Towards Data-Driven Approximate Circuit Design

Ling Qiu†, Ziji Zhang‡, Jon Calhoun† and Yingjie Lao†
†Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634
‡University of Electronic Science and Technology of China, Chengdu, China, 610054
Email: {lingq,jonccal,ylao}@clemson.edu, {201611030130}@uestc.edu.cn

***Abstract*—Approximate computing provides a paradigm shift to reduce the hardware cost or improve the speed of logic circuits by relaxing the quality of computation. This paper presents a novel approximate logic design methodology that integrates input data distribution to optimize the performance for specific applications. Furthermore, the proposed approach also provides sufficient scalability as only the circuit netlist and input data are needed in designing the approximate circuit. Our experimental results show that the data-driven designs generated by the proposed method achieve significantly better accuracy than the data-independent ones, while maintaining minimal hardware overheads. For example, in the design of an approximate finite impulse response (FIR) filter, our method reduces nearly 40% of the error metrics while still achieving a 29% area reduction, compared to the original exact circuit, which further validating the robustness of the proposed method beyond arithmetic elements.**

I. Introduction

Electronic devices are becoming smaller and faster each year while embedding more sophisticated functionalities. However, energy efficiency in hardware has become a major bottleneck constraining this trend, especially in the deep nanometer technologies. To this end, approximate computing in hardware design has emerged as one promising approach to ameliorate this issue by slightly sacrificing the accuracy of the computational result in exchange for a large amount of power/area reduction [1]. The effectiveness of approximate circuits is well-known in various levels ranging from transistors to architectures [1]–[9]. Another underlying reason behind this computing paradigm is that a large body of prevailing resource-hungry applications such as machine learning and image processing are error-resilient. This indicates that a tolerable amount of error in these applications can be potentially exploited to trade for reductions in hardware complexity.

Approximate computing in hardware design at the logic-level is broadly divided into two categories according to the design strategies: manual designs and automatic approaches. Manual design strategies have achieved excellent performance on arithmetic elements (e.g., adders [1]–[3] and multipliers [4]–[7]). These design strategies usually require analyzing the resiliency of each component in a target circuit, which are difficult to generalize. On the other hand, automatic design strategies focus on developing general methodologies for designing approximate circuits. For instance, the concept of approximate logic synthesis (ALS) [10] has been developed to automatically synthesize a Boolean function into either a two-level [11] or multi-level [10] approximate version under given error constraints. Other techniques such as probabilistic pruning has also been proposed to obtain approximate circuits by pruning the internal nodes selectively at the gate-level [12], [13]. In our recent work, we have also developed an algorithm that exploits the input-error pattern to automatically generate a lightweight compensation block to further optimize a given approximate circuit [14].

It is worth noting that most of the existing approximate logic designs in the literature [3]–[5] assume a uniform input distribution, which might not always be the case in practice. In particular, many computational tasks are data-oriented such that they mainly operate on a unique data pattern. In fact, capturing the input distribution is currently a very popular problem in the field of deep learning [15]. Therefore, it is imperative to take the data-driven perspective into account when designing approximate logic circuits.

In this paper, we propose a novel algorithm to automatically design the compensation circuit, a hardware block used to mitigate computational error, for a given approximate circuit with a specific input distribution. This method is able to cooperate with the existing approximate circuit designs and approaches to further improve their performance. This work builds upon the concept of using feature selection to design approximate logic circuits [14] and develops a more robust method while incorporating data-driven considerations into the approximate circuit design. The main contributions of this paper are summarized as follows:

- We propose a modified Forward Stepwise Selection technique for selecting the feature subset, which significantly improves the computing accuracy of the generated approximate circuit.
- As opposed to our prior work in [14] that only uses the primary inputs as features, we expand the candidate features to internal nodes of a circuit to further optimize the performance.
- The proposed methodology achieves better scalability and generality than the prior manual design methods.
- Our approach yields better accuracy while only imposing a minimal hardware overhead, especially from the timing perspective.

