18] (denoted as "statistical"). For fair comparison, we repeat the one-pass training (denoted as "one-pass") the same amount of times as the iterative co-training. The key differences of the proposed method are (i) selecting the "AC" category of the training data and (ii) dynamic threshold tuning. We select four benchmark applications from Esmaeilzadeh et al. [4] and the GNU GSL scientific computing library. The details of each benchmark, including the domain, the train/test data, and the quality loss metrics (QLMs), are listed in Table 1. The benchmarks (blackscholes, jmeint, fft, and bessel) we use are the publicly

Table 1. Benchmark Description

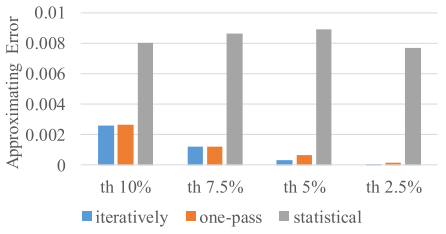
Benchmark	Domain	Train Data	Test Data	QLM	Accelerator Topology	Classifier Topology
blackscholes	Financial	4K inputs	16K inputs	MRE	6->8->1	6->8->2
jmeint	2D Gaming	10K 3D triangles	100K 3D triangles	MRE	18->32->16->2	18->16->2
fft	Signal Procession	4K fp numbers	32K fp numbers	MAE	1->2->2->2	1->2->2
bessel	Scientific Computing	10K fp numbers	20K fp numbers	MAE	2->4->4->1	2->4->2



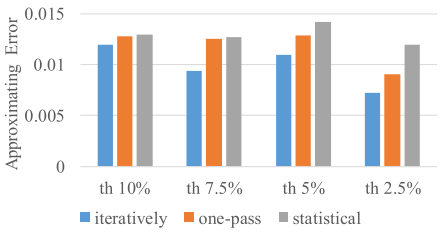
(a) Blackscholes.



(b) Jmeint.



(c) Fft.



(d) Bessel.

Fig. 9. Comparison on Approximating Error varying the threshold.

available benchmarks called *AxBench* [29]. The inputs are 6, 18, 1, and 2 for blacksholes, jmeint, fft, and bessel, respectively. When evaluating the performance of the proposed method, the QLM we use is either MRE or mean absolute error (MAE) depending on the benchmark [4], denoted as Approximating Error. The metric to evaluate the energy efficiency is Invocation mentioned in the previous section. We also use a synthetic metric Invocation/Error to take both the quality loss and the energy efficiency into account.

6.2 Results

Figures 9 and 10 show the Approximating Error and the Invocation/Error as the threshold of the classifier varies. In blackscholes, the iterative co-training reduces the approximating error by at most 83% and improves the Invocation/Error by 4.5× when the threshold is 10%. When the threshold decreases, the iterative co-training has little improvement on the approximating error but instead improves the Invocation. Therefore, we can still get 1.5× improvement on average on the Invocation/Error. A similar trend can be found in jeimt: as the threshold decreases, the iterative co-training has larger Approximating Error than the one-pass method does, but the iterative co-training outperforms the one-pass method in terms of the Invocation; thus, we still get 25% benefit on the Invocation/Error. In fft, on the contrary, our method has much lower Approximating Error

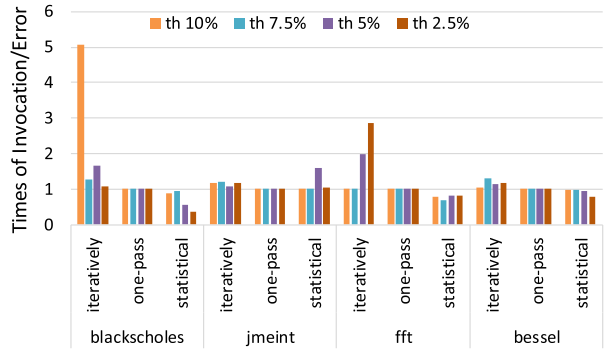


Fig. 10. Times of Invocation/Error varying the threshold, normalized to the "one-pass."

