CLAppED: A Design Framework for Implementing Cross-Layer Approximation in FPGA-based Embedded Systems

Salim Ullah, Siva Satyendra Sahoo, Akash Kumar
Center for Advancing Electronics Dresden (CfAED), Technische Universität Dresden, Germany
(salim.ullah, siva_satyendra.sahoo, akash.kumar)@tu-dresden.de

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586260

***Abstract*—With the rising variation and complexity of embedded workloads, FPGA-based systems are being increasingly used for many applications. The reconfigurability and high parallelism offered by FPGAs are used to enhance the overall performance of these applications. However, the resource constraints of embedded platforms can limit the performance in multiple ways. In recent years, *Approximate Computing* has emerged as a viable tool for improving the performance by utilizing reduced precision data structures and resource-optimized high-performance arithmetic operators. However, most of the related state-of-the-art research has mainly focused on utilizing approximate computing principles individually on different layers of the computing stack. Nonetheless, approximations across different layers of computing stack can substantially enhance the system's performance. To this end, we present a framework to enable the intelligent exploration and highly accurate identification of the feasible design points in the large design space enabled by cross-layer approximations. Our framework proposes a novel polynomial regression-based method to model approximate arithmetic operators. The proposed method enables machine learning models to better correlate approximate operators with their impact on an application's output quality. We use a 2D convolution operator as a test case and present the results for FPGA-based approximate hardware accelerators.**

***Index Terms*—Approximate Computing, Embedded Systems, Cross-layer System Design, FPGA, High-level Synthesis**

I. Introduction

The paradigm of Approximate Computing has shown promising capabilities for designing energy-efficient computing systems for error-resilient applications. A broad spectrum of applications in the domain of computer vision, data mining, and machine learning possess an intrinsic error-resilience to the inexactness of the computing algorithms, their corresponding implementations, and the data being processed. The error-tolerant elements of these applications are also, on average, the main contributors to the overall resource utilization, critical path delay, and energy consumption of the application [1]. The approximate computing paradigm leverages the error-resilience of these applications by trading the output accuracy of an application to realize computing systems with better resource utilization, performance, and energy efficiency [2].

The approximate computing paradigm encompasses different layers of the computation stack. For example, loop perforation [3], precision scaling [4], and utilization of inexact hardware blocks [5] are the commonly investigated techniques at the software, architecture, and circuit levels, respectively. At any layer of the computation stack, the deployed approximation techniques have a limited impact on the resulting output quality of the application and the corresponding performance gains of the implementation. However, depending upon the application, a specific computational layer, or a combination of layers, may prove a better candidate for approximations than other layers in the computation stack. For example, consider a 3×3 convolution kernel sliding over a 5×5 image in Fig. 1. The sliding step size of the kernel is defined by the *Stride* parameter of the convolution operation. For each position of the kernel, nine element-wise multiplications followed by an addition are performed to compute a single value of the output image. For $Stride = 1$, a total of 81 multiplications are performed to compute a 3×3 output image. As convolution is a commonly used operation in the error-resilient applications such as deep neural networks and image processing, the performance and energy efficiency of the convolution operation can be improved by utilizing approximate multipliers. However, the utilization of only approximate arithmetic units (circuit-level approximation) may not be feasible for resource-constrained embedded systems. To improve the overall performance of the convolution operator in Fig. 1, it is advantageous to examine other layers of the computation stack for feasible approximations. For example, the total number of energy-consuming multiplication operations can be reduced by making $Stride = 2$ (algorithmic-level approximation). The utilization of approximate multipliers for the reduced number of operations can further improve the energy efficiency of the convolution operation.

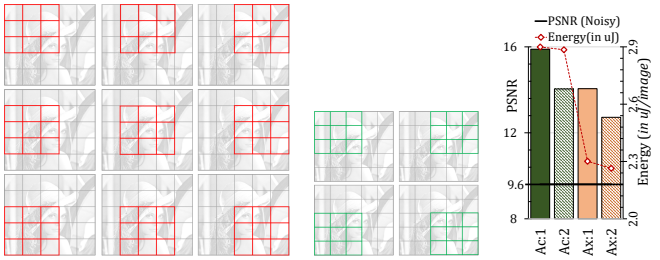


Fig. 1: Convolution operation using a 3×3 kernel (a) $Stride = 1$ (b) $Stride = 2$ (c) Accuracy/Energy trade-off for the Gaussian Image Smoothing filter using 2 *DoFs*

