CoOAx: Correlation-aware Synthesis of FPGA-based Approximate Operators

Salim Ullah
Chair of Processor Design, cfaed, Technische Universität Dresden
Dresden, Germany
salim.ullah@tu-dresden.de

Siva Satyendra Sahoo
Chair of Processor Design, cfaed, Technische Universität Dresden
Dresden, Germany
siva_satyendra.sahoo@tu-dresden.de

Akash Kumar
Chair of Processor Design, cfaed, Technische Universität Dresden
Dresden, Germany
akash.kumar@tu-dresden.de

ABSTRACT

The run-time reconfigurability and high parallelism offered by FPGAs make them an attractive choice for implementing hardware accelerators for ML algorithms. In the quest for designing efficient FPGA-based hardware accelerators for ML algorithms, the inherent error-resilience of ML algorithms can be exploited to implement approximate hardware accelerators to trade the output accuracy with better overall performance. As multiplication and addition are the two main arithmetic operations in ML algorithms, most state-of-the-art approximate accelerators have considered approximate architectures for these operations. However, these works have mainly considered the exploration and selection of approximate operators from an existing set of operators. To this end, we provide an efficient methodology for synthesizing and implementing novel approximate operators. Specifically, we propose a novel operator synthesis approach that supports multiple operator algorithms to provide new approximate multiplier and adder designs for AI inference applications. We report up to 27% and 25% lower power than state-of-the-art approximate designs, with equivalent error behavior, for 8-bit unsigned adders and 4-bit signed multipliers respectively. Further, we propose a correlation-aware design space exploration method that can improve the efficacy of randomized search algorithms in synthesizing novel approximate operators.

CCS CONCEPTS

• **Computer systems organization** → **Embedded hardware**; • **Hardware** → **Hardware accelerators**; **Arithmetic and datapath circuits**; *Circuit optimization.*

KEYWORDS

Approximate Computing, Arithmetic Operator Design, Circuit Synthesis, AI-based Exploration

ACM Reference Format:
Salim Ullah, Siva Satyendra Sahoo, and Akash Kumar. 2023. *CoOAx*: Correlation-aware Synthesis of FPGA-based Approximate Operators . In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3583781.3590222

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590222

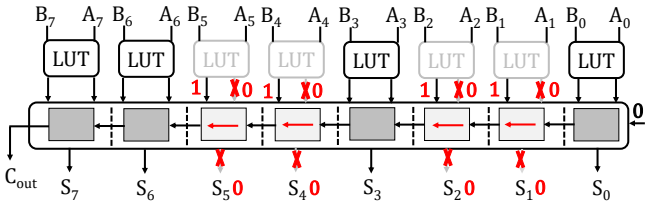


Figure 1: An 8-bit approximate adder from [3]. The disabled elements cannot be used for other operations.

1 INTRODUCTION

The recent developments in the field of Artificial Intelligence (AI) have made Machine Learning (ML) algorithms an attractive choice to perform various tasks, such as computer vision and natural language processing, for embedded systems at the edge. Recent studies have shown that state-of-the-art ML algorithms can outperform human-level output accuracy in many cases [1, 2]. However, these algorithms' high computational, memory, and energy requirements pose a challenge for their deployment on resource-constrained embedded systems at the edge. To this end, one possible option is to utilize the cloud-based execution of these models. Nonetheless, an always cloud-based execution has its own associated limitations, such as the cost, latency, power consumption, and privacy of data transfer over a network. This motivates the exploration of various techniques that can enable high-performance and energy-efficient execution of ML models on the resource-constrained embedded systems at the edge.

A plethora of recent works have focused on the various optimization techniques to reduce the overall computational complexity and memory footprints of ML models to execute them on resource-constrained embedded systems. These techniques mainly exploit the inherent error-resilience of ML models to introduce deliberate approximations at the various layers of the computation stack. The inherent error-resilience of an application enables it to produce acceptable quality outputs despite some approximations (inaccuracies) in the processed data and the associated intermediate computations. For ML algorithms, the commonly explored approximation techniques are model pruning, quantization of trained parameters, and the utilization of approximate arithmetic modules. As Multiply-accumulate (MAC) is the most commonly utilized operation in ML models, such as Deep Neural Networks (DNNs), most of the related state-of-the-art works have focused on proposing various Application-specific Integrated Circuit (ASIC)- and Field Programmable Gate Array (FPGA)-optimized approximate architectures for the multiplication and addition operations [4–13]. However, most of these works, such as [4–6], have focused on proposing one or two different architectures for approximate operators. The

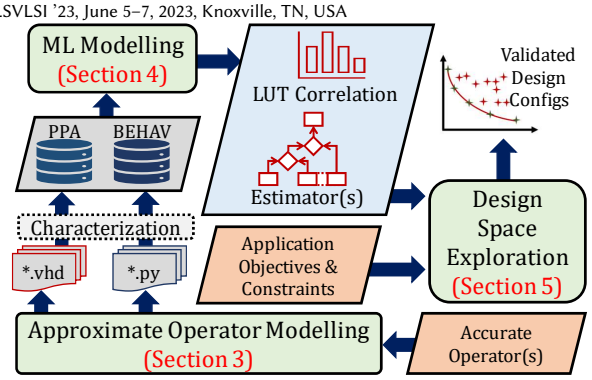


Figure 2: Design Methodology

authors of [10, 13] have focused on presenting libraries of approximate adders and multipliers having different accuracy-performance trade-offs. However, all these works follow an application-agnostic design methodology, and the usefulness of an operator is evaluated by deploying it in various applications. To this end, some works, such as [14], have focused on the exploration of the existing libraries of approximate operators to identify potential feasible operators that may satisfy the accuracy-performance constraints of an error-tolerant application. In general, the state-of-the-art approximate arithmetic operators have two main limitations:

1) These designs do not offer a consistent design methodology for trading accuracy with performance gains. Therefore, in cases where existing approximate operators fail to satisfy an application's accuracy-performance constraints, the process of defining a new approximate architecture is explored again.

2) Most of the state-of-art approximate arithmetic operators follow an application-agnostic design methodology. However, as presented in [3], such a design methodology can result in approximate operators that are not feasible for satisfying an application's accuracy and performance constraints.

To this end, [3] presents a framework that implements various versions of an approximate operator by disabling Lookup Tables (LUTs) and carry chain elements in the corresponding accurate implementation. A disabled LUT does not contribute to the computation of the final output. For example, Figure 1 presents an approximate version of an 8-bit unsigned adder according to the methodology proposed in [3]. As shown in the figure, the disabled LUTs and the associated carry chain elements are represented using light gray outlines and fonts. However, in this method, the disabled elements cannot be utilized for any other operation. For example, the implementation of the approximate adder, shown in Figure 1, on Xilinx 7-series FPGA would still require two complete logic slices[1]. Furthermore, a disabled carry chain element still contributes to routing carries, as shown using the red colored arrows in Figure 1. Therefore, this technique results in an insignificant reduction in the approximate operator's critical path delay and power dissipation compared with the corresponding accurate operator.

To address the aforementioned limitations of state-of-the-art approximate operator architectures, we present *CoOAx*, a framework for synthesizing FPGA-based approximate operators. Figure 2 presents an overview of the proposed framework. The proposed methodology involves circuit-level modeling and novel Design Space Exploration (DSE) methods for fast design of approximate arithmetic operators that can leverage the inherent robustness of error-tolerant applications. We have considered the Multilayer Perceptron (MLP)-based MNIST classification as an error-resilient benchmark application and multipliers as an example operator to discuss our methodology and present our results. However, the proposed methodology is generic and can be used for designing any *soft logic-based* operators for any error-resilient application. Our proposed implementations utilize the 6-input LUTs and associated carry chains of modern FPGAs as building blocks. This article uses the Configurable Logic Blocks (CLB) architecture of Xilinx FPGAs [15] to present the proposed approximate operator modeling methodology and the corresponding results. Our novel contributions include:

(1) *A systematic methodology for approximate operators generation:* Our proposed methodology utilizes the 6-input LUTs and the associated carry chains of FPGAs to implement approximate operators according to input configuration. For instance, the input configurations—a binary string—identify the LUTs, in an accurate operator implementation, that should be truncated (removed) to realize a corresponding approximate operator. For example, for an M × N accurate multiplier, utilizing 'L' LUTs, our methodology provides 2^L approximate multipliers with different accuracy and performance parameters.

(2) *An efficient DSE methodology:* We utilize various ML models to propose a correlation-based DSE methodology. Specifically, we use the correlation between the implemented LUTs and the performance metrics from a set of characterized designs to report improvements over randomized search-based methods.

The rest of the article is organized as follows. Section 2 provides a brief overview of related approximate operators. Section 3 describes the systematic modeling methodology used for designing arbitrary approximate multipliers. Section 4 presents the methodology adopted for designing ML-based performance estimators. The proposed DSE approach for designing approximate multipliers is described in Section 5. In Section 6, the results from the experimental evaluation of the proposed framework are presented, followed by a discussion on the scope of related future research in Section 7.

2 APPROXIMATE MULTIPLIERS

The authors of [8] and [9] have proposed 2 × 2 approximate unsigned multiplier architectures for ASIC-based systems. These architectures can be used to implement higher-order approximate multipliers using the modular design approach. The authors of [10] have utilized various existing approximate adders and multiplier architectures to implement a library of 8 × 8 unsigned approximate multipliers denoted as *EvoApprox*. In their follow-up work in [11], this library was extended to incorporate various signed multiplier designs. The authors of [3, 6, 12, 13] have focused on utilizing the structure of 6-input LUTs to propose various approximate multiplier architectures for FPGA-based systems. The work presented in [12] has proposed an approximate unsigned 4 × 2 multiplier architecture. This design completely utilizes all the 6-inputs of a LUT. The approximate 4 × 2 design is then used to implement 4 × 4 and other higher-order multipliers. The work presented in [13] proposes three unsigned approximate multiplier architectures. These designs

[1] A logic slice in Xilinx 7-series FPGA provides 4 LUTs [15].

