Using Machine Learning for Quality Configurable Approximate Computing

Mahmoud Masadeh, Osman Hasan, and Sofiène Tahar
Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada
Email:{m_masa, o_hasan, tahar}@ece.concordia.ca

***Abstract*—Approximate computing (AC) is a nascent energy-efficient computing paradigm for error-resilient applications. However, the quality control of AC is quite challenging due to its input-dependent nature. Existing solutions fail to address fine-grained input-dependent controlled approximation. In this paper, we propose an input-aware machine learning based approach for the quality control of AC. For illustration purposes, we use 20 configurations of 8-bit approximate multipliers. We evaluate these designs for all combinations of possible input data. Then, we use machine learning algorithms to efficiently make predictive decisions for the quality control of the target approximate application, based on experimentally collected training data. The key benefits of the proposed approach include: (1) fine-grained input-dependent approximation, (2) no missed approximation opportunities, (3) no rollback recovery overhead, (4) applicable to any approximate computation with error-tolerant components, and (5) flexibility in adapting various error metrics.**

I. Introduction

The pervasive, portable, embedded and mobile nature of the present age computing systems, has led to an ever increasing demand for ultra low power consumption, small footprint, and high performance systems [1]. Such battery-powered systems are one of the most essential componenets of IoT, near-sensor processing and edge computing. Approximate Computing (AC), is a nascent computing paradigm that allows us to achieve these objectives by compromising the arithmetic accuracy. Thus, its popularity is rising among the error resilient applications, such as multimedia, and visualization.

Approximation techniques necessitate a *quality management system* to control the degree of approximation and monitor the individual output quality. However, this important area has attained less attention from the scientific community compared to the design of AC. Setting design knobs of approximate circuits, can be realized with two strategies: i) *forward problem* design, which focuses on the effect of setting design knobs on the output of a tunable approximate design, ii) *inverse problem* of finding the optimal design knobs setting for a given bound of output quality. The inverse strategy is less explored compared to the forward problem because it has a huge input-dependent design space. In this paper, we tackle the inverse problem to find the optimal knob settings of the approximate design that meet the *target output quality* (*TOQ*) constraints for an application, given input data and specified user preferences.

There exist a few techniques for quality managenent of AC, e.g., Green [2] and SAGE [3] check the quality through *sampling*, where for every *N* invocations, the exact result is compared with the approximate. If the difference is greater than a pre-specified threshold, i.e., *TOQ*, then a *calibration* is done. However, the quality of unchecked invocations cannot be ensured, and the previous quality violations cannot be compensated. *Lightweight* quality checkers [4] [5] with high-efficiency are used in order to decide about the usage of rollback approximation. However, the quality checker proposed in [4] is application-specific, and [5] shows a low prediction accuracy for large applications where less than 20% of the inputs in the data space still exhibit unavoidable large approximate errors. Wang et al. [6] proposed using *multiple lightweight predictors* to get a high accurate prediction. Xu et al. [7] proposed an optimization framework, which supports an iteratively training process, to coordinate the training of the classifier and the accelerator with an intelligent selection of training data. However, the work [2]-[7], mainly target controlling software approximation, ignore input dependencies, and do not consider choosing an adequate design from a set of design choices.

Dynamic quality management of approximate hardware design is challenging. Raha et al. [8] designed a dual-mode, reconfigurable adder block. The authors of [9] proposed four designs of 4-to-2 compressors. Moreover, the deployment of a self-adaptive image filter, which is able to choose among different degrees of binary adder approximations at run time is also demonstrated in [10]. However, this approach changes the degree of approximation for a single design only. Likewise, the approaches reported in [8], [9] and [10], either have very limited configuration options (e.g., [8]), area overhead (e.g., [9]) or latency overhead (e.g., [10]).

In this paper, we propose an approach based on machine learning (ML) for controlling the quality of input-aware AC. We propose to change the approximate design dynamically for different input data to meet a user-defined TOQ, rather than having a static design. However, there is no clear relationship between the inputs of approximate designs and their errors, therefore we propose to build ML-based models to pick the proper configuration. In order to demonstrate the practical effectiveness of the proposed approach, we designed 8-bit approximate multipliers, which are intensively used in image processing applications, with 20 different configurations [1]. Then, we experimented with various ML algorithms to efficiently make predictive decisions for the quality control of approximate applications, based on experimentally collected training data. We were able to model the approximation error based on the input data and design settings. Moreover, for a specific input data and TOQ, we dynamically select the design with matched output quality and minimal cost. We applied the proposed methodology on image blending operations, where we processed each frame using three approximate designs, based on the value of its red, green and blue components, and matched the required TOQ.

