Efficient Neural Network Approximation via Bayesian Reasoning

Efficient Neural Network Approximation via Bayesian Reasoning

Alessandro Savino[1], Marcello Traiola[2], Stefano Di Carlo[1] and Alberto Bosio[2]
[1]*Politecnico di Torino, Control and Computer Engineering Department, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*

***Abstract*—Approximate Computing (AxC) trades off between the accuracy required by the user and the precision provided by the computing system to achieve several optimizations such as performance improvement, energy, and area reduction. Several AxC techniques have been proposed so far in the literature. They work at different abstraction levels and propose both hardware and software implementations. The standard issue of all existing approaches is the lack of a methodology to estimate the impact of a given AxC technique on the application-level accuracy. This paper proposes a probabilistic approach based on Bayesian networks to quickly estimate the impact of a given approximation technique on application-level accuracy. Moreover, we have also shown how Bayesian networks allow a backtrack analysis that automatically identifies the most sensitive components. That influence analysis dramatically reduces the space exploration for approximation techniques. Preliminary results on a simple artificial neural network shown the efficiency of the proposed approach.**

***Index Terms*—Approximate computing, Bayesian Networks, Neural Networks, Functional approximation, Accuracy prediction**

I. Introduction

Approximate computing (AxC) refers to the idea that computer systems can let applications trade off accuracy for efficiency [1]. Intuitively, instead of performing exact calculations, AxC aims to selectively relax the computation's accuracy to gain lower power consumption and faster execution time. So far, several AxC techniques have been proposed in the literature both at the hardware and software levels [2]. The resiliency of computation to approximation errors tightly depends on the application domain, as some are more resilient than others [3], [4], [5]. Several publications demonstrated the effectiveness of this approach when applied to applications showing an inherent resiliency to errors [6], [7]. Among them, Artificial Intelligence and, in particular, Deep Neural Networks showed an intrinsic resilience to computational errors and thus are well suited to leverage AxC to achieve better performances at lower costs [8].

Although the literature is rich in approximate implementation and strategy proposals, selecting the components to approximate, together with selecting the best approximation techniques for an application, remains a challenging problem [9], [10], [11]. Indeed, while it is pretty much simple to quantify the impact of an AxC technique on the performance or power budget of an application, measuring the error introduced on the result of the computation is still an open challenge.

978-1-6654-3595-6/21/$31.00 ©2021 IEEE

Most of the approaches proposed in the literature run the approximate application (i.e., the application implemented with some data, functional, or both approximation) several times and compare the outcomes with the precise application [12], [13], [14], [15], [16]. An appropriate error metric (e.g., the Structural Similarity Index (SSI) [17] in the case of image processing) allows the comparison. If the approximate application's accuracy is not satisfactory, a different approximate configuration must replace the actual one. Every new approximation, the application must be executed and analyzed again. The above process is an iterative one, and it goes until the approximation is good enough. This process's cost depends on the number of runs of the application required to reach the desired accuracy level.

Authors in [18], [19] proposed a different approach that analytically formalizes the error induced by C_{ax} and how it propagates in the application. This approach's benefit is that it is possible to evaluate the approximation impact without executing the application every time. However, the formalization is application-dependent. Therefore, building the formal model of an application is very complex and requires deeply analyzing its algorithms.

This paper presents a stochastic approach to predict the impact of an approximate technique on a Neural Network's (NN) accuracy. The approach delivers the characterization of the different approximate components, done only once, and then builds a Bayesian Network (BN) model of the NN, following the principles in [20], [21]. Eventually, by analyzing the network using the Bayesian inference theory, estimating the neural network's error distribution is possible. Moreover, BNs allow a backtrack analysis that automatically identifies the most sensitive components. That influence analysis dramatically reduces the space exploration for approximation techniques. This paper aims to show how BNs can be used to estimate approximation impacts on a DNN. The main benefits are a dramatic reduction of the search space for applying approximate computing techniques to DNNs.