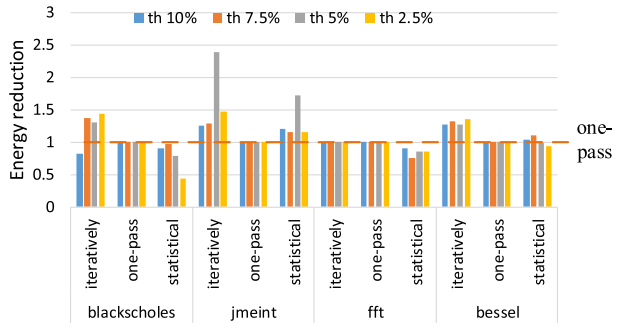


Fig. 11. Energy reduction varying the threshold, normalized to the "one-pass."

when the threshold is small, which contributes to 2× or more Invocation/Error. Bessel shows the improvements of Approximating Error and Invocation/Error in all threshold settings. On average, the iterative co-training gets 58% improvement on Invocation/Error compared to the one-pass method on different benchmarks and different threshold settings. These results show the advantages of the holistic training solution and the wise selection of the training data in each iteration. We also compare our proposed method to the statistical-control method [18] on approximating error and times of Invocation/Error in Figure 9 and Figure 10, respectively. The statistical-control method collects more data to approximate because the method chooses a 95% high confidence interval. Consequently, the approximating errors of the statistical-control method are higher than our proposed method and one-pass method across those four benchmarks. With a 95% high confidence interval, the invocations of the statistical-control method are therefore enhanced compared with our proposed method across those four benchmarks. As shown in Figure 10, the times of Invocation/Error of the statistical-control method is comparable to the one-pass method in jmeint and bessel benchmarks, but the times of Invocation/Error of the statistical-control method degrades 40% and 23% in blackscholes and fft benchmarks compared to the one-pass method.

Figure 11 shows the "energy reduction" of our proposed method, the statistical-control method, and the one-pass method on different benchmarks varying the error threshold from 10% to 2.5%. The results are normalized with respect to the one-pass method under the same approximating error. We estimate the energy consumption of our proposed method by scaling the energy consumption of NPU in work [4] based on the invocation. This is a valid estimation, as the proposed method is merely the same as the original NPU design. Compared to the one-pass method, the average energy reduction of our proposed method is about 1.23×, 1.59×, 1×, and 1.3× on

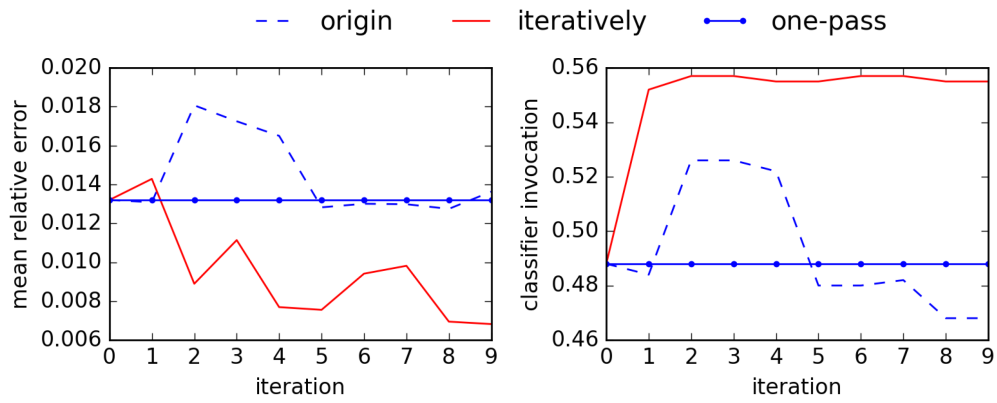
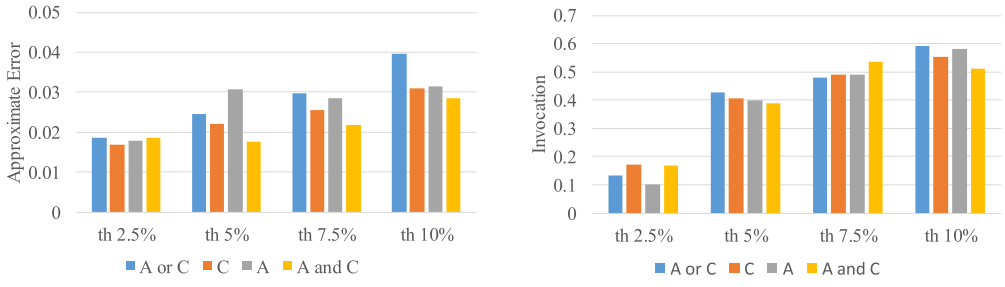


Fig. 12. Results on MRE and Invocation for blackscholes with a fixed threshold of 5% and varying iteration.



(a) Approximate error.

