IEEE Open Journal of the
Computer Society

Received 6 January 2021; accepted 10 January 2021. Date of publication 14 January 2021;
date of current version 11 February 2021. The review of this paper was arranged by Associate Editor Pedro Reviriego.

Digital Object Identifier 10.1109/OJCS.2021.3051643

A Hardware/Software Co-Design Methodology for Adaptive Approximate Computing in clustering and ANN Learning

PENGFEI HUANG [1], CHENGHUA WANG[1], WEIQIANG LIU [1] (Senior Member, IEEE), FEI QIAO [2] (Member, IEEE), AND FABRIZIO LOMBARDI [3] (Fellow, IEEE)

(Invited Paper)

[1] College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
[2] Department of Electronic Engineering, Tsinghua University, Beijing, China
[3] Department of Electrical Engineering, Northeastern University, Boston 40125 USA

CORRESPONDING AUTHOR: WEIQIANG LIU (e-mail: liuweiqiang@nuaa.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grants 62022041 and 61871217.

ABSTRACT As one of the most promising energy-efficient emerging paradigms for designing digital systems, approximate computing has attracted a significant attention in recent years. Applications utilizing approximate computing (AxC) can tolerate some loss of quality in the computed results for attaining high performance. Approximate arithmetic circuits have been extensively studied; however, their application at system level has not been extensively pursued. Furthermore, when approximate arithmetic circuits are applied at system level, error-accumulation effects and a convergence problem may occur in computation. Multiple approximate components can interact in a typical datapath, hence benefiting from each other. Many applications require more complex datapaths than a single multiplication. In this paper, a hardware/software co-design methodology for adaptive approximate computing is proposed. It makes use of feature constraints to guide the approximate computation at various accuracy levels in each iteration of the learning process in Artificial Neural Networks (ANNs). The proposed adaptive methodology also considers the input operand distribution and the hybrid approximation. Compared with a baseline design, the proposed method significantly reduces the power-delay product while incurring in only a small loss of accuracy. Simulation and a case study of image segmentation validate the effectiveness of the proposed methodology.

INDEX TERMS Approximate computing, approximate multiplier, k-means clustering, semi-supervised learning.

I. INTRODUCTION

As computer systems become pervasive, computing workloads have significantly increased due to new areas such as big data and IoT, hence the computing landscape has become more complex over the last decade. Although per-transistor speed has continuously improved, energy and power still remain a significant hurdle for chip design. Moreover, schemes such as the dark silicon limit the amount of usable silicon [1]. Significant efforts have already been made to improve energy efficiency at various levels, from software, to architecture all the way down to circuit and device levels. Among these techniques, approximate computing relies that a growing body of applications are inherently error resilient and energy requirements can benefit from a slight or acceptable quality loss [2], [3]. Based on the observation that inputs and outputs of some algorithms can tolerate imprecision, some exact operations may have only a small effect on the final quality. Approximate computing techniques have been extensively studied at both hardware (such as circuit designs and computing architectures) and software levels [4]–[6]. As key components in arithmetic circuits, several approximate adders [7] and multipliers [8] have been proposed; these circuits yield incorrect results for some input combinations.

Multiplication is one of the most common and complex op-erations, critical in computation for many applications. Multipliers are more complex than adders and therefore, they incur in a higher energy consumption; moreover, they are also slower than adders. Approximated multipliers have been widely discussed in recent years [9]–[16]. Approximate Booth multipliers have been proposed in [9], [10] by approximating the partial product generation process. [11] has considered a high-performance and low-power approximate partial product accumulation tree for a multiplier using a newly designed approximate adder. [12] has presented a design methodology for implementing accurate and approximate signed array-multipliers. In [16], efficient approximate redundant binary multipliers for error-tolerant applications with high accuracy have been proposed; however, most of these works only consider approximate designs at circuit level, so error effects at system or algorithm level are not fully addressed. [17] has proposed an algorithm-level method that applies approximate computing to an application. By applying approximation techniques to the most computationally intensive blocks, energy reduction can be obtained at a limited accuracy reduction. [18] has provided a dynamic auto-tuning framework for self-aware approximate computing at software level. In [19], an RnR (reduce and rank) accelerator for approximate computing has been proposed to reduce energy consumption. Three approximation strategies have been presented for the RnR computation pattern at algorithm level.

Approximate computing cannot be fully exploited by only considering hardware (circuits, architecture and memory) or software (application, algorithms and compile stack); the fundamental nature of machine learning (ML) workloads requires that the barriers between abstract layers to be clearly broken, so that an efficient ML system can be realized by AxC including cross-layer collaborative design. Therefore, hardware/software (HW/SW) co-design [5], [20], [21] must be considered to change the abstractions and relationships between hardware and software for a trade-off between accuracy and efficiency.

This paper is an extension of our previous work presented in [22]; the main differences and novel contributions are summarized as follows:

- Adaptive approximate computing for both supervised and unsupervised learning is discussed and hybrid approximate computing modes are presented in detail.
- In unsupervised learning, we present a gradient-oriented adaptive approximate k-means clustering; the semi-supervised adaptive approximate k-means clustering algorithm is also treated in more detail.
- In supervised learning, we propose an operand swapper and greedy approximate computing algorithms for weight stationary in artificial neural networks (ANNs).
- By considering error propagation, a generalized strategy is proposed for the accuracy in approximate computing.
- The advantages of the proposed method are theoretically analyzed and experimentally demonstrated through simulation.

The remaining part of this paper is organized as follows: In Section II, the preliminary is briefly reviewed. Section III discusses the analysis for adaptive approximate computing. Section IV presents the proposed HW/SW co-design method for both unsupervised, semi-supervised and supervised learning. In Section V, the error analysis and the simulation results are provided. The application of the proposed algorithm to image processing is given in Section 6. Section 7 concludes the paper.

II. PRELIMINARY AND REVIEW

A. SUPERVISED LEARNING AND UNSUPERVISED LEARNING

1) SUPERVISED LEARNING

Supervised learning trains a data sample from the data source with the correct classification already assigned. It infers a function from the labeled training data; for input variables (X) and output variables (Y), the algorithm employs the mapping function for the purpose of learning.

$$Y = f(X) \tag{1}$$

The goal is to approximate the mapping function, such that when a new input data (X) is provided, the output variable (Y) can be predicted. These techniques are used in feedforward or multilayer perceptron (MLP) models.

2) UNSUPERVISED LEARNING

Self-organizing neural networks use unsupervised learning algorithms to identify hidden patterns in unlabeled input data. This kind of unsupervised refers to the ability to learn and organize information to evaluate potential solutions [23]. When there is only input data (X) with no corresponding output variables, the goal of unsupervised learning is to model the underlying structure or distribution in data to further understand such data. In unsupervised learning, the lack of directionality of the learning algorithm is sometimes advantageous because it allows the algorithm to go back to patterns that have not been previously considered. These techniques are used in association and clustering models.

B. MULTILAYER PERCEPTRON (MLP) MODELS

The MLP is a type of feedforward ANN. The term MLP is ambiguous; it is used for any feedforward neural network, but it also strictly refers to a network composed of multilayer perceptron (with threshold activation). A MLP consists of at least three layers of nodes: input, hidden and output layers. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. A MLP uses a supervised learning technique called backpropagation for training. Its multilayer and non-linear activation features distinguish MLP from a linear perceptron because it can distinguish non-linearly separable data [24].

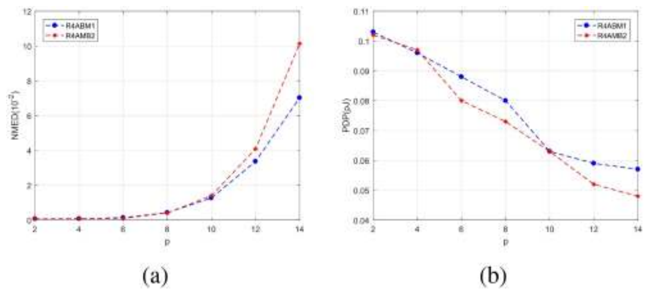

FIGURE 1. **(a) The error characteristics and (b) hardware requirement of the R4ABMs [10] with different approximate factors.**

C. SEMI-SUPERVISED K-MEANS

Semi-supervised learning is a type of supervised learning that uses unlabeled data to train a small amount of labeled data and a large amount of unlabeled data; k-means clustering divides n observations into k clusters, and each observation belongs to the nearest mean cluster. Semi-supervised k-means uses the paired constraints provided by the user to learn the appropriate distance metric in the feature space, or guide the clustering algorithm to develop the correct clustering direction. The semi-supervised k-means objective function $\mathcal{J}_{obj}$ using feature constraints is given as follows:

$$min\mathcal{J}_{obj} = D(\vec{X}, \vec{C}) + \mathcal{W}\varphi_D(\vec{X}_{pw}) \tag{2}$$

where, D is the distortion function, $\vec{X}$ and $\vec{C}$ are the samples and cluster centroids, $\mathcal{W}$ is the set of weights of the penalty for violating the feature constraints, and φ_D is an increasing function of the distance between two samples and $\vec{X}_{pw}$ is the sample with pair-wise constraints. For convenience, $\mathcal{W}$ is chosen as 1. In the k-means clustering algorithm, a 5% loss in classification accuracy permits a 50 times energy saving compared to the fully accurate classification [25]; therefore, k-means is a typical application that trades accuracy for performance.