The rest of the paper is organized as follows: In Section II, we present a brief overview of approximate circuit designs and data-driven methods. We then describe our proposed methodology in detail in Section III. Section IV presents the experimental results. Finally, Section V concludes this paper.

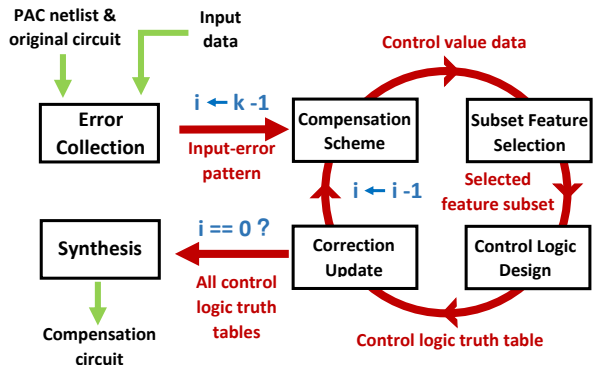


Fig. 1: General Design Flow

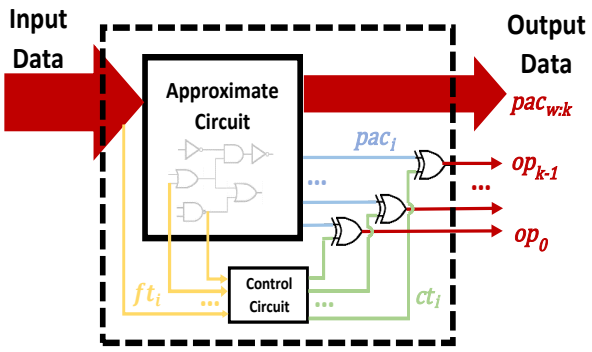


Fig. 2: Compensation Structure

II. Background

Many existing approximate logic circuit designs start with over-scaling or truncating exact circuits to reduce the hardware cost. We denote these intermediate circuit before compensation as the Primitive Approximate Circuit (PAC). Since PACs often carry relatively large amounts of errors, various techniques on designing lightweight compensation circuits via theoretical analysis have been proposed in the literature [4], [8], [9] to mitigate these errors,.

In our prior work, we have introduced an alternative method to treat the PAC as a black box and use machine learning based methods, specifically feature selection, to systematically design the compensation circuit [14]. For example, in order to compensate a truncated 10-bit multiplier, two compensation bits are added to the truncated multiplier's output. For each of these bits, only the most correlated primary inputs are selected to construct its compensation circuits. As a result, the overall compensation logic is simplified while still maintaining a high computational accuracy.

Although machine learning methods have already been investigated in the approximate circuit design [14], there are still several design spaces that have not been exploited. First, since we expect input patterns to vary across different tasks as we have discussed in Section I, the approximate circuit should also be adapted accordingly to better fit a specific computational task. Second, the methodology proposed in [14] considers area and power as the primary metrics of hardware complexity. It takes the difference between the erroneous output and the original output as the compensation value, which yields an additive compensation circuit that may an incur non-negligible timing overhead. Therefore, this scheme is not suitable for applications where speed is a major concern.

III. Methodology

In our proposed approach for designing approximate logic circuits, we incorporate data pattern as well as delay considerations into the design of compensation circuits. Both input samples and the netlist of the PAC are required for initiating the proposed methodology. The overall design flow is shown in Fig. 1, which is explained in details below.

A. Compensation Scheme

According to the input-error patterns of the PAC, the proposed methodology first determines the error dynamic range, which decides the bit-length of the compensation output, k. As opposed to the additive compensation circuit in [14], we introduce an XOR gate for the correction of each PAC's output bit within the error range. We use pac_i and ct_i to represent PAC's i-th output bit and its corresponding error control bit, respectively. A basic block for the compensation scheme is shown in Fig. 2, where ct_i and pac_i are the two inputs of the XOR gate and the corrected output, indicated as op_i is the XOR gate's output. According to the XOR logic, if pac_i needs a correction (i.e., from 1 to 0 or from 0 to 1), ct_i will be set to 1.