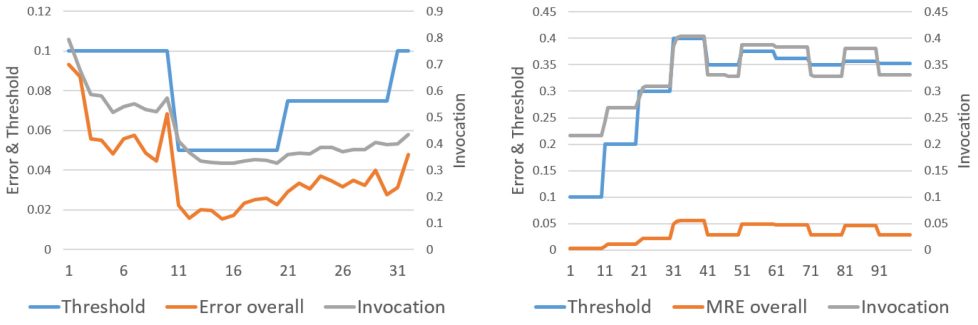
(b) Invocation.

Fig. 13. Comparison in Error and Invocation using different data selection policies.

the blackscholes, jmeint, fft, and bessel benchmarks, respectively. Compared to the statistical-control method, the average energy reduction of our proposed method is about 1.59×, 1.22×, 1.19×, and 1.28× on the blackscholes, jmeint, fft, and bessel benchmarks, respectively. The energy consumption is larger than the one-pass method and the statistical-control method when the threshold is 10% on the blackscholes benchmark. The reason is that when the threshold is 10% on the blackscholes benchmark, the invocation degrades for the approximating accuracy, which makes the energy consumption of our iterative co-training method in such scenarios higher than that of the one-pass method. Our proposed system gains 4.8×, 3.6×, 3×, and 3.9× on the jmeint, blackscholes, fft, and bessel benchmarks' energy reduction against the accurate data path, as the one-pass method [4] yields a 3× average energy reduction against the unmodified CPU.

Figure 12 shows a deeper view of the iterative co-training process by showing the change of MRE and Invocation in each iteration. We also show the result of one-pass training, which remains unchanged. The MRE and Invocation of the one-pass method fluctuate around those of the one-pass training. This result shows that more training effort is in vain without any judicious selection of the training data. We can observe that the iterative co-training process converges to fairly good MRE and Invocation after four iterations and two iterations, respectively.

Figure 13 shows the results using different categories of data in the iterative co-training in the blackscholes benchmark. We use Approximate Error and Invocation to represent the effectiveness of different data selection policies. We can see in blackscholes that the category "A and C" exceeds others in terms of error and invocation.



(a) Blackscholes. (b) Fft.

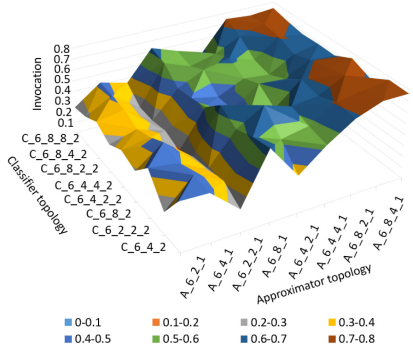
Fig. 14. Results for iterative co-training + threshold tuning, varying the iterations.

Figure 14 shows the results derived by the integration of iterative co-training and the threshold tuning. The initial threshold is set to 0.1, and the error bound is defined as 0.05. The *Y* axes represent the threshold/MRE (left *Y* axis) and Invocation (right *Y* axis). In blackscholes, the MRE violates the error bound after one-pass training. The threshold is first tuned down to be below the error bound. The threshold drags both MRE and Invocation down. Then, the tuning algorithm increases the threshold with a smaller magnitude and pushes up both MRE and Invocation. Finally, the MRE satisfies the error bound, and the Invocation is salvaged. On the contrary, in fft, both MRE and Invocation are low after the one-pass training. The proposed algorithm iteratively tunes the threshold in coordination with iterative co-training, resulting in 73% improvement on Invocation without violating the error bound.

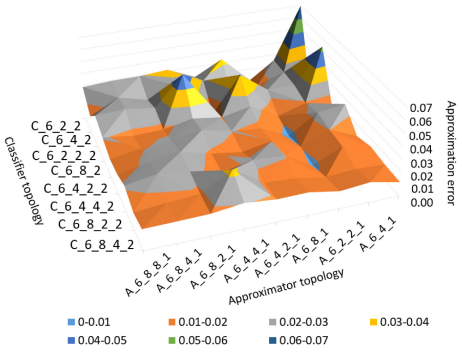
Figure 15(a) shows the Invocation in different topologies of the accelerator and classifier. We find that the topology of the accelerator has a global impact on the Invocation; as the number of synaptic decreases, the Invocation has an obvious descending trend. The topology of the classifier, however, has a local impact on the Invocation; interestingly, we find that the maximum Invocation can be reached under the same approximator no matter what classifier we use. Figure 15(b), however, shows that Approximating Error is mainly dominated by the topology of the classifier. Only when the classifier is super simple can the accelerator have some local impact on the Approximating Error. These results support our motivation to the comprehensive topology exploration for the classifier-accelerator hybrid architecture.