D. RADIX-4 APPROXIMATE BOOTH MULTIPLIER

In this paper, 8-bit approximate Booth multipliers of [10] are used and therefore, they are discussed for completeness. In [10], the complexity of the approximate Booth encoder has been reduced by at least an order of magnitude compared with an exact design. Furthermore, the so-called approximation factor $p\,(p = 1, 2, \ldots, 2N)$ is defined as the number of least significant partial product columns that are generated by the approximate Booth encoders as the approximate circuit (*i.e.*, approximate radix-4 Booth encoding (R4ABM)) can be used in all or only part of the partial product generation process. Fig. 1 presents a plot of the power-delay product (PDP) and the normalized mean error distance (NMED) under different approximation factors (p). It shows that the NMED increases, while the PDP of R4ABM decreases with an increase of p. Therefore, the impact of p on the performance of an approximate multiplier is that the increase in inaccuracy can be traded off for a higher energy efficiency by using an approximate multiplier at a higher value of p. A larger p leads to a simpler logic hardware; this results in a less complex hardware but more errors will be present. For a given data path in an implementation, p establishes a design space in which every multiplication can utilize a value of p such that this approximate multiplier ensures the desired computing accuracy while also achieving the highest energy efficiency.

TABLE 1. **Arithmetic Calculations of Error Propagation**

Type	Example	Standard Deviation (δ_Q)
Addition or Subtraction	$Q = a + b - c$	$\delta_Q = \sqrt{\delta_a^2 + \delta_b^2 + \delta_c^2}$
Multiplication	$Q = a * b$	$\frac{\delta_Q}{Q} = \sqrt{(\frac{\delta_a}{a})^2 + (\frac{\delta_b}{b})^2}$
Division	$Q = \frac{a}{b}$	$\frac{\delta_Q}{Q} = \sqrt{(\frac{\delta_a}{a})^2 + (\frac{\delta_b}{b})^2}$
Multiplication with constant	$Q = C * x$	$\delta_Q = \lvert C\rvert\delta_x$
Any function	$Q = f(x)$	$\delta_Q = \lvert\frac{\partial f}{\partial x}\rvert\delta_x$

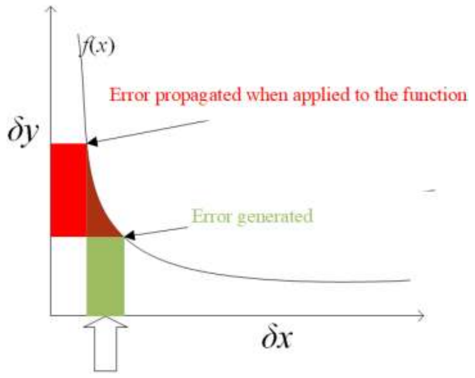

FIGURE 2. **General plot for error propagation.**

E. ERROR PROPAGATION

Usually, an error is quantified by the standard deviation of the expected value. When a calculation has multiple variables to solve, propagation of errors must be carefully considered. Consider a calculation with 4 input variables, i.e. a, b, c and x. The desired outcome is given by Q, so Q depends on these variables. As examples, the exact expressions of some calculations are given in Table 1. δ_a, δ_b, δ_c and δ_x are the standard deviations of these variables. Table 1 shows that addition and subtraction result in absolute standard deviations, while multiplication results in relative standard deviations. Fig. 2 shows that a small error in an independent input variable of a function may result in a larger error at the output.

F. DYNAMICALLY CONFIGURABLE APPROXIMATE COMPUTING

Approximate computing is application dependent; therefore, different designs have different features. Ansari *et al*. [26] has identified the statistically most relevant and critical features of 600 approximate multipliers to ensures the best selection. Few approximate designs of arithmetic circuits have been proposed to configure the approximation level [10], [11]. Most of these works consider configuration only at hardware level. Imani *et al*. [27] has proposed a multi-level configuration of approximation during run time to better adjust each application error tolerance but only in software. Many approximate designs found in the technical literature rely on static data in the approximation process [28]; however, workloads are often dynamic, so this type of approach has a very limited applicability. Mazahir *et al*. [29] has proposed a scheme to bound errors using detection and correction for specific workloads based on selected approximate modules to completely or partially cancel the generated error. Xu and Schafer [30] has presented an adaptable architecture that enables or disables approximations. This architecture tunes itself at runtime based on the workload for variable-to-constant or variable-to-variable adaptive approximation in hardware. The R4ABMs [10] has utilized the approximate factor p to select an approximate multiplier. The adaptive approximate data path design can be defined as an optimization problem:

$$
\begin{aligned}
max\Lambda = \alpha & \sum_{datapath} \Psi(EN_p(\vec{X},\vec{Y}) \\
& + \beta \left\| \sum_{datapath} \psi(ED_p(\vec{X},\vec{Y})) \right\| \\
s.t. & \begin{cases} EN_p = \dfrac{P_{DA}(\vec{X},\vec{Y}) - \widetilde{P}_{DA}(\vec{X},\vec{Y},p)}{P_{DA}(\vec{X},\vec{Y})} \\ ED_p = \dfrac{M(\vec{X},\vec{Y}) - \widetilde{M}(\vec{X},\vec{Y},p)}{M(\vec{X},\vec{Y})} \end{cases}
\end{aligned}
\tag{3}
$$

where Λ is the total benefit from the adaptive trade-off between the energy and accuracy. EN_p is the hardware saving with the factor p. ED_p is the error incurred by using the factor p. $\vec{X}$ and $\vec{Y}$ are the operands. α and β are the coefficient for the first and second parts respectively. Ψ and ψ are the application-aware mapping functions in order to better measure the advantage for an approximate design in terms of dynamic workloads. $P_{DA}(\vec{X},\vec{Y})$ is the hardware for an exact design (for different measures such as power, area and delay), while $\widetilde{P}_{DA}$ is for an approximate design. $M(\vec{X},\vec{Y})$ is the exact product, while $\widetilde{M}$ is the approximate product. EN_p is normally greater than 0, ED_p may be positive or negative. Along all data paths, ED can be accumulated or counterbalanced. Fig. 3 shows an approximate clustering with factor $p \in \{2, 4, 6\}$ achieves the same result as for the exact case, and the results are similar when the factor is greater than 6. Larger p leads to simpler logic, and so it results in lower hardware but not always more error.

G. FIXED-POINT QUANTIZATION

A higher precision in approximate computation usually incurs in a larger penalty. Recently, [31]–[33] have proposed error-tolerance at data level by utilizing fixed-point quantization. Based on the assumption that the dynamic range of the precision of floating points from a data set is bounded, an approach for fixed-point quantization is proposed. Floating-point numbers can be extended or compressed as fixed-point numbers according to the desired function as follows:

$$
\begin{cases} k * x_{max} + b = y_{max} \\ k * x_{min} + b = y_{min} \end{cases} \tag{4}
$$

where, x_{max} and x_{min} are the max and min values of the floating-point inputs, y_{max} and y_{min} are the max and min of the available fixed-point values. Consider an 8-bit width as example; y_{max} and y_{min} are given by $127(2^7 - 1)$ and -128(-2^7). For sake of efficiency, k is modulated as k' is given by the $m-th$ power of 2:

$$
\begin{aligned} & k' = 2^m \\ & s.t.\, 2^m \leq k < 2^{m+1} \end{aligned} \tag{5}
$$

After the modulation of k, x can be mapped into the range of (y_{min}, y_{max}) and its fixed-point quantization is obtained by shifting the bits of the fraction based on m and the exponent.

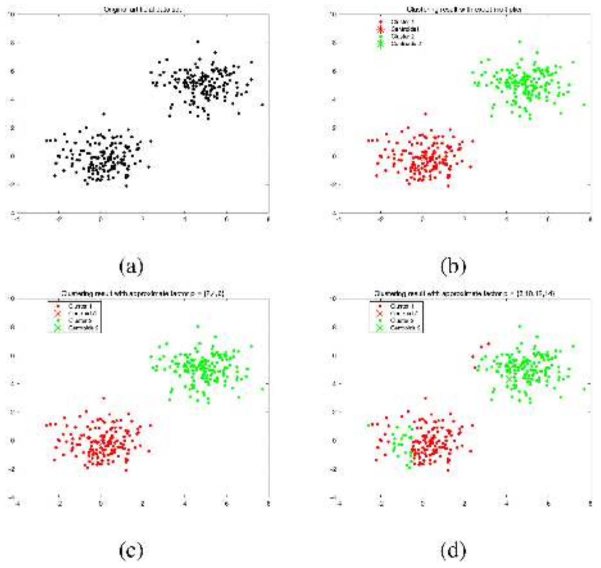

FIGURE 3. Clustering result with different multipliers: (a) original, (b) exact multiplier, (c) and (d) the approximate multiplier of the R4ABMs [10] with approximate factor $p \in \{2, 4, 6\}$ and $p \in \{8, 10, 12, 14\}$ respectively.

III. ADAPTIVE APPROXIMATE COMPUTING

A. INPUT OPERAND DISTRIBUTION

Most previous works utilize either the test-bench or a random input operand distribution (IoD) for evaluation. Different IoDs form different approximate circuits because approximations are data-dependent, where *IDD ={random (Rand), positive skew (PSkew), negative skew (NSkew), normal random distribution (NRD) and sorted normal random distribution (SNRD)}* distributions are employed in this paper.

As shown in Fig. 4, the dynamic power at a specific factor p is function of the IoD; it is highest when both operands are random in 9 IoDs. When only an operand is random, the IoD incurs in more dynamic power dissipation than when both

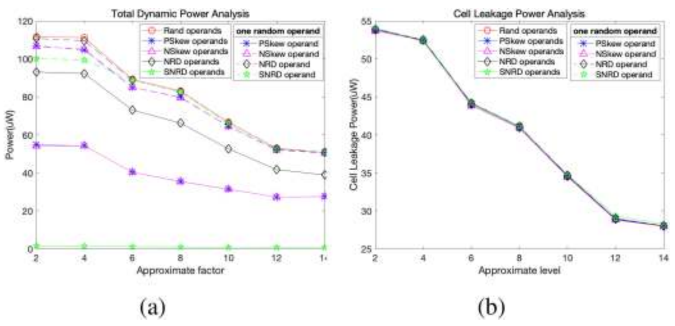

FIGURE 4. Power consumption analysis of R4ABMs with different approximate factor *p* from random (Rand), positive skew (PSkew), negative skew (NSkew), normal random distribution (NRD) and sorted normal random distribution (SNRD) operands: (a) total dynamic power and (b) cell leakage power.