The remainder of the paper is structured as follows. Section II overviews the main concepts of the proposed approach. Section II-C presents the proposed Bayesian model and how it describes a neural network, while Section II-B presents the necessary characterization of operators. Results are discussed in Section III. Finally, Section IV summarizes the main contributions and concludes the paper.

II. Methods

As introduced in Section I, the goal of the proposed approach is to develop a stochastic method able to assess the accuracy of an application exploiting a set of approximate functional components. We model the application with the same methodology of [21], in which three tasks allow us to build the model of the target application. One of them is application-independent (the component characterization), while two of them, application dependent, are the following:

1) Bayesian network construction, and
2) Approximation analysis.

The first application-dependent task aims to analyze the application source code to build the Bayesian Network. The analysis task will resort to BN solving algorithms to analyze the approximation's impact on the outcomes.

Within the BN, a set of Conditional Probability Tables (CPTs) should describe each node's probabilistic behavior. This task characterizes a library of approximate operators quantifying the error introduced by the approximation. An operator's characterization aims to build a CPT table to model the conditional probability of having approximation errors at the operator's output, depending on the approximation of the operator's inputs.

A. Bayesian network construction

This task aims at analyzing the application source code in order to build the Bayesian Network modeling. Therefore, the application must be modeled in such a way to represent formally:

- all data and operators involved in the computation;
- all relations between data and operators;
- the mechanisms that propagate approximation errors through the data of the application.

To achieve this goal, we model the application in the form of a Bayesian Network in which:

- nodes represent both data and operators by mean of stochastic variables;
- edges depict the dependency between data and operators;
- each node is associated with a CPT able to express how the approximation of the parents' nodes impacts a computation outcome.

All stochastic variables express different states according to the modeling described in Section II-C. Once built, this model analyzes how errors are propagated from the root nodes down to the leaves representing the application's outcome.

In this work, we aim at showing our approach on a case study based on a simple Neural Network (NN). Let us consider the example depicted in Figure 1-A, consisting of a single neuron. The neuron performs a weighted sum (`ws`) between Inputs (X_1, X_2, ... X_n) and Weights (W_1, W_2, ... W_n). The neuron's output is the result of the activation function evaluated on the `ws` value. In this paper, we focus on the Sigmoid as an activation function (eq. 5).

Figure 1-B shows the BN model of the considered neuron example. Yellow nodes are the weight nodes (W_i), while the black nodes are the inputs (X_i). Each input node is linked to a CPT reported in Figure 1-C indicating the marginal probability of the related data to be in one of the approximation classes defined in equation 3 from Section II-C.

All intermediate nodes represent computations involving different operators implemented by components analyzed during the characterization phase (see Section II-B), which provide the required CPTs. In the example, we instantiated one node for each multiplication ('(*)') between X_i and $\tilde{W}_i$. Then, we have a node modeling the accumulation ('(+)') and a node modeling the sigmoid function ('S(sw)'). The CPT reported in Figure 1-D is an example of the one computed for the ('(+)') operator.

Finally, orange nodes identify the leaf of the Bayesian network representing the output of the computation. It serves as an observation point, where the effect of the approximation accumulates, allowing us to analyze it from a probabilistic standpoint.

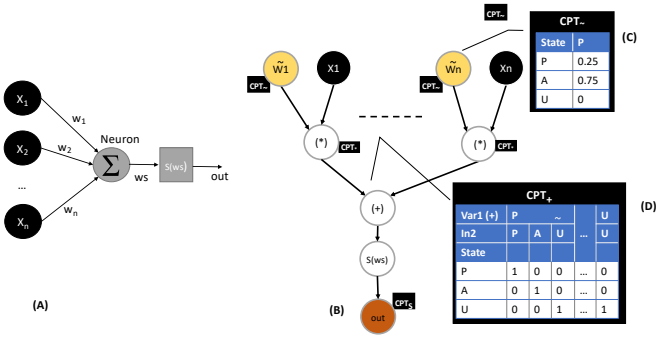


Fig. 1: Bayesian Network example

We used a publicly available BN library and engine [22] to create and analyze the BNs.

