# Machine-Learning-Based Self-Tunable Design of Approximate Computing

Mahmoud Masadeh, *Student Member, IEEE*, Osman Hasan, *Senior Member, IEEE*, and Sofiène Tahar, *Senior Member, IEEE*

***Abstract*— Approximate computing (AC) is an emerging computing paradigm suitable for intrinsic error-tolerant applications to reduce energy consumption and execution time. Different approximate techniques and designs, at both hardware and software levels, have been proposed and demonstrated the effectiveness of relaxing the *average* output quality constraint. However, the output quality of AC is highly input-dependent, i.e., for some input data, the output errors may reach unacceptable levels. Therefore, there is a dire need for an input-dependent tunable approximate design. With this motivation, in this article, we propose a lightweight and efficient machine-learning-based approach to build an input-aware design selector, i.e., *quality controller*, to adapt the approximate design in order to meet the target output quality (TOQ). For illustration purposes, we use a library of 8-bit and 16-bit energy-efficient approximate array multipliers with 20 different settings, which are commonly used in image and audio processing applications. The simulation results, based on two sets of images, including an *8 Scene Categories Dataset,* which is a benchmark of images data set, demonstrate the effectiveness of the lightweight selector where the proposed tunable design achieves a significant reduction in quality loss with relatively low overhead.**

***Index Terms*— Approximate computing (AC), approximate multiplier, decision tree (DT), image processing, input-aware approximation, machine learning (ML), neural network (NN), tunable design.**

## I. Introduction

Due to the enormous explosion in new data, i.e., big data, generated by billions of small, low-power, ubiquitous computing devices, the computer architects are rethinking the whole computing stack to process such big data in a timely and energy-efficient manner. Furthermore, the present-age embedded and mobile computing systems require an ultralow power consumption, small footprint, and high-performance designs [1], [2]. These battery-powered systems are integral components of the Internet of Things (IoT), cyber–physical systems (CPSs), near-sensor processing, and edge computing. Approximate computing (AC) [3], known as best-effort computing, is a nascent computing paradigm that promises to meet these objectives, by sacrificing the arithmetic accuracy of the results.

The popularity of AC is rising among the error-resilient applications, such as multimedia, web search, data mining, and visualization, where computation is naturally tolerant to some degree of imprecision. Moreover, the final output with reduced quality is tolerable by perceptual, i.e., visual or hearing, human limitations [4]. Despite the unprecedented power saving and reduced execution time introduced by AC, it is still an immature computing paradigm, where a formal model of the impact of approximation on other design metrics is still missing [4]. Various works have applied AC techniques at different layers of the computing stack, i.e., circuit design, architectures, and application software, to reduce power consumption but have not yet explored their impact on the quality of individual outputs. A detailed description of these computing layers and approximation techniques can be found in [3].

Both hardware and software approximation techniques require a *quality assurance* to adjust approximation settings/knobs and monitor the quality of fine-grained individual outputs. To the best of our knowledge, this pivotal direction has acquired less attention from the scientific community compared to the design of AC. In order to assure the quality of results, there are two strategies to adjust the settings of an approximate program: 1) *forward design* [5] that sets the design knobs and then observes the quality of results; however, the output quality of some inputs may reach unacceptable levels and 2) *backward design* [6] that tries to find the optimal knobs setting for a given bound of output quality; this requires exploring a tremendous space of knob settings for a given input, which is intractable [7].

To overcome the abovementioned limitations of design approaches, we propose a *tunable approximate design* that allows altering the settings of approximation, at runtime to meet the target output quality (TOQ). The main idea is to develop a machine learning (ML)-based input-aware design selector, which is able to modify the approximate design based on the applied inputs, in order to meet the required quality constraints. Our approach is general in terms of quality metrics and supported approximate designs. It is primarily based on a library of 8-bit approximate multipliers with 20 different configurations and well-known power dissipation, performance, and accuracy profiles [1]. Moreover, we utilize a backward design approach to dynamically adapt the design in order to meet the desired TOQ based on ML models [8]. The TOQ is a user-defined quality constraint, which represents the maximum allowable error for a given application. The *contributions* of this article are as follows:

1) a tunable approximate design approach based on fine-grained input data to satisfy output quality constraints set by the user;
2) an accuracy evaluation approach utilizing a magnitude-based clustering/quantization of input data of the approximate designs;

Manuscript received September 12, 2020; revised November 27, 2020; accepted January 27, 2021. Date of publication February 23, 2021; date of current version April 1, 2021. *(Corresponding author: Mahmoud Masadeh.)*

The authors are with the Department of Electrical and Computer Engineering, Concordia University, Montreal, QC H3H 2J2, Canada (e-mail: m_masa@ece.concordia.ca).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3056243.

Digital Object Identifier 10.1109/TVLSI.2021.3056243

3) an input-dependent ML-based, i.e., decision tree (DT) and neural network (NN), design selector;
4) a fully automated toolchain based on the abovementioned approaches;
5) a validation of the proposed approach based on image processing applications, i.e., image blending and filtering.

The rest of this article is organized as follows. Section II introduces the related work on AC quality assurance. Section III explains the proposed methodology. Section IV summarizes the accuracy and other design metrics of the considered library of approximate multipliers. Section V explains the AC design selector, based on DT and NN, and its characteristics. Section VI provides experimental results of an image/audio blending and filtering applications. The obtained results are compared with related work in Section VII. Finally, Section VIII concludes this article with some future directions.

## II. Related Work

The research efforts in the field of quality control of AC can be mainly classified in terms of targeting software or hardware designs. For instance, Green [9] and SAGE [10] are two frameworks for checking the quality of software approximation through *sampling*-based techniques. For every $N$ invocations, the exact result is computed and compared with the approximate result. Whenever the error distance (ED), i.e., the difference between the exact and approximate result, is greater than a prespecified threshold, calibration is done. However, the quality of unchecked samples cannot be ensured, and the previous quality violations cannot be compensated.

Similarly, a *rollback* recovery scheme in the case of quality violation is proposed in [11] and [12]. Accordingly, *classifiers* were developed to predict whether the input data, when processed by approximate accelerators, produces an output with the allowed relative error. This kind of approach cannot be applied exhaustively for every possible output since the associated area and power dissipation overhead violates the foremost motivation of using AC. Thus, *lightweight* quality checkers with high-efficiency are used to decide about the usage of rollback approximation. However, the quality checker proposed in [11] is application-specific, and Khudia *et al.* [12] show a low prediction accuracy for large applications. Khudia *et al.* [12] showed that less than 20% of the inputs exhibit significant approximate errors, which can lead to a notable quality degradation in most applications. With the motivation of achieving a high prediction accuracy, *multiple lightweight predictors* have been proposed in [13] by avoiding many unnecessary expensive rollback recoveries. Chengwen *et al.* [14] proposed an optimization framework to coordinate the training of the classifier (error predictor) and the accelerator (used for approximate computation) with an intelligent selection of training data, based on the iterative training process. The above approaches reported in [9]–[14] mainly target controlling software approximation only through program reexecution and, thus, are not applicable to hardware designs. Moreover, they ignore input dependencies and do not consider choosing an adequate design from a set of available designs.

Recently, Xu and Schafer [15] proposed a runtime a reconfigurable manager to select the most suitable approximate design based on the detected input data distribution. However, they utilize approximate accelerators designed at the behavioral level for various coarse-grained expected input data distributions. In addition, the proposed approximate circuits heavily depend on the training data used during the approximation process, where not all possible workload distributions can be precharacterized. Thus, the real workload may differ completely from the training one. Xu and Schafer [16] also presented a self-tunable runtime adaptive approximate architecture that is suitable for application-specific integrated circuit (ASIC) designs. However, the used approximation techniques are variable-to-variable (V2V) and variable-to-constant (V2C) optimization only. Overall, none of these state-of-the-art techniques, i.e., [9]–[16], exploits the potential of different settings of AC and their adaptations based on a user-specified quality constraint in order to ensure the accuracy of the individual outputs, which is the main idea proposed in this work. Our proposed work is complementary to [15] and [16] in the sense that our designs are approximated independently of the applied inputs (unlike [15]) and encompass various simplifications (unlike [16]).

## III. Proposed Methodology

The proposed methodology caters to the following challenges of AC [17]: 1) minimizing the approximate results with large error magnitudes; 2) selecting the approximate designs based on the input data as well as the user requirements; and 3) continuously monitoring the output quality and compensating the error where monitoring all the inputs is not a feasible solution. Therefore, we propose a design selector based on a computationally inexpensive ML-models [18]. The proposed methodology utilizes DTs and NNs to build an input-aware model, i.e., *AC quality manager*, to pick the most suitable approximate design based on the input data. The characteristics of the models, including their accuracy, execution time, power consumption, and computation cost, are extremely important for the successful implementation of the proposed methodology.

Without loss of generality, we consider an tunable approximate design with two settings, i.e., $S1$ and $S2$, that take values from the 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, we designed a set of energy-efficient approximate multipliers based on three design decisions [1]: 1) the type of the full adders (FAs) used to construct the multiplier where we used 11 different types; 2) the architecture, i.e., array or tree; and 3) the approximation degree (how much of the results to be approximated), where we have used two options: 1) all FAs are approximate (fully approximate design) and 2) FAs that contribute to the least significant 50% of the resultant bits are approximated. Based on the obtained results, we have selected the most energy-efficient designs to be used in this work that is: 1) utilizing five types of FAs that are called *approximate mirror adders*, i.e., Type $= s1 =$ {AMA1, AMA2, AMA3, AMA4, AMA5}, chosen from the low-power approximate FAs [19]; 2) its architecture is array; and 3) the approximation degree has four options, i.e., Degree $= s2 =$ {D1, D2, D3, D4}, where D1 has 7 bits approximated out of 16-bit result, while D2–D4 have 8, 9, and 16 approximate bits, respectively. Table I shows our library of approximate designs based on *Degree* and *Type* knobs, i.e., *ApprxMul* $=$ {Design1, ..., Design20}. Also, the library includes the exact design to be used whenever the required TOQ cannot be satisfied. The proposed design flow is adaptable, i.e., applicable to approximate functional units other

TABLE I
LIBRARY OF 20 STATIC APPROXIMATE DESIGNS BASED ON *Degree* AND *Type* KNOBS

| **Approximate Design** | | **Degree** | | | |
|---|---|---|---|---|---|
| | | **D1** | **D2** | **D3** | **D4** |
| **Type** | **AMA1** | *Design1* | *Design2* | *Design3* | *Design4* |
| | **AMA2** | *Design5* | *Design6* | *Design7* | *Design8* |
| | **AMA3** | *Design9* | *Design10* | *Design11* | *Design12* |
| | **AMA4** | *Design13* | *Design14* | *Design15* | *Design16* |
| | **AMA5** | *Design17* | *Design18* | *Design19* | *Design20* |

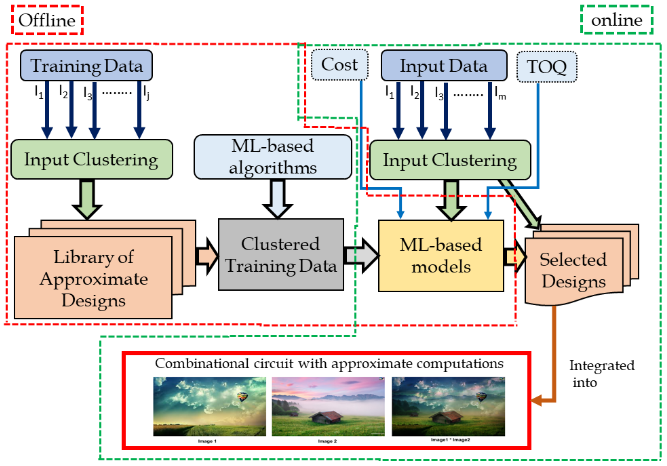

Fig. 1. ML-based methodology for tunable design to control the quality of AC.

than multipliers, e.g., adders, subtracters, dividers, square-root circuit, approximate multiply-accumulate (MAC) units, and approximate metafunctions.

