Data-Driven Feature Selection Framework for Approximate Circuit Design

Bingyin Zhao, *Student Member, IEEE*, Ling Qiu, *Student Member, IEEE*, and Yingjie Lao, *Senior Member, IEEE*

***Abstract*—The ever-growing data scale and computation complexity raise tremendous concerns about computer systems' efficiency (i.e., lower hardware overhead and power consumption). Orthogonal to the advancement in semiconductor manufacturing technologies, approximate computing provides an alternative paradigm to reduce the hardware cost and power dissipation by relaxing computation quality for error-resilient applications. Voltage over-scaling (VOS) and approximate logic design (ALD) have become two mainstream approaches of approximate computing due to their superior performance in efficiency-critical designs. VOS reduces the power in quadratic by scaling down supply voltage while ALD saves hardware overhead by redesigning an approximate version of a given circuit (e.g., trimming less significant circuitry). However, these primitive approximate circuits (PAC) inevitably introduce notable errors and require additional error compensation circuits (ECC) to preserve computation accuracy. In existing works of ECC design, there lacks a systematic method that can generalize well to different approximate computing approaches. In this paper, we present a data-driven feature selection framework for approximate circuit design, which is applicable to both VOS and ALD. We propose novel algorithms that profoundly analyze the correlation between input data and output errors and select the most critical features to generate compensation circuits. Extensive evaluations are performed over a variety of circuits using approximate finite impulse response (FIR) filters and the prevalent approximate computing benchmark AxBench. The experimental results show that the proposed approach achieves superior compensation performance, boosting the circuit accuracy while only introducing trivial area overhead.**

***Index Terms*—Approximate Computing, Error Correction, Feature Selection, Computer-Aided Circuit Design**

I. Introduction

Computer systems and electronic devices that can process large-scale data are emerging with the unprecedented development of information technology and the cumulative increment of data. However, these systems and devices are computationally expensive and energy-consuming to handle ever-increasing sophisticated tasks. Circuit overhead and power efficiency have become major bottlenecks in computer hardware design. A recent survey [1] shows that it is challenging and essential to significantly reduce hardware cost and improve energy efficiency for emerging workloads to keep pace with rapid information growth. The evolving semiconductor manufacturing technologies provide possible directions to tackle these issues. However, as Moore's law is nearing its end [2], it is imminent to exploit alternatives. To this end, approximate computing in hardware design has emerged as one promising workaround by slightly sacrificing the accuracy of the computational result in exchange for a large amount of power/area reduction [3]. The underlying reason behind this computing paradigm is that a large body of prevailing resource-hungry applications such as machine learning [4] and image processing are error-resilient [5]. For example, deep neural networks are elastic to parameter perturbations induced by rounding errors and preserve model accuracy even when transforming the representation of parameters from 32-bit floating-point numbers into 4 bits. This indicates that a tolerable amount of error in these applications can be potentially exploited to trade for reductions in hardware complexity and energy consumption. Plenty number of prior works have shown the effectiveness of approximate circuits in various levels ranging from transistors to architectures [3], [6]–[13].

In the existing literature, voltage over-scaling (VOS) [14]–[18] and approximate logic design (ALD) [3], [19]–[28] have been investigated for approximate computing. VOS scales the supply voltage to reduce power dissipation while ALD redesigns an approximate variant of a logic circuit to save circuit area. Despite the significant reduction in power and area, such approximate circuits inevitably render non-trivial computation errors. In consideration of the inclination to avoid "Robbing Peter to Pay Paul", researchers proposed distinctive error compensation strategies to optimize the performance of VOS and ALD [29]–[31], [31]–[35]. It is common practice for VOS to manually analyze computation errors and design hardware implementations that achieve better accuracy. On the other hand, the line of works of error compensation for ALD can be broadly divided into two categories according to the design strategies: manual designs and automatic approaches. Manual designs usually require analyzing the resiliency of each component in a target circuit, while automatic counterparts focus on developing general methodologies for designing low error approximate circuits. However, there lacks a systematic method that can generalize well to different approximate computing paradigms. We, for the first time, attempt to tackle this issue and propose a general data-driven framework of automatic error compensation circuits (ECC) design for both VOS and ALD.

It is worth noting that most of the existing approximate computing designs in the literature [7]–[9] 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

Bingyin Zhao, and Yingjie Lao are with the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA. E-mail: {bingyiz,ylao}@clemson.edu

Ling Qiu is with College of Information Sciences and Technology, Penn State University, State College, PA 16802, USA. Ling contributed to this work when he was with Clemson University. E-mail: lingq@psu.edu

a very popular problem in the field of deep learning [36]. Therefore, it is imperative to take the data-driven perspective into account when designing approximate logic circuits. In this paper, we propose novel algorithms 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 framework is able to cooperate with the existing approximate computing approaches to further improve their performance. This work is an extended version of our prior work [23], which builds upon the concept of using feature selection to design compensation circuits for approximate circuits and develops a more robust method by incorporating data-driven considerations into the approximate circuit design. The main contributions of this paper are summarized as follows:

- We extend the design concept of our prior work [23] and propose a data-driven framework that is applicable to both voltage over-scaling and approximate logic design, for automatic error compensation circuit design.
- We develop modified Forward Stepwise Selection and Backward Stepwise Selection algorithms for selecting the feature subset, which significantly improves the computing accuracy of the generated approximate circuit.
- We consider optimized strategies to determine candidate features by differentiating the characteristics of VOS and ALD. We employ primary inputs as features for VOS and expand the candidate features to internal nodes of a circuit to further optimize the performance for ALD.
- Our approach achieves superior performance under the comprehensive evaluation using approximate finite impulse response (FIR) filters and approximate computing benchmark AxBench. It yields significant reduction in multiple error metrics while only incurring a minimal hardware overhead and power consumption.

The rest of the paper is organized as follows: In Section II, we present a brief overview of related works. We then describe the proposed framework and algorithms in details in Section III. Section IV presents the experimental results. Finally, Section V concludes this paper.

II. Related Work

VOS is exceptionally effective in energy saving since power dissipation is quadratically correlated to supply voltage for CMOS circuits. In contrast to conventional voltage scaling techniques that scale the clock frequency simultaneously, VOS intentionally causes timing violations of critical paths. It is widely used in error-resilient applications such as video processing, data mining, and wireless communications. There has been a series of works to optimize the performance of VOS. [16] proposed an analytical method to select the proper computer arithmetic architecture in VOS signal processing systems by estimating the statistics of computation errors. [17] reduced the error caused by VOS in meta-functions (computational kernels) through dynamic segmentation with multi-cycle error compensation. [37] optimized trade-off between energy efficiency and error margin via modeling approximate operators using different operating triads. However, these works require carefully manual design and do not generalize well to more scenarios.