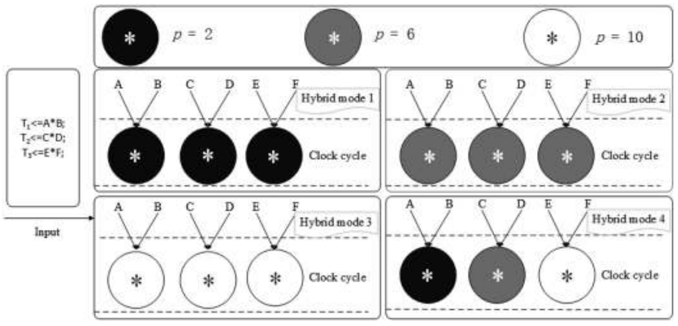

FIGURE 5. Hybrid approximate computing overview.

of them are not random. *PSkew* and *NSkew* IoDs have the same dynamic power; *SNRD* has the least dynamic power and is nearly constant of the approximate factor. Larger *p* leads to simpler logic, but not always to less hardware; these results show that the dynamic power is dependent on the input operand distribution.

B. HYBRID APPROXIMATE COMPUTING

One of the key requirements in approximate computing is that an approximation can be introduced only in non-critical data, because the approximation of critical data (e.g., control operations) can lead to catastrophic events, such as program failure or totally erroneous output. Hence, the level of approximate computing should be dynamic configured ahead by taking into consideration the load. However, a frequent change in workload may incur in a large reconfiguration overhead (such as in FPGA based circuits) for a single approximate circuit (eg, a multiplier) so it must be carefully considered (for ASIC configuration at run-time is not possible).

As not all possible workload distributions can be pre-characterized and to improve existing approximate designs, hybrid schemes are proposed as alternatives. Multiple approximate units usually interact in a datapath; moreover, many applications often require complex datapaths than just a single operation (such as multiplication), so this aspect must be also considered. Fig. 5 gives an overview of the four proposed

TABLE 2. Comparison of Hybrid Approximate Booth Multipliers

Hybrid Approximate Multipliers	Power (μW)	Delay (ps)	Area (μm^2)	Energy (pJ)
HybridMode0 (Exact multiplier)	564.3	0.70	788.4	0.395
HybridMode1 ($p = \{2,2,2\}$)	500.9	0.62	730	0.311
HybridMode2 ($p = \{6,6,6\}$)	441.8	0.60	670	0.265
HybridMode3 ($p = \{10,10,10\}$)	312.9	0.58	610	0.181
HybridMode4 ($p = \{2,6,10\}$)	407.2	0.62	680	0.252

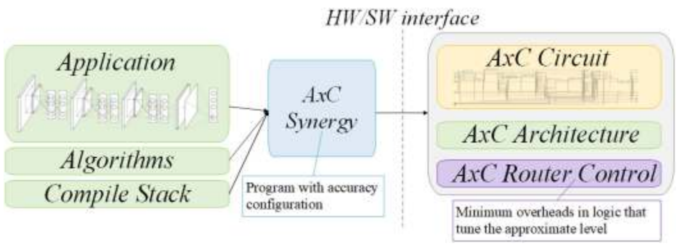

FIGURE 6. Overview of proposed HW/SW co-design framework for adaptive approximate computing.

modes for hybrid approximate multipliers. HybridMode4 utilizes a combination of 3 types of multipliers to fulfill the requirement of an application while the other three modes use one type. Table 2 shows that HybridMode4 achieves better energy performance than HybridMode0, HybridMode1 and HybridMode2; it also provides a better accuracy than HybridMode3 when the first two multipliers are used to process critical data.

IV. HARDWARE/SOFTWARE CO-DESIGN

Next hybrid computing and IoD are proposed to improve the performance of approximate computing; so we need to establish the best relationship between these two techniques using an adaptive scheme. Fig. 6 gives an overview of the proposed adaptive approximate computing scheme. Although ML applications have error resilience capabilities, it is best to use AxC in a principled manner to ensure that the impact on output quality is negligible (or acceptable). There is significant research work for improving the efficiency of artificial intelligence systems at every layer of the computing stack. For better synergy between ML and AxC, the hardware needs to be application-aware and the application must adaptively restrain the noises introduced by the AxC hardware design through the features of the application state.

Learning is usually classified as supervised, unsupervised and reinforced. Learning depends on the hardware for the interconnected neurons of the ANNs; these models address learning in different ways; moreover, when approximate hardware modules are used in these leaning processes, different learning rules are applicable to them. Fig. 7 presents four modes of approximate computing modules (AM) and exact computing modules (EM). Hence, approximate computing for supervised and unsupervised learning must be also considered.

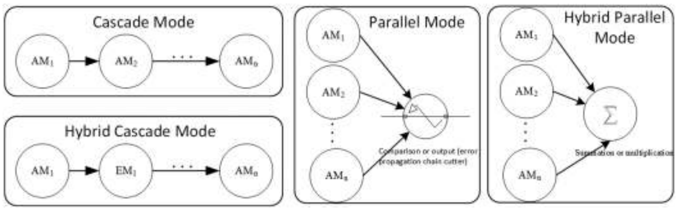

FIGURE 7. Compound modes of approximate computing modules.

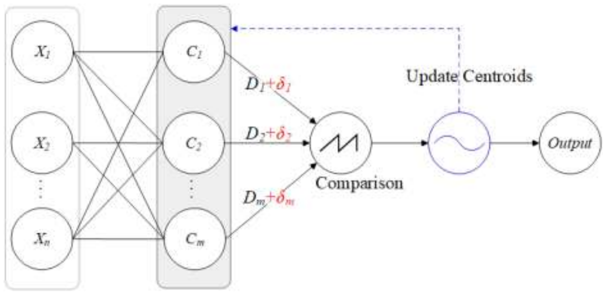

FIGURE 8. Error propagation pattern in the k-means learning process.

Algorithm 1: Gradient-Oriented Adaptive Approximate K-Means Clustering Algorithm.

```
Require:
      The dataset $\mathcal{X}$;
      The number of clusters $k$;
      A distance function $D$;
      A big value of approximate factor $p$, likely the most
       approximated level.
Ensure:
      A partition of $\mathcal{X}$ in $k$ groups.
1:    Assign initial centroids $C$;
2:    repeat
3:      Iteration $i$ increase;
4:      Re-assign the labels of the samples using the
        centroids $c_i$ to minimize $\mathcal{J}_{obj}$;
5:      if $p > p_{min}$ then
6:        $p--$;
7:        Update $\alpha, \beta$ if necessary;
8:      end if
9:    until Convergent or Pre-defined Max number of
      iterations reaches
```

A. ADAPTIVE APPROXIMATE COMPUTING FOR UNSUPERVISED LEARNING

The k-means is one of the most used unsupervised learning algorithm. It searches for several clusters within an unlabeled dataset. As approximate computing contributes to a loss of convergence; hence, when the approximate hardware modules are involved, the loss function of k-means is redefined as follows:

$$min\mathcal{J}_{obj} = \alpha D(\vec{X}, \vec{C}) + \beta D'_{m_p}(\vec{X}, \vec{C}) + \theta \tag{6}$$

where, D is the distortion function, m_p is the multiplier with the approximate factor p as defined for R4ABM, D'_{m_p} is the counterpart of D with m_p multiplier, α is the bonus factor due to exact computing, β and θ is the healing and mitigation factors for approximate computing receptively. The healing and mitigation factors must be determined by the selected approximate level. In this paper, for convenience, α is set to 1, θ is set to 0 and β is set to $\log_2 p$. Moreover, approximate computing can also be employed by several inputs X.

Fig. 8 illustrates the clustering process and the propagation of errors in the approximate computing modules in likely a parallel mode. Although errors (δ) can be propagated to the next iteration when centroids are updated, the error propagation chain may no longer exist.

1) GRADIENT-ORIENTED ADAPTIVE APPROXIMATE CLUSTERING

For ML, the learning rate is a tuning parameter in an algorithm to determine the step size at each iteration when moving toward the minimum of a loss function. While the descent direction is usually determined from the gradient of the loss function, the learning rate determines the step in that direction. To achieve faster convergence, prevent oscillations and getting stuck in an undesirable local minimum, the learning rate is often varied during the learning process. The simplest learning rate schedule decreases the learning rate linearly, so from a large initial value to a small value. This allows large weight changes in the beginning of the learning process and small changes or fine-tuning towards the end of the learning process because the early phases are more resilient to error. To match the reduction in learning rate, the approximate computing factor must also decrease. Moreover, since error propagation is not continued, the error generated by a higher approximate level has a very limited impact on the next iteration. The gradient-oriented adaptive approximate k-means clustering is given in Algorithm 1.

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm for density-based clustering. It can find clusters of different shapes and sizes from a large amount of data, which has noise and outliers. The gradient-oriented adaptive approximate DBSCAN clustering algorithm is given in Algorithm 2.