Furthermore, we use the preceding two results to evaluate our topology exploration method. We define the error bound to be 2.5%. Based on Figure 15(b), we discard those topology combinations incurring violations of error bound and enforce the corresponding invocation to be 0. Then, we can build an invocation graph as shown in Figure 15(c). Red color indicates high invocation. We plot the topology combinations searched by the proposed topology exploration algorithm, denoted as the dotted line. The topology search algorithm first searches the topology combination with a large classifier and large accelerator, which can deliver the large invocation. After that, the search algorithm keeps exploring the topology combination and finally stops at a point with the relatively small topology of the classifier-accelerator and very high invocation.

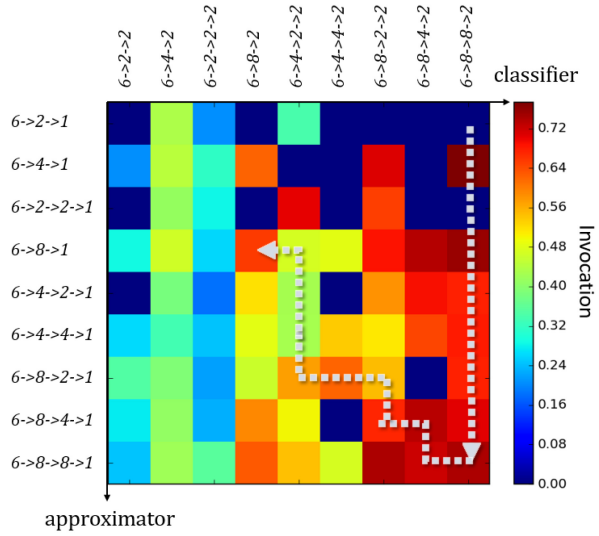
Figure 16 shows the case study of the bessel benchmark for the topology search using the proposed exponential enlarge and linear decrease with fine tuning. We start from the minimal topology, where the approximator is 2->1->1 and the classifier is 2->1->2. We only use four steps to get the large and error satisfied structure (the approximator 2->4->4->1 and the classifier 2->4->4->2) rather than beginning from the minimum structure and increasing one by one. Then, we use



(a) Invocation in different topologies.



(b) Approx. Error in different topologies.



(c) Topology finding.

Fig. 15. Performance in different topologies (blackscholes).

four steps to get the appropriate structure (the approximator 2->4->4->1 and the classifier 2->4->2) by reducing the structure one by one. In short, we use only eight steps to get the appropriate structure.

We compare our two topology exploration methods—linear enlarge with NN retraining algorithm (short for "linear enlarge") and exponential enlarge and linear decrease with NN fine tuning algorithm (short for "exponential enlarge")—with the brute force search algorithm in different benchmarks, as shown in Figure 17. Total training steps of different topologies measure the training overhead. As the brute force algorithm searches all possible topologies of approximator and classifier, the brute force algorithm costs the largest training overhead. The "linear enlarge" algorithm performs better than the brute force algorithm, and the "exponential enlarge" algorithm outperforms the others. We use a synthetic metric, Invocation/Synapses, to evaluate the final topology explored by different algorithms. This metric takes both invocation and size of topology into account. The result is normalized to that derived from the brute force algorithm, as shown in

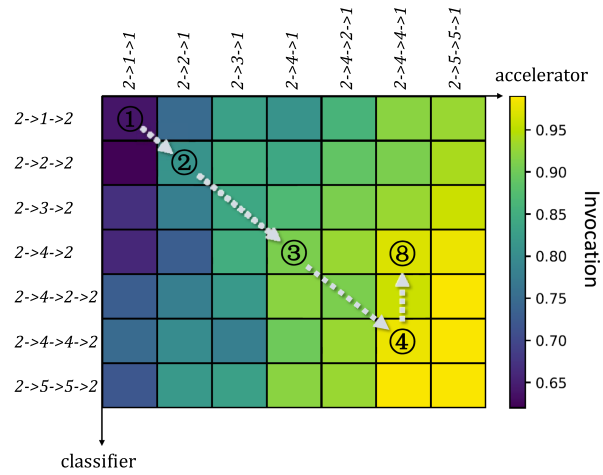


Fig. 16. Topology finding with the exponential enlarge algorithm.