On the other hand, ALD produces an approximate version (e.g., truncate less significant sub-circuitry, skip carry using carry prediction, etc.) based on a full-precision logic circuit, which shows superior results in the area and power reduction. Both manual and automatic methodologies have been proposed to retain better accuracy for ALD. Manual design strategies have achieved excellent performance on arithmetic elements (e.g., adders [3], [6], [7] and multipliers [8]–[11]) by inspecting the characteristics of these building blocks and revising the original circuit design. Automatic designs, such as approximate logic synthesis (ALS) [21], have been developed to automatically synthesize a Boolean function into either a two-level [22], [38], [39] or multi-level [21], [25], [40]–[42] approximate version under given error constraints. Other techniques, such as probabilistic pruning, have also been proposed to obtain approximate circuits by pruning the internal nodes selectively at the gate-level [43], [44]. Another emerging technique is the high level synthesis for approximate computing [45], [46] that makes use of approximate circuits at system level [47] and incorporates with HLS toolchains to automatically generate inexact circuits [48]. Note that these approaches are designed explicitly for ALD and barely consider the importance of input data distribution in circuit design. Our recent work [20] developed an algorithm that exploited the input-error pattern to automatically generate a lightweight compensation block to optimize a given approximate circuit and achieved surprisingly good performance, revealing a worthwhile direction of incorporating data distribution into approximate circuit designs.

III. Methodology

A. General Design Flow

We denote the approximate circuits that start with VOS or ALD as the Primitive Approximate Circuit (PAC). Note that PACs often carry relatively large amounts of errors, which requires the design of lightweight compensation circuits to mitigate the accuracy drop. Our goal is to design effective error compensation circuits that preserve high accuracy at the minimum overhead. No prior approaches can comply with the approximate design of both VOS and ALD. Our idea is inspired by the fact that many applications are designed for specific needs. Input data of such applications usually have unique distributions or patterns, and the outputs are highly correlated to particular inputs. Hence, we propose a general framework for approximate circuit design that can be generalized to different paradigms. Our approach generates the error compensation circuits offline, which does not require the continuous collection of new features nor iteratively modify the compensation circuit design. This is a once-for-all solution that significantly reduces development costs. A concrete account of the design flow is given in Fig. 1.

The proposed framework is a data-driven approach that incorporates with feature selection algorithms to automatically generate compensation circuits for all PACs. It is composed of three main phases: error collection, feature selection, and

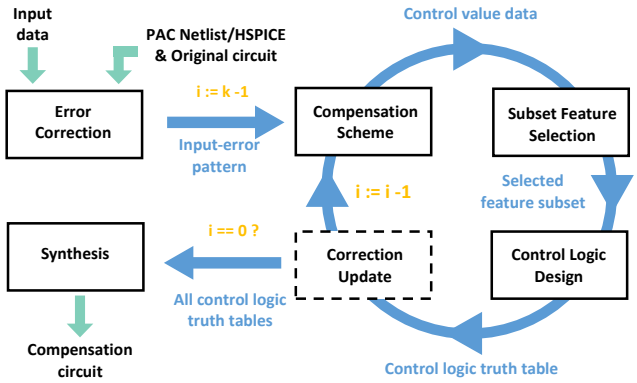


Fig. 1: General Design Flow

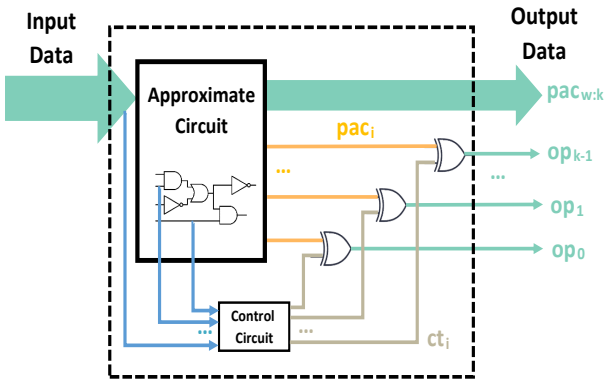


Fig. 2: Compensation Structure for ALD

compensation circuit generation. In the error correction phase, we gather the highly-biased outputs of PACs with given input data. We then seek the intrinsic connection between inputs and error outputs using tailored feature selection algorithms based on different compensation schemes due to the distinct PAC mechanisms (i.e., forward stepwise selection for ALD PACs and backward stepwise selection for VOS PACs). Finally, we generate the truth tables of control signals upon the selected features and corresponding responses and synthesize the compensation circuit accordingly. We present the detailed compensation schemes, feature selection algorithms, and control logic design of PACs generated by ALD and VOS in III-B and III-C, respectively.

B. Approximate Logic Design

In this work, we consider truncated circuits as the PACs, which is one of the most representative approaches of ALD. Since ALD usually revises the original circuit design (e.g., trims the less significant sub-circuitry) and produces a PAC with a modified structure, the circuit's internal signals are as important as the input samples, as the circuit functionality is highly correlated to both (some of the internal signals reflect how the circuit is modified). Therefore, we incorporate data distributions as well as delay considerations into the design of compensation circuits. Both input samples and internal signals (i.e., the netlist) of PACs are required for initiating the proposed methodology. The overall design flow is shown in Fig. 1, which is explained in detail below.

1) Compensation Scheme: The proposed methodology first determines the dynamic range of the error based on the input-error patterns of the PAC, , which decides the bit-length of the compensation output, k. In contrast to the additive compensation circuit in [20], we introduce an XOR gate to control the correction signal 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. The schematic of 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 comprises k control subcircuits and generates all the error control bits. It 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. Let the PAC's output bit-length be 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 $\tilde{\mathbf{P}}$.

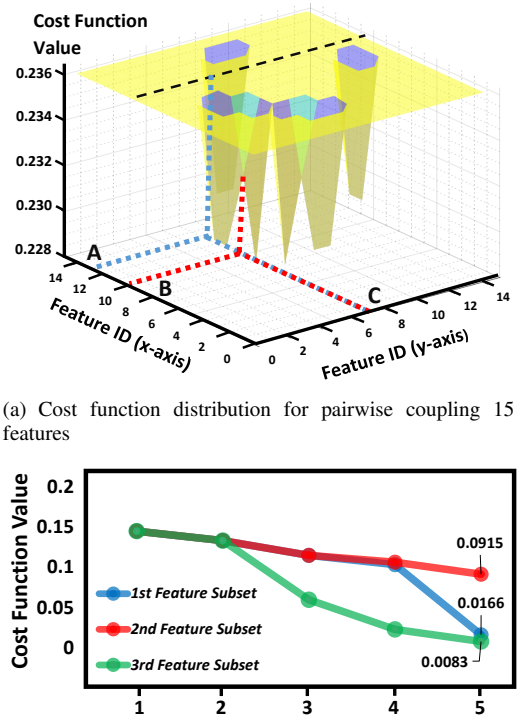
2) Modified Forward Stepwise Selection: For the purpose of saving the hardware overhead, we only select a small fraction (i.e., a subset) of nodes (features) to generate each control subcircuit. We denote the overall candidate feature pool as $\mathbf{F}$ and a single feature in $\mathbf{F}$ as ft. We expand $\mathbf{F}$ by including the internal nodes of the PAC to enrich the feature diversity. 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.