B. General approximate component characterization

This paper's BN modeling follows the work proposed in [21]. While in the original work, the approximation worked on integer numbers and precision cut, the NNs require some comma-based number representation. We chose to work with the `libfixmath` cross-platform library [23]. The library defines a fixed-point number of 32 bits. The point is placed in the middle of the representation, having a half-word representing the integer part and the other one dedicated to the fractional part. This representation is suitable for approximating NNs, especially in embedded environments [24], [25], [26].

In order to understand the error range from approximation, we can start by defining the fractional part calculation:

$$\sum_{x=0}^{15} b_x \cdot 2^{x-16} \tag{1}$$

where b_x is the bit value at position x, which counts the position starting from the bit closer to the point place. In this context, a precision reduction means removing bits from the fractional part (the lower the better), and then set them at 0 when using in full 32 bits operations. Based on the number

of bits to remove, this identifies an initial precision error (ϵ) on reduced data given by the k bits removed as:

$$\epsilon(k) = \sum_{x=0}^{k-1} b_x \cdot 2^{x-16} \tag{2}$$

which is bigger when b_x is 1 for all $x \in [0, k]$.

Following the model in [21], a stochastic variable represents each data node; hence a set of states defines each node's behavior based on the impact of the approximation. In [21], we proposed three different classes of accuracy (states) for each node: (i) *precise* (P), when the result is error-free regardless of the precision reduction operation; *acceptable* (A), when the introduced error remains under a user-defined threshold (α); *unacceptable* (U) when the introduced error rises above the threshold making value not acceptable. Therefore, each node's CPT expresses the node's probability belonging to one of these classes. When nodes are independent data nodes, e.g., input, weights, and biasses, CPTs represent the probability that they belong to each of those classes based on all precision reduction considered in the exploration. The CPT values can be evaluated using probability distribution (when values may vary) or based on the actual calculation of eq. 2 on its fixed value based on all possible evaluated ks. This latter case is the approach used on weights and biases in this paper. When nodes represent operations among data, following the Bayes Theorem, the operator's characterization means computing the operator's output's probability to be in a specific state (accuracy level) conditioned to its parent nodes' state. It is crucial to understand that those probabilities are independent of the probability expressed by the input states. They depend only on the number of combinations of states of all inputs' states. On the overall NN, the final accuracy depends on the probabilities of the output layer's nodes belonging to one of the three accuracy classes.

Since the paper's purpose is to prove the feasibility of an approximation exploration based on BN as opposed to a full experimental exploration, we start by defining how to model the input nodes. From the definition in eq. 2, the introduced approximation error classification ($\epsilon(k)$) on an input of NN ($\tilde{y}$), i.e., weight and biases as defined during the training, follows the rules:

$$Class(\tilde{y}) = \begin{cases} P \text{ (Precise)} & if\ \epsilon_y(k) = 0 \\ A \text{ (Acceptable)} & if\ \epsilon_y(k) \leq \alpha \\ U \text{ (Unacceptable)} & if\ \epsilon_y(k) > \alpha \end{cases} \tag{3}$$

With α the threshold is indicating the maximum value of ϵ_y that the exploration can tolerate. Since ϵ_y depends on the actual fixed value, and from the all admissible k cut in the current exploration, for each weight or bias, we can compute the probability of having $\tilde{y}$ in P, A, or U as:

$$\begin{aligned} P(\tilde{y}\ is\ P) &= P(\epsilon_y(k) = 0) = \frac{\sum_{\forall k \to \epsilon_y(k)=0} 1}{\#k} \\ P(\tilde{y}\ is\ A) &= P(\epsilon_y(k) \leq \alpha) = \frac{\sum_{\forall k \to \epsilon_y(k)\leq\alpha} 1}{\#k} \\ P(\tilde{y}\ is\ U) &= P(\epsilon_y(k) > \alpha) = \frac{\sum_{\forall k \to \epsilon_y(k)>\alpha} 1}{\#k} \end{aligned} \tag{4}$$

Where $\#k$ is the number of considered k cut the exploration is investigating. If α is more significant than the maximum $\epsilon(k)$ out of a precision reduction, input nodes will not show any U cases.