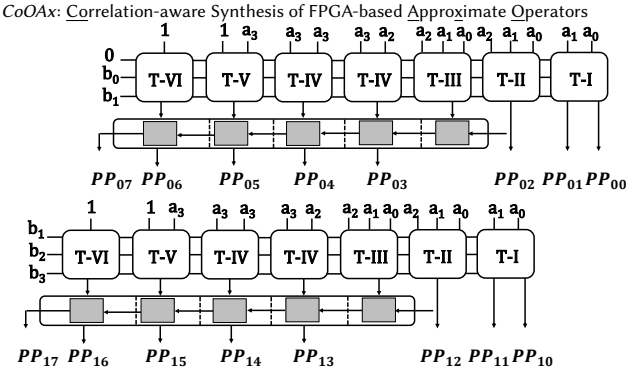


Figure 3: LUT mapping of partial products for 4×4 accurate signed multiplier architecture [17]

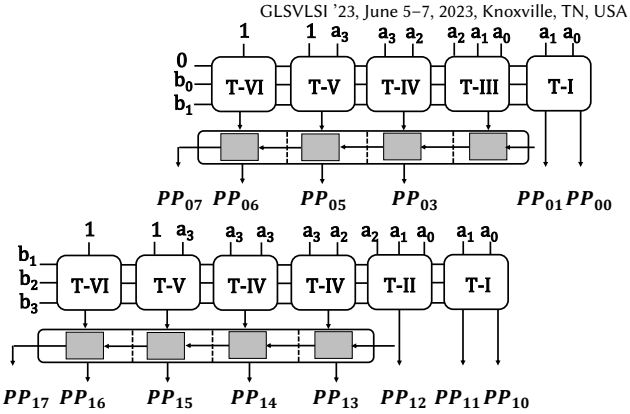


Figure 4: 4×4 approximate multiplier design according to input configuration '1101101101'. PP_{02} and PP_{04} are truncated to 0.

are based on the parallel generation of all product bits to reduce the critical path delay of the implementations. The authors of [6] have focused on signed operations and have proposed a radix-4 Booth's algorithm-based approximate multiplier architecture. The work presented in [3] has proposed a framework to implement various FPGA-optimized approximate versions of an accurate operator by utilizing various ML models and multi-objective optimization techniques to identify approximate operator configurations that satisfy an application's accuracy-performance constraints. The framework implements the various versions of an approximate operator by disabling LUTs and carry chain elements in the implementation. The authors of [16] have also proposed a technique to implement ASIC-based approximate operators for Artificial Neural Networks (ANNs). In their proposed technique, they have used Cartesian Genetic Programming (CGP) and 2-input logic gates for implementing approximate operators. However, in this technique, the various performance parameters, such as critical path delay and energy, of the approximate operators are not considered while designing an operator. The accuracy of the application (ANN) is the only evaluation criterion for selecting an operator.

3 OPERATOR MODELLING

CoOAx proposes a systematic methodology for designing approximate arithmetic operators from the corresponding accurate implementations of the operators. In this work, we have used the accurate, signed multiplier implementation presented in [17] to demonstrate the proposed approximate operator modeling technique. However, the proposed methodology for approximation is equally applicable to other multiplication algorithms and operators. As shown in Figure 3, the architecture presented in [17] utilizes five different LUT configurations to generate the signed partial products. In this implementation, LUT configurations *T-V* and *T-VI* are used to compute and transfer the sign of each partial product row. The generated partial products are added together to compute the final product. Our proposed approximation methodology is based on the truncation of LUTs (and the corresponding carry chain cell) in the accurate implementation to generate approximate designs. For this purpose, we have utilized an $L-bit$ string (denoted as input-configuration) to address the LUTs in every partial product row. However, LUTs *T-V* and *T-VI* are excluded from this $L-bit$ input-configuration to compute the sign of each approximate partial product row with high accuracy. For example, to address the LUTs in the two partial product rows of the 4×4 multiplier shown in Figure 3, the proposed approximation methodology utilizes a $10-bit$ string. The least significant $5-bits$ in the input-configuration correspond to the first partial product row, and the most significant $5-bits$ denote the second partial product row. A '0' at any location in the *L-bit* string represents the truncation of the corresponding LUT. The truncation of a LUT denotes that the LUT does not compute the logic associated with it in the accurate implementation, and the corresponding output of the LUT (or carry chain cell) is truncated to '0'. For example, Figure 4 shows the generation of approximate partial products for input-configuration '1101101101'. The configuration shows that truncation of two and one LUTs in the first and second partial product rows, respectively. For example, the LUT producing PP_{02} is truncated to zero. Similarly, the LUTs *T-III* (utilized for computing the input carry) in the second partial product row is also removed. Compared to the accurate multiplier architecture presented in Figure 3, the approximate multiplier implementation in Figure 4 results in saving three LUTs and two carry chain elements[2]. Since an accurate 4×4 multiplier utilizes 10 LUTs for the partial product generation, therefore, the proposed approximation methodology supports 2^{10} approximate multipliers. Similarly, for an accurate 8×8 multiplier utilizing 36-LUTs for partial product generation, *CoOAx*'s proposed approximation methodology provides 2^{36} different approximate multipliers with different accuracy and implementation performance metrics.