In [20], χ^2 univariate feature selection technique is used to rank each feature individually according to its correlation with $\mathbf{CT_i}$. However, it may not be able to capture the inter-correlation among the features in the subset due to its greedy nature. To address this problem, we propose a modified Forward Stepwise Selection (FSS) algorithm for feature subset selection. The detailed procedures are presented in Algorithm 1. The conventional FSS begins with a null feature subset, and then adds features to the subset one by one, until meeting a stopping criteria [49]. In our revised version, 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

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 j ← 0 to m − 1 do
        if j ≠ 0 and C(F̂_{x,j}) = C(F̂_{x,j−1}) then
            remove last added feature from both F̂_{x,j} and F;
            j ← j − 1;
        end
        else
            ft = arg min_F C(F̂_{x,j} ∪ ft);
            F̂_{x,j+1} ← F̂_{x,j} ∪ ft;
        end
    end
end
Select F̂_{x,j} 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

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-B3.

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

a) *Observation 1:* For a model whose features and response variables are both binary, conventional FSS sometimes settles at a local minimum. In particular, we empirically find that the feature with the highest score (i.e., cost function reduction) converges to a local minimum by directly employing the conventional FSS algorithm. There is no further reduction on the cost function by adding any additional feature, which also terminates the selection procedure. In addition, due to the binary representations, many features yield the same scores. Thus, we argue that always selecting the feature with the highest score at each step might not always be optimal. We present the cost function distribution from exhaustively pairing 15 feature candidates in Fig. 3(a) to illustrate this local optimum phenomenon. In this figure, each of the features individually reduces 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, indicating by the black dashed line. On the other hand, feature #10 (point B) and feature #7 (point C) are not local minima because the cost function can be further optimized 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. This approach gradually avoids bad local minima and eventually results in a better feature subset.

b) *Observation 2:* As prior mentioned, there are usually multiple candidate features that can achieve a same cost function reduction under the Boolean logic setting. Since it is challenging 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. We then select the best one from h subsets. In general, the larger h is, the better feature subset can be obtained. Note there is a trade-off between the feature subset's quality and the algorithm's running time. However, the feature subset's quality is critical to the performance of the error correction circuit, it is recommended to build as many feature subsets as possible and pick the best one. We present an example to showcase the importance of sampling multiple feature subsets in Fig. 3(b), where we generate 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 and yield different cost function reduction in the end. In the completion of all the iterations, we select the best 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).

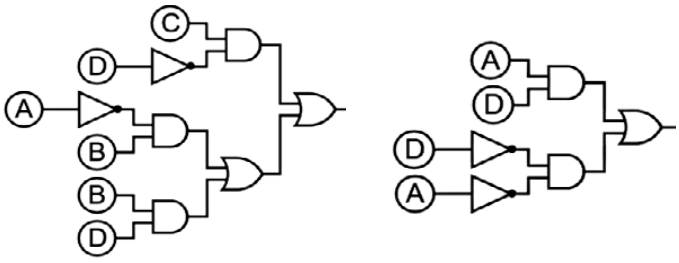
TABLE I: Reduced Truth Table Generation

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

Feature $\mathbf{F}$				Response
ft_A	ft_B	ft_C	ft_D	$\mathbf{CT_i}$
0	0	0	0	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
1	1	0	0	0
1	1	1	1	1

(b) Reduced Truth Table

Feature $\hat{F}$		Response
ft_A	ft_D	ct_i
0	0	$011 \rightarrow 1$
0	1	0
1	0	0
1	1	1



(a). Compensation circuit of orginal truth table

(b). Compensation circuit of reduced truth table

Fig. 4: Compensation circuits corresponding to the truth table.

3) Control Logic Design: We introduce the procedure of generating the reduced truth table for the selected feature subset ct_i in this subsection. This procedure is used in the selection of the feature subset as well as the construction of 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 and Fig. 4 to illustrate the basic idea. Fig. 4 demonstrates the compensation circuits corresponding to the truth tables in Table I. Our ultimate goal is to generate a highly accurate compensation circuit with minimal hardware overhead for all output bits that need to be corrected (i.e., each output bit has a specific control bit ct_i). Thus, we need to reduce the truth table from the original one and find the best features representing most of the output scenarios. Such optimization may cause minor errors, which is inevitable. For example, in the original truth 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_A and ft_D are selected to build a reduced truth table for ct_i. As it can be seen in Table I that when $\{ft_A, ft_D\} = 01$, ct_i is 0, regardless the values of ft_B and ft_C. When $\{ft_A, ft_D\} = 00$, $Pr(1) = \frac{2}{3}$, which is larger than $Pr(0) = \frac{1}{3}$. Therefore, when $\{ft_A, ft_D\} = 00$, we choose ct_i to be 1 (i.e., all values of $\mathbf{CT_i}$ in the original truth table are converted to 1 in the reduced truth table). In this example, one out of six rows in the original true table does not match the reduced truth table (i.e., the first row). Thus, according to Equation (2), we obtain $\mathcal{C}(\{ft_A, ft_D\}) = \frac{1}{6}$. As shown in Fig. 4, the hardware implementation of the reduced truth table is more lightweight than the compensation circuit derived from the original truth table. Every output bit has a control bit ct_i and the corresponding compensation circuit. The output bit is connected with the control bit via an XOR gate to produce the final corrected result.