The control circuit is responsible for generating all the error control bits; and thus it comprises k control subcircuits. The control circuit is built iteratively from the error dynamic range's most significant bit (MSB), $k-1$, to the least significant bit (LSB), 0, as shown in Fig. 1. Since the procedure in each iteration is identical, without the loss of generality, we use the i-th iteration (the construction of i-th control subcircuit) as an example to illustrate the methodology.

The first step of building the i-th control subcircuit is to determine the goal value of its error control bit, indicated as $\mathbf{CT_i}$, for every input sample. Assuming the PAC's output bit-length is w, we denote the PAC and the original circuit's w-bit output as $\tilde{\mathbf{P}}$ and $\mathbf{P}$, respectively. In order to correct the output of each sample as close to the original output as possible, we use the following scheme to determine $\mathbf{CT_i}$ for every sample:

$$\mathbf{CT_i} = \begin{cases} \mathbf{P}[i] \oplus \tilde{\mathbf{P}}[i] & \mathbf{P}[w-1:i] = \tilde{\mathbf{P}}[w-1:i] \\ 1 \oplus \tilde{\mathbf{P}}[i] & \mathbf{P}[w-1:i] > \tilde{\mathbf{P}}[w-1:i] \\ 0 \oplus \tilde{\mathbf{P}}[i] & \mathbf{P}[w-1:i] < \tilde{\mathbf{P}}[w-1:i] \end{cases} \quad (1)$$

where $\tilde{\mathbf{P}}[w-1:i]$ represents the partial binary number from the MSB to the i-th bit of $\tilde{\mathbf{P}}$.

B. Modified Forward Stepwise Selection

For the purpose of reducing the hardware complexity, we only select a small subset of nodes (features) to generate each

control subcircuit. We denote the overall candidate feature pool as $\mathbf{F}$ and a single feature in $\mathbf{F}$ as ft. In this paper, we largely expand $\mathbf{F}$ by including the internal nodes of the PAC, while the method in [14] only uses the primary inputs to construct $\mathbf{F}$. This feature pool expansion significantly improves the control circuit's performance as we observe in the experiment that the majority of selected features are indeed these internal nodes.

χ^2 univariate feature selection technique is used in [14] to rank each feature individually according to its correlation with $\mathbf{CT_i}$. Due to its greedy nature, it may not be able to capture the inter-correlation among the features in the subset. To address this problem, we propose a modified Forward Stepwise Selection (FSS) algorithm as described in Algorithm 1 for feature subset selection. The conventional FSS begins with a null feature subset, and then adds features to the subset one by one, until meeting a stopping criteria [16]. In our experiments, we set the maximum number of selected features as the stopping criteria. Note that this value can also be considered as a user-defined parameter for trading off between error and hardware complexity, as more selected features usually indicates higher hardware complexity. FSS requires the performance evaluation of a selected feature subset as a whole at each step of feature selection. In our method, we employ a cost function, $\mathcal{C}(\cdot)$, as expressed in Equation (2):

$$\mathcal{C}(\hat{F}) = \|\mathbf{CT_i} - \mathcal{TT}(\hat{F})\|_0 \tag{2}$$

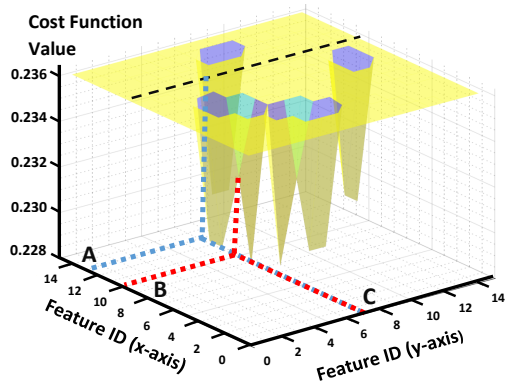
where $\hat{F}$ is the selected feature subset and $\mathcal{TT}$ is the generated truth table for $\hat{F}$, which will be explained in Section III-C.