Operation nodes propagate the error from their inputs by performing a specific computation. As stated in [21], since the input error can be either masked or amplified, depending on how the operation handles the inputs, their characterization requires determining how this propagation occurs and classify the results according to the three accuracy classes (P, A, or U).

In this NN case, we need to evaluate three operators: (i) sum, (ii) multiplication, and (iii) the sigmoid function ($S(x)$). The first two operations are trivial to define. However, the sigmoid is the activation function of each neuron in the reference NN employed in the paper, which will take as input the result of the multiply and accumulation across NN inputs, weights, and biases, and will provide new inputs to intermediate layers or the output of the NN. The following formula defines the $S(x)$ employed in this paper:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

In [21], we resorted to the enumeration of all cases to define the operators' CPTs. Since the fixed-point data type makes the enumeration a less trivial task, we devised an Algorithm to evaluate the CPTs automatically. It is important to stress out that this operation will be necessary only once, given the α threshold to be analyzed. Algorithm 1 depicts the algorithm for operators with two inputs, i.e., sum and multiplication. It expects four inputs: (i) the operator, (ii) the threshold, (iii) the error difference considered acceptable as change within the CPT values from the ones evaluated on the previous step, and (iv) the minimum number of values to test. The output is the CPT for the given operator.

The algorithm explores the huge space of all possible combinations of two inputs efficiently. It stops when it has evaluated a minimum number of combinations (`minNValues`) as well as when the maximum difference in the single values of the CPT is below a given value (`EM`), line 15 in Alg. 1. The algorithm is straightforward; at each iteration, it generates a random number (`p` at line 5), as well as two versions of it, one belonging to the 'A' class and one to the 'U' class (`a` and `u` at lines 6-7). Other two 'A' and 'U' numbers are generated from two different random numbers (`a2` and `u2` lines 8-9). Those five values define the set of combinations tested to update the CPT (`comb_to_test` in lines 10-12). Each combination outcome is compared against the same combination using

Algorithm 1: Algorithm to define the CPT for a single operator (OP)

Input: OP = Operator to Analyse;
α = threshold;
EM = acceptable error difference;
minNValues = minimum number of values to test
Output: CPT_{OP} = OP's CPT

```
1  e = MAX_ERROR;
2  CPT_OP = all columns sum equal to 1;
3  repeat
4  |  OLD_CPT_OP = CPT_OP;
5  |  p = random ();
6  |  a = gen_acceptable_version(p, α);
7  |  u = gen_unacceptable_version(p, α);
8  |  a2 = gen_acceptable_version(random (), α);
9  |  u2 = gen_unacceptable_version(random (), α);
10 |  comb_to_test = [(p,a), (p, u), (a,a2), (a, u), (u, u2)];
11 |  foreach c ∈ comb_to_test do
12 |  |  evaluate_and_update(c, OP, α, CPT_OP) );
13 |  end
14 |  e = max (OLD_CPT_OP - CPT_OP);
15 |  minNValues = minNValues - 1;
16 until e > EM || minNValues > 0;
```

the corresponding precise version of all 'A's and 'U's. The classification of the `OP` outcome of each combination, used as input, allows updating the CPT. It is the purpose of the `evaluate_and_update` function. Eventually, line 14 evaluates the difference between the new CPT and the old one.

Algorithm 1 is designed for two input operators. Since the third operand in a NN is the sigmoid, we employed the same approach to the $S(x)$ operator of eq. 5. The only difference when investigating $S(x)$ is that lines 8-9 are unnecessary because the operator is a unary one. Still, for the same reason, line 10 does not define combinations, but all the three values generated at lines 5-7 are going to be the input of the evaluation loop at lines 11-13.

The results included in this paper leverage on a `minNValues` set as the 1E+9 of all possible combinations of two inputs, and a `EM` equal to 1E-6. Moreover, all random numbers follow a uniform distribution, which can be changed if shreds of evidence might show a different distribution in values.

C. Approximation analysis

The Bayesian inference allows the analysis of the model to predict the approximation at the output of the NN [27].