Algorithm 2: Control Circuit Generation

Input: $\mathbf{P}$, $\tilde{\mathbf{P}}$ and $\mathbf{F}$;
Output: $\mathcal{TT}$ and $\hat{F}$ 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
```

4) 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:

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

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 sub-circuits are determined, the generated compensation circuit is synthesized and connected to the PAC as shown in Fig. 2.

C. Voltage Over-Scaling

Any circuit under VOS can be considered as the PACs. The data samples of PACs and HSPICE files that contain the error information are fed as initial inputs. The overall design flow is similar to ALD and shown in Fig. 1, but the correction update is not required for VOS PACs due to the different feature selection strategies. Next, we explain the detailed procedures.

1) Compensation Scheme: We follow a similar compensation strategy where we employ an XOR gate to correct the output bit of each PAC. However, we propose several non-trivial modifications to fit the VOS design. **First**, as can be seen in Fig. 6, we only consider the error pattern between input and output rather than taking intermediate signals into account. Unlike the truncated PACs that partially trim the

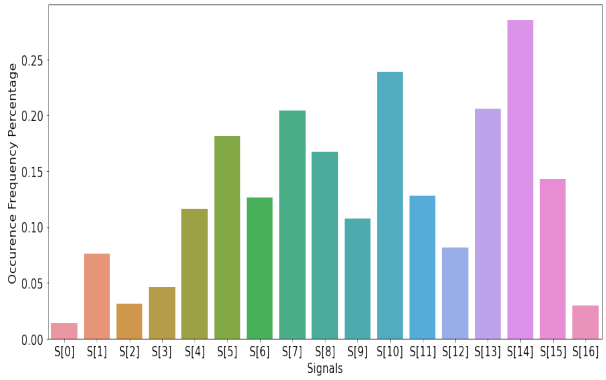


Fig. 5: Error Occurrence Frequency of 16-bit Brent Kung Adder

circuit and break the internal structure, VOS PACs keep the original circuit design, rendering circuitry functionality the same. Therefore, the circuit output primarily relies on the input samples and the interior signals of VOS PACs play a less significant role in influencing the output. **Second**, we compensate a part of error outputs for VOS PACs (following the order of significance of these signals) while correcting all output signals for truncated PACs. Note that truncated PACs usually have fewer error bits (i.e., two least significant bits) while VOS PACs have much more incorrect bits (i.e., all output bits). Therefore, the feature selection and error correction circuit design are more challenging. To accommodate the design change, we propose the Backward Stepwise Selection algorithm to select the most significant features, as described below.

2) Backward Stepwise Selection: The proposed Backward Stepwise Selection (BSS) is the reverse version of the FSS. Recall that in FSS, we start from an empty (null) set and perform the feature selection algorithm to add features to the subset and set the maximum number of selected features as the stopping criteria. In BSS, we begin with a complete feature set (i.e., a group with all input bits), eliminate elements from the collection, and choose the maximum number of remaining features as the stopping criteria. A concrete account of the BSS algorithm is given in Algorithm 3. Note that in BSS, it is important to determine the order of compensated bits in circuitry outputs (i.e., which bit has a higher priority to be compensated given area overhead constraints). Thus, we propose a rule to designate the priority of each bit by assigning different weights to them. In a nutshell, bits with a higher weight (i.e., more important) are always compensated before those with lower weights. The principle of weight assignment is based on the following two observations.

a) *Observation 1:* **Bit significance matters.** The significance of each bit is different. The most significant bit plays a more critical role than the least significant bit in all cases. An error that occurs in the most significant bit renders more severe accuracy degradation and compensation towards such bit is more important. To this end, we assign different weights to different bits to signify their importance, a more influential bit will be compensated ahead of those less important

Algorithm 3: Backward 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 $x \leftarrow 1$ ***to*** h **do**
  let $\hat{F}_{x,L}$ denote the full feature set;
  for $j \leftarrow L$ ***to*** $L-m+1$ **do**
    $ft = \arg\min_{\mathbf{F}} \mathcal{C}(\hat{F}_{x,j} \setminus ft)$;
    $\hat{F}_{x,j-1} \leftarrow \hat{F}_{x,j} \setminus ft$;
  end
end
Select $\hat{F}_{x,L-m+1}$ with lowest $\mathcal{C}$ as $\hat{F}$;
return $\hat{F}$

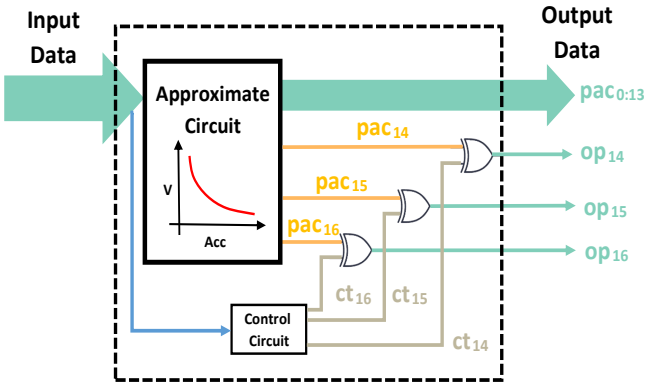


Fig. 6: Compensation Structure for VOS

ones. The weight of each bit can be simply represented as $w = 2^{bit\_position}$.

b) *Observation 2:* **Error occurrence frequency matters.** As shown in Fig. 5, the error occurrence frequency (EOF) of each bit is different. The EOF depends on the propagation delay caused by VOS and varies across different circuits. We compute the error occurrence frequency for each circuit and multiply EOF with the bit weight to determine the priority of bits for compensation. Thus, the final order of compensated bits follows the ranking of $EOF \times w$ of each bit. We show in our experiments that by using the proposed scheme to determine the order of compensation, the proposed method achieves the best performance at the cost of minimal overhead.

3) Control Logic Design & Correction Data Update: We adopt the same approach as for ALD for the logic control design. Given the selected feature subset, we determine the control logic circuit ct_i following Equation (3) and update the corresponding output bit using Equation 4. Once all the control signals are finalized, we synthesize and generate the entire compensation circuit according to Algorithm 2, and connect it to the VOS PAC, as shown in Fig. 6.

IV. Experiments

A. Experimental Setup

Experiment Settings for ALD: We implement the proposed design methodology in Python and employ scikit-learn

[50], a well-known machine learning library, to perform the χ^2 feature selection as our baseline method. We conduct experiments on two real-world datasets, ALYA and GADGET [51], to demonstrate the advantage of the proposed data-driven method for compensation circuit generation. We randomly select 10000 samples (a small fraction of the entire dataset) from ALYA and all 601 samples from GADGET to create input-error patterns. We consider a relatively small sample size for two reasons: 1). The underlying assumption of data-driven approaches is that data are not uniformly distributed and have specific patterns. Thus, a small number of data can reflect the common pattern in the entire dataset; 2). Data collection is costly in real-world scenarios. We attempt to select features that can be well generalized to all data from the small fraction of the dataset. All the circuits including PACs and the final approximate circuits in our experiments are synthesized into netlists using a 32nm technology node. We use the Synopsys Verilog Compiler and Simulator (VCS) to generate the Value Change Dump (VCD) file and obtain features from it. The VCD file is an ASCII-based format file storing circuit signal transitions, where we can extract signal state information. Mean absolute error (MAE) and error rate (ER) are considered as the error metrics to evaluate the performance of the generated approximate logic circuits.