4 ML MODELLING FOR PERFORMANCE ESTIMATION

Given the operator model described in Section 3, any arbitrary arithmetic operator implementation in FPGAs can be represented by the ordered tuple $O_i(l_0, l_1, ..., l_l, ..., l_{L-1}), \forall l_l \in \{0, 1\}$. The term l_l represents whether the LUT corresponding to the operator's accurate implementation is being used (1) or not(0) and L represents the total number of LUTs of the accurate implementation that may be removed to implement approximation. So, the accurate implementation can be represented as $O_{Ac}(1, 1, ..., 1)$. Similarly, $O = \{O_i\}$ represents the set of all possible implementations of the

[2]A carry chain element consists of multiple cells. For example, a carry chain element in Xilinx 7-series FPGAs is $4-bit$ wide [15], whereas it is $8-bit$ wide in Xilinx Ultrascale architecture [18].

operator. Any operator/application's behavior can be represented as a function $\mathcal{S}$. So the output of the operator/application for a set of inputs can be abstracted as shown in (1). The term Err_{O_i} represents the error in the operator/application's behavior as a result of using an approximate operator O_i compared to using the accurate operator O_{Ac}. Similarly, the operator/accelerator's hardware performance can be abstracted as a set of functions as shown in (2).

While (2) represents complex relationships between O_i and the Power-Performance-Area (PPA) and behavioral accuracy (BEHAV) metrics, ML-based *proxy-functions* can be used as estimators for each metric. We represent such estimators for any arbitrary PPA and BEHAV metric as shown in (3). We used AutoML [19] to explore across different ML algorithms to find the appropriate ones for each metric. The selection of such algorithms was based on their Root Mean Squared Error (RMSE) and R^2 score for both training and test datasets, which is obtained from true characterization of randomly selected O_i configurations. In addition we generated the correlation coefficient between the metrics and the LUT instances of the operator. These coefficients can be represented as shown in (4).

$$\begin{aligned} Out_{O_i} = \mathcal{S}(O_i, Inputs);\ Out_{O_{Ac}} = \mathcal{S}(O_{Ac}, Inputs) \\ Err_{O_i} = Out_{O_{Ac}} - Out_{O_i} \end{aligned} \tag{1}$$

$$\begin{aligned} Power\ Dissipation: \quad \mathcal{W}_{O_i} = \mathcal{H}_W(O_i, Inputs) \\ LUT\ Utilization: \quad \mathcal{R}_{O_i} = \mathcal{H}_R(O_i) \\ Critical\ Path\ Delay: \quad C_{O_i} = \mathcal{H}_C(O_i) \\ Power\ Delay\ Product: \quad PDP_{O_i} = \mathcal{W}_{O_i} \times C_{O_i} \\ PDPLUT_{O_i} = \mathcal{W}_{O_i} \times \mathcal{R}_{O_i} \times C_{O_i} \end{aligned} \tag{2}$$

$$\begin{aligned} PPA\ Metric: \quad PPA_{O_i} = \hat{P}_{ML}(O_i) \\ BEHAV\ Metric: \quad BEHAV_{O_i} = \hat{B}_{ML}(O_i) \end{aligned} \tag{3}$$

$$\begin{aligned} Corr_{PPA} = \left\{ \underset{O_i \in O_{train}}{correlation}\left(\{PPA(O_i)\}, l_l\right) \right\} \\ Corr_{BEHAV} = \left\{ \underset{O_i \in O_{train}}{correlation}\left(\{BEHAV(O_i)\}, l_l\right) \right\} \\ \forall\ l_l \in \{l_0, ..., l_l, ..., l_{L-1}\} \end{aligned} \tag{4}$$

5 DSE METHODOLOGY

With the proxy function representation shown in (3), the corresponding unconstrained multi-objective optimization problem can be represented as shown in (5). In our current work, we used a randomized search for the DSE problem. This involves generating random samples of O_i and evaluating them using the estimators to generate a set of Pseudo Pareto Front (PPF) design points as candidate solutions for true characterization. The results from the characterization are again filtered to generate the Validated Pareto-front (VPF) designs. While a purely randomized search assumes uniform probability across all (removable) LUTs in the implementation, we use the correlation values, shown in (4), to generate a probability distribution across the LUTs, l_l. Figure 5 shows the

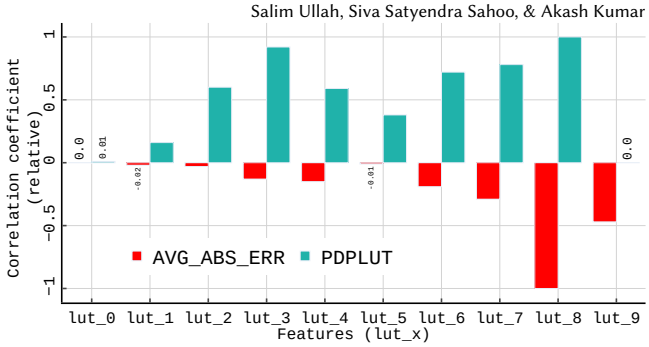


Figure 5: Comparison of relative correlation coefficient of all the LUTs across all approximate signed 4×4 multipliers using Baugh-Wooley's algorithm and proposed operator modelling

relative correlation coefficient of $PDP \times LUT$ (PDPLUT) and the average absolute relative error (AVG_ABS_REL_ERR) across the ten removable LUTs of a 4×4 signed multiplier. As evident from the figure, assuming a uniform probability does not account for the varying impact of each LUT's usage in the design on the PPA and BEHAV metrics. We use a weighted sum of the PPA and BEHAV correlation coefficients for determining the probability distribution, as shown in (6), while generating the random samples. Varying the relative weights for $Corr_{PPA}$ and $Corr_{BEHAV}$ allows us to implement varying prioritization of PPA and BEHAV metrics while generating the candidate design points for characterization.