Fig. 1(c) shows the performance trade-offs from such a *Cross-layer Approximation* approach by comparing the peak signal-to-noise ratio (PSNR) and energy consumption values of a Gaussian Image Smoothing filter having a 3×3 convolution kernel using accurate (*Ac*) and approximate (*Ax*) multipliers[1] for two different values of the *Stride* parameter. The convolution operation using $Stride = 1$ and accurate multipliers (Ac:1) produces an output image with the highest PSNR and energy consumption values. The convolution operation utilizing algorithmic- and circuit-level approximations ($Stride = 2$ with approximate multipliers Ax:2) produces the most energy-efficient output, albeit with a low PSNR value. Such a cross-layer approach presents the designer with multiple tuning knobs for application-specific optimizations across multiple degrees of freedom (*DoFs*) in each layer.

However, each *DoF* and the available choices for it expand the design space exponentially. For instance, increasing the choice of multipliers for each operation, in Fig. 1(c), from two to three increases the possible design points from 2×2^9 to 2×3^9. Therefore, efficient design space exploration (DSE) frameworks, that enable the joint analysis of multiple *DoFs* across layers, are necessary to implement cross-layer approximations. Further, the required DSE technique should provide methods for the fast and accurate estimations of the applications' output accuracy and their performance parameters.

Most state-of-the-art works in the domain of approximate computing have focused on harvesting the performance gains by designing and utilizing approximation techniques at a single layer of the design stack. For example, the work presented in [7] utilizes a data sampling technique to process only a subset of input data. The authors of [8] have used reduced precision of data to decrease the application's computational complexity. Techniques presented in [3], [9], [10] utilize loop perforations and task skipping to bargain output accuracy of applications with performance gains. Many works, such as [5], [6], [11], employ approximate arithmetic blocks for realizing energy-

The work presented in this article is supported by the German Research Foundation (DFG) funded project ReAp (Project Number: 380524764)

[1] mul8s_1KVL from [6] has been used.

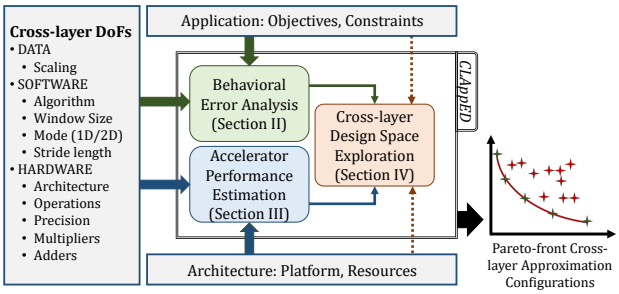


Fig. 2: *CLAppED* proposed framework

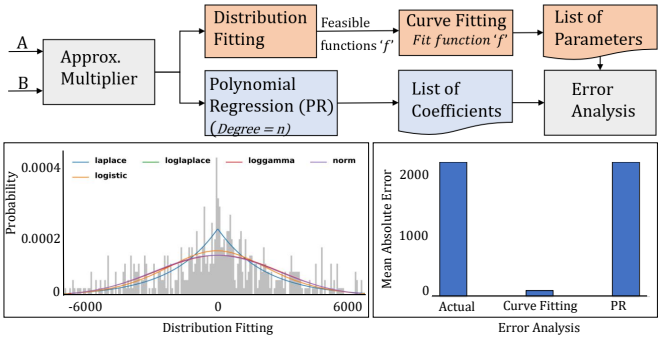


Fig. 3: Proposed behavioral Error analysis of approximate multipliers with results shown for mul8s_1KR3 approximate multiplier from [6]

efficient hardware accelerators. The work presented in [12] has focused on the circuit-level approximations by performing fast estimations of the optimal design points for implementing area-efficient approximate hardware accelerators.