We compute the posterior probability of the leaves of the network (orange nodes in Figure 1) to be in one of the three approximation classes defined in equation 3 to have the necessary prediction. Literature is full of different update belief algorithms for this scope. In particular, the library used to implement the proposed framework [22] provides two solvers: (1) the exact solver proposed by Lauritzen in [28], and (2) the Estimated Posterior Importance Sampling (EPIS) approximate stochastic solver proposed in [29]. The first one might prove helpful with medium-size models (i.e., tens of nodes), while the second one suits huge models (i.e., thousands of nodes). Results in this paper come from the exact solver since the analyzed BN was small enough.

The proposed model's flexibility can help insight the accuracy reduction of the application quickly, thus quickly exploring different design solutions.

III. Experimental Results

While in [21] we demonstrated the approach on single benchmarks, in this paper, we focus on correctly predicting the approximation quality of a Neural Network outcome. For this reason, we work on a small network able to implementing the XOR bit-wise operation. This example is quite simple but, at the same time, complex enough to show the benefits of our approach. Fig. 2a depicts the topology of the XOR neural network. It is composed of two inputs, two layers, and one output. The first layer contains two neurons, while the last layer contains only one neuron. Each neuron has the same structure as the example shown in Fig. 1 plus an extra input called bias. All the activation functions are Sigmoid. The XOR network implementation consists of a C code where the data types are 32-bit fixed-point with 16 bits representing the fractional part and 16 the fractional part.

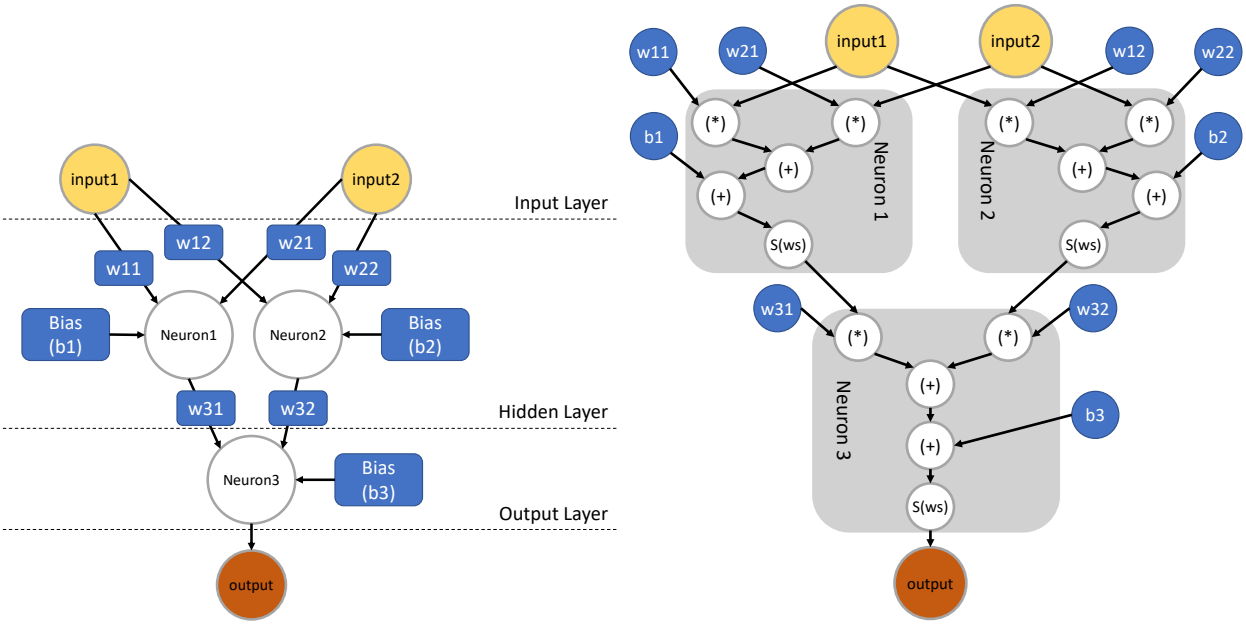
The experimental setup defines a small test set of all possible combinations of input values, and the primary constraint is to avoid approximation on the input values. The network train was a precise one. The reduction was applied in the range of $[0, 31]$ bits, meaning that 0 is no approximation and 31 refers to all but the most significant bit set to 0. The range was set as a huge one on purpose, with the sole intention of proving the BN's predictive capability. As mentioned in Section II-B, this reduction range will only change the probability of the three classes based on the trained values of each weight or bias, leaving the propagation estimation to the Bayes theorem unaltered in methods.