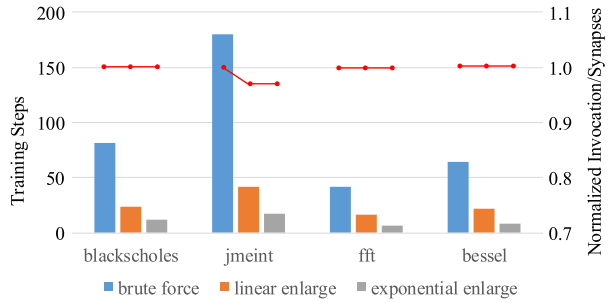


Fig. 17. Total training steps using different topology exploration methods.

Figure 17. The final results show that both of our proposed algorithms explore the same topology as the brute force in blackscholes, fft, and bessel. However, since the search space in jmeint is too large, our two topology exploration algorithms explore different topologies from brute force, but the Invocation/Synapses is almost the same as the brute force algorithm.

7 CONCLUSION AND FUTURE WORK

In this article, we proposed a novel training framework to provide a novel iterative co-training process for classifier-accelerator hybrid approximate computing architecture with judicious training data categorization and selection and with dynamic threshold tuning. We also proposed more than one topology exploration method to find the sweet point in the tradeoff between computation quality and energy-efficiency algorithms. The experimental results show that we can get about 1.5×, 1.25×, 2×, and 1.58× improvement on the Invocation/Error in blackscholes, jmeint, fft, and bessel benchmarks, respectively. The average energy reduction is about 1.6×, 1.2×, 1×, and 1.3× in blackscholes, jmeint, fft, and bessel benchmarks, respectively. In summary, our experimental results show significant improvement on the quality and energy efficiency compared to the existing NN-based approximate computing frameworks.

APPENDIX

A MATH PROOF

This appendix gives a mathematical proof to demonstrate effectiveness and practicability of our iterative co-training method theoretically. Two optimization objectives—accuracy and invocation—are considered here to achieve better quality and energy efficiency. We can prove that if one of the two optimization objectives remains unchanged for two successive training iterations, the other one increases.

A.1 Assumption

First, we assume that the approximator is always well trained, which means that at iteration i, the approximator can perfectly achieve minimum mean squared error (MSE).

$$MSE = \frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_i(x) - y(x)|^2.$$

Here, $\mathcal{D}^i$ denotes the input space at iteration i, $f_i(x)$ presents the approximator, and $y(x)$ is the precise result of given input x. As $f_i(x)$ is well trained using $\mathcal{D}^i$, so compared to $f_{i-1}(x)$, we know that

$$\frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_i(x) - y(x)|^2 \leq \frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_{i-1}(x) - y(x)|^2. \tag{1}$$

Second, we assume that the classifier can perfectly discriminate the input data space, so we can always select those data that satisfy the error bound for the next iteration's training.

$$\forall x \in \mathcal{D}^{i+1},\ |f_i(x) - y(x)| \leq err. \tag{2}$$

Here, err denotes the error bound. And, at last, we assume that the probability distribution of the training set is the same as the test set, so we can use $\mathcal{D}^{i+1}$ to evaluate the performance of our method at iteration i.

$$acc_i = 1 - \frac{1}{|\mathcal{D}^{i+1}|} \sum_{x \in \mathcal{D}^{i+1}} |f_i(x) - y(x)| \tag{3}$$

$$inv_i = \frac{|\mathcal{D}^{i+1}|}{|\mathcal{D}^0|}. \tag{4}$$

Here, acc_i denotes the accuracy at iteration i, and inv_i denotes the invocation. We train the approximator using all training data at iteration 0, so $\mathcal{D}^0$ denotes the whole training set.

A.2 Proof of Ascending Accuracy If Invocation Is Unchanged

First, we split $\mathcal{D}^{i+1}$ into two parts, $\mathcal{D}^{i+1} \cap \mathcal{D}^i$ and $\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}$, where $\widetilde{\mathcal{D}^i}$ denotes the complementary set of $\mathcal{D}^i$. Apparently, we can rewrite the evaluation MSE of $f_i(x)$ at $\mathcal{D}^{i+1}$ as

$$\frac{1}{|\mathcal{D}^{i+1}|} \sum_{x \in \mathcal{D}^{i+1}} |f_i(x) - y(x)|^2 = \frac{1}{|\mathcal{D}^{i+1}|} \left[\sum_{x \in \mathcal{D}^{i+1} \cap \mathcal{D}^i} |f_i(x) - y(x)|^2 + \sum_{x \in \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}} |f_i(x) - y(x)|^2 \right]. \tag{5}$$