Some recent related works, such as [3], [13]–[16] have presented the opportunities offered by the cross-layer approximations. However, most of these works, such as [13], discuss approximation techniques on various layers of computation stack in isolation. These works do not exploit the challenges and opportunities offered by approximations on a combination of layers. The authors of [3] have explored approximations at the algorithm-, architecture- and circuit-levels of design abstraction to implement a processor, with $1.2x$ to $5x$ energy efficiency, for the recognition and mining (RM) applications. However, their work does not consider utilizing already available open-source approximate arithmetic modules for circuit-level approximation. Further, they do not consider the fast estimations of the feasible design points for their RM processors. The authors of [14] and [15] have proposed simulators for evaluating the impact of three *DoFs* (low bit-width quantization schemes, activations pruning, and approximate multipliers) on the output accuracy of a deep neural network (DNN). The work in [16] has also considered these three *DoFs* for energy-efficient approximate DNNs. However, these works do not consider the thorough exploration of the design space and fast estimations of the feasible design points provided by various available *DoFs*. Further, to the best of our knowledge, none of the related works provide a solution for analyzing the application-level impact of a new approximate arithmetic unit without the time- and resource-consuming process of actual behavioral (or synthesis) testing of the application.

Towards this end, we present an efficient exploration framework, referred to as *CLAppED*, that incorporates a joint analysis of tuning the *DoFs* across multiple layers of the computation stack. Fig. 2 presents the various stages of the *CLAppED* framework. The related novel contributions are:

Contributions:

- We propose a novel polynomial regression-based characterization of approximate arithmetic units. Compared to the traditional distribution-based models, we report significant reductions in estimation errors.
- We provide a behavioral framework that utilizes various machine learning models for analyzing the impact of various *DoFs* on an application's output accuracy. The polynomial regression-based coefficients enable machine learning models to correlate an approximate arithmetic operator's impact on an application's output quality. This behavior allows the trained machine learning models to characterize the application-level impact of new unseen approximate arithmetic units.
- Although primarily focused on behavioral analysis, we present a complete framework for enabling cross-layer approximation-aware DSE. *CLAppED* utilizes behavioral error-analysis and accelerator's performance estimates to provide design points that offer better trade-offs between application error and hardware performance.

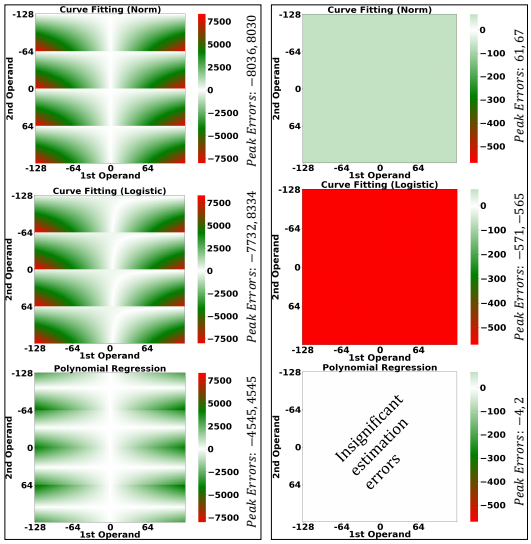
II. Error-analysis of Approximate Arithmetic Units

For our current work, we have considered only the approximate multipliers to describe the proposed error analysis. However, a similar analysis is also valid for other types of approximate circuits, such as adders and dividers. Further, due to the significance of signed numbers in various modern applications, such as machine learning, we have examined the open-source signed multipliers provided by [6] and [11]. Traditionally, statistical error metrics like Average Relative Error, Error Probability, and Mean Error Distance [17] are used to characterize approximate circuits. However, as observed in [18], the approximate arithmetic units are static non-linear systems and may violate several fundamental arithmetic principles such as *commutativity* and *associativity*. Further, the utilization of a statistical error metric to estimate an approximate circuit's impact on an application's output accuracy is mostly unknown. This lack of correlation between statistical error metrics and the corresponding quality of an application's output makes it difficult to select an approximate arithmetic unit out of many available choices. The problem is exacerbated in scenarios where a machine learning model is trained to predict an application's final output with a given configuration of approximations and performs poorly on novel approximate circuits. Towards this end, we perform an error-analysis of approximate signed multipliers to represent each component by a set of parameters that can be utilized to estimate the approximate result of a multiplier for an arbitrary input dataset. These parameters are utilized in our high-level behavioral framework to train machine learning models for fast estimation of an application's output quality for a given configuration of cross-layer *DoFs*.