As depicted in Fig. 1, our proposed methodology encompasses two phases: 1) an off-line phase (executed once) where the training inputs are applied to the library of approximate designs to generate the training data for building the ML-based *design selector* and 2) an online phase where the design selector continuously accept inputs and select the most suitable design to match the required TOQ for the given inputs. Overall, the proposed methodology consists of the following main steps.

1) *Library of Approximate Designs:* The first step is to generate a set of approximate designs, i.e., approximate library. An approximate design with two knobs, i.e., *S1* and *S2*, will have $|S1| \times |S2|$ different settings that constitutes our library. For the example of 8-bit approximate array multiplier, as shown in Table I, we have $|S1| \times |S2| = 5 \times 4 = 20$ different settings with reduced power consumption and execution time compared to the exact design. Section IV evaluates various design characteristics of the approximate library.
2) *Training Inputs:* For an approximate design with $m$ inputs, i.e., $i_1, i_2, \ldots, i_m$, each of $n$-bit width, and value ranging from 0 to $2^n-1$, we apply $j$ inputs of $k$-bit width each to the approximate design to generate the training data, where $j \leq m$ and $k \leq n$. For example, for an 8-bit approximate array multiplier with $m = 2$ and $n = 8$, and in order to enhance model's accuracy, we choose using the whole range of inputs for training, i.e., $j = m = 2$ and $k = n = 8$, where we have 65 536 different input combinations. For 16- and 32-bit designs, the size of the input combinations is $2^{32}$ and $2^{64}$, respectively. Thus, a sampling of the training data could be used because it is impossible to generate an exhaustive training data set.
3) *Input Clustering/Quantization:* Evaluating the design accuracy for a single input can provide the ED metric only, as explained in [20]. Therefore, input clustering is indispensable while generating the training data to evaluate the design accuracy over a range of consecutive inputs that belong to the same cluster. Thus, mean error metrics, e.g., mean square error (MSE) and normalized mean error distance (NMED), are evaluated for each cluster. Considering the example of an 8-bit approximate multiplier, where each design has a $(2^k)^j = (2^8)^2 =$ 65 536 possible input combinations, we propose to cluster every 16 consecutive input values, where each input encompasses $C = (2^k/16) = 16$ clusters rather than 256 inputs. Thus, the total number of possible input combinations per design is reduced to 256. Similarly, for the 16-bit design, the number of clustered inputs is reduced to $2^{24}$ rather than $2^{32}$.
4) *Training Data:* For an approximate design with $|S1| \times |S2|$ different settings adapted from [1] and $(2^k/C)^j$ different combinations of clustered input, we generate a training data of $|S1| \times |S2| \times (2^k/C)^j$ instances. Each instance includes the clusters of the applied inputs, design settings, and various accuracy metrics obtained from the approximate results, such as error rate, ED, relative ED, MSE, peak-signal-to-noise ratio (PSNR), and normalized error distance (NED), which are described in [20]. Moreover, each design has its area, power, delay, and PDP reduction compared to the exact design. For the example of an 8-bit approximate multiplier, we have a total number of $5 \times 4 \times 16^2 = 5120$ training instances. Similarly, for the library of 16-bit designs, we have $20 \times 2^{24}$ training instances.
5) *Design Selector:* It enables design adaptation for changing inputs to match the required TOQ where the error distribution is input-dependent [21]. We realized this in a two-steps design. The first step determines the first setting of the design, i.e., approximation *Degree*. Then, the second step determines the other setting, i.e., design *Type*. To train the model, the data were divided into two groups: the training-testing data set and the validation data set; 70% of the original data set were selected randomly for model training using repeated fivefold cross-validation. The most accurate model was selected for the testing stage based on 15% for the data set. Finally, we validated the model using the remaining 15% for the data set. Thus, we ensure that no training data leak into the test set. The obtained accuracy of the DT-based model depends on the settings of various parameters. Even when the model's prediction is inaccurate, the area, power, and delay of the selected approximate design are still lower than the exact design because all the approximate designs have less power, area, delay, frequency, and energy compared to the exact design. When the TOQ required by the user is higher than the best value attainable by approximate designs, the model selects the exact design. The characteristics of these DT and NN-based models are summarized in Table IV. Finally, the selected design is adapted within an error-resilient multiplication-intensive application. For example, 8-bit approximate multipliers are used

**Algorithm 1** Proposed Tunable Approximate Design

```
Input:
 1: (1) Input1: first input data; (2) Input2: second input data;
 2: (3) TOQ: specified quality;
Output: (1) Result:
 3: Offline Phase::
 4: (1) Build a Library of Approximate Designs
 5: (2) Apply Clustered Training Inputs
 6: (3) Obtain Training Data
 7: (4) Build ML-based Error and Cost Models
 8: (5) Build ML-based Design Selector, i.e., AC quality manager
 9: Online Phase::
10: N ← Configure_Window_Size (TOQ) ;
11: L ← Length_of_Input (Input1);
12: $T \leftarrow L/N$;                          ▷ number of reconfiguration times
13: $i \leftarrow 1$;
14: while $i \leq T$ do
                                                  ▷ determine the $i^{th}$ cluster
15:     $C1_i$= Detect_Cluster (Input1[N*(i-1)+1:i*N])
16:     $C2_i$= Detect_Cluster (Input2[N*(i-1)+1:i*N])
                                    ▷ check if any of the $i^{th}$ inputs changed
17:     if $(C1_i \neq C1_{i-1})$ $Or$ $(C2_i \neq C2_{i-1})$ then
18:         $(Degree_i, Type_i)$ = Selector $(C1_i, C2_i, TOQ_i)$
19:     else                                      ▷ check if $TOQ_i$ changed
20:         if $TOQ_i \neq TOQ_{i-1}$ then
21:             $(Degree_i, Type_i)$ = Selector $(C1_i, C2_i, TOQ_i)$
                                                  ▷ Use previous settings
22:         else
23:             $Degree_i \leftarrow Degree_{i-1}$
24:             $Type_i \leftarrow Type_{i-1}$
25:         end if
26:     end if
27:     UpdateBuffer($C1_i$, $C2_i$, $Degree_i$, $Type_i$, $TOQ_i$)
28: end while
29: function Selector($C1_i$, $C2_i$, $TOQ_i$)
30:     Find $Degree_i$, $Type_i$
31:     Return $Degree_i$, $Type_i$
32: end function
33: function AxD($C1_i$, $C2_i$)
34:     Perform approximate Computation for $C1_i$ and $C2_i$
35:     Return $Result$
36: end function
37: function UpdateBuffer($C1_i$, $C2_i$, $Degree_i$, $Type_i$, $TOQ_i$)
38:     Save the inputs, used design and quality result
39: end function
```

in multiplication intensive applications, such as image processing, i.e., blending and filtering.

The proposed methodology is easily applicable to other approximate designs, such as approximate adders, subtracters, dividers, and MAC units. The first step of the proposed methodology is generating the AC blocks. Based on this, the remaining four steps of the methodology (generating training inputs, performing input quantization, generating training data, and building ML-based design selector) are general, where they are applicable to any AC block. The flow of the proposed methodology is shown in Algorithm 1. The main off-line steps are done once. During the online phase, the user specifies the TOQ, where we build our models based on NED and PSNR error metrics. An important design decision is to determine the *configuration granularity*, i.e., how much data to process before readapting the design, which is termed the *window size* ($N$). For image processing applications, we select $N$ to be equal to the size of colored components of a frame, i.e., $250 \times 400 = 100\,000$ pixels. Then, based on the length of inputs, i.e., $L$ and $N$, we determine the number of times to reconfigure the design such that the final approximation benefits, i.e., reduced energy and/or execution time, are still significant. After $N$ inputs, a design adaptation is done if any of the inputs or TOQ changes. The first step in such adaptation is input *quantization*, i.e., specifying the corresponding cluster for each input based on its magnitude. In order to evaluate the inputs of an approximate design, various metrics, such as median, skewness, and kurtosis, have been used [15]. However, our approximate library is designed irrespective of the applied inputs. Thus, the input magnitude is the most suitable characteristic for design selection.

## IV. Library of Approximate Multipliers

Multipliers are an integral component for many digital signal processors, embedded systems, and microprocessors. They are extensively used in the operations of various applications related to speech processing, digital filtering, digital signal processing, convolution, correlation, communication, and video coding. Moreover, multipliers are one of the most energy-hungry components with complex logic design [22]. Usually, multipliers have a complex structure, as well as being laid in the critical path of digital circuits. Thus, approximate multipliers have a major impact on the performance and energy dissipation of the overall hardware design. Accordingly, we noticed from the literature that approximate multiplier designs with their quality control have gained a great research interest. In this article, as a case study of our methodology, we use a library of approximate multipliers, which we developed in a previous work [1]. The methodology proposed in this article, however, is equally applicable to other functional units, e.g., adders, subtracters, dividers, and MAC units.

In this section, we analyze various characteristics, i.e., accuracy, area, delay, energy, and power consumption, of our approximate library. We consider 8-bit multipliers, which are commonly used in low-power embedded systems and image/video processing applications [23]. For example, the CEVA-NP4000 DSP processor includes 4096 8-bit MAC units [24], which are used in various high-performance energy-constrained edge processing applications. Moreover, in [1], we developed 8- and 16-bit approximate multipliers, and the simulation results for 16-bit approximate *array* multipliers exhibit significant similarities with the 8-bit version. Similarly, the characterization of 16- and 8-bit approximate *tree* multipliers is also very similar.