$$\underset{O_i \in O}{\text{minimize}}(BEHAV_{O_i}, PPA_{O_i}) \tag{5}$$

$$\begin{aligned} Corr_{SUM}(l_l) = wt_B \times Corr_{BEHAV}(l_l) + wt_P \times Corr_{PPA}(l_l) \\ Prob(l_l) = \underset{MinMax}{scale}\ Corr_{SUM}(l_l) \\ where,\ l_l \in \{l_0, ..., l_l, ..., l_{L-1}\} \\ and,\quad wt_B + wt_P = 1;\quad wt_B,\ wt_P \in [0, 1] \end{aligned} \tag{6}$$

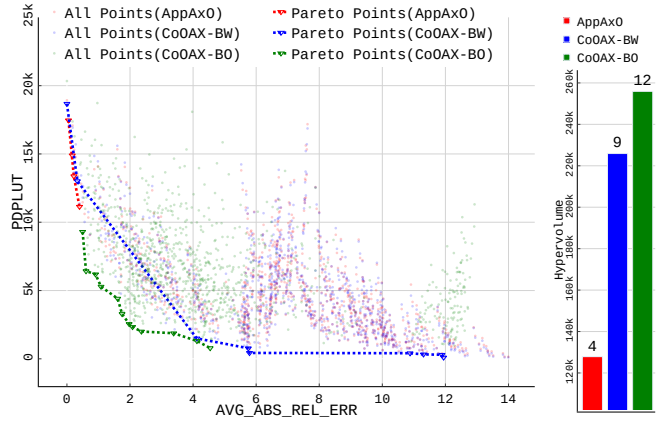
6 EXPERIMENT AND RESULTS

6.1 Experimental Setup

We have utilized VHDL to implement the proposed approximate operator modeling technique. All the presented multipliers have been synthesized for the $7VX330T$ device of the Virtex-7 family using Xilinx Vivado 19.2. We report design metrics for critical path delay (CPD)-optimized implementations and use Vivado Simulator and Power Analyzer tools to calculate the dynamic power. The benchmark application (MLP) accelerator has been implemented for the Zynq UltraScale+ MPSoC ($xczu3eg - sbva484 - 1 - e$ device). All the behavioral models and the multi-objective optimization-based DSE have been implemented in Python using multiple packages, such as scikit-learn and PyGMO.

6.2 Approximate Operator Modelling Analysis

6.2.1 Approximate Adders. With the proposed framework, the synthesis of novel approximate adders does not require complex DSE methods owing to the low LUT utilization for each adder. For instance, an accurate 8-bit unsigned adder uses 8 LUTs and can only result in 2^8 approximate designs. We used an exhaustive search



(a) (b)

Figure 6: Pareto-front analysis of signed 4×4 multiplier. (a) Pareto-front with PDPLUT and average absolute relative error. (b) Comparison of hypervolume and number of design contributions to the combined Pareto-front.

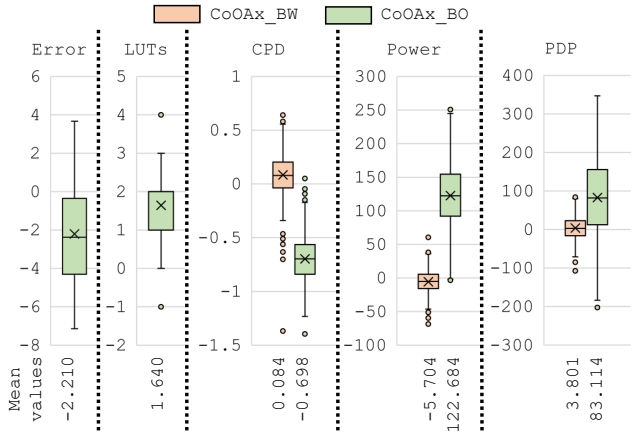
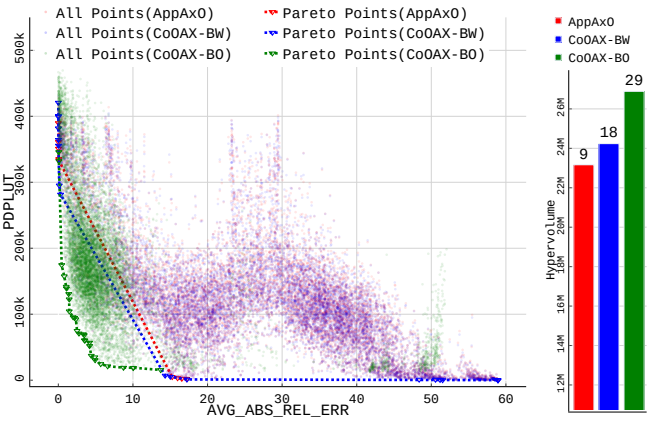