The rest of the paper is organized as follows: Sections II and III describe the proposed methodology and the ML-based models we built, respectively. Section IV provides an image blending application. Finally, Section V concludes the paper.

II. Proposed Methodology

The proposed methodology for the quality control of AC is based on ML techniques to build input-aware quality and cost models. Such models are suitable for configuring the knobs of input-dependent approximate designs. Consider a *to-be-controlled* approximate design with two knobs/settings, i.e., *S1* and *S2*, that take values from finite sets *s1* and *s2*, respectively, where *s1* represents the type of the approximate block used to build the approximate design, and *s2* is the approximation degree of the design. For example, in the case of approximate 8-bit multipliers, *s1*= $\{AMA1, AMA2, AMA3, AMA4, AMA5\}$, i.e., selected from the low power approximate full adders [11] and *s2*=$\{D1, D2, D3, D4\}$, where $D1$ has 7 bits approximated out of 16, while $D2$, $D3$, and $D4$ have 8, 9, and 16 approximate bits, respectively. Thus, for the case of approximate multipliers, we obtain 20 different approximate designs. Now, we define two functions to characterize the quality of the considered approximate design: (1) *Error Function* $f_e(i, S1, S2)$ that defines the output error for a specific input data and knob settings; (2) *Cost Function* $f_c(i, S1, S2)$ that defines the cost of computing the output for a specific input data and knob settings. This cost may be based on power consumption, area, delay or power-delay-product (PDP).

We define the feasibility constraint as follows: Given an approximate design with two knob settings *S1* and *S2*, and a set of possible inputs I. For a given input value $i \in I$, and *TOQ*, find s1 $\in$ *s1*, and s2 $\in$ *s2*, such that the objective cost function $f_c(i, S1, S2)$ is minimized, and the error function is within the error bound limit, i.e., $f_e(i, S1, S2) \leqslant$ *TOQ*. The solution is a point, or a set of points, which minimizes the cost function. Changing the knob settings for every possible input is power-inefficient. For example, for an 8-bit binary multiplier based on 20 different settings, the design space includes 20x2^{16}=$1,310,720$ different settings. Therefore, we propose to cluster the input data based on its magnitude into C=16 different clusters. As depicted in Figure 1, the proposed methodology for quality control of AC encompasses the following steps:

(1) *Training Inputs*: For an approximate design with m inputs, i.e., i_1, i_2, ..., i_m, of n-bit width, and value ranges from 0 to 2^n-1, we apply j inputs of k-bit width to the approximate design in order to generate the training data, where $j \leq m$ and $k \leq n$. For example, for an 8-bit binary approximate multiplier with $m = 2$ and $n = 8$, we choose to use the whole range of inputs for training, i.e., $j = m = 2$ and $k = n = 8$.

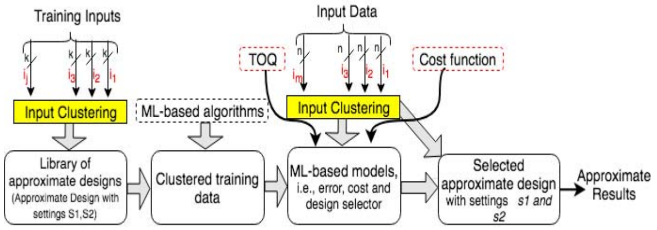

Figure 1: Proposed ML-based methodology for quality configurable AC

(2) *Input Clustering*: Rather than having j inputs, each with 2^k possible values, applied to an approximate design with $|S1|$x$|S2|$ different settings, we cluster such inputs into C different clusters. The clustering is indispensable to evaluate design accuracy over a range of $(\frac{2^k}{C})^j$ inputs, where evaluating the design accuracy for a single input is inapplicable. For the example of an 8-bit approximate multiplier, each design has a $(2^k)^j = 65536$ possible input combinations. For that, we use a cluster size of 16, thus the total number of input combinations is reduced to $(\frac{2^k}{C})^j = (\frac{2^8}{16})^2 = 256$ combinations.

(3) *Library of Approximate Designs*: An approximate design with two knobs, i.e., *S1* and *S2* will have $|S1|$x$|S2|$ different design settings that constitute our library of approximate designs. For the example of the 8-bit approximate multiplier [1], we have $|S1|$x$|S2|$=5x4=20 different settings with 13% to 65.9% power saving compared to the exact design.