Experiment Settings for VOS: Similarly, the feature selection algorithm is implemented in Python. We employ 16-bit and 32-bit adders from the widely-used approximate computing benchmark AxBench [52] as PACs in the experiment. For each circuit, we randomly generate 4000 (if not specified) samples as input. We synthesize the PACs into netlist using a 32nm technology node first and then use Synopsys IC Validator to convert the source Verilog netlists (.v file) to the HSPICE netlists (.sp file), which is the ASCII file that contains an HSPICE Simulation Deck for circuit simulation. The standard voltage is set to 1.05V while down-scaling to 0.8V for VOS. We perform the simulation and collect circuit errors using Synoposys HSPICE. We evaluate the performance of the compensated approximate circuits in three dimensions: mean squared error (MSE), relative mean error (RME) and average bit-wise error (BWE).

B. Evaluation on ALD PACs

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

We first compare the performance on the feature subset selection algorithm between our modified FSS and χ^2 used in [20]. Fig. 7 demonstrates the cost function trends for the 3rd and 2nd LSBs in the design of a 12-bit approximate multiplier. Both methods achieve better performance than randomly selecting features. However, the χ^2 feature selection may converge to a bad local minimum due to its greedy nature, as showed in Fig. 7(a). In contrast, the proposed modified FSS tends to escape these local minima and achieve lower cost function values by gradually eliminating these features from the candidate pool.

2.) Approximate Fixed-width Multiplier Design

To assess the effectiveness of our proposed method, we generate the compensation circuits for a truncated fixed-width

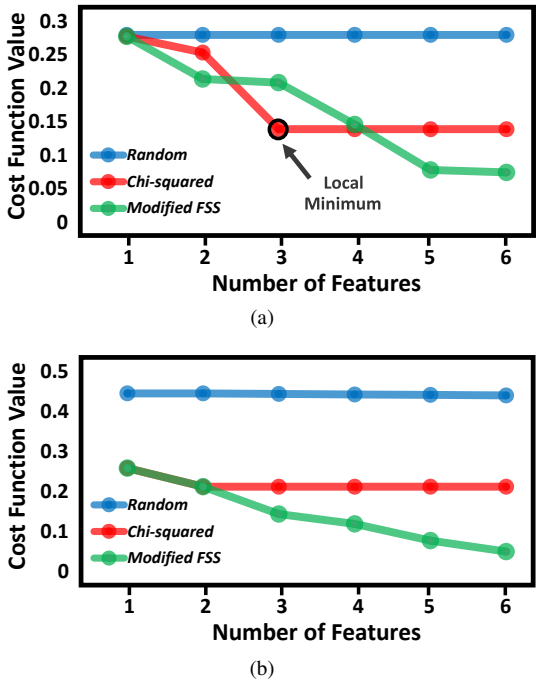


Fig. 7: Comparison between the Modified FSS and χ^2 Feature Selection on a 12-bit Truncated Fixed-width Multiplier: (a) the 3rd LSB, (b) the 2nd LSB

radix-4 booth multipliers under various settings based on three input data: ALYA, GADGET and UNIFORM (uniformly distributed inputs). We use UNIFORM 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 performance over different input data and the corresponding hardware costs of compensated circuits (i.e., power, area consumption and delay) are summarized in Table II and Fig. 8, respectively. "Original" represents the completed multiplier without approximation, "Truncated" represents the approximated multiplier (i.e., the PAC) that partially trims the original circuit, and DD-ALYA, DD-GADGET and DD-UNIFORM are compensated multipliers generated by our proposed method with different data features. PACs usually have low hardware consumption and high error compared to completed circuits. Our target is to improve the accuracy of the PAC with a minimal increment in hardware overhead. It can be seen that the error is significantly reduced from the PAC (Table II) while only introducing small hardware overheads (Fig. 8) using our proposed method. In the design of DD-AYLA, the delay of the compensated circuit would remain unchanged if we constrain the candidate feature pool to the internal nodes close to the primary inputs. However, if we sample multiple feature subsets to find the optimal one by considering all the internal nodes into the candidate feature pool, the delay might slightly increase (e.g., DD-GADGET and DD-UNIFORM) when some of the selected features locate near to the output of the circuit. In this case, the propagation

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

(a) ALYA

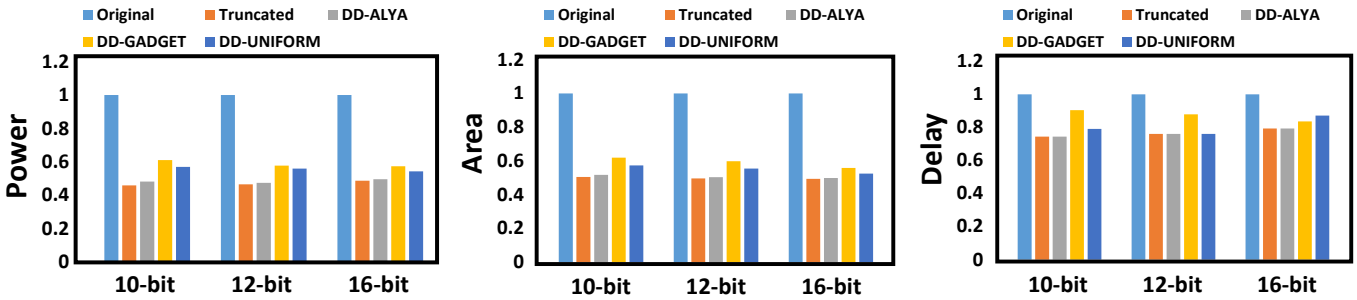
	Truncated Circuits	**DD-ALYA Circuits**	DD-GADGET Circuits	DD-UNIFORM Circuits
10-bit	0.9114 (84.94%)	**0.4613 (39.93%)**	0.5452 (48.32%)	1.4636 (70.55%)
12-bit	0.9242 (85.46%)	**0.4641 (39.63%)**	1.0924 (94.05%)	0.5480 (48.04%)
16-bit	0.9242 (86.27%)	**0.4623 (40.08%)**	1.3041 (93.91%)	1.4492 (70.43%)

(b) GADGET

	Truncated 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

	Truncated 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%)**



(a) (b) (c)

Fig. 8: Hardware Consumption Comparison between Different Compensated Approximate Multipliers: Truncated: multiplier PAC; DD-ALYA: compensated multiplier with ALYA data as features; DD-GADGET: compensated multiplier with GADGET data as features; DD-UNIFORM: compensated multiplier with uniformly distributed data as features. Y axis: normalized hardware consumption to the original circuit. X axis: different input bit widths. All the compensation circuits generated using our approach have much lower hardware overhead than the original multiplier while only introducing trivial power, area, and delay consumption compared to the multiplier PAC.

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 we suggest limiting the candidate feature pool to only internal nodes close to the primary inputs for applications where the delay is a primary concern.

The proposed data-driven methodology shows superior performance on all of the approximate logic circuits in terms of accuracy when tested on data with a similar pattern, which is indicated by the bold numbers in Table II. Our method achieves a 45.68%, 41.21%, and 30.47% accuracy improvement on ALYA, GADGET, and uniformly distributed datasets, respectively. 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 is crucial to consider the input data distribution in the design of approximate logic circuits for data-oriented tasks.

3.) Approximate FIR Filter

We then scale up the circuit complexity to demonstrate the effectiveness of the proposed method on designing larger and more sophisticated approximate circuits, which is one key merit of the proposed data-driven methodology compared to manual analysis. Note that the final output may accumulate to an intolerable error magnitude if we construct a large circuit with approximate arithmetic elements. For instance, we build an approximate 10-bit 4-tap finite impulse response (FIR) filter circuit and replace the multipliers using the manually designed approximate multiplier in [9]. As shown in Table III, the original FIR represents the completed circuit without

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

	Original FIR	[9] FIR	DD-GADGET FIR
Area	5074.94	3640.77	**3859.39**
Power	902.075	682.69	**723.82**
MAE (ER%)	0 (0%)	0.4732 (39.96%)	**0.3013 (23.92%)**

TABLE IV: Compensation Performance on 16-bit Brent Kung and Kogge Stone Adders

Mean Squared Error									