### *A. Related Approximate Multipliers*

Many methods for designing approximate multipliers were proposed in the literature. For example, Sabetzadeh *et al.* [25] proposed ultraefficient approximate 4:2 compressor and multiplier designs based on majority logic, where the proposed compressor can tradeoff between ED and transistor count. Moreover, the proposed compressor is based on the majority logic, which leads to a more efficient imprecise multiplier. The proposed designs are evaluated using FinFET technology. Ansari *et al.* [26] used approximate multipliers to improve the accuracy of NN. For that, they used a set of carefully simplified approximate multipliers in addition to another set of approximate multipliers designed based on the Cartesian genetic programming (CGP). On the other hand, it is known that array multipliers have periodic structures and, thus, lead to compact hardware due to short wiring, which leads to efficient pipelining. Due to its benefits, Mahdiani *et al.* [27], Shao and Li [28], Shirane *et al.* [29], Yamamoto *et al.* [30], and Sato and Ukezono [31] designed a series of approximate

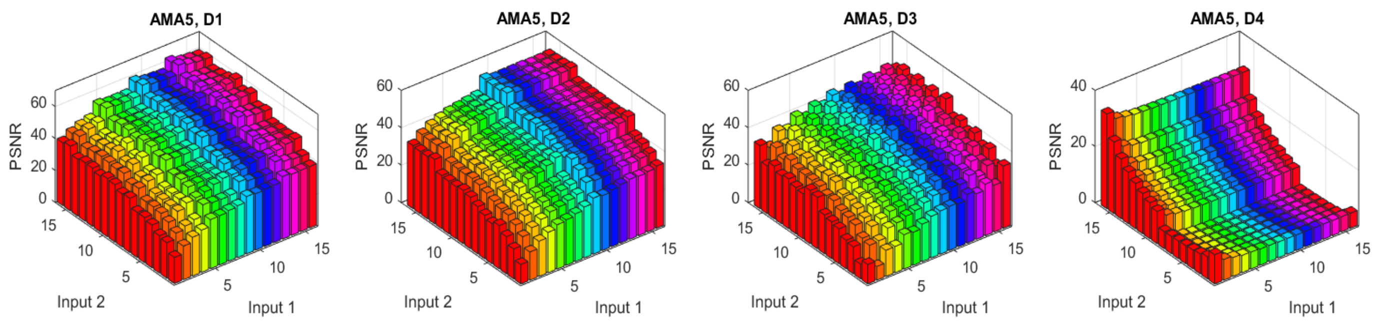

Fig. 2. PSNR for approximate multiplier based on *AMA5* type and D1–D4 approximation degrees.

TABLE II

ACCURACY METRICS FOR THE LIBRARY OF APPROXIMATE MULTIPLIERS

| Approx Degree | FA Type | ER | MED | NEDx$10^{-3}$ | MRED | MSE | PSNR | Approx Degree | FA Type | ER | MED | NEDx$10^{-3}$ | MRED | MSE | PSNR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **D1** | **AMA1** | 0.931 | 102 | 16.5 | 0.0380 | 1.69x$10^4$ | 39.35 | **D2** | **AMA1** | 0.962 | 246 | 36.6 | 0.0823 | 9.61x$10^4$ | 32.10 |
| | **AMA2** | 0.978 | 101 | 21.3 | 1.0447 | 1.44x$10^4$ | 39.97 | | **AMA2** | 0.990 | 227 | 51.5 | 2.1470 | 7.23x$10^4$ | 33.26 |
| | **AMA3** | 0.996 | 200 | 41.6 | 2.8055 | 4.76x$10^4$ | 34.78 | | **AMA3** | 0.999 | 474 | 107.7 | 7.0425 | 2.68x$10^5$ | 27.65 |
| | **AMA4** | 0.932 | 51 | 8.3 | 0.0189 | 4.11x$10^3$ | 45.58 | | **AMA4** | 0.963 | 107 | 16.4 | 0.0350 | 1.79x$10^4$ | 39.15 |
| | **AMA5** | 0.870 | 44 | 6.9 | 0.0148 | 3.14x$10^3$ | 46.80 | | **AMA5** | 0.922 | 93 | 13.8 | 0.0280 | 1.38x$10^4$ | 40.32 |
| **D3** | **AMA1** | 0.977 | 538 | 75.8 | 0.1679 | 4.53x$10^5$ | 25.80 | **D4** | **AMA1** | 0.9882 | 13162 | 1186.2 | 2.1 | 3.27x$10^8$ | 5.54 |
| | **AMA2** | 0.996 | 464 | 97.6 | 4.2929 | 2.97x$10^5$ | 27.41 | | **AMA2** | 0.9998 | 16902 | 5003.3 | 272.6 | 4.00x$10^8$ | 8.18 |
| | **AMA3** | 1.000 | 1010 | 228.0 | 13.4428 | 1.17x$10^6$ | 21.75 | | **AMA3** | 0.9999 | 17211 | 3930.3 | 180.7 | 4.13x$10^8$ | 7.36 |
| | **AMA4** | 0.977 | 185 | 28.6 | 0.0582 | 5.27x$10^4$ | 34.30 | | **AMA4** | 0.9882 | 6334 | 470.5 | 0.6039 | 7.91x$10^7$ | 10.38 |
| | **AMA5** | 0.952 | 185 | 26.1 | 0.0488 | 5.34x$10^4$ | 34.56 | | **AMA5** | 0.9825 | 8096 | 530.9 | 0.6642 | 1.17x$10^8$ | 8.85 |

array multipliers with different accuracy, area, power, and delay properties.

Ansari *et al.* [26] indicated that there is no single design of approximate multiplier, which is best for all applications. Thus, selecting the most suitable design for a specific application is a challenge. Similarly, in this work, we declared that the most suitable approximate design depends on the application, applied inputs, and user preferences. Therefore, we propose using a DT-based model to help in selecting the most suitable approximate design for image processing applications. Our proposed methodology is perpendicular to designing approximate multipliers, e.g., [25]–[27], where we can integrate any set of approximate multipliers to the library of approximate designs. Then, we built an ML-based model that selects the most suitable design to perform the specified operation.

## B. Accuracy of Approximate Library

In [20], we performed an exhaustive error analysis for the library of approximate designs. The analysis revealed a strong correlation between the applied inputs and the various accuracy metrics, i.e., ED, NED, and PSNR. Table II summarizes various error metrics, i.e., error rate (ER), mean error distance (MED), NED, mean relative error distance (MRED), and PSNR, which we obtained based on an exhaustive simulation for the approximate library. The shown results are averaged over the full range of the inputs and provide useful insights about design accuracy. Based on the analysis of ED in [20], we identified a strong correlation between the ED and both design *type* and *degree*. However, considering the ED metric as our TOQ mandates changing the design for every applied input, which is impractical due to the associated overhead

$$\mathrm{PSNR} = 10 \times \log_{10}\left(\frac{255^2}{\mathrm{MSE}}\right). \qquad (1)$$

For image processing, PSNR that depends on the MSE as given by (1) is an image quality indication. Thus, we use it as a TOQ metric in the proposed design selector. A low value of PSNR indicates a low-quality image associated with a large MSE. Similar to [32], as an example, we consider PSNR $\geq$ 25 dB as our threshold for an *acceptable* quality. Fig. 2 shows—as an example—the PSNR for designs based on *AMA5* type. The *D1*-based design has an average of 46.8 dB with a minimum of 16.3 dB, where it has five input combinations with TOQ $<$ 25 dB. Similarly, the design based on *D2* has an average of 40.3 dB with a minimum value of 10.4 dB, and 14 input combinations have TOQ $<$ 25 dB. The design with *D3* has an average of 34.6 dB for the PSNR, and 33 input combinations have TOQ $<$ 25 dB. Regarding the *D4*-based design, 239 clusters have a reduced quality with an average of 8.8 dB. The analysis of the PSNR of 20 different approximate designs [20] shows that every design has some input combinations that violate the specified TOQ. Therefore, the tunable design should avoid such quality violating cases. Designs with low PSNR (high error) either: 1) have all bits of the results being approximated based on designs with D4 approximation *Degree* or 2) have a low magnitude applied inputs, where the low bits of the result are approximated, while there are no bits applied to the most significant part of the design causing the whole result being approximated.

## C. Power, Area, Delay, and Energy of Approximate Designs

For the analysis of design metrics, of the approximate library, we utilized the XC6VLX75T FPGA, which belongs to the Virtex-6 family, and the FF484 package. For functionality verification, we use VHDL simulation based on Mentor Graphics *Modelsim*. We use *Xilinx XPower Analyser* for the power calculation based on exhaustive design simulation. For logic synthesis, we use the *Xilinx ISE 14.7 tool suite*.

Table III shows the characteristics of the approximate designs. As depicted in Fig. 3, all designs have reduced power, area, and energy compared to the exact design. However, few designs have a negligible longer critical path. Power reduction varies from 27.2% to 93.4% with an average of

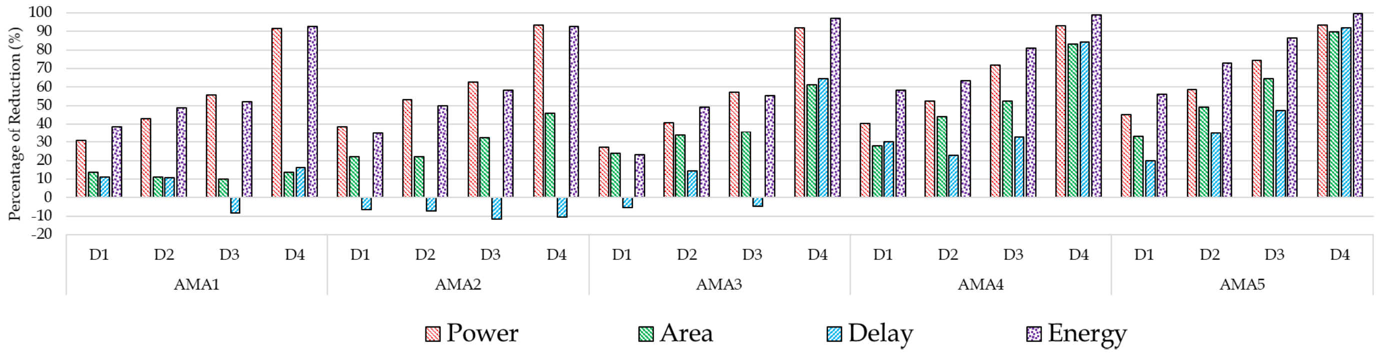

Fig. 3. Power, area, delay, and energy reduction for the 20 static approximate multipliers.