Algorithm 1: Modified Forward Stepwise Selection

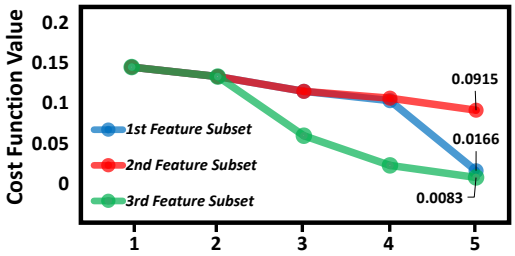
Input: Candidate feature pool $\mathbf{F}$, error control value $\mathbf{CT}_i$, number of subsets h, number of selected features m, cost function $(\mathcal{C})$ $(\cdot)$

Output: Selected feature subset $\hat{F}$

```
for x ← 1 to h do
    reset F to initial;
    let F̂_{x,0} denote the null subset;
    for y ← 0 to m − 1 do
        if y ≠ 0 and C(F̂_{x,y}) = C(F̂_{x,y−1}) then
            remove last added feature from both F̂_{x,y} and F;
            y ← y − 1;
        end
        else
            ft = arg min_F C(F̂_{x,y} ∪ ft);
            F̂_{x,y+1} ← F̂_{x,y} ∪ ft;
        end
    end
end
Select F̂_{x,y} with lowest C as F̂;
return F̂
```



(a) Cost function distribution for pairwise coupling 15 features



(b) Cost function value for different selected feature subsets

Fig. 3: Two Observations on Forward Stepwise Selection

Based upon the conventional FSS, we modify the feature selection procedure specifically for our methodology according to the following two observations.

1) Observation 1: For a model whose features and response variables are both binary, conventional FSS sometimes settles at a local minimum. In particular, we have seen many such cases in our experiments that the feature with the highest score (i.e., cost function reduction) converges to a local minimum where no further reduction on the cost function is possible by adding any additional feature, which also terminates the conventional FSS. In addition, due to the binary representations, many features yield the same scores. Thus, in our case, we argue that always selecting the feature with the highest score at each step might not always be optimal. Fig. 3(a) illustrates a case in our experiments of the cost function distribution from exhaustively pairing 15 feature candidates. In this figure, each of the features individually reduce the cost function value to 0.236. In other words, any of these features may be selected at the first iteration of FSS. However, feature #13 (point A) is a local minimum, since pairing with any additional features does not reduce the cost function. This is indicated by the black dash line. Neither feature #10 (point B) nor feature #7 (point C) is a local minimum, as the cost function can be further lowered by adding an additional feature into the feature subset. As shown in Algorithm 1, we propose a modification such that when a local minimum is found, it is removed from both the feature subset ($\hat{F}$) and the candidate feature pool ($\mathbf{F}$) to escape from the local minimum. Through this approach, bad local minima are gradually avoided, which eventually results

in a better feature subset.

2) Observation 2: As we have mentioned above, there are usually multiple candidate features that achieve a same cost function reduction under the Boolean logic setting. Since we find it difficult to predict which feature could yield the best performance in the subsequent feature selections, we repeat the modified FSS for h times and hence build h different feature subsets. In general, the larger h is, a better feature subset can be obtained; however, the time complexity of the algorithm will increase. In the example of Fig. 3(b), we generated 3 different feature subsets with 5 selected features for each subset. Although the first 2 selected features for all the 3 feature subsets have the same cost function values, they progress differently in the subsequent steps. Lastly, after completing all the iterations, we select the feature subset with the lowest cost function among the h subsets as the final $\hat{F}$ (i.e., the 3rd feature subset in this example).