	1bit	**2bit**	**3bit**	**4bit**	**5bit**	**6bit**	**10bit**	**12bit**	**17bit**
BK (VOS)	**MSE = 1.700** ($\times 10^8$)								
BK (Compensated)	1.679	0.983	0.647	0.613	0.613	0.614	0.611	0.611	**0.610**
Perf. Gain	1.28%	42.19%	61.93%	63.93%	63.93%	63.87%	64.07%	64.06%	**64.14%**
KS (VOS)	**MSE = 2.133** ($\times 10^8$)								
KS (Compensated)	2.124	1.383	0.893	0.825	0.818	0.817	0.814	0.814	**0.814**
Perf. Gain	0.41%	35.16%	58.14%	61.33%	61.63%	61.69%	61.82%	61.83%	**61.83%**
Relative Mean Error									
	1bit	**2bit**	**3bit**	**4bit**	**5bit**	**6bit**	**10bit**	**12bit**	**16bit**
BK (VOS)	**RME = 10.25%**								
BK (Compensated)	10.20%	7.87%	4.93%	4.56%	4.48%	4.30%	3.96%	3.94%	**3.94%**
Perf. Gain	0.51%	23.25%	51.91%	55.48%	56.27%	58.06%	61.42%	61.55%	**61.55%**
KS (VOS)	**RME = 18.63%**								
KS (Compensated)	18.60%	16.39%	9.19%	7.78%	7.55%	7.33%	6.69%	6.94%	**6.94%**
Perf. Gain	0.16%	12.01%	50.67%	58.27%	59.45%	60.68%	62.66%	62.74%	**62.75%**
Average Bit-Wise Error									
	1bit	**2bit**	**3bit**	**4bit**	**5bit**	**6bit**	**10bit**	**12bit**	**16bit**
BK (VOS)	**BWE = 1.80**								
BK (Compensated)	1.79	1.70	1.59	1.52	1.48	1.40	0.92	0.69	**0.55**
Perf. Gain	0.68%	5.45%	11.95%	15.36%	18.08%	22.01%	49.17%	61.65%	**69.44%**
KS (VOS)	**BWE = 2.18**								
KS (Compensated)	2.16	2.06	1.85	1.70	1.64	1.56	1.05	0.81	**0.61**
Perf. Gain	1.00%	5.31%	15.06%	22.00%	24.56%	28.44%	51.94%	62.63%	**72.06%**

approximation, the FIR built upon [9] is the approximated FIR without compensation (i.e., the PAC), and DD-GADGET FIR is the compensated FIR using the proposed method. The MAE and ER of the corresponding approximate 4-tap FIR filter get accumulated to 0.4732 and 39.96%, which indicates the drawback of the manual design in building large approximate circuits with approximate arithmetic elements. On the other hand, our proposed data-driven method achieves a significantly better performance to further compensate this approximate FIR filter. Our method yields a design with nearly a 16% error reduction and only a 5.6% hardware overhead compared to the PAC. Also, the area and power consumption are significantly lower than the original FIR. Furthermore, compared to manual design strategies, this proposed systematic method dramatically reduces the design workload..

C. Evaluation on VOS PACs

1.) Brent Kung and Kogge Stone Adders

We first evaluate our proposed method on VOS using 16-bit Brent Kung (**BK**) and Kogge Stone (**KS**) adders from AxBench. To thoroughly assess the effectiveness of our proposed method, we compensate different numbers of bits and compare their performance. For each bit, only ten out of thirty-two features are selected to generate the compensation circuit. As shown in Table IV, "1bit" means only one output bit is compensated while "17bit" means all output bits are compensated (i.e., a 16-bit unsigned adder has 17-bit output). We follow the rule discussed in Section III-C2 to decide the order of compensated bits. For both circuits, we observe the same trend in all three evaluation metrics that the more bits are compensated, the better accuracy we can retain. For example, we can significantly reduce the MSE of the BK adder from 1.7×10^8 to 0.61×10^8, achieving a 64.14% performance gain. More importantly, our approach can reach a good-enough performance by only compensating a small number of bits (e.g., 3-4 bits), which greatly saves the hardware overhead of the compensation circuits. Specifically, it achieves a 63.93% and a 61.33% performance gain on MSE, a 55.48% and a 58.27% performance gain on RME when only four output bits are compensated for BK and KS adders, respectively. Note these circuits are complex, and hence keeping the accuracy of such circuits is challenging under the constraint of a limited hardware budget. Thus, the proposed approach is particularly useful in real-world scenarios where the chip size is always

TABLE V: Compensation Performance on a Larger Circuit (32-bit Brent Kung Adder)

Mean Squared Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**33bit**
BK (VOS)	**MSE = 7.259** ($\times 10^{17}$)								
BK (Compensated)	4.295	4.256	5.190	4.269	4.264	4.262	4.262	4.262	4.262
Perf. Gain	40.83%	43.05%	28.51%	41.19%	41.25%	41.28%	41.28%	41.28%	41.28%
Relative Mean Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**32bit**
BK (VOS)	**RME = 10.21%**								
BK (Compensated)	6.58%	5.81%	6.35%	5.74%	5.72%	5.72%	5.71%	5.71%	5.71%
Perf. Gain	35.54%	43.05%	37.79%	43.78%	43.95%	43.95%	44.01%	44.01%	44.01%
Average Bit-Wise Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**32bit**
BK (VOS)	**BWE = 4.09**								
BK (Compensated)	3.80	3.65	3.58	3.48	3.41	3.28	2.75	1.96	1.42
Perf. Gain	7.11%	10.78%	12.46%	14.91%	16.76%	19.91%	32.83%	52.02%	65.20%

TABLE VI: Compensation Performance on Different Data Size (32-bit Kogge Stone Adder)

Mean Squared Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**33bit**
KS (VOS)	**MSE = 1.938** ($\times 10^{17}$)								
KS (Compensated)	0.024	0.020	0.018	0.018	0.018	0.018	0.017	0.017	0.017
Perf. Gain	98.72%	98.94%	99.08%	99.08%	99.09%	99.10%	99.10%	99.10%	99.10%
Relative Mean Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**32bit**
KS (VOS)	**RME = 3.10%**								
KS (Compensated)	0.23%	0.19%	0.15%	0.16%	0.13%	0.12%	0.11%	0.11%	0.11%
Perf. Gain	92.69%	93.82%	95.31%	94.82%	95.65%	96.04%	96.39%	96.39%	96.39%
Average Bit-Wise Error									
	4bit	**5bit**	**6bit**	**7bit**	**8bit**	**10bit**	**16bit**	**24bit**	**32bit**
KS (VOS)	**BWE = 1.44**								
KS (Compensated)	1.33	1.32	1.31	1.30	1.28	1.24	0.77	0.27	0.22
Perf. Gain	7.63%	8.46%	9.43%	9.85%	11.51%	14.15%	46.60%	81.55%	84.47%

concerned. One last observation from Table IV is that we achieve relatively low-performance gain on BWE when few output bits are compensated compared to its fully compensated counterparts, which is dissimilar to MSE and RME. This is because different bits contribute differently to errors (e.g., the flip of the four most important bits may cause over 70% errors). Therefore, the slopes of the performance gain of MSE and RME are not linearly correlated to that of BWE.

2.) Generalizability

Next, we evaluate the generalizability of the proposed framework to more complex circuits. We scale up the complexity and use a 32-bit Brent Kung adder in this experiment. The results are summarized in Table V. Since the 32-bit adder has more bits to compensate, we increase the minimum number of compensated bits from 1 bit to 4 bits. Note that the number of gates dramatically boosts with the increment of circuit complexity while the candidate features for each output bit only increase from 32 to 64. Moreover, we only select ten out of sixty-four features to generate the compensation circuit to minimize the hardware overhead. Compared to the 16-bit BK adder, the performance merely slightly drops under such strict constraints, indicating strong generalizability to more sophisticated designs. Another observation from Table V is that the proposed method may overcompensate in some rare cases (i.e., "6bit"), rendering a worse performance gain. However, this can be avoided by increasing the number of features.