TABLE III
POWER, AREA, DELAY, FREQUENCY, AND ENERGY FOR THE LIBRARY OF APPROXIMATE MULTIPLIERS

| Design | | Dynamic | Slice | Occupied | Period | Frequency | Energy |
|---|---|---|---|---|---|---|---|
| Type | Degree | Power (mW) | LUTs | Slices | (ns) | (MHz) | (pj) |
| **AMA1** | **D1** | 306 | 79 | 23 | 7.763 | 128.82 | 2375.5 |
| | **D2** | 253 | 76 | 29 | 7.814 | 127.98 | 1976.9 |
| | **D3** | 196 | 77 | 29 | 9.487 | 105.41 | 1859.5 |
| | **D4** | 38 | 75 | 27 | 7.339 | 136.26 | 278.9 |
| **AMA2** | **D1** | 271 | 69 | 23 | 9.306 | 107.46 | 2521.9 |
| | **D2** | 207 | 63 | 29 | 9.373 | 106.69 | 1940.2 |
| | **D3** | 165 | 57 | 23 | 9.775 | 102.30 | 1612.9 |
| | **D4** | 29 | 46 | 18 | 9.672 | 103.39 | 280.5 |
| **AMA3** | **D1** | 322 | 67 | 23 | 9.223 | 108.42 | 2969.8 |
| | **D2** | 262 | 58 | 20 | 7.488 | 133.55 | 1961.9 |
| | **D3** | 189 | 55 | 21 | 9.139 | 109.42 | 1727.3 |
| | **D4** | 36 | 32 | 14 | 3.093 | 323.31 | 111.3 |
| **AMA4** | **D1** | 263 | 56 | 29 | 6.121 | 163.37 | 1609.8 |
| | **D2** | 210 | 47 | 19 | 6.743 | 148.30 | 1416.0 |
| | **D3** | 124 | 40 | 16 | 5.889 | 169.81 | 730.2 |
| | **D4** | 31 | 13 | 7 | 1.383 | 723.07 | 42.9 |
| **AMA5** | **D1** | 242 | 53 | 26 | 7.017 | 142.51 | 1698.1 |
| | **D2** | 183 | 41 | 19 | 5.694 | 175.62 | 1042.0 |
| | **D3** | 113 | 31 | 11 | 4.625 | 216.22 | 522.6 |
| | **D4** | 30 | 6 | 6 | 0.706 | 1416.43 | 21.2 |
| **Exact** | | **442** | **85** | **33** | **8.747** | **114.32** | **3866.2** |

60.8%. Similarly, energy reduction ranges between 23.2% and 99.4% with an average of 65.5%. The designs based on D1–D4 have an average of 42.2%, 56.9%, 66.6%, and 96.2% of energy reduction, respectively. These designs exhibit a 21.3% execution time reduction on average. Each design of the 8-bit approximate array multipliers implemented on FPGA consists of 64 full adders (FAs). The designs D1–D4 have 25, 33, 40, and 64 approximate FAs, respectively. Based on the experimentally obtained values, the power consumption is reduced approximately by a factor of 1.3 for every bit of the results being approximated, e.g., going from D1 to D2. The dynamic power difference between D1 and D4 is about 10×, which equals $\approx 1.3^8$. This clearly shows the benefits of design approximation regarding power consumption, which is obtained mainly due to the simplified design of approximate FA with reduced switching activity. The approximate FAs have a simplified structure compared with the exact FA. Thus, approximated designs have a noticeable reduction in the occupied LUTs

$$\text{Reduced Energy} = \frac{1}{X \times \left(\frac{E}{E_{\text{TOQ}}}\right) + (1 - X)}. \tag{2}$$

The final energy reduction enabled by approximation is weighted by the portion of circuit design that is *truly* beneficial for approximation. This is similar to the improvement of computer performance, which is obtained based on the enhanced

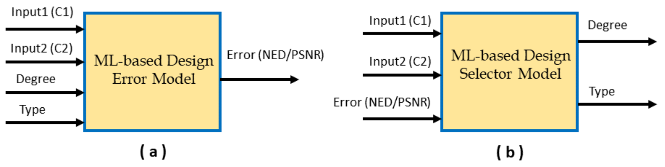

Fig. 4. Models for AC quality manager. (a) Forward design. (b) Inverse design.

performance of specific components, as quantified by Amdal's law [33]. The final reduced energy can be expressed in the form of Amdal's law, as given in (2), where $E$ is the energy for the exact component, and $E_{\text{TOQ}}$ is the energy of the approximate component, which satisfies the TOQ. The ratio $(E/E_{\text{TOQ}})$ is the energy saving in the approximate component of the design, and $X$ is the fraction of the design, which is amenable to approximation. Thus, for best energy efficiency, we aim to achieve $(E/E_{\text{TOQ}}) \gg 1$ and $X \approx 1$. Therefore, we use image blending and filtering that are a pure multiplication operation, i.e., $X = 1$.

## V. ML-Based Models

ML-based algorithms find solutions by learning through training data [18]. Supervised learning allows for a fast, flexible, and scalable way to generate accurate models that are specific to the set of application inputs and TOQ. It allows automated data analysis with a set of methods, which detects patterns in the given data set and predict future data. In supervised learning, a map between a set of input attributes and an output variable is used to predict the unseen data.

The error for an approximate design with specific settings can be predicted based on the applied inputs. For this purpose, we developed a *forward design*-based model, as shown in Fig. 4(a). The obtained accuracy for this model is 97.6% and 94.5%, for PSNR and NED error metrics, respectively. Such high accuracy is attributed to the straightforward nature of the problem. However, we mainly target the *inverse design* of finding the most suitable design settings for given inputs and error threshold, as shown in Fig. 4(b).

Utilizing the *Rattle* package [8], we designed and evaluated various ML-based models, based on the analyzed data and several algorithms, developed in the statistical computing language R. These models represent the *design selector* for the tunable design. Linear regression models (LMs) are the simplest to develop; however, their accuracy is the lowest, i.e., around 7%. Thus, they are not suitable to work with our methodology. DT models based on both *C5.0* and *rpart* algorithms achieve an accuracy of up to 64%. Random forest (RF) models, with an overhead of 25 DTs, achieve an accuracy of up to 68%. The most accurate models have been found to be based on NNs, but they suffer from long development time, design complexity, and high energy overhead [13]. Next, we implement and evaluate two versions of the *design selector*, based on DT and NN models. Accordingly, we identify and select the most suitable one to be implemented in our methodology.

### A. DT-Based Design Selector

The DT algorithm uses a flowchart-like structure to partition a set of data into various predefined classes, thereby providing the description, categorization, and generalization of the given data sets [34]. Unlike the LM, it models nonlinear relationships quite well. Thus, it is applied in a wide range of applications, such as credit risk of loans and medical diagnosis [35]. DTs are usually employed for *classification* over data sets, through recursively partitioning the data, such that observations with the same label are grouped together [35].

Initially, we implemented and evaluated three different DT-based models: 1) using the *C5.0* function; 2) using the *rpart* function; and 3) based on the *cubist* function [36]. The accuracy of the *rpart*-based models is found to be lower than that of *C5.0*-based models. *Cubist* is a rule-based model, which is an extension of Quinlan's M5 model tree [37], where a tree is developed such that the terminal leaves represent LMs. Therefore, we have to discretize the results, which degrades its accuracy. Moreover, the leaves with linear models need more processing than the scalar leaves in the *C5.0*. Thus, we implemented a two-step design selector utilizing the C5.0 function by predicting the design *Degree* and then its *Type*. This kind of *design selector* is supposed to be lightweight for constantly monitoring the workloads and continuously adapting the design of every ($N$) input.

The gradient boosting machine (GBM) [38] is a well-known ML algorithm. Similar to RF, GBM is a set of DTs. However, RFs build an ensemble of deep independent trees, while GBMs build an ensemble of shallow and weak successive trees where each tree is capable of learning and improving on the previous ones [39]. GBMs are computationally expensive, where they often require many trees, e.g., more than 1000, which can be time and memory exhaustive [40]. As shown in Table I, a model to predict the approximation degree can achieve an accuracy of 91.2% with the overhead of 30 000 trees. The other four models to predict the design type have better accuracies of 39.4%, 61.85%, 60.3%, and 46.2%, with an overhead of 100, 500, 500, and 50 trees, respectively. However, these GBM-based accuracies are less than the DT-based ones. The GBM-based model to predict the design degree has higher accuracy than the DT-based model with the overhead of 5000 trees. However, the four GBM-based models to predict the design type have an accuracy lower than the DT-based models with an overhead of hundreds of trees. Therefore, in this work, we discarded GBMs due to the associated overhead and reduced accuracy in all four models compared to DTs.

A DT model, which is built based on full training data, could be replaced by a simple lookup table or exhaustive search, especially for a small search space [8]. In general, for embedded and limited resources systems, a lookup table is not a viable solution if the number of entries becomes very large which causes a significant area overhead [41]. For instance, for a circuit with two 16-bit inputs, we need to generate $2^{32}$ input patterns to cover all possible scenarios of a circuit, which is unsuitable for our target embedded system applications [42].

Based on the error analysis of the approximate designs [20], we noticed that the error magnitude is correlated with the approximation *Degree* in a more significant manner than the design *Type*. Such correlation is manifested in the accuracy of the models, where these models have an average accuracy of 77.8% and 74.3% for predicting the design *Degree* and *Type*, respectively, as shown in Table IV. The time for executing the implementation in MATLAB of these models is very small; for example, around 8.87 ms are required to predict the design *Degree*, and a maximum of 25.03 ms are required to predict the design *Type*. This time is negligible compared to the time of running an application, such as image blending or filtering.

This work aims to show the effectiveness of the proposed methodology based on a software-based implementation (in MATLAB) of the tunable approximate design. Moreover, in order to prove its applicability to hardware designs, we evaluate the power, area, delay, and energy of the *hardware* implementation of the *design selector* in a similar manner as evaluating the approximate library in Section IV-C. Table IV shows the obtained results. The power consumption of the model is less than 44 mW. This value is insignificant compared to the power consumption of approximate multipliers, as shown in Table III, where these multipliers are used for $N$ inputs. Similarly, the introduced area, delay, and energy overhead are amortized by running the approximate design for $N$ inputs. The area of the model, represented in terms of the number of slice LUTs, is 1099, at maximum. Also, the number of occupied slices could reach 452 slices. The worst case (slowest) frequency that the model could run is 43.65 MHz with a period of 22.91 ns. The designed model could consume maximum energy of 733.7pj. The set of approximate multipliers exhibit an acceptable saving in their characteristics, i.e., area, power, delay, and energy, compared to the exact design, as shown in Table III. It is important to note that the *design selector*, which is synthesized only once, is specific for the considered set of approximate designs.

### B. NN-Based Design Selector

NNs have generally been implemented in embedded software. However, recently, with the exploding number of embedded devices, the hardware implementation of the NNs is gaining significant attention. FPGA-based implementation of the NN is complex due to the large number of neurons and the calculation of complex equations, such as activation function [43]. We use the sigmoid function $f(x)$, which is given by

$$f(x) = \frac{1}{1+e^{-x}}, \tag{3}$$

as an activation function. Targeting an *inexpensive* hardware, a piecewise second-order approximation scheme for the implementation of the sigmoid function is proposed in [44], as given by

$$f(x) = \begin{cases} 1, & x > 4.0 \\ 1 - \frac{1}{2}\left(1 - \frac{|x|}{4}\right)^2, & 0 < x \leq 4.0 \\ \frac{1}{2}\left(1 - \frac{|x|}{4}\right)^2, & -4.0 < x \leq 0 \\ 0, & x \leq -4.0. \end{cases} \tag{4}$$

It has one multiplication, no lookup table, and no addition.