C. Control Logic Design

In this subsection, we introduce the procedure of generating the reduced truth table for the selected feature subset, ct_i. This procedure is used in selecting the feature subset as well as constructing the final control logic (ct_i) truth table.

Given the selected feature subset, we can construct its reduced truth table for the control subcircuit, $\mathcal{TT}$. The error control bit value for each entry of the truth table is determined by the larger appearance between 0's and 1's to minimize the overall error as expressed in Equation (3):

$$TT_j = \begin{cases} 0 & Pr(0) > Pr(1) \\ 1 & Pr(0) < Pr(1) \\ X & Pr(0) \approx Pr(1) \end{cases}. \tag{3}$$

where TT_j represents the response value for the jth entry, and $Pr(0)$, $Pr(1)$ are the appearance percentages of 0's and 1's, respectively. We assign don't cares when $Pr(0)$ and $Pr(1)$ are equal or close, which could be exploited in logic optimization for reducing the hardware cost. We use an example in Table I to illustrate the basic idea. In this table, we have a total number of 4 candidate features and 1 response, which is the value of the i-th error control bit, $\mathbf{CT_i}$. Suppose features ft_w and ft_z are selected to build a reduced truth table for ct_i. As it can be seen in Table I that when $\{ft_w, ft_z\} = 01$, ct_i is 1, regardless the values of ft_x and ft_y. When $\{ft_w, ft_z\} = 00$, $Pr(0) = \frac{2}{3}$, which is larger than $Pr(1) = \frac{1}{3}$. Therefore, when $\{ft_w, ft_z\} = 00$, we choose ct_i to be 0. In this example, there is one out of four rows in the original true table that does not match the reduced truth table. Thus, according to Equation (2), we obtain $\mathcal{C}(\{ft_w, ft_z\}) = \frac{1}{4}$.

D. Correction Data Update

After the circuit for ct_i is determined, we need to update PAC's corresponding i-th output bit, i.e., $\tilde{\mathbf{P}}[\mathbf{i}]$, since the current bit may not be corrected completely thus may affect the compensation scheme of lower bits. We update these compensation values for each sample as following:

TABLE I: Reduced Truth Table Generation

(a) Original $\mathbf{CT_i}$ Truth Table

Feature $\mathbf{F}$				Response
ft_w	ft_x	ft_y	ft_z	$\mathbf{CT_i}$
0	1	1	0	1
0	0	0	0	0
0	0	1	1	1
0	0	1	0	0

(b) Reduced Truth Table

Feature $\hat{F}$		Response
ft_w	ft_z	ct_i
0	0	0
0	1	1

$$\tilde{\mathbf{P}}[\mathbf{i}] = \tilde{\mathbf{P}}[\mathbf{i}] \oplus \mathcal{TT}(\hat{F}) \tag{4}$$

After the completion of the current iteration, the algorithm moves on to the next lower bit. The overall design methodology is presented in Algorithm 2. After the logic for all the control subcircuits are determined, the generated compensation circuit is synthesized and connected to the PAC as shown in Fig. 2.

Algorithm 2: Control Circuit Generation

```
INPUT: P, P̃ and F;
OUTPUT: TT and F̂ for all correction bit;
for i = k − 1 to 0 do
    for each sample do
        assign CT_i according to Equation (1);
    end
    Feature Selection: Modified FSS;
    Construct truth table TT for ct_i;
    if i = 0 then break;
    for each sample do
        Update P̃ by Equation (4);
    end
end
```

IV. Case Study and Experimental Results

A. Experimental Setup

The proposed design methodology is implemented on Python Jupyter Notebook. For comparison, we also use scikit-learn [17], a well-known machine learning library in Python, to perform the χ^2 feature selection. We use two real-world datasets, ALYA and GADGET [18], to demonstrate the advantage of the proposed data-driven method for compensation circuit design. We randomly select 10000 samples from ALYA and all of the 601 samples from GADGET to generate input-error patterns. All the circuits including the PAC and the final approximate circuits in our experiments are synthesized into netlists using a 32nm technology node. The features are obtained from the Value Change Dump (VCD) file, which is generated by the Synopsys Verilog Compiler and Simulator (VCS). The VCD file is an ASCII-based format file storing circuit signal transitions, from which we extracted signal state information. Mean absolute error (MAE) and error rate (ER) are used as the error metrics to evaluate the performance of the generated approximate logic circuits.