3.) Different Data Sizes

Data size is a critical factor that affects the effectiveness of our method. In this experiment, we show that it performs particularly well on small data sizes. In the previous assessment, we generate 4000 samples as the input of BK and KS adders, which is a relatively large data size, and it demonstrates decent compensation performance. However, in many real-world scenarios where approximate circuits can be applied, the data size is usually limited to a small number. We down-scale the input data size from 4000 to 1000 and present its capability on a smaller data size in Table VI using a 32-bit KS adder. The proposed approach reveals surprisingly good performance in such a scenario and achieves up to 99.10% and 96.39% performance gain on MSE and RME. The rationale behind this is that the features selected by the feature selection

TABLE VII: Compensation Performance on Different Voltages

KS	MSE ($\times 10^9$)			RME		
	VOS	Compensated	Gain	VOS	Compensated	Gain
0.5V	1.463	1.041	28.85%	64.75%	41.32%	36.18%
0.6V	1.377	0.666	51.63%	63.90%	34.30%	46.32%
0.7V	0.743	0.388	47.48%	57.89%	29.31%	49.37%
0.8V	0.213	0.083	61.33%	18.63%	7.78%	58.27%

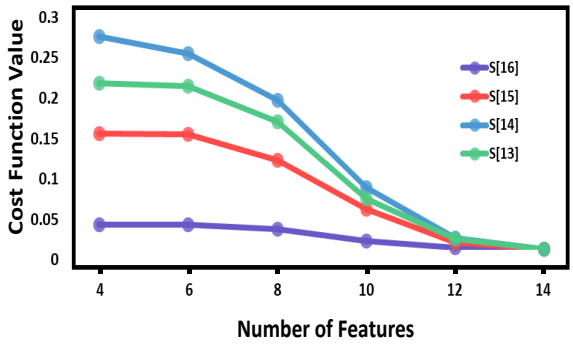


Fig. 9: Loss change over selection of different number of features (KS-16)

algorithm cover the majority of errors caused by the most weighted bits. The results again validate the effectiveness of the Backward Stepwise feature selection design.

4.) Impact of Number of Features

Another factor that affects the compensation performance is the number of features for each output bit. In general, the more features we select, the better performance we can achieve. However, we only pick a small portion of features from the candidate pool (i.e., 10/32 for 16-bit adders and 10/64 for 32-bit adders) due to the overhead constraint. We evaluate the impact of the number of features and present the loss function value change of the four bits with the highest weight under the different number of features in Fig. 9. We conduct the experiment using the 16-bit KS adder and vary the number of features from 4 to 14. The results show that the loss constantly drops with the increasing number of features. Although 10 features already yield a decent loss reduction, there is still a non-trivial improvement from 10 to 14 features. Undoubtedly, we can further improve the performance by selecting more features (e.g., increasing the number of features from 10 to 14). However, there is always a trade-off between the number of features and the area increment of compensation circuits. The Backward Stepwise Selection algorithm in the proposed method consistently selects the most influential features, which significantly improves the efficiency and effectiveness of error compensation circuits while only introducing minimal area overhead.

5.) Different Voltages

We also evaluate the effectiveness of our method when different voltages are applied. We use the KS 16-bit adder as the VOS PAC and compensate for four output bits. The results are summarized in Table VII. It can be seen that the absolute error increase with lower voltage as expected (i.e., 0.5V and 0.6V have larger MSE than 0.7V and 0.8V). Although voltages yield different errors, our proposed method can always reduce the error by a large margin, rendering at least a 28.85% and a 36.18% improvement in MSE and RME, respectively. Note that 0.5V is only 50% of the standard working voltage, which can drastically lower the power consumption of a circuit. Our method achieves a decent error compensation in such an extreme case, indicating good generalizability and effectiveness.

V. Discussion

Since we are the first to consider compensating approximate circuits from the data-driven perspective, we acknowledge the limitations of the proposed method and honestly discuss them in this section. First, like many widely-used optimization techniques in machine learning, such as stochastic gradient descent, our feature selection algorithms have no guarantees of reaching a global minimum. However, we attempt to avoid local minima by running the feature selection algorithm multiple times (e.g., h times) and keeping features that yield the lowest loss, as we show in Algorithm 1 and Algorithm 3. Such a process does not guarantee the global minimum but can effectively result in a better feature subset. In principle, it is straightforward to enumerate all possible feature subsets and compare the loss to find the global optima. Yet the optimization problem is intractable due to the complexity of computation. Thus there is a trade-off between the feature subset's quality and the algorithm's run time. It is recommended to build as many feature subsets as possible depending on the time budget and pick the best one. Additionally, although the proposed method significantly reduces the error rate of PACs, the compensation performance may vary and depends on circuits, voltages, datasets, and feature sets. We encourage future work to improve our baseline.

VI. Conclusions

This paper presented a data-driven feature selection framework of compensation circuit generation for both approximate logic design and voltage over-scaling. We proposed forward stepwise and backward stepwise feature selection algorithms for these two approximate computing paradigms, respectively. The design of the proposed approach and the algorithm analysis are discussed in detail. We extensively evaluate the proposed approach over various circuits and our experimental results show that the proposed approach achieves superior performance, which indeed indicates the need for developing scalable data-driven approximate computing methods.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation award 2047384.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[2] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

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

[4] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 701–706.

[5] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–9.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] G. Karakonstantis, D. Mohapatra, and K. Roy, "System level dsp synthesis using voltage overscaling, unequal error protection & adaptive quality tuning," in *2009 IEEE Workshop on Signal Processing Systems*. IEEE, 2009, pp. 133–138.

[15] R. Hegde and N. Shanbhag, "Soft digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 6, pp. 813–823, 2001.