2) SEMI-SUPERVISED ADAPTIVE APPROXIMATE K-MEANS CLUSTERING ALGORITHM

The approximate factor in Algorithm 1 is established a-priori and only reducing with iterations; the feature constrains of the supervising information are not as common as for the entire samples; so, it is possible to control the error tolerance using an approximate circuit. Therefore, a co-design framework for semi-supervised approximate k-means clustering is proposed by using feature constraints. Its objective function can be

Algorithm 2: Gradient-Oriented Adaptive Approximate DBSCAN Clustering Algorithm.

```
Require:
    The dataset $\mathcal{X}$;
    The radius of neighborhood $\varepsilon$;
    The minimum number of neighbors $M$;
    A distance function $D$;
    A big value of approximate factor $p$, likely the most
      approximated level.
Ensure:
    A partition of $\mathcal{X}$ in certain groups.
 1: Initialize the first cluster label $C = 0$;
 2: repeat
 3:    Assign the cluster label $C = C + 1$;
 4:    if the label of point $x$ is undefined then
 5:       Assign an empty neighbor set $N$, $i = index(x)$
          and $p = p_{max}$;
 6:       repeat
 7:          Set $i = i + 1$, find the next $x_i$;
 8:          if $D(x, x_i) < \varepsilon$ then
 9:             Add the $x_i$ into the neighbor set $N$ of $x$;
10:          end if
11:          if ($i$ Mod $M == 0$) and ($p > p_{min}$) then
12:             $p - -$;
13:          end if
14:       until All points in $\mathcal{X}$ are scaned;
15:       if sizeof($N$) $< M$ then
16:          Label the point $x$ as noise;
17:       else
18:          Label the point $x$ as cluster C;
19:          repeat
20:             if The label of point $x'$ in $N$ is noise or
                undefined then
21:                Label the point $x'$ as cluster C;
22:                Assign the $x$ as $x'$, goto 5;
23:             end if
24:          until All the points of neighbor set $N$ is
             scaned;
25:       end if
26:    end if
27: until Until all points in $\mathcal{X}$ are labeled
```

extended from semi-supervised k-means as follows:

$$min\mathcal{J}_{obj} = \alpha D(\vec{X}, \vec{C}) + \beta D'_{m_p}(\vec{X}, \vec{C}) + \gamma \varphi_{D_{m_p}}(\vec{X}) + \theta$$
$$s.t. \begin{cases} \varphi_{D_{m_p}}(\vec{X}) \leq \varphi'_{D_{m'_p}}(\vec{X}) \,\&\&\, p < p_{max},\ p++ \\ \varphi_{D_{m_p}}(\vec{X}) > \varphi'_{D_{m'_p}}(\vec{X}) \,\&\&\, p > p_{min},\ p-- \end{cases} \tag{7}$$

where $\varphi_{D_{m_p}}(\vec{X})$ is the penalty value of the violations of the constraints at the current stage based on the multiplier m_p and $\varphi'_{D_{m_p}}(\vec{X})$ is at the previous stage based on the previous multiplier m'_p with an approximate factor p'. When the new penalty value $\varphi_{D_{m_p}}(\vec{X})$ is higher than the previous value, the approximate level must be reduced, and the approximate factor p decreases if p is greater than p_{min}. Equivalently, the approximate factor p increases if p is less than p_{max}.

Algorithm 3: Semi-Supervised Adaptive Approximate K-Means Clustering.

```
Require:
    The dataset $\mathcal{X}$;
    The number of clusters $k$;
    A set of must and cannot links;
    A distance function D;
    A set of weights for violating the feature constrains;
    A default value of approximate factor $p$.
Ensure:
    A partition of $\mathcal{X}$ in $k$ groups.
 1: Assign initial centroids $C$;
 2: repeat
 3:    Iteration $i$ increase;
 4:    Re-assign the labels of the examples using the
       centroids $c_i$ to minimize $\mathcal{J}_{obj}$;
 5:    Check the penalty $\varphi$ of the violation of the feature
       constraints;
 6:    if $\varphi_i > \varphi_{i-1}$ and $p > p_{min}$ then
 7:       $p - -$;
 8:       Update $\alpha, \beta, \gamma$ if necessary;
 9:    else
10:       if $p < p_{max}$ then
11:          $p + +$;
12:          Update $\alpha, \beta, \gamma$ if necessary;
13:       end if
14:    end if
15: until Convergent or Pre-defined Max number of
    iterations reaches
```

The procedures for semi-supervised approximate k-means clustering are summarized in Algorithm 3. The entire approximate clustering process is supervised using the semi-supervised feature constraints for different approximation levels, so reducing the accumulated errors and exploiting more aggressively approximate computing at algorithm level with a small amount of supervised information. This can adaptively find the best approximate factor not only for the supervised data, but also for most unlabeled data. The approximate factor is gradually improved to reduce the large accuracy loss introduced by the approximation operations. The approximation level is controlled by the approximation factor p and can be adjusted according to the violations of the supervising information. The procedures for semi-supervised approximate k-means clustering are given in Algorithm 3. The entire approximate clustering process is supervised using the semi-supervised feature constraints for different approximation levels, so reducing the accumulated errors at algorithm level with a small amount of supervised information. After each iteration or computation of supervised data (in every iteration), the approximate factor can be adaptively adjusted. The approximate factor can be updated through the multiplication of the supervised data as minority. Then the (majority) unlabeled data can also benefit from the better approximate multiplier for the current phase. The approximate factor can be updated not only according to the penalty value after each iteration, but also at each multiplication of the supervised process. In this work, we only update the approximate level through the factor after each iteration in the next evaluation. Hence, the cost of the adaptive transformation is very small within the entire hybrid process.

B. ADAPTIVE APPROXIMATE COMPUTING FOR SUPERVISED LEARNING

MLP processes a type of supervised learning. The MLP system is provided with a large amount of input data during the training phase; such data guides the system to obtain the output from each specific input value. The trained model is then used to infer the actual data. For ANNs, many works have been proposed for approximate computing in the training process to have a better model for errors. As result of back propagation in each iteration, every weight is affected by the error generated by approximate computing; this error may accumulate and change the descent direction of the loss function. An approximate training process requires time and often suffers from a loss of convergence. The model trained by approximate computing is highly dependent on the specific approximate modules and the training data set. In Fig. 9, the dots are the training data; the green or red backgrounds are the inference results after traversing all data in the specific range. Fig. 9(a) shows that the loss function of an approximate training process still oscillates after 4,000 iterations; moreover, the inference result of exact computing with the approximate trained model remains unacceptable. Existing approximate computing techniques mainly focus on pre-trained artificial neural networks, which may lead to the generation of sub-optimal solutions. Even after training with exact computing, a high accuracy is not necessarily a good indicator, because it may also imply that the model is suffering from overfitting. Such a data set may perform well in the test scenario, but it may fail in specific applications. Fig. 9(b) shows that after 23 iterations of approximate training, a learning model satisfying all training data is found. However, inference with approximate training overfits more than the exact approach. Hence, a more accurate and generalized model is required for a high quality inference. ANN algorithms are improved for accuracy under the assumption of ideal hardware implementation without considering error tolerance.

The focus of this paper is on flexible approximate hardware modules for inference an exact training model. In Table 1, the multiplication equations lead to relative standard deviations; it is quite like the error metric of the mean absolute percentage error (MAPE). To better illustrate the relative standard deviations of noises caused by the approximate computing, the enhanced MAPE (EMAPA) is defined as follows:

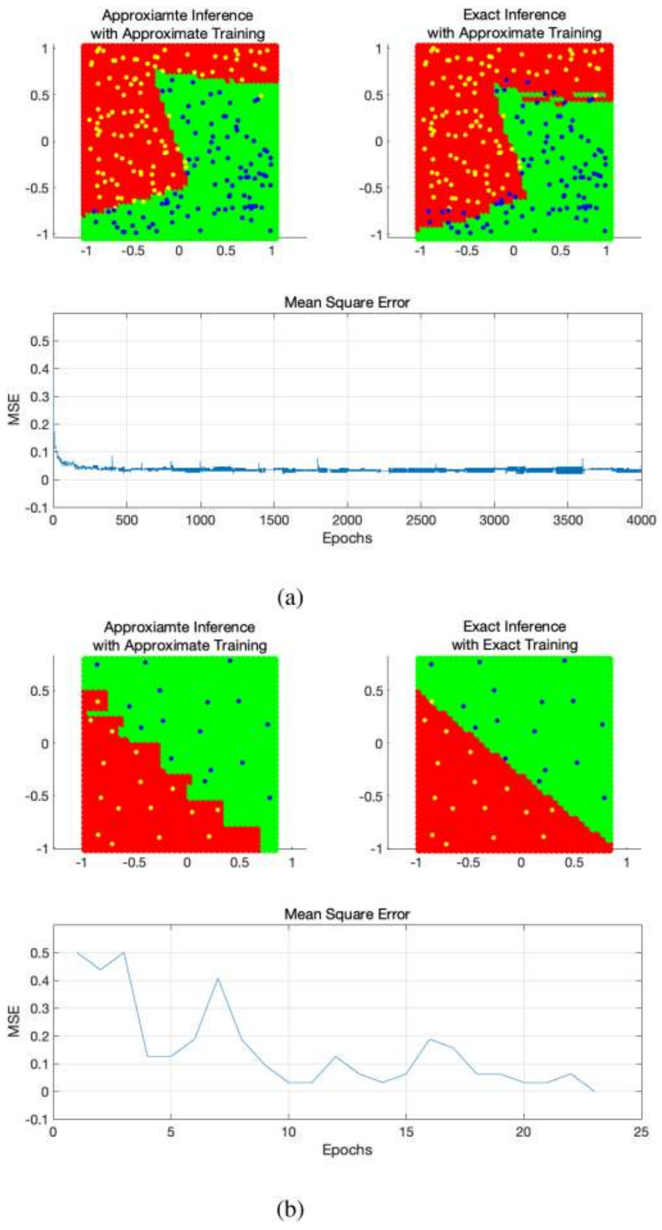

FIGURE 9. Comparison of approximate and exact inference: (a) with approximate training and (b) with approximate and exact training respectively.