A. Application Independent Error-analysis of Approximate Multipliers

We have considered *Curve Fitting* and *Polynomial Regression* (PR) techniques, as shown in Fig. 3, to represent each approximate multiplier by a set of unique parameters. The curve fitting-based technique utilizes the non-linear least-squares method to fit a function *'f'* to the results of an approximate multiplier for all input combinations. The function *'f'* utilizes a set of parameters to reduce the error between actual approximate products and the fitted results. Due to the various types of approximations in the available multipliers, a single function cannot be fitted to estimate all multipliers accurately. To identify feasible fitting functions, we perform distribution fitting of all approximate multipliers using several data distributions and evaluate their efficacy using Kolmogorov-Smirnov's (K-S) fitness metric (a commonly used technique to compare a sample with a reference probability distribution) [19]. We select five top distributions to implement the corresponding fitness functions for each approximate multiplier. For example, the distribution fitting subfigure in Fig. 3 shows the top-5 distributions for the multiplier *mul8s_1KR3* from [6]. However, as shown by the *Mean Absolute Error* (MAE) graph in Fig. 3, the fitting of the normal distribution-based function *'f'* significantly mismatches the actual error value. Similar results with large disparities between actual and estimated statistical error metrics are observed for other multipliers with various *fitting functions*.

The error plot in Fig. 3 shows that the *Polynomial Regression* (PR)-based technique can better estimate the approximate results of *mul8s_1KR3*. The PR technique utilizes the *'degree'* parameter to show the number of coefficients utilized for estimating the target function. For example, for an approximate multiplier with inputs *'x'*



(a) mul8s_1KR3 [6] (b) mul8s_1KVA [6]

Fig. 4: Comparison of estimation-induced errors for multipliers modeled using Curve Fitting and Polynomial Regression techniques

and 'y', a degree 2 PR generates six coefficients ($c0$ — $c5$), as represented by Eq. (1). The fitting of a PR-based model trains these coefficients to minimize the sum of squared errors between actual and estimated outputs for all input combinations of a multiplier. For each approximate multiplier, the PR-based model is trained separately to compute the corresponding coefficients.

$$f(x, y) = c_0 + c_1x + c_2y + c_3x^2 + c_4xy + c_5y^2 \quad (1)$$

We have also evaluated the PR-based models' efficacy on the 8-bit approximate adders from [6]. In this regard, we report as low as 18% estimation errors (MAE) with PR-based models compared to 84% estimation errors (MAE) with the curve fitting-based technique.

As shown by the error analysis results in Section V, all the trained coefficients, for a particular degree, do not have comparable significance, and the number of actually employed coefficients can be varied to deliver acceptable estimates with a reduced number of trained coefficients. During our error analysis, we have observed that the PR-based technique produces better estimates of the approximate multipliers than the curve fitting-based technique. For example, Fig. 4 presents the error distributions (the difference between the actual approximate and the estimated results) of mul8s_1KR3 (a *highly approximate multiplier*) and mul8s_1KVA (*a highly accurate multiplier*) from [6] for all input combinations. For mul8s_1KR3, the logistic-based model[2] produces slightly smaller estimation errors than the norm-based model. For mul8s_1KVA, the norm-based model produces better results than the logistic-based model. However, for both multipliers, the PR-based models produce fewer and smaller estimation errors than the curve fitting-based models. For example, for mul8s_1KVA, the PR-based model generated estimation errors range from -4 to $+2$. We have observed similar efficacy of the PR-based models for other approximate multipliers. Therefore, we have considered only the PR-based models for application-level error analysis and training of machine learning models for accuracy and performance predictions.

B. Application Specific Error Analysis

The behavioral error estimation constitutes the most application-specific analysis stage in *CLAppED*. Although in the current work, *CLAppED* is tested with one application—*Gaussian Smoothing*—the proposed framework is application-agnostic in principle. With appropriate interfaces between the generic DSE method and the application-specific estimation methods, the proposed framework can be used for any arbitrary application. For instance, in the behavioral error estimation for Gaussian Smoothing, we implemented a version of the two-dimensional (2D) convolution that integrates the impact of different *DoFs* such as scaling, stride-length variation—with and without downsampling[3], convolution mode (2D or 1D-Horizontal(1DH) → 1D-Vertical(1DV)) and using different approximate multipliers. Consequently, the impact of any arbitrary configuration across these *DoFs* can be evaluated directly by executing the corresponding executable over a set of images.