Figure 7: Increase in AVG_ABS_REL_ERR and PPA metrics across all approximate signed 4×4 multipliers for *CoOAx*_BW and *CoOAx*_BO compared to *AppAxO*[3]

for synthesizing novel approximate adders. Compared to *AppAxO*, the *CoOAx*-based 8-bit unsigned adders used up to 2 (66.7%) lesser carry chains (CCs), 101.14uW (27.1%) lower power, 0.31nS (22.5%) lower CPD. Overall, the *CoOAx*-based adders result, on average, 246.97(32.09%) lower PDP × CC than *AppAxO*-based adders. It must be noted that the same approximate configuration results in an equivalent error and LUT utilization for both *CoOAx* and *AppAxO*.

6.2.2 Approximate Multipliers. To evaluate the efficacy of the generated approximate arithmetic operators, Figure 6 presents an accuracy-performance analysis of 4×4 approximate multipliers. For this experiment, our proposed approximate operator modeling technique utilizes radix-4 Booth's and Baugh-Wooley's algorithms [20, 21] to implement approximate 4×4 multipliers denoted as *CoOAx-BO* and *CoOAx-BW*, respectively. These designs are compared with the 4×4 approximate multipliers generated by the AppAxO framework [3]. It should be noted that both the *AppAxO* framework and

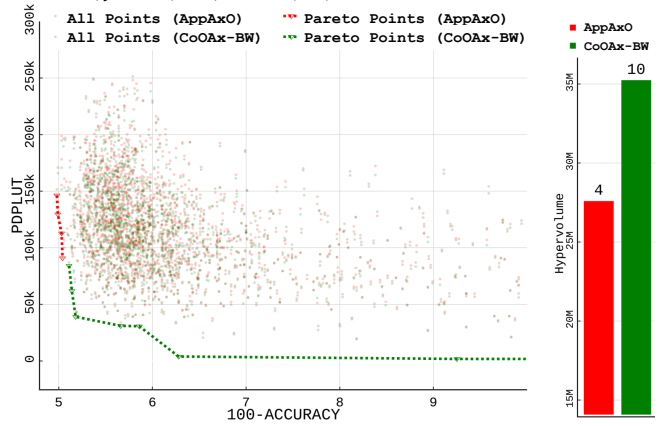


(a) (b)

Figure 8: Pareto-front analysis of signed 8×8 multiplier (a) Pareto-front (b) Comparison of hypervolume and number of pareto-point contributions

our proposed operator modeling methodology utilize a L-bit string to address the LUTs in an accurate $N \times N$ multiplier. Therefore, each of the three architectures in Figure 6 contributes 1024 distinct approximate designs. For the analysis, we have used AVG_ABS_REL_ERR and PDPLUT as the BEHAV and PPA metrics, respectively The analysis presented in Figure 6 utilizes two commonly utilized metrics, i.e., the total number of non-dominated design points and the significance of these points, known as the hypervolume indicator [22]. The bar plots in Figure 6(b) show the individual hypervolume contribution of the non-dominated design points by each technique, and the number above the plot represents the number of contributed non-dominated design points in the combined Pareto front analysis.

The results in Figure 6 show that compared to the four non-dominated design points provided by *AppAxO*, our proposed approximate operator implementation technique provides 9 and 12 approximate multipliers. Moreover, the hypervolume contribution of our proposed designs is also more than that of *AppAxO*'s designs. The non-dominated accurate multiplier configuration '1023' (binary value '1111111111') utilizes all LUTs for the partial product generation and is provided by *CoOAx-BW* architecture. Furthermore, compared to Baugh-Wooley's algorithm-based non-dominated design points, Booth's algorithm-based designs have a lower average absolute relative error. Interestingly, some non-dominated input-configurations, such as 128 and 512, provided by *CoOAx*, have utilized only a single LUT for partial product generation. Furthermore, compared to *AppAxO*'s non-dominated design points, *CoOAx* non-dominated design points provide better coverage of the whole design space. Figure 7 shows the distribution of the increase in the AVG_ABS_REL_ERR and the PPA metrics for all 1023 signed 4×4 multiplier designs, using the proposed operator modelling, over that proposed in *AppAxO* [3]. The *CoOAx*-BW shows average reduction of nearly 5.7 μW of power with marginally higher CPD. The error and LUT utilization comparison are not shown for *CoOAx*-BW as they are same as that in *AppAxO*. The *CoOAx*-BO model shows average reduction in error and CPD with increased average power dissipation and LUT utilization.



(a) (b)

Figure 9: Pareto-front analysis of signed 8×8 multiplier. (a) Pareto-front with PDPLUT and MNIST classification error (b) Comparison of hypervolume and number of pareto-point contributions to the combined pareto-front

Figure 8 compares the accuracy-performance trade-offs of various 8×8 approximate multipliers. In this experiment, each architecture (*CoOAx-BO*, *CoOAx-BW*, and *AppAxO*) utilizes an identical 36-bit string to identify the LUTs in the corresponding accurate multiplier implementation. Utilizing the 36-bit string, we generated 10650 configurations (random and patterned) and utilized them to implement corresponding approximate multipliers using each technique. The results in Figure 8 show that the *CoOAx*'s multiplier implementations mainly contribute to the non-dominated design points. Compared to the 9 Pareto designs provided by *AppAxO*, *CoOAx* provides 18 and 29 non-dominated design points. Similar to Figure 6, the non-dominated *CoOAx-BO* designs have better output accuracy, and the non-dominated *CoOAx-BW* designs offer reduced $PDP \times LUT$ values.