TABLE IV

CHARACTERISTICS, I.E., ACCURACY, EXECUTION TIME, POWER, AREA, DELAY, FREQUENCY, AND ENERGY, OF DESIGN SELECTORS BASED ON DT AND NN MODELS

| Model | | Accuracy | | Execution Time (ms) | | Dynamic Power (mW) | | Slice LUTs | | Occupied Slices | | Period (ns) | | Frequency (MHz) | | Energy (pj) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inputs | Output | DT | NN | DT | NN | DT | NN | DT | NN | DT | NN | DT | NN | DT | NN | DT | NN |
| C1, C2, PSNR | Degree | 77.8% | 82.17% | 8.87 | 18.9 | 16 | 155 | 602 | 7835 | 231 | 2683 | 22.910 | 31.504 | 43.65 | 31.74 | 366.6 | 4883.1 |
| C1, C2, PSNR, s2=D1 | Type | 75.5% | 66.52% | 25.03 | 18.0 | 19 | 164 | 497 | 8427 | 189 | 2791 | 18.596 | 31.746 | 53.78 | 31.50 | 353.3 | 5206.3 |
| C1, C2, PSNR, s2=D2 | Type | 76.1% | 70.21% | 19.3 | 9.0 | 23 | 153 | 449 | 6625 | 221 | 2309 | 15.962 | 31.718 | 62.65 | 31.53 | 367.1 | 4852.8 |
| C1, C2, PSNR, s2=D3 | Type | 71.3% | 73.22% | 11.94 | 18.7 | 23 | 159 | 390 | 5360 | 149 | 1731 | 15.494 | 29.164 | 64.54 | 34.29 | 356.4 | 4637.0 |
| C1, C2, PSNR, s2=D4 | Type | 74.1% | 59.08% | 6.61 | 7.4 | 28 | 170 | 298 | 4549 | 134 | 1420 | 11.838 | 28.678 | 84.47 | 34.87 | 331.5 | 4875.3 |

We implemented a two-step design selector by predicting the design *Degree* first and then the *Type*. The model for *Degree* prediction has an accuracy of 82.17% (the highest among all built models), while the four models for *Type* prediction have an accuracy between 73.22% and 59.08% with an average of 67.3%, as shown in Table IV. The time for design prediction ranges between 37.6 and 26.3 ms with an average of 32.7 ms.

We implemented the NN-based model on FPGA, and its dynamic power consumption, slice LUTs, occupied slices, operating frequency, and consumed energy are shown in Table IV. These values are found to be insignificant compared to the characteristics of approximate multipliers, as shown in Table III, where these multipliers are used for $N$ inputs. However, compared to the DT-based model, the NN-based model has an execution time, which is 1.31× higher than the DT, while its average accuracy is almost 0.98× of the accuracy achieved by the DT-based model. Other design metrics, including power, slice LUTs, occupied slices, period, and energy, have a value of 8.6×, 13.93×, 11.74×, 1.81, and 13.6×, consecutively, compared with the DT-based model. Unexpectedly, the DT-based model is better than the NN-based model in all design characteristics, including accuracy and execution time. Section VI evaluates the software-based implementation of the proposed methodology, which utilizes the DT-based *design selector* that we described earlier. We discarded the NN-based design selector due to the absence of advantages over DT.

## VI. RESULTS

This section evaluates the effectiveness of the proposed tunable approximate design, including the *approximate library* and the DT-based *design selector*. Regarding the setup for the execution time, we run MATLAB on a machine with 8-GB DRAM and i5 CPU with a speed of 1.8 GHz. We evaluate the proposed methodology based on two applications of images processing: 1) image blending, where we use two sets of images, i.e., *Set-1* with five examples and *Set-2* with 50 examples and 2) image filtering, where we use two images, i.e., *Lina* and *Cameraman*. Section VI-C evaluates an audio mixing application based on 16-bit models. The *execution time* is considered as a quality metric, where its overhead is found to be relatively small compared to the original applications, as shown next.

### A. Image Blending

Image blending in multiplication mode allows blending multiple images together to look like a single image. This process is widely used in developing animation and effect movies where video blending requires the multiplication of several consecutive images. For example, blending two colored videos, each with $N_f$ frames of size $N_r$ rows by $N_c$ columns per image, involves a total of $3N_f \times N_r \times N_c$ pixels. Each image has three colored components/channels, i.e., red, green, and blue, where the values of their pixels are expected to differ. A static configuration uses a single design, to perform all multiplications, even when their pixels are different. Therefore, for enhanced output quality, we propose to adopt the approximate design per channel parameter, as shown in Fig. 5. This way, the *design selector* continuously monitors the inputs and efficiently locates the most suitable design for each colored component to meet the required TOQ.

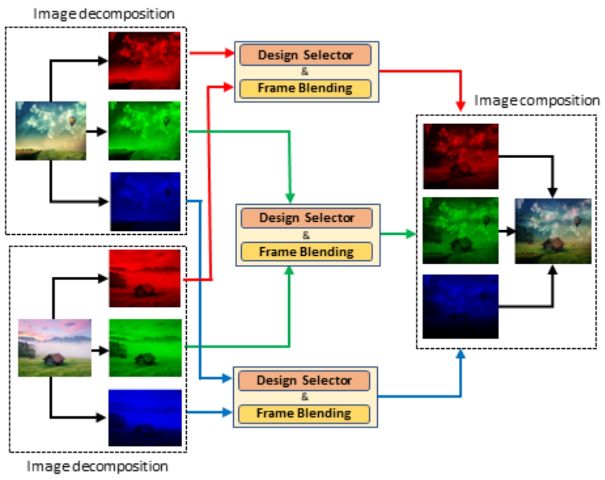

Fig. 5. Tunable image blending at the component level.

Various metrics, e.g., median, skewness, and kurtosis, have been used in the literature to characterize the inputs of approximate designs [15]. However, their proposed approximate circuits heavily depend on the training data used during the approximation process. Our approximate library is designed regardless of the applied inputs. Thus, the relationship between the consecutively applied inputs, such as skewness and kurtosis, is insignificant for our designs. Since the error magnitude depends on the applied inputs, we rely on pixel values to select the suitable design. However, setting the *configuration granularity* at the pixel level is impractical. On the other hand, the design selection per colored component is more suitable.

We evaluate the average of the pixels of each colored component to select the most suitable design. Two completely different images may have the same average of their pixels. Unfortunately, this could result in the same selected approximate design because the training is performed at the multiplier level and not the image level. To avoid this scenario, we reduce the *configuration granularity* by dividing the colored component into multiple segments, e.g., four segments. Thereafter, we propose to use multiple designs, rather than a single design, for each colored component. This triggers training and

TABLE V
CHARACTERISTICS OF *Set-1* BLENDED IMAGES

| Example | (Image1, Image2) | Frame Characteristic | Input 1 (Image1) | | | Input2 (Image2) | | |
|---|---|---|---|---|---|---|---|---|
| | | | Red | Green | Blue | Red | Green | Blue |
| 1 | (Frame, City) | Average | 131 | 163 | 175 | 172 | 153 | 130 |
| | | Cluster | 9 | 11 | 11 | 11 | 10 | 9 |
| 2 | (Sky, Landscape) | Average | 121 | 149 | 117 | 160 | 156 | 147 |
| | | Cluster | 8 | 10 | 8 | 11 | 10 | 10 |
| 3 | (Text, Whale) | Average | 241 | 241 | 241 | 48 | 156 | 212 |
| | | Cluster | 16 | 16 | 16 | 4 | 10 | 14 |
| 4 | (Girl, Beach) | Average | 177 | 158 | 140 | 168 | 176 | 172 |
| | | Cluster | 12 | 10 | 9 | 11 | 12 | 11 |
| 5 | (Girl, Tree) | Average | 102 | 73 | 40 | 239 | 193 | 118 |
| | | Cluster | 7 | 5 | 3 | 16 | 13 | 8 |

building the model at the image level to control the quality of approximation in image processing applications. Next, we analyze the results of applying the proposed methodology on two sets of images: 1) *Set*-1 with ten images used in our prior work [45] and 2) *Set-2* with 100 images based on a well-known database of images [46]. The images of each set are then blended at the component level, as shown in Fig. 5, to evaluate the efficiency of the proposed methodology.

*1) Blending of Set-1 of Images:* We use a set of ten different images, each of size $N_r \times N_c = 250 \times 400$ pixels. Table V shows the characteristics of the images, where each is segmented into three colored components. The average values of the pixels of each component and the associated input cluster are denoted as *Average* and *Cluster*, respectively.

We target 49 different values of TOQ, i.e., PSNR ranges from 17 to 65 dB, for each blending example. Thus, we run the methodology 245 times, i.e., $5 \times 49$. For every invocation, based on $C1$, $C2$, and the associated target PSNR, one of the 20 designs is selected and used for blending. For illustration purposes, we explain *Example5* in detail. As shown in Table V, the *Girl* image has a red-component with an average of 102, which belongs to *Cluster 7*, i.e., $C1_R = 7$. Similarly, the *Tree* image has a red-component with an average of 239, which belongs to *Cluster 16*, i.e., $C2_R = 16$. The green components belong to *Clusters* 5 and 13 ($C1_G = 5$ and $C2_G = 13$), while the blue components belong to *Clusters* 3 and 8 ($C1_B = 3$ and $C2_B = 8$). Then, we adapt the design by calling the design selector thrice, i.e., once for every colored component, assuming TOQ = 17 dB. The selected designs (based on line 29 of Algorithm 1) are given by

$$\text{Selector}(C1_{\text{R}}, C2_{\text{R}}, \text{TOQ}) \rightarrow \text{Design}_{\text{R}} \rightarrow \text{Design}_8 \quad (5)$$
$$\text{Selector}(C1_{\text{G}}, C2_{\text{G}}, \text{TOQ}) \rightarrow \text{Design}_{\text{G}} \rightarrow \text{Design}_{16} \quad (6)$$
$$\text{Selector}(C1_{\text{B}}, C2_{\text{B}}, \text{TOQ}) \rightarrow \text{Design}_{\text{B}} \rightarrow \text{Design}_{11}. \quad (7)$$

The selected $\text{Design}_{\text{R}}$, $\text{Design}_{\text{G}}$, and $\text{Design}_{\text{B}}$ are $\text{Design}_8$, $\text{Design}_{16}$, and $\text{Design}_{11}$, respectively. Based on that, the obtained quality is 16.9 dB, which is close to the TOQ.

*a) Accuracy analysis of tunable design:* Fig. 6 shows the minimum, maximum, and average curves of the obtained output quality, each evaluated over five examples of image blending. Out of the 245 selected designs, 49 predicted designs are violating the TOQ, i.e., the obtained output quality is below the red line. The unsatisfied output quality is attributed mainly to model imperfection. The best achievable prediction accuracy is based on the accuracy of the two models executed consecutively, i.e., *Degree* model with 77.8% and *Type* model with 76.1%. Thus, the best achievable accuracy is $77.8\% \times 76.1\% = 59.2\%$. However, a false predicted design can still achieve an acceptable output quality because there can be multiple designs that may achieve the required quality constraints. The accuracy of our model prediction is 80%, which is in agreement with the average accuracy of the DT-based models, as shown in Table IV.