[2]The logistic- and norm-based models for these multipliers have been decided based on the corresponding distribution fitting ranking.

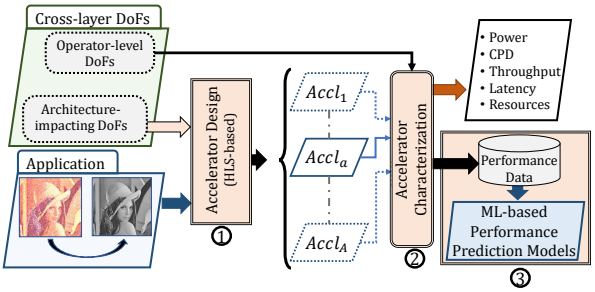


Fig. 5: Accelerator Performance Estimation

Further, we also provide an interface where a supervised machine learning (ML)-based model, that has been trained over a set of randomly generated configurations, can be used to predict the application's output quality for newer cross-layer approximation configurations. In this regard, we use the various available *DoFs*, as shown in Fig. 2, to produce input configurations for generating training and testing data for our machine learning models. A configuration denotes an arbitrary combination of the various *DoFs*. For each configuration in the training and testing set, we implement the applications' behavioral functions to generate corresponding *true labels* (actual outputs). The approximate multipliers in the applications' behavioral functions can be implemented by either using the multipliers' behavioral models or the PR coefficients-based estimates. These methods can be used for finding the appropriate *feasible* cross-layer approximation-based designs for any arbitrary application.

III. Accelerator Performance Estimation

Accelerator performance estimation constitutes the architecture-specific design method of *CLAppED*. Fig. 5 shows the various stages in this method. Stage ① involves designing the various accelerators that can be used for the application. This stage should consider the effect of only those *DoFs* that can result in different accelerator architectures and not the operator-level approximations. For instance, in the example of 2D convolution for Gaussian smoothing, varying the window size or the mode of convolution results in different numbers and types of accelerators. For the current work, we implemented the line-buffer-based sliding window [20] accelerators for 2D, 1DH, and 1DV operations. Further, the High-level Synthesis (HLS)-based designs support the variation in odd-numbered window sizes and different stride-lengths, resulting in varying accelerator designs. The second stage, ②, considers the effect of using operator-specific approximations in the accelerator. For our current work, we include the accelerator characterization with approximate signed multipliers from [5], [6]. It must be noted that, in our current work, we do not use the proposed framework for exploring the impact of HLS directives. We limit the exploration to configuring the approximate multipliers and other application-specific *DoFs*. Prior works such as [21], [22] present exploration methodologies for HLS and can be used alongside ours.

While the accurate characterization of the accelerators with varying *DoFs* provides high-quality results, the *actual* performance estimation using synthesis tools is very time-consuming. For instance, the characterization of a 2D convolution accelerator of 3×3 window size takes around 15 minutes. To this end, stage ③ in Fig. 5 shows an ML-based

[3]Downsampling refers to not using any values for the skipped pixels, leading to reduced output image size.

performance prediction method to estimate the hardware metrics. For 2D convolution, the dimensions of the model for each performance metric are a subset of the following features–the input image size, the stride-length, a down-sampling flag, and the precision and type of each multiplier operation in the design. Similar to the behavioral error analysis, the actual and the ML-based approaches provide alternative methods of varying result quality and estimation time to the DSE methodology of *CLAppED*.

IV. DSE Methodology

The error and accelerator performance estimation methods described earlier can be used for DSE with any randomized algorithm-based optimization, such as Genetic Algorithms and Simulated Annealing. However, these methods usually involve the evaluation of a large number of configurations. Hence, if the fitness function has a large evaluation time, it can lead to large DSE times as well. In contrast, Bayesian Optimization [23] provides a more directed search method and can also benefit from faster design point evaluations. In our current work, we implement a Multi-objective Bayesian Optimization (MBO) as the DSE method in *CLAppED*. Every iteration of Bayesian optimization can be divided into three major stages. The first stage involves using a *true Objective Function* to evaluate the metrics under consideration for some initial random samples. The results of this evaluation function form the training set for the second stage. In the second stage, the training data is used to design the *Surrogate Function*, which is a probabilistic prediction model. This prediction model is used in the third stage by the *Acquisition Function* to generate new candidate configurations to be evaluated by the Objective Function in the next iteration. This step completes one iteration of the search process. This process is repeated over multiple iterations to generate the final design points.