(4) *Training Data*: Based on $|S1|$x$|S2|$ different design settings, and $(\frac{2^k}{C})^j$ different clustered inputs, we generate a training data of $|S1|$x$|S2|$x$(\frac{2^k}{C})^j$ instances. Each instance includes the clusters of the applied inputs, design settings, and various accuracy metrics derived from the approximate results, such as error rate, accuracy, error distance, relative error distance, mean square error, peak-signal-to-noise ratio and normalized error distance. Moreover, each approximate design has its area, power, delay and PDP reduction compared to the exact design. For the example of the 8-bit approximate multiplier, we have 5x4x16^2=5120 training instances. To build ML-based models, we randomly partitioned the training data into 80% for training and 20% for testing purposes.

(5) *Error and Cost Models*: Error and cost functions of an approximate design for specific input data, do not have a closed-form analytic expression, therefore we built and evaluated various ML-based models using the training and testing data, respectively, as will be described in Section III-A.

(6) *Design Selector*: A distinctive characteristic of our proposed methodology is its input-data dependency. Therefore, for each set of specific inputs and given *TOQ*, we first identify the most appropriate approximate designs and then select the ones with the lowest cost function, e.g., area, power and delay as will be described in Section III-B. For the example of the 8-bit approximate multiplier, the selected design is used within an error-resilient multiplication-intensive application.

III. Machine Learning Based Models

ML-based algorithms find solutions by learning through a training data. We built various ML-based models, based on the analysed data and several regression and classification

Table I: ML-based models for error prediction

Algorithm	Category	Accuracy
Simple Linear Model (Independent variable is Approximation Degree)	Regression	R^2=6.8%
Simple Linear Model (Independent variable is FA Type)	Regression	R^2=0.3%
Simple Linear Model (Independent variable is Input 1)	Regression	R^2=1.3%
Simple Linear Model (Independent variable is Input 2)	Regression	R^2=2.1%
Multiple Linear Model	Regression	R^2=10.2%

C5.0 based Decision Tree	Classification	Accuracy=66.8% (Train)
		Accuracy=56.2% (Test)

rpart based Decision Tree	Classification	Accuracy=54.0% (Train)
		Accuracy=50.6% (Test)

rpart based Decision Tree	Regression	R^2=63.65
Random Forest	Regression	R^2=67.7%

algorithms, given in R, which is a programming and statistical computing language. The predicted result for such models falls into two categories, classification and regression. *Classification* is the problem of identifying to which set of specific categories does a new observation belong. The *accuracy* of a classifier is defined as the fraction of correct classifications made on the test data set. *Regression* estimates a *continuous* dependent variable, through explaining how such dependent variable changes in response to changes in independent variable. R^2 is used as a measure of regression model accuracy, which is the proportion of the variance in the dependent variable that is predictable from the independent variables. Next, we build models utilizing different algorithms, where we evaluate their accuracy for selection.

A. Error Models

Several metrics have been proposed for quantifying errors of approximate designs. We chose the normalized mean error distance (NED) as our error metric as its results are almost independent of the size of an approximate design and can be used for comparing designs of different sizes. We built various ML-based models to predict the design error based on the input data and design settings, i.e., full adder (FA) type and approximation degree for the case of an approximate multiplier. Table I summarizes these models including its settings, category, i.e., regression or classification and the obtained accuracy.

Linear regression models are simple to build, however, they are unable to identify a clear relationship between the approximation error and the associated design settings and input data. On the other hand, *decision tree* models exhibit a better accuracy for both training and testing data, including classification and regression. The lowest obtained accuracy is 56.2% and 50.6% for testing data based on C5.0 and *rpart* algorithms, respectively. The *random forest* based model has a slightly improved accuracy compared to the decision tree models with the overhead of constructing 300 trees instead of one. We have chosen the *decision tree regression models*, based on *rpart*, due to their good accuracy, small training time, and ease to understand. Neural network models are expected to have a high accuracy with high training time, therefore, due to space limitation we kept it for future work. The previously built models consider the whole range of the input data to build a single model. However, as approximation is data dependent, we built a decision tree model for each input cluster, where we clustered the first input data into 16 clusters as explained in

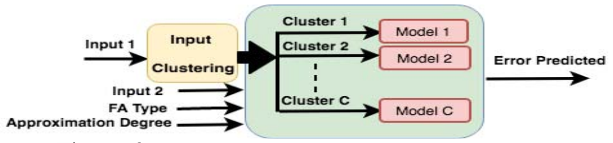

Figure 2: C ML-based error models for clustered input

Section II. Figure 2 depicts the general overview for training C error models. These models, compared to a single model, are smaller, faster to construct and more accurate.