Similarly, we can also split $\mathcal{D}^i$ into two parts, $\mathcal{D}^i \cap \mathcal{D}^{i+1}$ and $\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}$, so that the training MSE of $f_i(x)$ at $\mathcal{D}^i$ is

$$\frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_i(x) - y(x)|^2 = \frac{1}{|\mathcal{D}^i|} \left[\sum_{x \in \mathcal{D}^i \cap \mathcal{D}^{i+1}} |f_i(x) - y(x)|^2 + \sum_{x \in \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}} |f_i(x) - y(x)|^2 \right]. \tag{6}$$

Considering Equations (5) and (6), we can find that the key problem here is the relationship between $\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}$ and $\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}$, which separately present for the data moved from the right-hand side of the error bound to the left-hand side after training and, contrarily, from left to right. In some extreme cases, if $|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|$ is far greater than $|\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}|$, which means that more data satisfy the error bound, then the invocation at iteration i will increase substantially, but accuracy may decrease. Unfortunately, due to the randomness of NNs, we can hardly explore all of those cases. To give a rigorous proof, we set one of the two optimization objectives unchanged and try to prove the ascending of the other one, which can also demonstrate effectiveness of our method. Thus, here, if invocation remains unchanged, we will get

$$|\mathcal{D}^{i+1}| = |\mathcal{D}^i|$$

$$|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}| = |\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}|.$$

And according to Equation (2), we know that $\mathcal{D}^{i+1}$ presents for those input data that satisfy the error bound after i iteration's training, so we have

$$\forall x \in \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i},\ |f_i(x) - y(x)| \leq err.$$

On the contrary, $\widetilde{\mathcal{D}^i}$ presents for those input data with predicted error larger than the error bound at iteration i, so obviously we have

$$\forall x \in \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}},\ |f_i(x) - y(x)| > err.$$

Thus, we can derive

$$\sum_{x \in \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}} |f_i(x) - y(x)|^2 \leq \sum_{x \in \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}} |f_i(x) - y(x)|^2.$$

Combining Equation (5) and Equation (6), we can get

$$\frac{1}{|\mathcal{D}^{i+1}|} \sum_{x \in \mathcal{D}^{i+1}} |f_i(x) - y(x)|^2 \leq \frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_i(x) - y(x)|^2.$$

And using Equation (1), we have

$$\frac{1}{|\mathcal{D}^{i+1}|} \sum_{x \in \mathcal{D}^{i+1}} |f_i(x) - y(x)|^2 \leq \frac{1}{|\mathcal{D}^i|} \sum_{x \in \mathcal{D}^i} |f_{i-1}(x) - y(x)|^2. \tag{7}$$

This inequality shows that the evaluation MSE of $f_i(x)$ is less than that of $f_{i-1}(x)$. According to Equation (3) and the equivalence of L1 norm and L2 norm, we can get the final conclusion:

$$acc_i \geq acc_{i-1}, \quad if\ inv_i = inv_{i-1}. \tag{8}$$

A.3 Proof of Ascending Invocation If Accuracy Is Unchanged

We can also prove that if accuracy remains unchanged between two successive iterations, invocation increases. We rewrite Equation (5) as follows:

$$
\begin{aligned}
MSE(f_i, \mathcal{D}^{i+1}) &= \frac{1}{|\mathcal{D}^{i+1}|} \sum_{x \in \mathcal{D}^{i+1}} |f_i(x) - y(x)|^2 \\
&= \frac{1}{|\mathcal{D}^{i+1}|} \left[\sum_{x \in \mathcal{D}^{i+1} \cap \mathcal{D}^i} |f_i(x) - y(x)|^2 + \sum_{x \in \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}} |f_i(x) - y(x)|^2 \right] \\
&= \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} \frac{1}{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|} \sum_{x \in \mathcal{D}^{i+1} \cap \mathcal{D}^i} |f_i(x) - y(x)|^2 \\
&\quad + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} \frac{1}{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|} \sum_{x \in \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}} |f_i(x) - y(x)|^2 \\
&= \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i) + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}).
\end{aligned} \tag{9}
$$

Similarly, we can rewrite Equation (6):

$$
MSE(f_i, \mathcal{D}^i) = \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) + \frac{|\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}). \tag{10}
$$