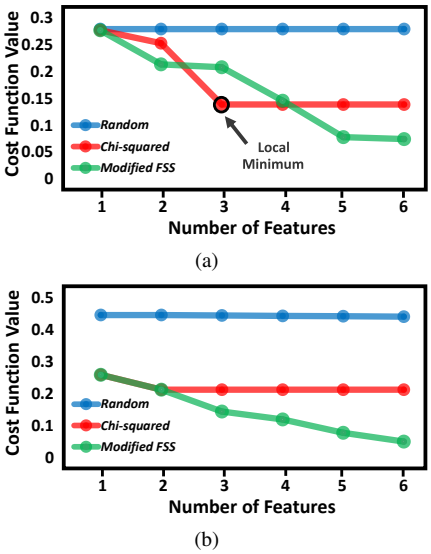


Fig. 4: Comparison between the Modified FSS and χ^2 Feature Selection on a 12-bit Truncated Fixed-width Multiplier: (a) the 3rd LSB, (b) the 2nd LSB

B. Comparison between the Modified FSS and χ^2 Feature Selection

We first compare the performance on the feature subset selection between the modified FSS and χ^2 used in [14]. Fig. 4 shows the cost function trends for the 3rd and 2nd LSBs during the design of a 12-bit approximate multiplier. Both the proposed modified FSS and χ^2 feature selection achieve better performance than randomly selecting features. However, due to the greedy nature of the χ^2 feature selection, it might converge to a bad local minimum, as indicated in Fig. 4(a). In contrast, the proposed modified FSS escapes from these local minima to achieve lower cost functions by gradually eliminating these features from the candidate pool.

C. Approximate Fixed-width Multiplier Design

To illustrate the effectiveness, we apply the proposed methodology to compensate truncated fixed-width radix-4 booth multipliers under various settings based on three input data: ALYA, GADGET and UNIFORM (uniformly distributed inputs). UNIFORM is used to build reference circuits, which corresponds to the data-independent compensation circuits. Since different input data result in different error distributions, the number of compensation bits is determined by the error dynamic range and the final approximate circuit is generated as described in Section III.

The performances on different input data are summarized in Table II, where DD-ALYA represents the approximate multiplier generated from the ALYA dataset using the data-driven approach. The corresponding hardware costs are shown in Fig. 5. Using the proposed method, the error can be significantly reduced from the PAC while incurring very small hardware overheads. If we constrain the candidate feature pool when designing DD-AYLA to the internal nodes that are close to the primary inputs, the delay of the compensated circuit would not increase. However, if we want to find the optimal feature subset by including all the internal nodes into the candidate feature pool, the delay might be increased slightly (e.g., DD-GADGET and DD-UNIFORM) when some of the selected features are close to the output of the circuit. In this case, the propagation delay of the compensation block added into the path from the primary inputs to the selected internal nodes exceeds the critical path of the PAC. Therefore, for applications that delay is the primary concern, we should limit the candidate feature pool to only internal nodes that are close to the primary inputs.

All of the approximate logic circuits designed using the proposed data-driven methodology show superior performance in terms of accuracy when tested on their own data, which is indicated by the bold numbers in Table II. It can also be observed that the approximate circuits designed without considering data patterns are sub-optimal in applications that have specific input distributions. For example, both ALYA and GADGET datasets yield larger errors on the 10-bit DD-UNIFORM approximate multiplier than the uniformly distributed data. Therefore, it can be concluded that considering the input data distribution is important and desired in designing approximate logic circuits for data-oriented tasks.