$$EMAPE = \begin{cases} |\dfrac{E-A}{E}|, E \neq 0 \\ 1, E = 0 \end{cases} \tag{8}$$

where E and A refer to the exact and approximate computing results, respectively. From the Eq. (9) depicted in Table 1, if the relative errors $\frac{\delta_a}{a}$, $\frac{\delta_b}{b}$ and $\frac{\delta_c}{c}$ are random and independent, the largest dominates the last relative deviation $\frac{\delta_Q}{Q}$. Therefore, it is critical to limit the largest relative deviation from the approximate computing within an acceptable range.

$$\frac{\delta_Q}{Q} = \sqrt{\left(\frac{\delta_a}{a}\right)^2 + \left(\frac{\delta_b}{b}\right)^2 + \left(\frac{\delta_c}{c}\right)^2} \tag{9}$$

Eq. (10) presents the best strategy for a target accuracy where δ is the largest allowed EMAPE, θ is the largest acceptable tolerance, and $P_{robability}$ is the percentage as $EMAPE > \delta$. Therefore, a design using an adaptive approximate modules can be attained by adjusting δ and θ.

$$P_{robability}(EMAPE > \delta) < \theta \tag{10}$$

In many ANNs, the computation of the score is actually the dot product of the feature ($\vec{x}$) and the weight ($\vec{w}$) (i.e., $\sum_i w_i x_i$). Therefore, research has focused on reducing the overhead of multiplying and accumulating (MAC) [35]. In the inference process, the MAC operations of the feature extraction (convolutional layer) and the classification can be easily parallelized. Sze *et al*. [34] has reported an ANN acceleration scheme that considers fixed weights, as stored in the register file of the processing element (PE). For a more aggressive and adaptive approximation strategy, we further discuss the characteristics of fixed weight, and present two approximation strategies for ANNs. These two methods can be used simultaneously.

1) OPERAND SWAPPER ALGORITHM (OSA)

In most approximate multipliers, the upper right corner of the irregular partial product array is considered less significant and so an approximate design is implemented. In R4ABM, when an approximate factor $p = 2$ is employed, there are over half non-zero multiplication results while the multiplier is 0. Multiplication is commutative, thus the designation of multiplier and multiplicand does not affect the result of multiplication. Still in R4ABM with $p = 2$, if the multiplicand is 0, all multiplication results are 0. Since the weight in ANNs is pre-determined, we can distinguish the multipliers needed to be swapped. The overall operand swapper algorithm is summarized in Algorithm 4. Fig. 11(a) is the original result and (b) is the result after swapping; after swapping, the EMAPE is reduced; hence, some operand swapper PEs can be designed as in Fig. 10(b) with no impact for inference.

2) GREEDY APPROXIMATE COMPUTING ALGORITHM FOR ANNS

To fully exploit approximate computing, a greedy approximate computing algorithm (GACA) is proposed for ANNs using adaptive PEs. GACA is summarized in Algorithm 5. GACA can be employed over all approximate arithmetic techniques when one of the operands is fixed. GACA is implemented when designing the PEs for the inference phase. Multiple PEs are designed for the specific weights; hence, the workload for the different PEs should be further considered.

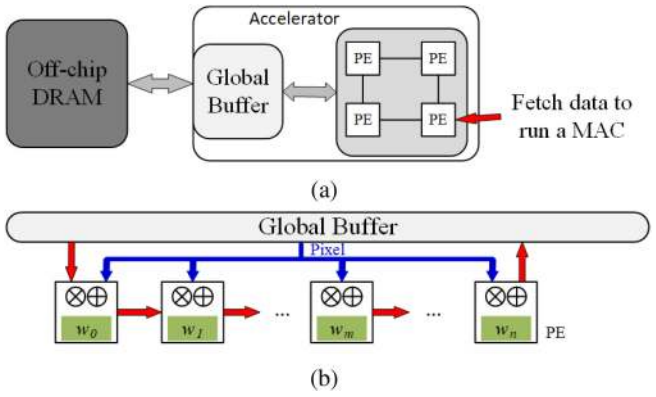

FIGURE 10. Parallel compute paradigm for ANNs [34]: (a) processing elements (PEs) for multiple arithmetic logic units (ALU) and (b) characteristic of weight stationary.

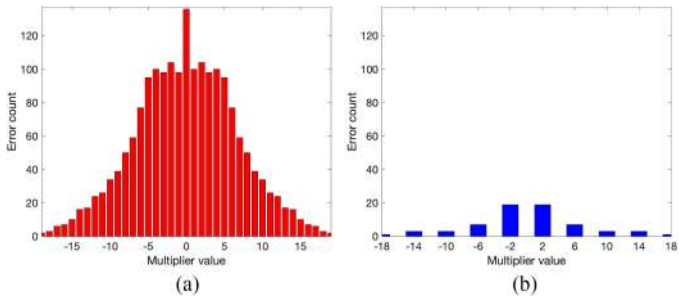

FIGURE 11. Count of EMAPE > 0.1: (a) fixed multiplier and (b) swapped multiplier.

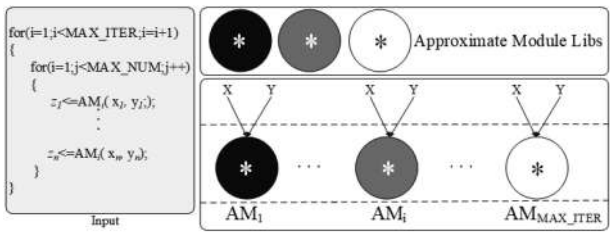

FIGURE 12. Simplified hardware resource evaluation model.

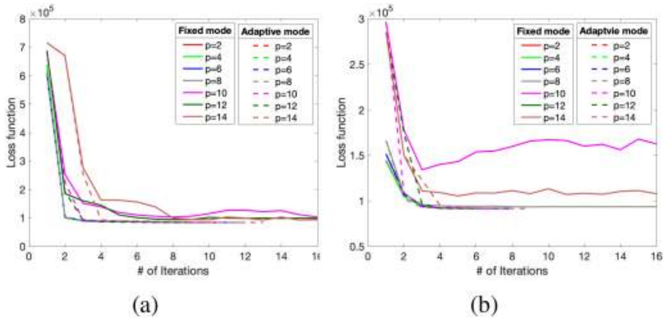

FIGURE 13. Loss function of each iteration on (a) Iris dataset and (b) Seeds dataset.

Algorithm 4: Operand Swapper Algorithm for ANNs.

```
Require:
    The set of weight $W$;
    The EMAPE of each $w_i$ in $W$;
    The maximum allowed EMAPE $\delta$;
    The approximate operator $A_p$ with pre-determined
      approximate factor $p$;
    The overall potential operand set of $X$;
Ensure:
    Choices of the first or second operand for each $w_i$
1:  Set $i = 0$;
2:  repeat
3:      Set $w \leftarrow w_i, i \leftarrow i + 1$;
4:      if $P_{robability}(EMAPE(A(w_i, X)) > \delta) > P_{robability}(EMAPE(A_p(X, w_i)) > \delta)$ then
5:          Define the weight $w_i$ as the second operand for
            the approxiamte computing
6:      end if
7:  until Swapping strategy for all weights are found
```

Algorithm 5: Greedy Approximate Computing Algorithm for ANNs.

```
Require:
    The set of weight $W$;
    The EMAPE of each $w_i$ in $W$;
    The largest allowed EMAPE $\delta$;
    The largest acceptable tolerance $\theta$;
    The largest value of approximate factor $p$;
Ensure:
    A greedy approximate factor for each $w_i$
1:  Set $i = 0$;
2:  repeat
3:      Set $w \leftarrow w_i, i \leftarrow i + 1$;
4:      repeat
5:          if $P_{robability}(EMAPE > \delta) > \theta$ and $p > p_{min}$
            then
6:              $p - -$;
7:          end if
8:      until $p$ does not decrease or $p = p_{min}$
9:  until Greedy approximate factor for all weights are
    found
```

V. EVALUATION AND ANALYSIS

Hardware metrics, such as power consumption, area, critical path delay, and PDP, are considered in this section.

A. SIMULATION SETUP

Although the proposed method is applicable to most reconfigurable multipliers and adders, the method of [10] is considered because multipliers are more complex and with a higher energy consumption than adders. To evaluate the effectiveness of the proposed method and the energy reduction, the classical k-means algorithm is selected using the standard UCI datasets [36] and the approximate ANNs by embedding

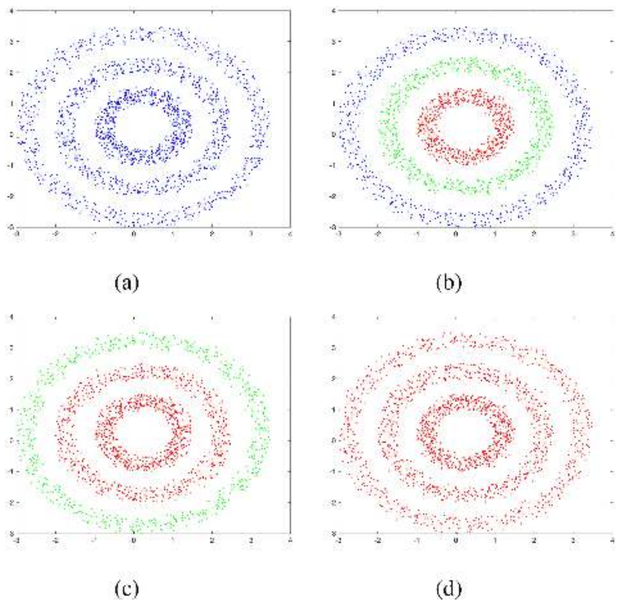

FIGURE 14. Clustering result of DBSCAN: (a) original, (b) the exact, adaptive and fixed AxC ($p = 2$), (c) fixed AxC ($p = 4, 6$) and (d) fixed AxC (other p).

fixed weights. Logic synthesis and simulation tools from the Synopsys Design Compiler and VCS are utilized; the target synthesis technology is given by the Nangate 45-nm open cell library. Moreover, an approximate multiplier of each iteration is considered as an individual module because the IoD of each iteration is different. The operations of each iteration are kept the same and defined by the number of samples. Therefore, as depicted in Fig. 12, a simplified evaluation model is presented. For performance comparison, different combinations of approximate multipliers (AMs) are used in the application as corresponding to diverse hybrid modes. The AMs are kept same for each iteration when a specific approximate factor is employed.