A. Prediction Reliability

To effectively demonstrate the modeling's prediction capability if applied to NNs, we run the iterative exploration against the BN modeled based on different α values. This process requires generating the CPT tables of all operators for each α and then use them on the previously one-time modeled BN. Together with the CPT, the weights and biases distribution have been computed and assigned to the CPT's proper nodes. Table I shows the three classifications for all three alpha used.

Reported results confirm the reliability of the BN prediction, showing a maximum percentage point error of 5.42pp. The absolute error variability reflects the data dependency of the multiplication, as already demonstrated in [21]. Moreover, it also accounts for the difference between the input node actual distribution and the uniform distribution used for the operators' characterization. Using a characterization based on a different distribution is a choice we made to demonstrate

Fig. 2: XOR network



(a) The original network

(b) The BN equivalent network

TABLE I: Experimental Results

α		P	A	U
1	BN	76.52%	23.48%	0.00%
	App. Run	76.37%	23.63%	0.00%
	Abs. Error	**0.15pp**	**0.15pp**	**0.00pp**
0.1	BN	71.83%	0.01%	28.15%
	App. Run	76.37%	0.89%	22.73%
	Abs. Error	**4.54pp**	**0.88pp**	**5.42pp**
0.0001	BN	57.79%	0.00%	42.20%
	App. Run	61.58%	1.45%	36.96%
	Abs. Error	**3.79pp**	**1.45pp**	**5.25pp**

the characterization's re-usability when feasible. The possible values in this case (0 and 1 only) were not worth a specific characterization. Nonetheless, since the outcome comes from a non-linear function such as the sigmoid, it is interesting to notice that we could adequately model it using the BN. Moreover, having the three α expressing a considerable variation in the quality tolerance, the absolute errors confirm the BN prediction's reliability.

B. Efficiency & optimization

While it is still on a tiny scale, the first row of Table II shows the number of combinations required to explore all possible configurations of precision reduction of the six weights and three biases of the network. The same row also reports the time required to fully explore all of them, i.e., to apply the select combination of bits to cut on each of the nine items, run all test cases, collect and classify the results. The second row reports how the BN behaves when used to explore the NN approximation. The second column displays the number of alternatives that remain to explore after running a backward influence analysis, as possible in the engine included in the framework in [21]. The algorithm can analyze the BN in order to highlight the dependency of influence between nodes. Using this tool, we selected the three biases as the most critical nodes that influenced the NN's capability to produce precise outcomes, even when other nodes see a heavy approximation. This exclusion from the approximation set reduces the design space to 6 out of 9 nodes, and the second column of Table II reports the overall reduction in the exploration space, as the number of new combinations to check, which is the 3.6% of the full-size space. Therefore, the backtrack analysis results made it possible to have precise outcomes on every test input, just avoiding approximation on the biases. This outcome was correctly predicted by the BN, imposing the node states as precise. In the NN implementation, we run all combinations left, avoiding approximating the biases simultaneously, checking the NN results. From a timing perspective, it is essential to remark that reported times, while a few seconds higher than the iterative approach, include operators' characterization process, reusable when another NN employs the same data type α values. The analysis timing opens for potential scalability gain when the NN to analyze grows. Suppose the operators are not going to change. In that case, the number of combinations to explore will see the exponential growth of the iterative approach's timing, while the BN analysis time has room for coping with the growth by switching from an exact solving algorithm to an approximated one.