TABLE II: Comparison of the Approximate Multipliers on Different Input Data: MAE (ER%)

(a) ALYA

	Truncate Circuits	**DD-ALYA Circuits**	DD-GADGET Circuits	DD-UNIFORM Circuits
10-bit	0.9114 (84.94%)	**0.4613 (39.93%)**	0.5452 (48.32%)	1.4636 (70.55%)
12-bit	0.9242 (85.46%)	**0.4641 (39.63%)**	1.0924 (94.05%)	0.5480 (48.04%)
16-bit	0.9242 (86.27%)	**0.4623 (40.08%)**	1.3041 (93.91%)	1.4492 (70.43%)

(b) GADGET

	Truncate Circuits	DD-ALYA Circuits	**DD-GADGET Circuits**	DD-UNIFORM Circuits
10-bit	1.4792 (96.38%)	1.444 (94.84%)	**0.6772 (49.75%)**	1.0183 (70.38%)
12-bit	1.6855 (96.33%)	1.7154 (96.83%)	**0.8136 (56.57%)**	0.9617 (76.04%)
16-bit	2.7704 (99.80%)	2.7221 (98.66%)	**0.9417 (62.56%)**	1.0881 (72.21%)

(c) UNIFORM

	Truncate Circuits	DD-ALYA Circuits	DD-GADGET Circuits	**DD-UNIFORM Circuits**
10-bit	1.8653 (97.53%)	1.8586 (93.72%)	2.4257 (84.21%)	**1.0525 (65.38%)**
12-bit	2.2464 (99.03%)	2.2474 (98.05%)	1.8789 (83.77%)	**1.0907 (68.72%)**
16-bit	2.9999 (99.80%)	2.9836 (99.36%)	1.6411 (77.66%)	**1.1342 (70.84%)**

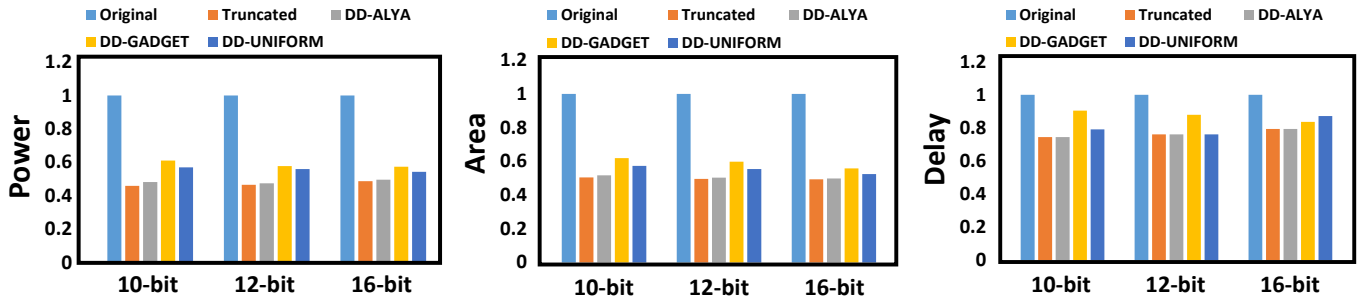
TABLE III: Quantum Circuit Implementation Comparison For Different Schemes for $p = 1$

(a) Factoring 143

	DIRECT	**SCHALLER**	GROBNER	NAIVE
qubit number	4	4	6	6
single operation	14	15	1.0924 (94.05%)	0.5480 (48.04%)
cnot operation	34	**0.4623 (40.08%)**	1.3041 (93.91%)	1.4492 (70.43%)

(b) Factoring 271311

	Truncate Circuits	DD-ALYA Circuits	**DD-GADGET Circuits**	DD-UNIFORM Circuits
10-bit	1.4792 (96.38%)	1.444 (94.84%)	**0.6772 (49.75%)**	1.0183 (70.38%)
12-bit	1.6855 (96.33%)	1.7154 (96.83%)	**0.8136 (56.57%)**	0.9617 (76.04%)
16-bit	2.7704 (99.80%)	2.7221 (98.66%)	**0.9417 (62.56%)**	1.0881 (72.21%)