B. SIMULATION RESULTS

1) UNSUPERVISED AND SEMI-SUPERVISED LEARNING

The clustering results using 32-bit full-precision, 8-bit fixed-point quantized, for various ($p \in \{2, 4, 6, 8, 10, 12, 14\}$) and adaptive p 8-bit approximate multipliers are presented in Table 3.

The metric of the F-value is used and is defined as follows:

$$F = \frac{(1 + \beta^2) * p * r}{\beta^2 * p + r} \tag{11}$$

where, p is the precision, r is the recall and β is used to balance the precision and the recall. It is usually set to be 1.

As Fig. 14 shows, the adaptive AxC scheme achieves the same accuracy as the exact computing but with a more aggressively approximation than a fixed AxC. Since there are many approaches for hardware acceleration for k-means, in order to better measure the ability of our proposed method, the energy consumption of the multiplication operations for different AxCs is summarized in Fig. 15. The data-set is the

TABLE 3. **F-Value of K-Means Clustering**

Data Sets	Exact		8b R4ABM with Fixed Approximate Factor [10]							8b R4ABM with Adaptive Approximate Factor(Initial value)						
	32b Float	8b	p=2	p=4	p=6	p=8	p=10	p=12	p=14	p=2	p=4	p=6	p=8	p=10	p=12	p=14
Iris	0.828	0.818	0.818	0.818	0.818	0.834	0.814	0.800	0.782	0.818	0.818	0.818	0.818	0.818	0.818	0.818
Seeds	0.805	0.805	0.812	0.812	0.816	0.823	0.843	0.624	0.745	0.812	0.812	0.816	0.816	0.816	0.816	0.816

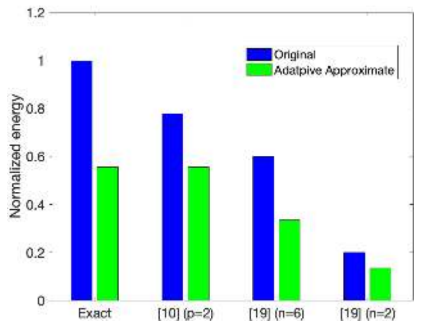

FIGURE 15. **Energy consumption of k-means multiplication operations for different AxCs.**

OCR digits in UCI. The energy values are normalized with respect to the exact value. The methods of [10] and [19] effectively reduce energy consumption by circuit design and algorithm acceleration respectively. Our proposed method is more application-aware than [10] and more error tolerant to hardware noises than [19]. As shown in Fig. 15, the bars indicate the exact computing value (blue) as well as AxC ($p = 2$) in [10], AxC ($n = 6$) and AxC ($n = 2$) in [19] respectively; the green bars are for the corresponding approximate method with no significant accuracy loss. In [19], when n is significantly less than the number of clusters (k), the accuracy quite depends on the initial status. However, the proposed adaptive approximate method can always maintain nearly the same accuracy with a considerable energy reduction. These results further validate our co-design method for synergy between AxC in hardware and software.

The results of Table 3 accomplishes good results with a small loss of accuracy; an adaptive approximate method achieves a very good and fixed accuracy. It can even achieve a better accuracy (for example for the Seeds data set with an adaptive p) than exact computing, because the error introduced by approximate computing avoids overfitting the initial centroids (to which k-means clustering is sensitive). Performance of the fixed approximate factor becomes worse when the factor is very large. The adaptive approximate factor can achieve a good energy reduction and a small accuracy loss despite the initial value of the approximate factor even when the initial approximate level is high. The highest accuracy (such as p_8 and p_{10} for the Iris and Seeds dataset respectively), can be attained, however, the results are not reliable. The algorithm ends when the loss function is convergent, or a pre-defined maximum number of iterations is reached. Fig. 13 shows that, even though the results of the fixed approximate computing are acceptable, several of them lose convergence and exceed the largest number of iterations.

The total energy consumption of the multipliers can be obtained using the simplified evaluation model of Fig. 12. The results on the Iris dataset are presented in Table 3. The best initial factor is 12; also it is assumed that the error at the beginning is not as large as the errors accumulated along the data path. For fast convergence, larger steps are always utilized and therefore, the early stage of computation of an algorithm should be resilient to errors.

Many machine learning algorithms, as based on the gradient descent method, can easy fall into a local point; therefore, the error introduced by approximate computing, causes the loss function to fluctuate but not gradually decrease. The PDP can be very large if an unsuitable approximate factor (such as $p \in \{10, 12, 14\}$) is utilized, thus contributing to a loss of convergence in the algorithm. The calibration of the loss function may also solve this problem because approximate computing can achieve a higher accuracy with a lower hardware. However, calibration of the loss function cannot be easily established, because there are many algorithms with a complex loss function. The proposed adaptive method achieves a small accuracy loss at a lower energy consumption despite an initial approximate factor and it overcomes the loss of convergence.

2) EVALUATIONS OF SUPERVISED LEARNING

Most supervised learning models are based on artificial neural net-works (ANNs). For generality of the models, in this paper, we focus on approximate inference (i.e., by only applying approximate techniques to the inference process). Fig. 16 shows the hardware noise cancellation of OSA under different approximate factors. OSA takes the advantage of the feature that the bottom left partial array is less approximate than the upper right. As the approximate factor increases, its benefit is reduced, but it is still affecting the result because there are still few less approximate (bottom left) partial arrays, so more significant. Fig. 17 shows the original and operand swapped approximate computing results with different EMAPE δ are presented. Different values of δ may lead to a different hardware noise cancellation; the best value of δ is determined by the accuracy requirement of the application, moreover it is dependent on the specific approximate technique.

The δ and θ will confine noise at an acceptable level. N PEs are utilized where N is the number of all possible weights (i.e., 256 for 8 bits). Each PE consists of an approximate module. Table 4 summarizes the power, delay, area and power-delay product. Overall adaptive approximate computing of GACA achieves a better energy performance than the fixed AMs (p=2,6), and certainly more accuracy than fixed AM ($p = 10$).

VI. CASE STUDY: IMAGE SEGMENTATION

In this section, the proposed method is applied to image segmentation. The semi-supervised feature constraints are chosen

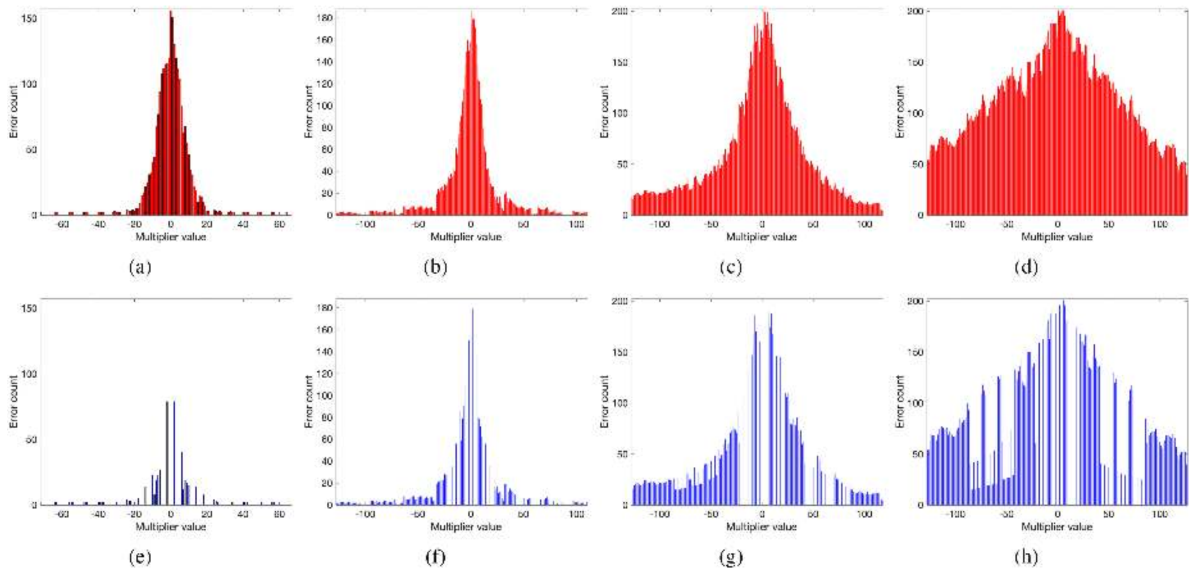

FIGURE 16. Error count of original and operand swapped approximate computing results with different approximate factor under a specific EMAPE $\delta > 0.1$: (a), (b), (c) and (d) are original approximate computing with $p = 4, 6, 8$ and 10 receptively; (e), (f), (g) and (h) are operand swapped approximate computing with $p = 4, 6, 8$ and 10 receptively.