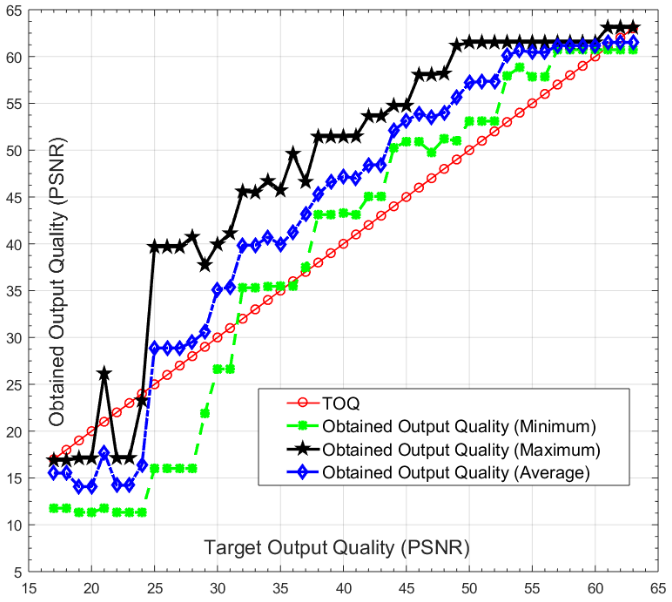

Fig. 6. Obtained output quality for image blending of *Set-1*.

Fig. 7 shows the fluctuation in the value of the PSNR, which is obtained by applying different input images for a specific static design. Based on the images of *Set-1*, *Design3*, *Design7*, *Design11*, *Design15*, and *Design19* have a fluctuation of 14.2%, 2.4%, 7.9%, 4.1%, and 3.1% of the obtained PSNR, respectively. Similarly, for the images of *Set-2*, the obtained PSNR fluctuates by 15.4%, 13.7%, 15.2%, 9.6%, and 9.8% for *Design3*, *Design7*, *Design11*, *Design15*, and *Design19*, respectively. Thus, for any design, the PSNR fluctuates for different images due to the dependence of the output quality on inputs, as observed in [47].

*b) Execution time analysis of tunable design:* Fig. 8 shows the average execution time of the five examples of image blending evaluated over 20 static designs. The shown time is normalized with respect to the execution time of the exact design. All designs have a time reduction ranging from 1.9% to 13.7% with an average of 3.96%, while the average execution time of the tunable design is 98.2% compared to the exact design. Thus, the tunable design is able to satisfy the user required TOQ with an execution time similar to the static design. For the five examples of image blending, we evaluated the execution time of the tunable design, where the target PSNR ranges between 17 and 65 dB for each example. Fig. 9 shows the execution time for the five examples using the exact design, the tunable design averaged over 49 different TOQ, and the static design averaged over 20 approximate designs. The time difference between the static time and the tunable time is due to design adaptation overhead. The time for design adaptation is 30.5, 93.9, 164.6, 148.6, and 42.1 ms for examples 1–5, which have a data processing time of 50.90, 50.91, 51.10, 51.69, and 51.04 s, respectively. For these five examples, the design adaptation time represents 0.06%, 0.18%, 0.32%, 0.28%, and 0.08% of the total execution time, respectively, which is negligible.

*c) Energy analysis of tunable design:* One of the foremost goals of designing a library of approximate multipliers is to enhance energy efficiency. To calculate the energy

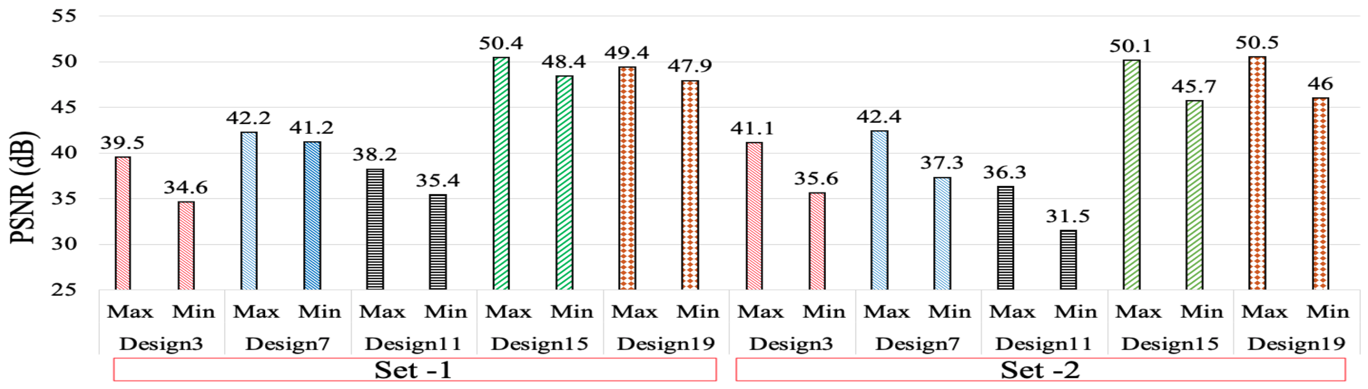

Fig. 7. Dependence of the output quality on the applied inputs.

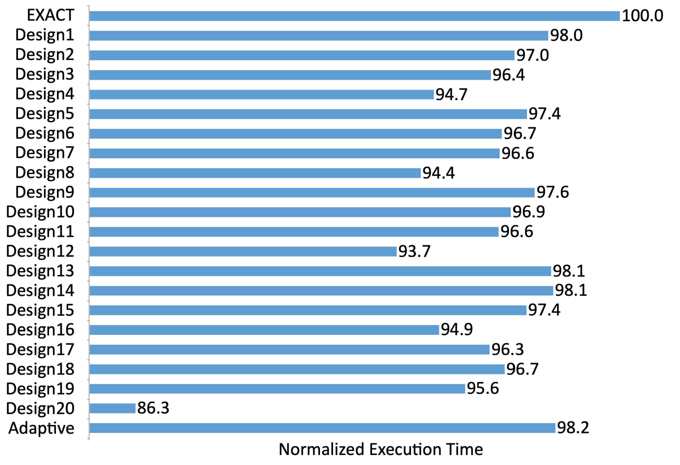

Fig. 8. Normalized execution time for blending of *Set-1* images using 20 static designs.

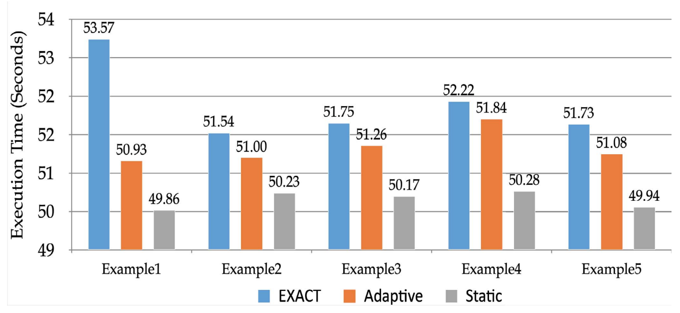

Fig. 9. Execution time of the exact, static, and tunable design.

consumed by the approximate multiplier to process an image, we use the following equation:

$$\text{Energy} = \text{Power} * \text{Delay} * N \tag{8}$$

where *Power* and *Delay* are obtained from the synthesis tool, as shown in Table III, and $N$ is the number of multiplications required to process an image, which equals 250*400 = 100 000 pixels. As shown in Table III, *Design9* has the highest energy consumption with 2970*pj* and a saving of 896*pj* compared to the exact design. Thus, the design adaptation overhead of 733.7*pj* (based on Table IV) is almost negligible compared to the total energy savings of 89.6 $\mu$j (896*pj* $\times$ 100 000) obtained by processing a single image. This validates our lightweight *design selector*.

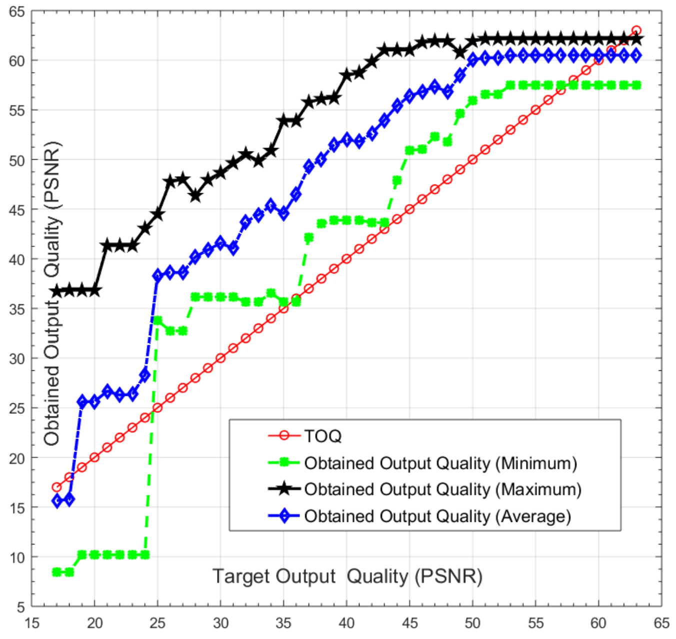

Fig. 10. Obtained output quality for image blending of *Set-2*.

*2) Blending of Set-2 of Images:* We used a set of 100 images from the database of *8 Scene Categories Data set* [46], which is downloadable from [48]. It contains eight outdoor scene categories: coast, mountain, forest, open country, street, inside city, tall buildings, and highways. Similar to *Set-1*, we target 49 different value of TOQ, for 50 examples of blending, and execute the proposed methodology 2450 times. Fig. 10 shows the minimum, maximum, and average curves of the obtained output quality, each evaluated over 50 examples of image blending. Out of the 2450 selected designs, 430 predicted designs are violating the TOQ, i.e., the obtained output quality is below the red line. Thus, the accuracy of our model prediction is 82.45%, which is slightly higher than the accuracy obtained for *Set-1* images. We consider PSNR $\geq$ 25 dB as a threshold for an *acceptable* quality of images, as proposed in [32]. As shown in Figs. 6 and 10, the proposed methodology has a high prediction accuracy for an *acceptable* PSNR, while its prediction accuracy is low when TOQ $<$ 25 dB.

### *B. Gaussian Smoothing/Filtering of Two Images*

Here, we evaluate the accuracy of the tunable design on a Gaussian smoothing low-pass filter, which reduces image

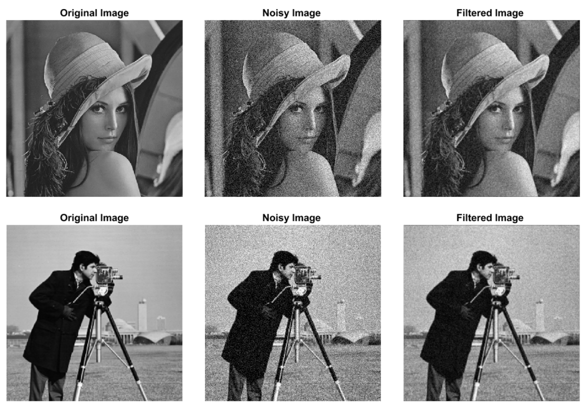

Fig. 11. Exact, noisy, and filtered images.