6.3 Application-level Analysis of Approximate Operators

Figure 9 presents the application-level accuracy-performance analysis of *CoOAx*-generated approximate multipliers. To demonstrate the efficacy of *CoOAx*'s operator modeling, we have evaluated the 3071 approximate 8×8 multiplier configurations utilized in *AppAxO* [3] for MNIST classification. Furthermore, we have considered *CoOAx-BW*- and *AppAxO*-based designs only for the analysis[3]. For this experiment, we have implemented a lightweight MLP in Python to classify the MNIST dataset [23]. The MLP consists of two hidden layers having 100 and 32 neurons, respectively. We have considered only the last layer of the MLP to evaluate the impact of various approximate multipliers on the output accuracy for the 10,000 testing images dataset. For this purpose, the trained floating-point weights and the input activations to the last layer are quantized to 8-bit fixed-point representation. The results in Figure 9 compare the performance ($PDP \times LUT$) and accuracy ($100 - accuracy$) of the MLP. The results show that *AppAxO*-based designs contribute to only four non-dominated design points, whereas, *CoOAx-BW*-based designs result in ten non-dominated design points. An interesting observation is that three *CoOAx-BW* non-dominated design points have configurations where only a single LUT is utilized for partial product generation. The significance of the *CoOAx*'s non-dominated designs is also more than that of the *AppAxO*'s designs, as shown by the respective hypervolume contributions of the points. *CoOAx*'s approximate operator modeling technique provides better resource utilization and performance than *AppAxO*-generated approximate operators.

[3] *AppAxO*'s designs presented in [3] utilize only Baugh-Wooley's algorithm for approximate multipliers implementation.

Table 1: Comparison of prediction accuracy of XGBoost models for PPA (PDP x LUT) and BEHAV (AVG_ABS_REL_ERR) for signed 8×8 multipliers using different operator algorithms

Metric	Operator Model	MSE (TRAIN)	MSE (TEST)	R2 (TRAIN)	R2 (TEST)
PPA	AppAxO	65431521.46	182850873.8	0.985	0.948
	CoOAx-BW	59016613.75	165053592.9	0.989	0.963
	CoOAx-BO	87545742.93	220018017.6	0.982	0.947
BEHAV	AppAxO	0.157	0.245	0.999	0.998
	CoOAx-BW	0.157	0.245	0.999	0.998
	CoOAx-BO	2.461	3.342	0.891	0.879

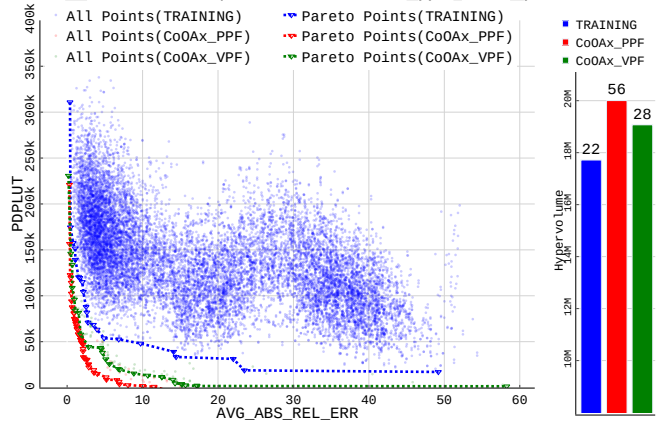
6.4 ML-based Estimator Performance

For the current work, we used XGBoost-based models for the estimation of PPA and BEHAV metrics. 6000 random configurations were used for generating the modelling data. Table 1 shows the prediction accuracy metrics (regression) for each of the model used in operator-level DSE. As seen in the table, for both PPA and BEHAV, the *CoOAx-BO* predictions are the worst, while the *AppAxO* and *CoOAx-BW* models exhibit similar and much better prediction statistics. The larger Mean Square Error (MSE) values of PDPLUT can be attributed to the relatively larger magnitude of those values.