B. Design Selector

Approximate design selector uses the input data and the allowed error to select the most suitable design, i.e., *FA Type* and *Approximation Degree*. When the model is unable to simultaneously predict the two variables, then we join them into one variable, and call it *Full Design*.

Table II: Various ML-based models for design selector

Algorithm	Category	Accuracy
Multivariate Multiple Regression Model (Dependent variable is Approximation Degree)	Regression	R^2=6.09%
Multivariate Multiple Regression Model (Dependent variable is FA Type)	Regression	R^2=0.324%
Multivariate Multiple Regression Model (Dependent variable is Full Design)	Regression	R^2=0.033%

C5.0 based Decision Tree for Approximation Degree	Classification	Accuracy=73.05% (Train)
		Accuracy=63.5% (Test)

C5.0 based Decision Tree for FA Type	Classification	Accuracy=45.1% (Train)
		Accuracy=31.5% (Test)

C5.0 based Decision Tree for Full Design	Classification	Accuracy=45.1% (Train)
		Accuracy=26% (Test)

rpart based Decision Tree for Approximation Degree	Classification	Accuracy=69.1% (Train)
		Accuracy=60.2% (Test)

rpart based Decision Tree for FA Type	Classification	Accuracy=43.2% (Train)
		Accuracy=31.2% (Test)

rpart based Decision Tree for Full Design	Classification	Accuracy=40% (Train)
		Accuracy=24.6% (Test)

Table II summarizes various models for the design selector based on multivariate regression models, decision trees and random forest. *Regression models* with low prediction accuracy are unsuitable. The *decision tree* classification model, based on the C5.0 algorithm, is the most accurate for predicting the *Full Design*. The exhaustive search over the entire design space of knob settings is applicable, to find the optimal design settings. The first input (*Input1*) is clustered based on its magnitude into C different clusters. Then, based on the matched cluster, a reduced search space from the training data is selected with size of $\frac{1}{C}$ of the original search space. Consequently, an exhaustive search is performed on the matched cluster, and the best settings are identified. Theoretically, we may get up to $|S1|$x$|S2|$ different settings. Among these candidate designs, we select the one with the maximum matched TOQ, which is quite likely to have the greatest approximation degree with a reduced cost.

IV. VIDEO/IMAGE BLENDING APPLICATION

Image blending allows us to blend multiple images together to look like a single image. *Video blending* consists of blending multiple consecutive frames. For example, blending two videos, each with N_f frames of size N_r rows by N_c columns per frame, where each pixel in the frame has 3 components, i.e., red, green and blue, involves $3N_f$xN_rxN_c pixels. The static configuration uses a single approximate design to perform all multiplications. Whereas, in the dynamic reconfiguration, we have the choice to decide the task granularity and the suitable design to be used. To control output quality,

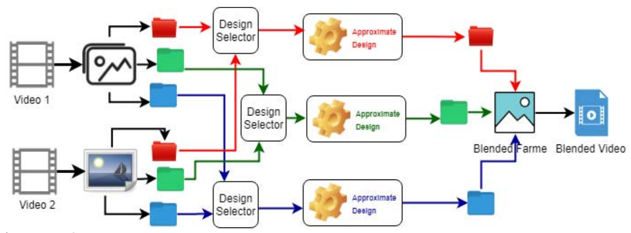

Figure 3: Video blending using reconfigurable multipliers for RGB components of each frame

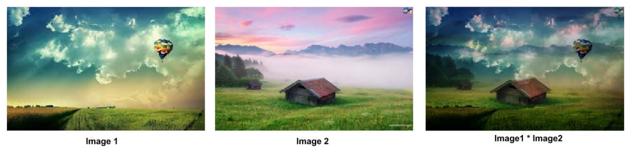

Figure 4: Example of image blending

we select the most suitable design for all RGB components of a frame, e.g., R, G and B, as shown in Figure 3. We propose to change the design $3N_f$ times, where each frame is processed using three designs, with task granularity of N_rxN_c pixels. The approximate design is selected based on the average of RGB components of the two frames. Due to space limitation, we discuss an image-based example here, rather than a full-movie example having N_f images.