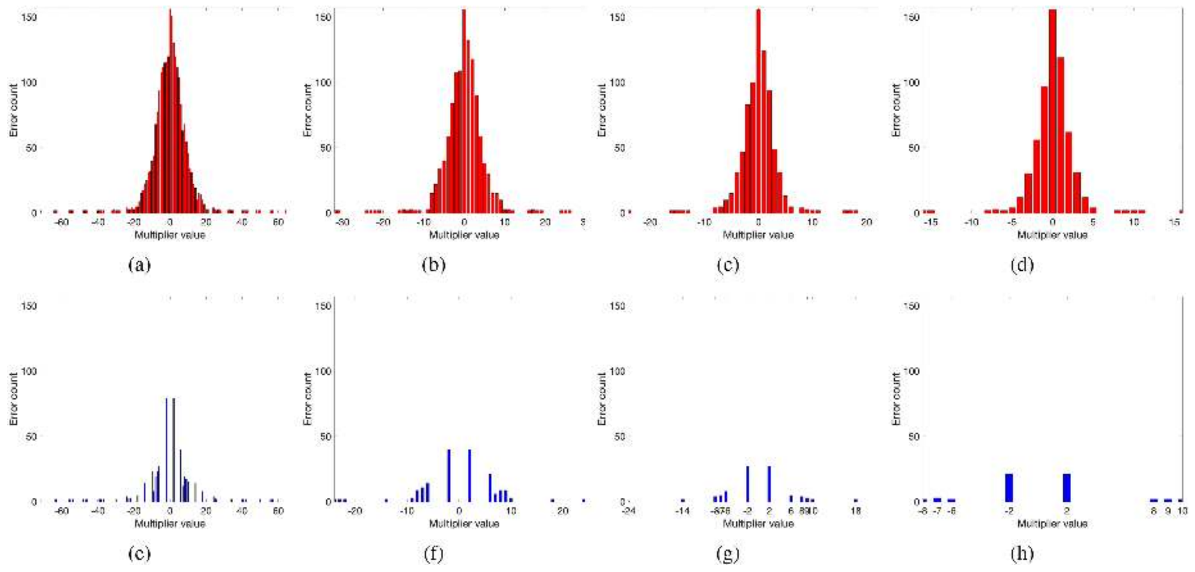

FIGURE 17. Error count of original and operand swapped approximate computing results with different EMAPE δ for an approximate factor ($p = 4$): (a), (b), (c) and (d) are original approximate computing with $\delta = 0.1, 0.2, 0.3$ and 0.4 receptively; (e), (f), (g) and (h) are operand swapped approximate computing with $\delta = 0.1, 0.2, 0.3$ and 0.4 receptively.

TABLE 4. Energy Evaluation of Greedy Approximate Computing Algorithm

Approximate Multipliers	Power (mW)	Delay (ps)	Area (mm^2)	Energy (nJ)
GACA($\theta = 0.1, \delta = 0.1$)	35.458	0.63	0.1327	22.338
GACA($\theta = 0.1, \delta = 0.2$)	35.097	0.63	0.1241	20.851
GACA($\theta = 0.2, \delta = 0.2$)	27.476	0.63	0.1084	17.310
GACA($\theta = 0.3, \delta = 0.3$)	25.250	0.63	0.0997	15.908
Fixed AM with p=2 [10]	44.385	0.63	0.1592	27.963
Fixed AM with p=6 [10]	36.855	0.61	0.1373	22.482
Fixed AM with p=10 [10]	25.319	0.58	0.1009	14.685
Exact multiplier	48.179	0.71	0.2011	34.207

as more than 10 must-links and 10 cannot-links. The semi-supervised k-means uses this information to segment the image, but it also dynamically calibrates the approximate levels for energy reduction and accuracy.

As shown in Fig. 18, the quality of the processed image deteriorates with an increase of p as factor during the segmentation process from (f) to (h). When the fixed approximate factor is larger than 8, the segmentation results are unacceptable. The results from (i) to (o) are based on different initial factors, all achieve an acceptable segmentation result.

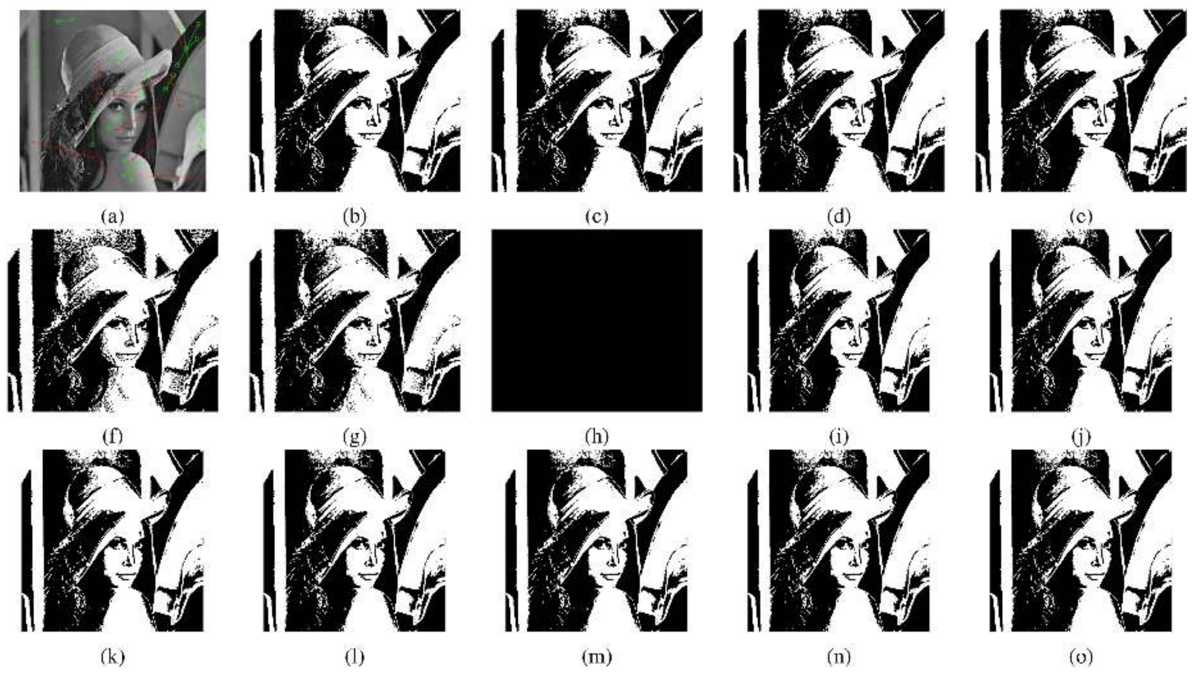

FIGURE 18. Image segmentation: (a) original image with feature constrains, approximate segmentation with fixed approximate factor (b) p = 2, (c) p = 4, (d) p = 6, (e) p = 8, (f) p = 10, (g) p = 12, (h) p = 14 and adaptive approximate segmentation using the proposed method with initial approximate factor (i) p = 2, (j) p = 4, (k) p = 6, (l) p = 8, (m) p = 10, (n) p = 12, (o) p = 14.

TABLE 5. Energy Consumption of Approximate Multiplication

Approximate Multipliers	Total Number of Iterations	Change of p in Each Iteration	Power (mW)	Delay (ps)	Area (μm^2)	Energy (nJ)	PSNR (dB)
Adaptive AM with initial p=2	11	(2,4,6,4,2,2,2,2,2,2,2)	1.997	6.74	1,261.11	13.463	Inf
Adaptive AM with initial p=4	11	(4,2,4,6,4,2,2,2,2,2,2)	1.885	6.54	1,261.11	12.331	Inf
Adaptive AM with initial p=6	10	(6,4,2,4,6,4,2,2,2,2)	1.712	6.37	1,261.11	10.905	Inf
Adaptive AM with initial p=8	10	(8,6,4,6,4,2,4,2,2,2,)	1.564	6.34	1,261.11	9.917	Inf
Adaptive AM with initial p=10	10	(10,8,6,4,2,4,6,4,2,2,)	1.427	6.23	1,261.11	8.889	Inf
Adaptive AM with initial p=12	9	(12,10,8,6,4,2,4,2,2)	1.223	5.58	1,261.11	6.822	Inf
Adaptive AM with initial p=14	9	(14,12,10,8,6,4,2,2,2)	0.949	4.40	1,261.11	4.175	Inf
Fixed AM with p=2 [10]	11	(2,2,2,2,2,2,2,2,2,2,2)	1.821	6.82	667.90	12.416	Inf
Fixed AM with p=4 [10]	12	(4,4,4,4,4,4,4,4,4,4,4,4)	1.964	7.20	649.30	14.139	Inf
Fixed AM with p=6 [10]	11	(6,6,6,6,6,6,6,6,6,6,6)	1.469	6.38	575.00	9.370	18.8929
Fixed AM with p=8 [10]	11	(8,8,8,8,8,8,8,8,8,8,8)	1.366	6.38	538.60	8.717	18.8929
Fixed AM with p=10 [10]	Infinite	(10,10,10,10,10,...,10,10,10,10,10)	N/A	N/A	480.40	N/A	13.5954
Fixed AM with p=12 [10]	Infinite	(12,12,12,12,12,...,12,12,12,12,12)	N/A	N/A	423.20	N/A	16.1819
Fixed AM with p=14 [10]	Infinite	(14,14,14,14,14,...,14,14,14,14,14)	N/A	N/A	405.30	N/A	3.3498

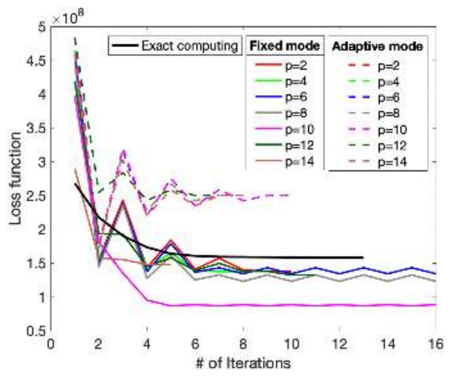

FIGURE 19. Loss function of the segmentation.