[16] Y. Liu, T. Zhang, and K. K. Parhi, "Computation error analysis in digital signal processing systems with overscaled supply voltage," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 4, pp. 517–526, 2010.

[17] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6.

[18] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[19] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design*. IEEE, 2011, pp. 409–414.

[20] 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.

[21] 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.

[22] S. Su, C. Zou, W. Kong, J. Han, and W. Qian, "A novel heuristic search method for two-level approximate logic synthesis," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 3, pp. 654–669, 2020.

[23] L. Qiu, Z. Zhang, J. Calhoun, and Y. Lao, "Towards data-driven approximate circuit design," in *2019 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2019, Miami, FL, USA, July 15-17, 2019*. IEEE, 2019, pp. 397–402.

[24] C. Chen, W. Qian, M. Imani, X. Yin, and C. Zhuo, "PAM: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Trans. Computers*, vol. 71, no. 10, pp. 2473–2486, 2022.

[25] Y. Wu and W. Qian, "Alfans: Multilevel approximate logic synthesis framework by approximate node simplification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1470–1483, 2019.

[26] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, "Vecbee: A versatile efficiency-accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[27] C. Meng, X. Wang, J. Sun, S. Tao, W. Wu, Z. Wu, L. Ni, X. Shen, J. Zhao, and W. Qian, "SEALS: sensitivity-driven efficient approximate logic synthesis," in *DAC '22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022*. ACM, 2022, pp. 439–444.

[28] X. Wang and W. Qian, "Minac: Minimal-area approximate compressor design based on exact synthesis for approximate multipliers," in *IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022.*, 2022.

[29] J. Hu and W. Qian, "A new approximate adder with low relative error and correct sign calculation," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*. ACM, 2015, pp. 1449–1454.

[30] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*. ACM, 2016, pp. 128:1–128:6.

[31] R. Zhou and W. Qian, "A general sign bit error correction scheme for approximate adders," in *Proceedings of the 26th edition on Great Lakes Symposium on VLSI, GLVLSI 2016, Boston, MA, USA, May 18-20, 2016*. ACM, 2016, pp. 221–226.

[32] J. Hu, Z. Li, M. Yang, Z. Huang, and W. Qian, "A high-accuracy approximate adder with correct sign calculation," *Integr.*, vol. 65, pp. 370–388, 2019.

[33] Y. Wu, Y. Li, X. Ge, Y. Gao, and W. Qian, "An efficient method for calculating the error statistics of block-based approximate adders," *IEEE Trans. Computers*, vol. 68, no. 1, pp. 21–38, 2019.

[34] C. Lou, W. Xiao, and W. Qian, "Quantified satisfiability-based simultaneous selection of multiple local approximate changes under maximum error bound," in *IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022.*, 2022.

[35] Z. Zhang, R. Wang, Z. Zhang, R. Huang, C. Meng, W. Qian, and Z. Zhou, "Reliability-enhanced circuit design flow based on approximate logic synthesis," in *GLSVLSI '20: Great Lakes Symposium on VLSI 2020, Virtual Event, China, September 7-9, 2020*. ACM, 2020, pp. 71–76.

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

[37] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys, "Pushing the limits of voltage over-scaling for error-resilient applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 476–481.

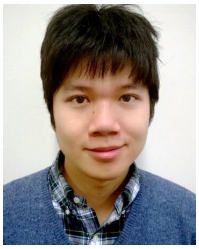
[38] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 957–960. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870926.1871159

[39] C. Zou, W. Qian, and J. Han, "DPALS: A dynamic programming-based algorithm for two-level approximate logic synthesis," in *2015 IEEE 11th International Conference on ASIC, ASICON 2015, Chengdu, China, November 3-6, 2015*. IEEE, 2015, pp. 1–4.

[40] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 504–510.
[41] C. Meng, P. Weng, S. Su, and W. Qian, "Advanced ordering search for multi-level approximate logic synthesis," in *2019 International Workshop on Logic and Synthesis (IWLS), Lausanne, Switzerland, 2019*, 2019, pp. 89–96.
[42] S. Su, Y. Wu, and W. Qian, "Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*. ACM, 2018, pp. 54:1–54:6.
[43] 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.
[44] 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.
[45] M. T. Leipnitz and G. L. Nazar, "High-level synthesis of approximate designs under real-time constraints," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–21, 2019.
[46] G. Zervakis, H. Saadat, H. Amrouch, A. Gerstlauer, S. Parameswaran, and J. Henkel, "Approximate computing for ml: State-of-the-art, challenges and visions," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 189–196.
[47] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.
[48] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 187–192.
[49] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.
[50] 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.
[51] UEABS, "Unified european applications benmark suite," 2013, http://www.prace-ri.eu/ueabs/.
[52] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2016.

Bingyin Zhao is currently a Ph.D. student in the Department of Electrical and Computer Engineering at Clemson University. He received the B.S. degree from East China University of Science and Technology, China, in 2012, and the M.S. degree from Rochester Institute of Technology in 2014. His research interests include robust and trustworthy AI, security of deep neural networks and machine learning for VLSI design.

Ling Qiu is currently a Ph.D. student in the College of Information Sciences and Technology at Pennsylvania State University. He received the B.S. degree from University of Nebraska, Lincoln in 2016 and the M.S. degree from Clemson University in 2019, respectively. He is the recipient of ISCAS Student Travel Award in 2018. His research interests include hardware design of approximate computing and Human-Computer Interaction in Health domains.



Yingjie Lao is currently an assistant professor in the Department of Electrical and Computer Engineering at Clemson University. He received the B.S. degree from Zhejiang University, China, in 2009, and the Ph.D. degree from the Department of Electrical and Computer Engineering at University of Minnesota, Twin Cities in 2015. He is the recipient of an NSF CAREER Award, a Best Paper Award at the International Symposium on Low Power Electronics and Design (ISLPED), and an IEEE Circuits and Systems Society Very Large Scale Integration Systems Best Paper Award. He is currently a member of 4 technical committees in the IEEE Circuits and Systems Society and IEEE Signal Processing Society.