An example of pixel-by-pixel image multiplication is shown in Figure 4. The results of using the 20 approximate designs and their peak signal to noise ration (PSNR), are evaluated, where each static approximate design is used for R, G and B components of the images. The results of designs with $D1$, $D2$ and $D3$ approximation degree, have a good quality with acceptable PSNR between 39 and 62.

Table III: Dynamic approximate designs selected based on magnitude of RGB components and TOQ

		Average value of the frame			Selected Designs		
NED		**R**	**G**	**B**	**R**	**G**	**B**
NED ≤ 0.03	Image1	120	149	117	AMA4_9, AMA5_9	AMA4_9, AMA5_9	AMA4_9, AMA5_9
	Image2	160	156	147			
NED ≤ 0.60	Image1	120	149	117	AMA2_16, AMA4_16 AMA5_16	AMA4_16 AMA2_16	AMA4_16, AMA2_16 AMA5_16
	Image2	160	156	147			

Table III summarizes the obtained results, for two examples with different TOQ, by applying the proposed methodology based on the average value of R, G and B image components. For example, the selected design *AMA4_9* means that the design is based on *AMA4* FA with 9 approximate bits of the result. We notice that, as the TOQ decreases (NED increases), the degree of approximation of the selected designs increases.

Now, we compare the proposed approach with the state-of-the-art [2]-[10] in terms of several requirements: (1) *Hardware/Software*: The work [2]-[7] exhibit a limited applicability for software approaches only. The approaches, presented in [8]-[10], are applicable to hardware with limited configuration options. The proposed approach is applicable to both HW and SW approximate applications with approximable components; (2) *Input Data Dependency*: To the best of our knowledge, none of the previous works, targeted fine-grained input-dependency of approximate designs to control the output quality. Instead, the average of all inputs is considered to meet an average output quality. Our approach uniquely clusters the input data, then uses these clusters with a set of approximate designs to generate training data. Thereafter, we build ML-based models for each input data cluster; (3) *Quality Assurance & overhead*: The quality of the output is assured through rollback and program reexecution in [2]-[7]. These methods require extra time and out-of-order-execution. The approaches [8]-[10] rely on having a few designs in order to meet the required quality, which also leads to additional area and the requirement to have complex controllers for multiple designs. On the other hand, the proposed approach needs a one-time data generation, training and model building. Then, the most suitable design is identified, and used through reconfiguration, without the overhead of quality checkers and rollback recovery.

V. Conclusion

In this paper, we proposed a novel fine-grained input-based quality control approach for AC based on ML models. The proposed solution considers the input data in generating the training data, building ML-based models and selecting the approximate design that satisfies the TOQ with minimal cost. This approach is applicable to both hardware and software designs. However, it has the overhead of one-time generating the training data, building and evaluating various models, and then selecting the most suitable models. As future work, we plan to implement a fully automated version of quality control of AC using ML-models built based on the input data, for both hardware and software approximate designs.

References

[1] M. Masadeh, O. Hasan, and S. Tahar, "Comparative study of approximate multipliers," in *Great Lakes Symposium on VLSI*. ACM, 2018, pp. 415–418.

[2] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," *SIGPLAN Notices*, vol. 45, no. 6, pp. 198–209, Jun. 2010.

[3] M. Samadi, J. Lee, D. Jamshidi, A. Hormati, and S. Mahlke, "SAGE: Self-tuning approximation for graphics engines," in *International Symposium on Microarchitecture*, 2013, pp. 13–24.

[4] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing," in *International Symposium on HPC Architecture*, 2015, pp. 615–626.

[5] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *International Symposium on Computer Architecture*, 2015, pp. 554–566.

[6] T. Wang, Q. Zhang, N. Kim, and Q. Xu, "On effective and efficient quality management for approximate computing," in *International Symposium on Low Power Electronics and Design*, 2016, pp. 156–161.

[7] X. Chengwen, W. Xiangyu, Y. Wenqi, X. Qiang, J. Naifeng, L. Xiaoyao, and J. Li, "On quality trade-off control for approximate computing using iterative training," in *Design Automation Conference*, 2017, pp. 1–6.

[8] A. Raha, H. Jayakumar, and V. Raghunathan, "Input-based dynamic reconfiguration of approximate arithmetic units for video encoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 3, pp. 846–857, 2016.

[9] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 4, pp. 1352–1361, 2017.

[10] J. Pirkl, A. Becher, J. Echavarria, J. Teich, and S. Wildermann, "Self-adaptive FPGA-based image processing filters using approximate arithmetics," in *International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES. ACM, 2017, pp. 89–92.

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