Since the segmentation involves many multiplications, the accumulated error has a large impact; however the loss function of exact computing is convex and decreases smoothly. The proposed adaptive approximate computing prevents divergence while reducing the loss function. As per the evaluation model in Fig. 12, Table 5 shows the simplified energy consumption of multiplication for image segmentation. The peak signal-to-noise ratio (PSNR) is used to assess the quality of the output image. The signal in this case is the approximate segmentation with a fixed approximate factor $p = 2$; this confirms that the adaptive approximate computing utilizes less energy and resolves the non-convergence problem.

VII. CONCLUSION

A HW/SW co-design method for adaptive approximate computing has been investigated using multi-precision approximate multipliers with various approximate factors. A fixed-point quantization for floating point data has been proposed to extend or compress the number range, thus resulting in resilience to the so-called approximate factor. We have presented a novel approach by considering hybrid approximate modes for dynamic workloads. The proposed approximate technique is adaptive and has been utilized for both unsupervised and supervised learning. A gradient-oriented adaptive approximate algorithm has considered noise with respect to different gradient descent phases of the clustering application (i.e., more error tolerant in the initial epochs). The proposed semi-supervised adaptive approximate algorithm employs supervised information (a very small portion of the entire data set) to better utilize the provided hardware. Algorithms such as the operand swapper algorithm (OSA) and the greedy approximate computing algorithm (GACA) have been used to take advantage of a fixed operand in the learning model. It has been shown that OSA achieves a significant accuracy improvement with no energy trade-off, while GACA adaptively reduces the hardware noise to reduce the side effects of approximate computing in an application, such as image processing. The proposed HW/SW co-design for adaptive approximate computing has been used in several machine learning and pattern recognition applications.

REFERENCES

[1] H. Esmaeilzadeh, E. R. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp Comput. Architecture*, 2011, pp. 365–376.

[2] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing," *Proc. IEEE,* vol. 108, no. 3, pp. 394–399, Mar. 2020.

[3] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th Annu. Des. Automat. Conf.*, 2013, pp. 113: 1–113:9.

[4] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Computing approximately, and efficiently," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2015, pp. 748–751.

[5] A. Sampson, "Hardware and Software for Approximate Computing," 2015.

[6] S. He, S. K. Lahiri, and Z. Rakamaric, "Verifying relative safety, accuracy, and termination for program approximations," *J. Autom. Reasoning*, vol. 60, no. 1, pp. 23–42, 2018.

[7] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. 49th Annu. Des. Automat. Conf.*, 2012, pp. 820–825.

[8] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[9] S. Venkatachalam, E. Adams, H. J. Lee, and S. Ko, "Design and analysis of area and power efficient approximate booth multipliers," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1697–1703, Nov. 2019.

[10] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1435–1441, Aug. 2017.

[11] H. Jiang, C. Liu, F. Lombardi, and J. Han, "Low-power approximate unsigned multipliers with configurable error recovery," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 189–202, Jan. 2019.

[12] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, "Area-optimized accurate and approximate softcore signed multiplier architectures," *IEEE Trans. Comput.*, pp. 1–1, 2020.

[13] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2019, pp. 928–931.

[14] F. Sabetzadeh, M. H. Moaiyeri, and M. Ahmadinejad, "A majority-based imprecise multiplier for ultra-efficient approximate image multiplication," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4200–4208, Nov. 2019.

[15] K. Manikantta Reddy, M. H. Vasantha, Y. B. Nithin Kumar, and D. Dwivedi, "Design of approximate booth squarer for error-tolerant computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 5, pp. 1230–1241, May 2020.

[16] W. Liu *et al.*, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, Jun. 2019.

[17] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-level approximate computing applied to energy efficient hevc decoding," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 5–17, Jan.–Mar. 2019.

[18] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, "mARGOT: A. dynamic autotuning framework for self-aware approximate computing," *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 713–728, May 2019.

[19] A. Raha, S. Venkataramani, V. Raghunathan, and A. Raghunathan, "Energy-efficient reduce-and-rank using input-adaptive approximations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 462–475, Feb. 2017.

[20] A. Sampson, J. Bornholt, and L. Ceze, "Hardware-software co-design: Not just a cliché," in *Proc. 1st Summit Adv. Program. Languages*, 2015, pp. 262–273.

[21] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, Mar./Apr. 2017.

[22] P. Huang, C. Wang, R. Ma, W. Liu, and F. Lombardi, "A hardware/software co-design method for approximate semi-supervised k-means clustering," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2018, pp. 575–580.

[23] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *Int. J. Adv. Res. Artif. Intell.*, vol. 2, no. 2, pp. 34–38, 2013.

[24] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals, Syst.*, vol. 2, no. 4, pp. 303–314, Dec. 1989.

[25] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62: 33, 2016.

[26] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Feb. 2020.

[27] M. Imani, R. Garcia, A. Huang, and T. Rosing, "Cade: Configurable approximate divider for energy efficiency," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2019, pp. 586–589.

[28] Z. Liu, K. Jia, W. Liu, Q. Wei, F. Qiao, and H. Yang, "Ina: Incremental network approximation algorithm for limited precision deep neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2019, pp. 1–7.

[29] S. Mazahir, O. Hasan, and M. Shafique, "Adaptive approximate computing in arithmetic datapaths," *IEEE Des. Test*, vol. 35, no. 4, pp. 65–74, Aug. 2018.

[30] S. Xu and B. C. Schafer, "Toward self-tunable approximate computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 4, pp. 778–789, Apr. 2019.

[31] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *Comput. Res. Repository*, 2017, vol. abs/1702.03044. [Online]. Available: https://arxiv.org/abs/1702.03044

[32] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *Comput. Res. Repository*, 2016, vol. abs/1606.06160. [Online]. Available: https://arxiv.org/abs/1606.06160

[33] P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," *Comput. Res. Repository*, 2016, vol. abs/1606.01981. [Online]. Available: https://arxiv.org/abs/1606.01981

[34] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2017, pp. 1–8.

[35] A. Raha and V. Raghunathan, "qLUT: Input-aware quantized table lookup for energy-efficient approximate accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5 s, p. 130, Sep. 2017.

[36] M. Lichman, "UCI machine learning repository," 2013. Accessed: Nov. 6, 2020. [Online]. Available: http://archive.ics.uci.edu/ml

PENGFEI HUANG received the B.Sc. degree in information engineering from Jilin University, Jilin, China, in 2006 and the M.Sc. degree in computer application technology from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 2009. Since September 2016, he has been working toward the Ph.D. degree with the College of Electronic and Information Engineering, NUAA. His research interests include computer arithmetic, machine learning, fault tolerance systems, and low-power technologies in approximate computing.

CHENGHUA WANG received the B.Sc. and M.Sc. degrees from Southeast University, Nanjing, China, in 1984 and 1987, respectively. In 1987, he joined the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China, where he became a Full Professor in 2001. He has authored or coauthored six books and more than 100 technical papers in journals and conference proceedings. His current research interests include testing of integrated circuits, and circuits and systems for communications. He was the recipient of more than ten teaching and research awards at the provincial and ministerial level.

WEIQIANG LIU (Senior Member, IEEE) received the B.Sc. degree in information engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 2006 and the Ph.D. degree in electronic engineering from Queen's University Belfast, Belfast, U.K., in 2012. In December 2013, he joined the College of Electronic and Information Engineering, NUAA, where he is currently a Professor and the Vice Dean. He has authored or coauthored one research book by Artech House and more than 100 leading journal and conference papers. His research interests include approximate computing, hardware security and VLSI design for digital signal processing, and cryptography. His paper was selected as the Feature Paper of IEEE TC in the 2017 December issue. He has two Best Paper Candidates in the IEEE ISCAS 2011 and the ACM GLSVLSI 2015. He was the recipient of the prestigious Outstanding Young Scholar Award by the National Natural Science Foundation of China in 2020. He is currently the Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM I: REGULAR PAPERS from January to December 2020 , the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from May 2019 to April 2021, and the IEEE TRANSACTIONS ON COMPUTERS from May 2015 to April 2019, a Steering Committee Member of the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS from January 2018 to December 2019. He is the program Co-Chair of the IEEE ARITH 2020, and also a Technical Program Committee Members for ARITH, DATE, ASAP, ISCAS, ASP-DAC, ISVLSI, GLSVLSI, SiPS, NANOARCH, AICAS, and ICONIP. He is a Member of the CASCOM and VSA Technical Committee of the IEEE Circuits and Systems Society.

FEI QIAO (Member, IEEE) received the B.Sc. degree from Lanzhou University, Lanzhou, China, in 2000 and the Ph.D. degree from Tsinghua University, Beijing, China, in 2006. He is currently an Associate Professor with the Department of Electronic Engineering, Tsinghua University. He has authored or coauthored around 90 papers and holds more than 30 invented patents. His research interests include low-power circuits design, and energy efficient integrated perception circuits and systems for intelligent robots, wearables, and IoT devices.

FABRIZIO LOMBARDI (Fellow, IEEE) received the B.Sc. (Hons.) degree in electronic engineering from the University of Essex, Colchester, U.K., in 1977, the master's degree in microwaves and modern optics from Microwave Research Unit, University College London, London, U.K., in 1978, and the diploma in microwave engineering and the Ph.D. degree from the University of London, London, U.K., in 1978 and 1982, respectively. He has extensively authored or coauthored papers in his research fields and coauthored or edited seven books. His research interests include bio-inspired and nano manufacturing or computing, VLSI design, testing, and fault or defect tolerance of digital systems. He is currently the Endowed Chair Professor of the International Test Conference, Northeastern University, Boston, MA, USA. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING. He is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY. Since 2019, he has been the Vice President for Publications of the IEEE Computer Society.