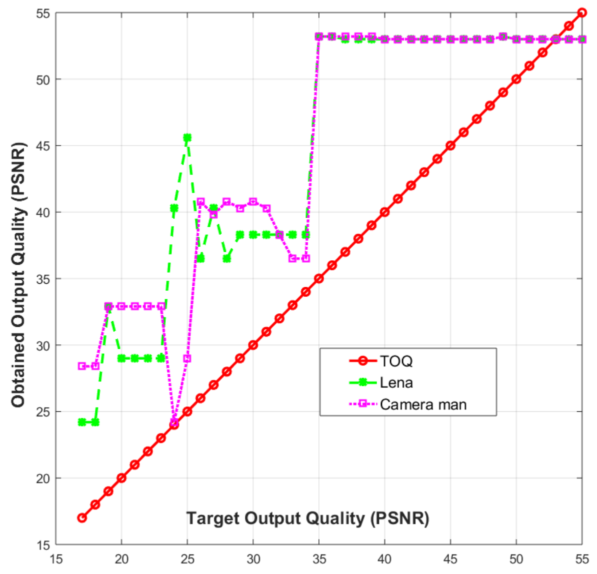

Fig. 12. Obtained output quality for tunable image filtering.

details through attenuating high-frequency signals. Applying a Gaussian smoothing is the same as convolving the image with a circularly symmetric 2-D Gaussian function, given in [49]

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \quad (9)$$

The Gaussian distribution is nonzero everywhere. Practically, it is effectively zero more than three standard deviations ($\sigma$) from the mean ($\mu$). The Gaussian smoothed output is a weighted average of each pixel's neighborhood. We use a ($3 \times 3$) kernel, based on $\sigma = 1.5$, where the kernel average weight depends significantly on the value of the central pixels.

We use the benchmarks *lena* and *Cameraman* as input images, with the addition of zero-mean Gaussian white noise, with a variance of 0.01 to the original gray-scale image. Fig. 11 shows the benchmark images with Gaussian noise added and the noisy images filtered with the exact design. The Gaussian kernel, as given in (9), is applied to the 8-bit gray-scale input images of size ($512 \times 512$) pixels.

Fig. 12 shows the obtained PSNR for two examples of image filtering based on tunable design where the TOQ ranges between 17 and 53 dB. The proposed methodology was able to satisfy the required TOQ. When it ranges between 17 and 34 dB, the obtained output quality for the two examples is different. However, when the TOQ is 35 dB or more, both examples have almost the same obtained PSNR. Fig. 13 shows the PSNR obtained by using the 20 static designs. The resulting values are computed with respect to the image obtained by applying the Gaussian filter with the exact multiplier design on the noisy image. The approximate designs were able to obtain a maximum PSNR of 53.2 dB.

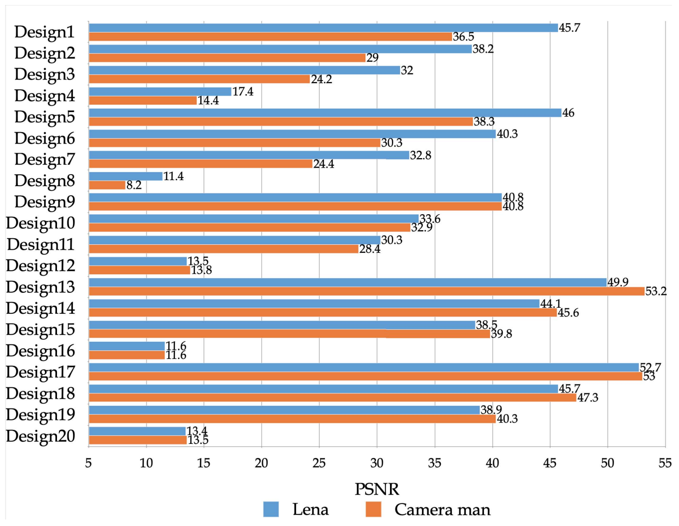

Fig. 13. Output quality (PNSR) for two examples of image filtering based on 20 static designs.

## C. Audio Mixing/Blending

Sounds are propagating waves represented in a 16-bit binary depth, which is able to cover a wide range of amplitudes with enhanced quality. We perform a set of audio blending applications, to evaluate the DT-based model for 16-bit designs. We evaluate the proposed methodology over 45 examples, where the target PSNR ranges between 15 and 70 dB. The used WAV sound files were obtained. Fig. 14 presents the achieved PSNR, i.e., minimum, maximum, and average, based on the DT model, which is designed to learn from the limited input data, to predict a result for the unseen data. The accuracy of the "average obtained TOQ" (blue curve) is 85.7%. Figs. 6 and 10 for the image blending application were showing a smoother curve of predicted results since the DT-model was built using a full set of training data. However, the stairs/steps in the predicted PSNR, as shown in Fig. 14, are due to the expectations of the model, which was built based on the sampled training data. The comparable accuracy of both models shows the effectiveness of ML even with the sampled training data.

## VII. Comparison With Related Work

We compare the output accuracy achieved by our tunable design with the accuracy of two static approximate designs based on approximate multipliers proposed by Kulkarni *et al.* [50] and Kyaw *et al.* [51], which have *similar structures* as our approximate array multipliers. Moreover, we compare the accuracy of our work with a third approximate design based on the approximate tree compressor multiplier (ATCM), as proposed by Yang *et al.* [52], which is a Wallace tree multiplier.

Kulkarni *et al.* [50] construct a large ($8 \times 8$) multiplier—which we call *$2 \times 2$-based* multiplier—using

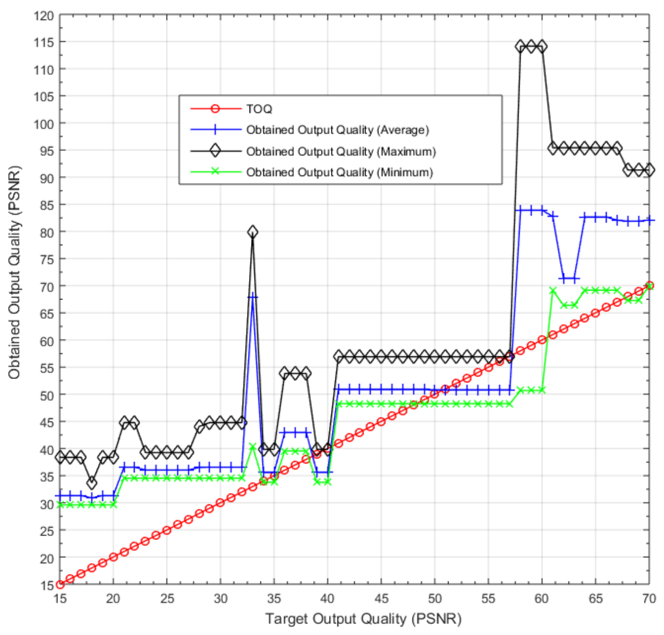

Fig. 14. Obtained output quality for audio blending.

TABLE VI

COMPARISON OF THE OBTAINED ACCURACY (PSNR) FOR VARIOUS APPROXIMATE DESIGNS

| Application | | KUL [50] | ETM [51] | ATCM [52] | Tunable Design (Proposed) |
|---|---|---|---|---|---|
| **Blending** | ***Set-1*, Ex. 1** | 24.8 | 27.9 | 41.5 | 61 |
| | ***Set-1*, Ex. 2** | 29.2 | 29.1 | 43.7 | 61.1 |
| | ***Set-1*, Ex. 3** | 20.3 | 24.8 | 33.1 | 63 |
| | ***Set-1*, Ex. 4** | 23 | 28 | 38.2 | 60.7 |
| | ***Set-1*, Ex. 5** | 27.6 | 29.4 | 40.3 | 61.5 |
| **Filtering** | **Lena** | 36.5 | 36.3 | 38.4 | 52.7 |
| | **Cameraman** | 35.9 | 36.9 | 29.2 | 53.2 |

smaller ($2 \times 2$) approximate multipliers as building blocks. The work in [51] presents an ($8 \times 8$) error-tolerant multiplier (ETM) based on the truncation principle by dividing the multiplier into two parts, i.e., accurate and approximate. For an 8-bit multiplier, the most significant 8 bits of the result are generated based on exact multiplication, while the least significant 8 bits of the result are generated based on probabilistic bit manipulation. ATCM utilizes a 4-to-2 compressor [53], which equally partitions the rows of the partial product tree array to reduce power and delay. In the rest of this article, we call the above three multiplier designs: *KUL*, *ETM*, and *ATCM*, respectively.

Table VI shows a summary of the obtained PSNR for image blending and filtering, based on KUL [50], ETM [51], ATCM [52], and the proposed adaptive design, which achieves better output quality than static designs due to the ability to select the most suitable design from the approximate library.

Table VII shows the power, area, delay, frequency, and energy of various multipliers, including the exact design, and three related designs, i.e., KUL, ETM, and ATCM, in addition to the approximate designs of our library with minimum and maximum values. We notice that the various circuit metrics, i.e., area, power, delay, and energy, of the related designs

TABLE VII

POWER, AREA, DELAY, FREQUENCY, AND ENERGY OF VARIOUS MULTIPLIERS

| Multiplier Design | Dynamic Power (mW) | Slice LUTs | Occupied Slices | Period (ns) | Frequency (MHz) | Energy (pj) |
|---|---|---|---|---|---|---|
| **Exact** | **442** | **85** | **33** | **8.747** | **114.32** | **3866.2** |
| **KUL [50]** | **232** | **72** | **34** | **5.218** | **191.64** | **1210.6** |
| **ETM [51]** | **143** | **23** | **12** | **3.668** | **272.63** | **524.5** |
| **ATCM [52]** | **136** | **63** | **24** | **6.410** | **156** | **871.8** |
| **Approx (Max)** | **322** | **79** | **29** | **9.775** | **1416** | **2970** |
| **Approx (Min)** | **29** | **6** | **6** | **0.706** | **102.3** | **21** |

are within the range of our designed approximate multipliers. The related designs, i.e., KUL, ETM, and ATCM, have a saving of $2656pj$, $3341pj$, and $2999pj$, respectively, compared to the exact design. Thus, for processing a single image, the design adaptation overhead of $733.7pj$ (based on Table IV) is negligible compared to the total energy savings of 265.6, 334.1, and 300 $\mu$j for KUL, ETM, and ATCM designs, respectively.

We compare the proposed methodology with the state of the art [9]–[16] in terms of several requirements related to hardware/software applicability, data dependence, quality assurance, and methodology overhead. Table VIII summarizes the comparison between the different approaches for AC quality control. Generally, all approaches are able to control the average quality of approximation results.

## *A. Hardware/Software*

Existing work [9]–[14] exhibit limited applicability for software approaches only, while the approaches in [15] and [16] are applicable to hardware with limited configurations. The methodology that we propose in this article is applicable to both hardware and software approximate applications with approximable components. This work explains a full software-based implementation (in MATLAB) of the proposed methodology. Moreover, we presented a hardware implementation of the approximate library and design selector.