6.5 Correlation-aware DSE Analysis

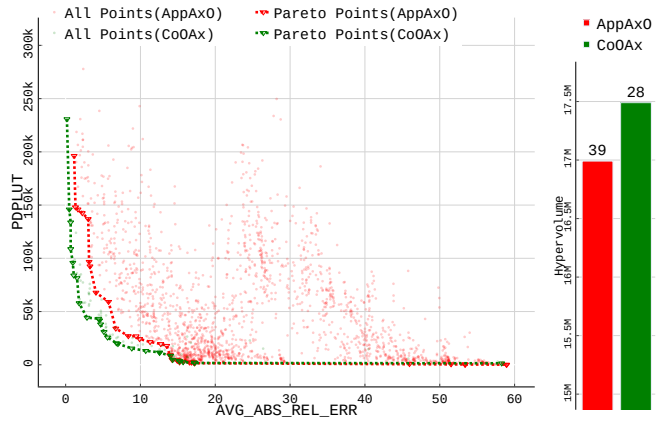
6.5.1 Searching for novel approximate multipliers. Figure 10 shows the Pareto front analysis of the results obtained from the proposed correlation-aware DSE method for a signed 8×8 multiplier. The figure shows three set of design points: the training data obtained from 6000 purely random approximate configurations, the PPF obtained from the DSE across the three operator models—*AppAxO*, *CoOAx-BW*, *CoOAx-BO*—and, the VPF obtained from the implementation of the PPF design configurations. The PPF reports rather optimistic results and the resulting VPF has lower hypervolume. However, using the proposed DSE method, we see significant improvement in the hypervolume with the synthesis and implementation of just 97 additional approximate multiplier designs.

6.5.2 Comparing DSE results with State-of-the-art. As was shown in Figure 9, adopting the proposed operator modelling for the results of *AppAxO* can lead to improved accuracy-performance trade-offs. Similarly, Figure 11 shows the Pareto front and hypervolume comparison of the implementation results of new design points with *AppAxO* and *CoOAx* (across all operator models). It must be noted that the *AppAxO* results correspond to 1927 new design points suggested from DSE, compared to just 93 newly discovered design



(a) (b)

Figure 10: Pareto-front analysis of signed 8×8 multiplier designs from training and correlation-aware DSE results (a) Pareto-front (b) Comparison of hypervolume and number of design points on each pareto-front.



(a) (b)

Figure 11: Pareto-front analysis of signed 8×8 multiplier designs from training and correlation-aware DSE results (a) Pareto-front (b) Comparison of hypervolume and number of pareto-point contributions

points using the correlation-aware randomized search. As can be seen from the figure, *CoOAx* clearly performs better than *AppAxO*, owing largely to improved operator models and additional operator algorithms.

7 CONCLUSION

This paper presents our *CoOAx* framework for synthesizing novel approximate operators. *CoOAx*'s operator modeling methodology utilizes a binary string to specify the 6-input LUTs in an accurate operator implementation. This binary string identifies the LUTs that should be removed from intermediate computation to implement various approximate versions of an accurate operator. *CoOAx*'s DSE methodology utilizes various ML models and a multi-objective optimization technique to generate configurations that provide efficient trade-offs between output accuracy and performance. Our experimental results show that compared to the state-of-the-art designs, the approximate operators generated by *CoOAx* provide more non-dominated design points with better hypervolume contribution for both operator-level designs and for AI inference application. The proposed correlation-aware search technique improves over purely randomized search and can be combined with other state-of-the-art approaches for efficient DSE.

ACKNOWLEDGEMENT

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) under the X-ReAp project (Project number 380524764).

REFERENCES

[1] O. Russakovsky, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
[2] P. Tschandl, et al. Comparison of the accuracy of human readers versus machine-learning algorithms for pigmented skin lesion classification: an open, web-based, international, diagnostic study. *The Lancet Oncology*, 20(7):938–947, 2019.
[3] S. Ullah, et al. AppAxO: Designing Application-specific Approximate Operators for FPGA-based Embedded Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.
[4] M. Shafique, et al. A low latency generic accuracy configurable adder. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.
[5] V. Gupta, et al. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2012.
[6] S. Ullah, et al. Area-optimized accurate and approximate softcore signed multiplier architectures. *IEEE Transactions on Computers*, 70(3):384–392, 2021.
[7] B. S. Prabakaran, et al. Demas: An efficient design methodology for building approximate adders for fpga-based systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 917–920. IEEE, 2018.
[8] P. Kulkarni, et al. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351, 2011.
[9] S. Rehman, et al. Architectural-space exploration of approximate multipliers. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2016.
[10] V. Mrazek, et al. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 258–261, 2017.
[11] V. Mrazek, et al. Libraries of approximate circuits: Automated design and application in cnn accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):406–418, 2020.
[12] S. Ullah, et al. High-performance accurate and approximate multipliers for fpga-based hardware accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.
[13] S. Ullah, et al. SMApproxlib: Library of FPGA-based approximate multipliers. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.
[14] V. Mrazek, et al. AutoAx: An Automatic Design Space Exploration and Circuit Building Methodology Utilizing Libraries of Approximate Components. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.
[15] Xilinx. 7 Series FPGAs Configurable Logic Block: User Guide. https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB, 2016.
[16] V. Mrazek, et al. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, New York, NY, USA, 2016. Association for Computing Machinery.
[17] S. Ullah, et al. Energy-efficient low-latency signed multiplier for fpga-based hardware accelerators. *IEEE Embedded Systems Letters*, 13(2):41–44, 2020.
[18] Xilinx. UltraScale Architecture Configurable Logic Block. https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb, 2017.
[19] A. Płońska et al. Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3, 2021.
[20] A. D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.
[21] C. R. Baugh et al. A two's complement parallel array multiplication algorithm. *IEEE Transactions on computers*, 100(12):1045–1047, 1973.
[22] E. Zitzler, et al. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 862–876. Springer, 2007.
[23] Anonymous. MNIST-cnn. https://github.com/integeruser/MNIST-cnn, 2021.