Here $MSE(f, \mathcal{D})$ presents for MSE of approximator $f(x)$ at dataset $\mathcal{D}$. And if accuracy stays the same, we have

$$
MSE(f_i, \mathcal{D}^{i+1}) = MSE(f_{i-1}, \mathcal{D}^i).
$$

According to Equation (1), we can get

$$
MSE(f_i, \mathcal{D}^{i+1}) \geq MSE(f_i, \mathcal{D}^i). \tag{11}
$$

Then, after combining Equations (9) and (10), we can derive that

$$
\begin{aligned}
MSE(f_i, \mathcal{D}^{i+1}) &\geq MSE(f_i, \mathcal{D}^i) \\
&\geq \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) + \frac{|\mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \\
&\geq \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) + \frac{|\mathcal{D}^i| - |\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \\
&\geq \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) + \left(1 - \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|}\right) MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \\
&\geq \frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} \left[MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) - MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \right] \\
&\quad + MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}).
\end{aligned} \tag{12}
$$

According to Equation (2), we have

$$
MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \geq MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i).
$$

And $MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})$ equals $MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i)$ if and only if $\mathcal{D}^i$ equals $\mathcal{D}^{i+1}$, which leads to our conclusion directly. Thus, we only consider $MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) > MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i)$ here:

$$
MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) - MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) < 0.
$$

Then, if we divide the left and right part of Equation (12) by $MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) - MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})$, we have

$$\begin{aligned}\frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} &\geq \frac{MSE(f_i, \mathcal{D}^{i+1}) - MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})}{MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1}) - MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})} \\ &\geq \frac{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1})}{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^i \cap \mathcal{D}^{i+1})}.\end{aligned} \tag{13}$$

Then, we can rewrite $MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})$ as

$$\begin{aligned}MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) &= \frac{|\mathcal{D}^{i+1}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \\ &= \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i| + |\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \\ &= \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}).\end{aligned} \tag{14}$$

And according to Equation (9), we know that

$$MSE(f_i, \mathcal{D}^{i+1}) = \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i) + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}).$$

Thus, combining Equations (9) and (14), we can get

$$\begin{aligned}&MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1}) \\ =&\left[\frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}})\right] \\ &- \left[\frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i) + \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|} MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i})\right] \\ =&\frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|}\left[MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i)\right] \\ &+ \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|}\left[MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i})\right].\end{aligned} \tag{15}$$

Based on the Equation (13) and Equation (15), we can get

$$\frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} \geq \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|} + \frac{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i})}{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i)} \frac{|\mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i}|}{|\mathcal{D}^{i+1}|}. \tag{16}$$

And according to Equation (2), we have

$$MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \geq MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i})$$
$$MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) \geq MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i).$$

Obviously, we can derive that

$$\frac{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \widetilde{\mathcal{D}^i})}{MSE(f_i, \mathcal{D}^i \cap \widetilde{\mathcal{D}^{i+1}}) - MSE(f_i, \mathcal{D}^{i+1} \cap \mathcal{D}^i)} > 0.$$

Then, according to Equation (16), we have

$$\frac{|\mathcal{D}^i \cap \mathcal{D}^{i+1}|}{|\mathcal{D}^i|} \geq \frac{|\mathcal{D}^{i+1} \cap \mathcal{D}^i|}{|\mathcal{D}^{i+1}|}$$
$$|\mathcal{D}^{i+1}| \geq |\mathcal{D}^i|.$$

Apparently, according to the definition of invocation, Equation (4), we can get the conclusion that

$$inv_i \geq inv_{i-1}, \quad if\, acc_i = acc_{i-1}. \tag{17}$$

According to the preceding math prove, we can get the conclusion that if accuracy remains unchanged for two successive training iterations, invocation will increase and vice versa. This conclusion demonstrates effectiveness and practicability of our method compared to previous works.

REFERENCES

[1] Mohammed Alawad and Mingjie Lin. 2016. Survey of stochastic-based computation paradigms. *IEEE Transactions on Emerging Topics in Computing* 7, 1 (2016), 98–114.

[2] Woongki Baek and Trishul M. Chilimbi. 2010. Green: A framework for supporting energy-conscious programming using controlled approximation. *ACM SIGPLAN Notices* 45, 6 (2010), 198–209.

[3] Zidong Du, Krishna Palem, Avinash Lingamneni, Olivier Temam, Yunji Chen, and Chengyong Wu. 2014. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *Proceedings of the 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC'14)*. IEEE, Los Alamitos, CA, 201–206.

[4] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, Los Alamitos, CA, 449–460.

[5] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. 2016. Proactive control of approximate programs. *Architectural Support for Programming Languages and Operating Systems* 51, 4 (2016), 607–621.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, et al. 2014. DaDianNao: A machine-learning supercomputer. In *Proceedings of the International Symposium on Microarchitecture*. IEEE, Los Alamitos, CA, 609–622.