IV. Conclusion

In this paper, we proposed a probabilistic method to analyze the impact of the precision reduction approximation to data nodes of a Neural Network. The proposed approach can estimate the effect of the approximation on the network

TABLE II: Comparison between experimental exploration and BN approach when Influence analysis's results are applied to the exploration

Technique	# alternatives to explore	Time (s)
Iterative Approach	$\binom{31}{9} = 2.0160075E+7$	204.96
BN	$\binom{31}{6} = 736281$	231.10 = 00.02 (solver + analysis) 81.88 (add) + 83.95 (mul) + 65.26 (sigmoid)

outcomes, covering all possible numbers of bits of reduction and selecting a specific approximation. The neural network translates into a Bayesian Network model in which each of the three operators handling data, i.e., sum, multiplication, and the sigmoid function, are characterized only once. The characterization is a one-time task that produces a library of CPTs re-used on different neural network models. To assess the accuracy of the obtained estimations, we evaluated the neural network responses against the BN's prediction, using different quality thresholds for the classifications. A backtrack influence analysis demonstrates the capability of efficiently support the design space exploration by reducing the number of weights and biases to approximate. Results show the accuracy of the prediction and the gain in terms of design exploration time.

ACKNOWLEDGMENT

This study has been achieved thanks to the financial support of the projects "IDEX Lyon OdeLe"

REFERENCES

[1] A. Bosio, S. D. Carlo, P. Girard, E. Sanchez, A. Savino, L. Sekanina, M. Traiola, Z. Vasicek, and A. Virazel, "Design, verification, test and in-field implications of approximate computing systems," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–10.

[2] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356

[3] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing."

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*. IEEE, 2013, pp. 1–6.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 97–107, 2014.

[6] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 113.

[7] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 120.

[8] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," *arXiv preprint arXiv:1511.06488*, 2015.

[9] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 124–137, 2013.

[10] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *Computers, IEEE Transactions on*, vol. 62, no. 9, pp. 1760–1771, 2013.

[11] A. Savino, M. Portolan, R. Leveugle, and S. Di Carlo, "Approximate computing design exploration through data lifetime metrics," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–7.

[12] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 187–192.

[13] K. Nepal, Y. Li, R. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 361.

[14] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[15] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 27.

[16] M. Barbareschi, F. Iannucci, and A. Mazzeo, "Automatic design space exploration of approximate algorithms for big data applications," in *IEEE International Conference on Advanced Information Networking and Applications (AINA-2016). IEEE*, 2016.

[17] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[18] K. N. Parashar, D. Menard, and O. Sentieys, "Accelerated performance evaluation of fixed-point systems with un-smooth operations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 599–612, April 2014.

[19] R. Rocher, D. Menard, P. Scalart, and O. Sentieys, "Analytical approach for numerical accuracy estimation of fixed-point systems based on smooth operations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2326–2339, Oct 2012.

[20] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo, and A. Bosio, "Predicting the impact of functional approximation: from component- to application-level," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 61–64.

[21] M. Traiola, A. Savino, and S. Di Carlo, "Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications," *Microelectronics Reliability*, vol. 102, p. 113309, 2019.

[22] BayesFusion, LLC. SMILE Engine. [Online]. Available: www.bayesfusion.com

[23] Libfixmath: Cross platform fixed point maths library. [Online]. Available: https://github.com/PetteriAimonen/libfixmath

[24] M. Baranowski, S. He, M. Lechner, T. S. Nguyen, and Z. Rakamarić, "An smt theory of fixed-point arithmetic," in *Automated Reasoning*, N. Peltier and V. Sofronie-Stokkermans, Eds. Cham: Springer International Publishing, 2020, pp. 13–31.

[25] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "Sensitivity analysis and compression opportunities in dnns using weight sharing," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2020, pp. 1–6.

[26] ——, "On the automatic exploration of weight sharing for deep neural network compression," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1319–1322.

[27] G. E. Box and G. C. Tiao, *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011, vol. 40.

[28] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International journal of approximate reasoning*, vol. 15, no. 3, pp. 225–263, 1996.

[29] C. Yuan and M. J. Druzdzel, "An importance sampling algorithm based on evidence pre-propagation," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 624–631.