## *B. Input Data Dependence*

A distinctive characteristic of our methodology is its data dependence. To the best of our knowledge, none of the previous works targeted fine-grained input dependence of approximate designs to control the output quality. The proposed approach clusters the input data and then uses it with a library of approximate designs to generate training data. Another peculiar feature of the proposed approach is the development of a lightweight DT-based model for design selection with satisfying accuracy.

## *C. Quality Assurance and Overhead*

The approaches [9]–[14] control the quality of results through rollback and program reexecution. However, they require extra time and out-of-order-execution. The approach in [15] depends on the training data used during the approximation process, which may differ significantly from the real workload, while the approach in [16] relies on V2V and V2C approximation techniques only. On the other hand, the proposed approach needs a one-time data generation, training, and model building. Then, the design is adapted, with negligible overhead.

TABLE VIII
COMPARISON BETWEEN AC QUALITY CONTROL APPROACHES

| **Characteristics** | **Software-based Design [9] – [14]** | **Hardware-based Design [15] – [16]** | **Our ML-based Adaptive Design** |
|---|---|---|---|
| **Hardware/Software** | Applicable to software approximation only, i.e., loop unrolling, at a very coarse granularity | Applicable to hardware approximation with limited configuration options | Applicable to both hardware and software approximate techniques with multiple approximate componentes |
| **Input Data Dependency** | Completely ignore differences between input data | [15] Workload distribution used during training may not be characterized<br>[16] Use only V2V and V2C input dependent optimization | Significantly relies on the input data for design adaptation |
| **Quality Assurance** | Through execution rollback and program re-execution | Quality can not be assured if the real workload differs from the one used for training | Utilizes ML-based models to pick the most suitable approximate component |
| **Overhead** | Rollback recovery time and cost | [15] Runtime overhead is negligible<br>[16] introduces 8.5% area and 8.1% delay overhead | Overhead of generating training data and build the models<br>Energy and delay overheads are negligible |

## VIII. CONCLUSION

AC reduces execution time or/and energy consumption of error-resilient applications by relaxing the quality constraints. However, when the inputs are dynamic, a static design may lead to extremely large output errors. Previous work has ignored considering the changing inputs to assure the quality of individual outputs. In this work, targeting *approximate programs*, we proposed and implemented a novel fine-grained input-based tunable design based on ML models. The proposed solution considers the input data in generating the training data, building ML-based models, and adapting the design to satisfy the TOQ. This approach is applicable to both hardware and software designs where we were able to satisfy the TOQ with negligible energy and delay overhead more than 80% of the time. These benefits come at the one-time cost of generating the training data, deploying and evaluating the model. With design adaptation, the most suitable design is always identified and selected for controlling the quality loss. Our ongoing work seeks to expand the approximate library to encompass approximation techniques across various levels of designs, including algorithmic, architectural, and functional units. For follow up work, we are targeting a fully *hardware* implementation of our proposed system.

## REFERENCES

[1] M. Masadeh, O. Hasan, and S. Tahar, "Comparative study of approximate multipliers," in *Proc. Great Lakes Symp. (VLSI)*, May 2018, pp. 415–418.

[2] M. Masadeh, O. Hasan, and S. Tahar, "Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency," *IEEE Access*, vol. 7, pp. 147129–147142, 2019.

[3] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surveys*, vol. 48, no. 4, pp. 1–33, May 2016.

[4] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, pp. 1–6.

[5] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, p. 86.

[6] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," in *Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2016, pp. 607–621.

[7] M. Masadeh, O. Hasan, and S. Tahar, "Approximation-conscious IC testing," in *Proc. 30th Int. Conf. Microelectron. (ICM)*, Dec. 2018, pp. 56–59.

[8] M. Masadeh, O. Hasan, and S. Tahar, "Using machine learning for quality configurable approximate computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1554–1557.

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

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

[11] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing," in *Proc. Int. Symp. HPC Archit.*, 2015, pp. 615–626.

[12] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 554–566.

[13] T. Wang, Q. Zhang, N. Kim, and Q. Xu, "On effective and efficient quality management for approximate computing," in *Proc. Int. Symp. Low Power Electron. Design*, 2016, pp. 156–161.

[14] X. Chengwen *et al.*, "On quality trade-off control for approximate computing using iterative training," in *Proc. Design Autom. Conf.*, 2017, pp. 1–6.

[15] S. Xu and B. C. Schafer, "Approximate reconfigurable hardware accelerator: Adapting the micro-architecture to dynamic workloads," in *Proc. Int. Conf. Comput. Design*, 2017, pp. 113–120.

[16] 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, Dec. 2018.

[17] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Quality control for approximate accelerators by error prediction," *IEEE Des. Test. Comput.*, vol. 33, no. 1, pp. 43–50, Feb. 2016.

[18] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[19] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[20] M. Masadeh, O. Hasan, and S. Tahar, "Error analysis of approximate array multipliers," *CoRR*, vol. 1908.01343, pp. 1–14, Aug. 2019. [Online]. Available: http://arxiv.org/abs/1908.01343

[21] W.-T.-J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 47–53.

[22] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY, USA: Oxford Univ. Press, 1999.

[23] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261.

[24] (2019). *CEVA NeuPro a Family of AI Processors for Deep Learning at the Edge*. Accessed: Nov. 19, 2019. [Online]. Available: https://www.ceva-dsp.com/product/ceva-neupro/

[25] 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.

[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] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.
[28] B. Shao and P. Li, "Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1081–1090, Apr. 2015.
[29] K. Shirane, T. Yamamoto, I. Taniguchi, Y. Hara-Azumi, S. Yamashita, and H. Tomiyama, "Maximum error-aware design of approximate array multipliers," in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2019, pp. 73–74.
[30] T. Yamamoto, I. Taniguchi, H. Tomiyama, S. Yamashita, and Y. Hara-Azumi, "A systematic methodology for design and analysis of approximate array multipliers," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Oct. 2016, pp. 352–354.
[31] T. Sato and T. Ukezono, "A dynamically configurable approximate array multiplier with exact mode," in *Proc. 5th Int. Conf. Comput. Commun. Syst. (ICCCS)*, May 2020, pp. 917–921.
[32] Y. Xu, J. D. Deng, and M. Nowostawski, "Quality of service for video streaming over multi-hop wireless networks: Admission control approach based on analytical capacity estimation," in *Proc. Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process.*, 2013, pp. 345–350.
[33] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2018.
[34] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. London, U.K.: Chapman & Hall, 1984.
[35] R. C. Barros, A. C. de Carvalho, and A. A. Freitas, *Automatic Design of Decision-Tree Induction Algorithms*. New York, NY, USA: Springer, 2015.
[36] (2019). *Package Cubist*. Accessed: Nov. 19, 2019. [Online]. Available: https://cran.r-project.org/web/packages/Cubist/Cubist.pdf
[37] J. R. Quinlan, "Learning with continuous classes," in *Proc. Austral. Joint Conf. Artif. Intell.*, 1992, pp. 343–348.
[38] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proc. Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2017, pp. 3149–3157.
[39] O. Isakova, "Application of machine learning algorithms for classification and regression problems for mobile game monetization," M.S. thesis, Dept. Appl. Comput. Sci. Biosci., Study Programme, Appl. Math. Digit. Media, Univ. Appl. Sci., Mittweida, Germany, 2019.
[40] A. Zhukov, D. Sidorov, A. Mylnikova, and Y. Yasyukevich, "Random forest, support vector regression and gradient boosting methods for ionosphere total electron content nowcasting problem at mid-latitudes," in *Proc. 2nd URSI Atlantic Radio Sci. Meeting*, 2018, pp. 1–3.
[41] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínský, M. Weininger, and M. Zamani, "dtControl: Decision tree learning algorithms for controller representation," in *Proc. Int. Conf. Hybrid Syst., Comput. Control*, 2020, pp. 1–7.
[42] P. Ashok, J. Křetínský, K. G. Larsen, A. Le Coënt, J. H. Taankvist, and M. Weininger, "SOS: Safe, optimal and small strategies for hybrid Markov decision processes," in *Quantitative Evaluation of Systems*. Cham, Switzerland: Springer, 2019, pp. 147–164.
[43] S. Ngah, R. A. Bakar, A. Embong, and S. Razali, "Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 7, pp. 4882–4888, 2016.
[44] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049, Sep. 1996.
[45] M. Masadeh, O. Hasan, and S. Tahar, "Machine learning-based self-compensating approximate computing," in *Proc. Int. Syst. Conf.*, 2020, pp. 1–6.
[46] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.
[47] M. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang, "Input responsiveness: Using canary inputs to dynamically steer approximation," in *Programming Language Design and Implementation (PLDI)*. New York, NY, USA: ACM, 2016, pp. 161–176.
[48] (2020). *Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope*. Accessed: Jan. 8, 2020. [Online]. Available: http://people.csail.mit.edu/torralba/code/spatialenvelope/
[49] C. Solomon, *Fundamentals of Digital Image Processing a Practical Approach With Examples in MATLAB*. Hoboken, NJ, USA: Wiley, 2011.
[50] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. Int. Conf. VLSI Design*, 2011, pp. 346–351.
[51] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. Int. Conf. Electron Devices Solid-State Circuits*, 2010, pp. 1–4.
[52] T. Yang, T. Ukezono, and T. Sato, "Low-power and high-speed approximate multiplier design with a tree compressor," in *Proc. Int. Conf. Comput. Design*, 2017, pp. 89–96.
[53] Z. Yang, J. Han, and F. Lombardi, "Approximate compressors for error-resilient multiplier design," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Nano Syst.*, 2015, pp. 183–186.

**Mahmoud Masadeh** (Student Member, IEEE) received the B.Sc. degree from Yarmouk University, Irbid, Jordan, in 2003, the M.Sc. degree from Delft University, Delft, The Netherlands, in 2011, both in computer engineering, and the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, QC, Canada, in 2020, under the supervision of Prof. Sofiéne Tahar.

Currently, he is an Assistant Professor of computer engineering at Yarmouk University. His research focuses on approximate computing and energy-efficient VLSI circuit design.

**Osman Hasan** (Senior Member, IEEE) received the B.Eng. degree (Hons.) from the University of Engineering and Technology, Lahore, Pakistan, in 1997, and the M.Eng. and Ph.D. degrees from Concordia University, Montreal, QC, Canada, in 2001 and 2008, respectively.

Currently, he is an Associate Professor at the School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan. He is the Founder and Director of the System Analysis and Verification Laboratory, NUST, which mainly focuses on the design and formal verification of safety-critical systems.

**Sofiène Tahar** (Senior Member, IEEE) received the Engineering Diploma degree from the University of Darmstadt, Darmstadt, Germany, in 1990, and the Ph.D. degree (Hons.) from the University of Karlsruhe, Karlsruhe, Germany, in 1994.

He is currently a Professor and Senior Research Chair in formal verification of systems-on-chip with the Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada. He is the Founder and Director of the Hardware Verification Group with a research interest in formal verification of hardware and physical systems, and safety and reliability analysis.