[7] Jie Han and Michael Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings of the 2013 18th IEEE European Test Symposium (ETS'13)*. IEEE, Los Alamitos, CA, 1–6.

[8] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 2016 ACM/IEEE 43rd International Symposium on Computer Architecture (ISCA'16)*. 243–254.

[9] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems, Volume 1 (NIPS'15)*. 1135–1143.

[10] Muhammad Abdullah Hanif, Alberto Marchisio, Tabasher Arif, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. 2018. X-DNNs: Systematic cross-layer approximations for energy-efficient deep neural networks. *Journal of Low Power Electronics* 14, 4 (2018), 520–534. DOI: https://doi.org/10.1166/jolpe.2018.1575

[11] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4, 2 (1991), 251–257.

[12] M. Imani, R. Garcia, A. Huang, and T. Rosing. 2019. CADE: Configurable approximate divider for energy efficiency. In *Proceedings of the 2019 Design, Automation, and Test in Europe Conference and Exhibition (DATE'19)*. 586–589. DOI: https://doi.org/10.23919/DATE.2019.8715112

[13] M. Imani, A. Rahimi, and T. S. Rosing. 2016. Resistive configurable associative memory for approximate computing. In *Proceedings of the 2016 Design, Automation, and Test in Europe Conference and Exhibition (DATE'16)*. 1327–1332.

[14] Mohsen Imani, Alice Sokolova, Ricardo Garcia, Andrew Huang, Fan Wu, Baris Aksanli, and Tajana Rosing. 2019. ApproxLP: Approximate multiplication with linearization and iterative error control. In *Proceedings of the 56th Annual Design Automation Conference (DAC'19)*. ACM, New York, NY, Article 159, 6 pages. DOI: https://doi.org/10.1145/3316781.3317774

[15] Daya S. Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. 2015. Rumba: An online quality management system for approximate computing. *ACM SIGARCH Computer Architecture News* 43, 3 (2015), 554–566.

[16] Y. Kim, S. Venkataramani, K. Roy, and A. Raghunathan. 2016. Designing approximate circuits using clock overgating. In *Proceedings of the 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. 1–6. DOI: https://doi.org/10.1145/2897937.2898005

[17] J. Liang, J. Han, and F. Lombardi. 2013. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers* 62, 9 (2013), 1760–1771. DOI: https://doi.org/10.1109/TC.2012.146

[18] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. 2016. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. *ACM SIGARCH Computer Architecture News* 44, 66–77.

[19] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy. 2011. Design of voltage-scalable meta-functions for approximate computing. In *Proceedings of the 2011 Design, Automation, and Test in Europe Conference and Exhibition (DATE'11)*. 1–6. DOI: https://doi.org/10.1109/DATE.2011.5763154

[20] Debabrata Mohapatra, Georgios Karakonstantis, and Kaushik Roy. 2009. Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 195–200.

[21] Abbas Rahimi, Luca Benini, and Rajesh K. Gupta. 2016. An approximation workflow for exploiting data-level parallelism in FPGA acceleration. In *Proceedings of the 2016 Design, Automation, and Test in Europe Conference and Exhibition (DATE'16)*. 1279–1284.

[22] Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. 2013. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, New York, NY, 13–24.

[23] Muhammad Shafique, Rehan Hafiz, Muhammad Usama Javed, Sarmad Abbas, Lukas Sekanina, Zdenek Vasicek, and Vojtech Mrazek. 2017. Adaptive and energy-efficient architectures for machine learning: Challenges, opportunities, and research roadmap. In *Proceedings of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'17)*. IEEE, Los Alamitos, CA, 627–632.

[24] S. Sinha and W. Zhang. 2016. Low-power FPGA design using memoization-based approximate computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 8 (2016), 2665–2678. DOI: https://doi.org/10.1109/TVLSI.2016.2520979

[25] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. 2015. Texture cache approximation on GPUs. In *Proceedings of the Workshop on Approximate Computing across the Stack*.

[26] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, New York, NY, 1–12.

[27] Ting Wang, Qian Zhang, Nam Sung Kim, and Qiang Xu. 2016. On effective and efficient quality management for approximate computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 156–161.

[28] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation* 26, 2 (2007), 289–315. DOI: https://doi.org/10.1007/s00365-006-0663-2

[29] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. 2017. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design & Test* 34, 2 (2017), 60–68.

[30] Hang Zhang, Mateja Putic, and John Lach. 2014. Low power GPGPU computation with imprecise hardware. In *Proceedings of the Design Automation Conference*. 1–6.

Received December 2018; revised June 2019; accepted June 2019