(a) (b) (c)

Fig. 5: Hardware Consumption Comparison between Different Compensated Approximate Multipliers (Normalized to the Original Circuits)

TABLE IV: Compensation performance on an approximate 4-tap FIR filter

	Original FIR	[5] Multiplier	[5] FIR	DD-GADGET FIR
Area	5074.94	792.68	3640.77	**3859.39**
Power	902.075	120.67	682.69	**723.82**
MAE (ER%)	0 (0%)	0.2379 (23.79%)	0.4732 (39.96%)	**0.3013 (23.92%)**

D. Approximate FIR Filter

In this experiment, we demonstrate the effectiveness of the proposed method on designing large and complex approximate circuits, which is one advantage of the proposed data-driven methodology compared to manual analysis. Alternatively, if we construct a large circuit with approximate arithmetic elements, the final output may accumulate to an intolerable error magnitude. As an example for demonstration, we build an approximate 10-bit 4-tap finite impulse response (FIR) filter circuit whose multipliers are replaced by the manually designed approximate multiplier in [5]. For a single 10-bit approximate multiplier, the MAE and ER are 0.2379 and 23.79%, respectively, as shown in Table III. However, the MAE and ER of the corresponding approximate 4-tap FIR filter are aggravated to 0.4732 and 39.96%, respectively, which indeed proves that the straightforward implementation of building large approximate circuits with approximate arithmetic elements may not achieve a satisfying performance.

A significantly better performance can be achieved by using the proposed data-driven method to further compensate this approximate FIR filter. Five-fold cross validation is adopted. As presented in Table III, our method leads to a design with nearly a 40% error reduction and only a 6% hardware overhead. Furthermore, compared to manual design strategies, this proposed systematic method also greatly reduces the design workload.

V. Conclusion

This paper presented a novel data-driven approximate logic design methodology. The steps of the proposed approach were discussed in detail. Experimental results were also provided to illustrate the effectiveness of the proposed approach, which indeed indicates the need for developing scalable data-driven approximate computing methods.

References

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

[2] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 343–348.

[3] W. Liu, L. Chen, C. Wang, O. Maire, and F. Lombardi, "Design and analysis of inexact floating-point adders," *IEEE Transactions on Computers*, no. 1, pp. 1–1, 2016.

[4] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified booth multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 522–531, May 2004.

[5] Z. Zhang and Y. He, "A low-error energy-efficient fixed-width booth multiplier with sign-digit-based conditional probability estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 2, pp. 236–240, Feb 2018.

[6] S. Hashemi, R. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proceedings of the IEEE/ACM international conference on computer-aided design*. IEEE Press, 2015, pp. 418–425.

[7] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–4.

[8] B. Moons and M. Verhelst, "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2015, pp. 237–242.

[9] S. Zhang and N. R. Shanbhag, "Embedded algorithmic noise-tolerance for signal processing and machine learning systems via data path decomposition," *IEEE Transactions on Signal Processing*, vol. 64, no. 13, pp. 3338–3350, July 2016.

[10] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. IEEE, 2012, pp. 796–801.

[11] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 957–960. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870926.1871159

[12] J. Schlachter, V. Camus, C. Enz, and K. V. Palem, "Automatic generation of inexact digital circuits by gate-level pruning," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 173–176.

[13] A. Lingamneni, C. Enz, K. Palem, and C. Piguet, "Synthesizing parsimonious inexact circuits through probabilistic design techniques," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 93, 2013.

[14] L. Qiu and Y. Lao, "A systematic method for approximate circuit design using feature selection," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[15] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[16] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[17] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[18] UEABS, "Unified european applications benmark suite," 2013, http://www.prace-ri.eu/ueabs/.