It can be noted that MBO is a well-established optimization method, and we do not claim novelty regarding the implementation. However, we model the cross-layer approximation design as an optimization problem and implement a framework that allows the novel ML-based estimation methods to be used as low-cost fitness functions required for faster DSE. In our current work, we generate multiple probabilistic models, one for each design objective. For instance, in the joint optimization for application accuracy and the LUT utilization, the Surrogate Function consisted of separate probabilistic models for both the objectives. The implementation of the Acquisition Function involved generating random cross-layer approximation configurations, followed by predicting their objective metrics with the corresponding probabilistic models in the Surrogate Function and ranking the samples based on their exclusive hypervolume contributions. A fixed number of the top-ranked samples are added to the configurations set for the next iteration.

The previous two sections present both *true* and ML-based estimation methods. The *true* estimation involved reporting the metrics from actual implementation of an accelerator with the cross-layer approximation configurations. Both *true* and ML-based methods serve as alternatives for the Objective Function only. Although the Surrogate Function is also based on a probabilistic model, unlike the ones used in the Objective Function, it is independent of the application under test. The proposed framework allows the designer to choose either of the methods (*true*/ML-based) independently for error and hardware performance estimation during Objective Function evaluation. In case of using ML-based methods, the DSE results can be used in a neighborhood search with the *actual* evaluation to further improve the quality of results.

V. Experiments and Results

A. Experiment Setup

The implementation of *CLAppED* involved both probabilistic analysis as well as hardware design. The HLS-based accelerator designs were implemented in C++ and synthesized with Xilinx Vivado Design Suite. All designs have been implemented for Xilinx Zynq UltraScale+ MPSoC (xczu3eg-sbva484-1-e device). The probabilistic analysis for curve fitting, polynomial regression (PR), application-level ML-based modeling, and the MBO-based DSE methodology were implemented in Python using multiple packages, including scikit-learn, TensorFlow [24], PyGMO [25].

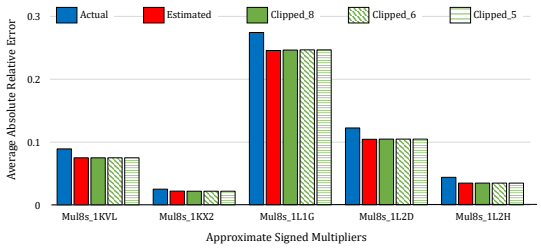


Fig. 6: Error analysis of various approximate signed multipliers from [6]

B. Behavioral Analysis

A PR model's efficacy to predict an approximate multiplier's output is computed using the model's coefficient of determination denoted as R^2. A large value of R^2 (≈ 1) denotes a good fitting of the model to predict the actual outputs. As shown in Eq. (1), the complexity of a PR model is defined by its degree parameter. During the error analysis of the approximate multipliers from [6] and [11], we have observed that PR models with at least degree 3 produce significantly accurate estimations of the actual approximate results. Further, all the generated coefficients for a PR model (after training the model) do not have equal significance. We can analyze the generated coefficients of all multipliers, under consideration, for a specific degree-based PR model and rank their overall significance. Based on the ranking of coefficients, we can remove the less significant coefficient for each multiplier. For example, Fig. 6 compares the *actual* and *estimated* (using degree 3 PR models) average absolute relative errors of five approximate multipliers from [6]. The *Clipped_8*, *Clipped_6*, and *Clipped_5* bars in the figure show the utilization of only 8, 6, and 5 trained coefficients for PR models to estimate approximate multipliers. The PR models provide significantly accurate estimates of the actual average relative error values, on average a difference of 15%. The *Clipped_5* graphs show, on an average, only a 0.06% degradation in the estimated values for the multipliers under consideration.

Utilizing the trained coefficients' ranking, we can also implement custom PR models that are *retrained* with a reduced number of coefficients. For example, Fig. 7 compares the average relative and maximum errors of *mul8s_1KR3*, from [6], using different numbers of coefficients (C2 — C9) based retraining. The *Predicted* bar in the figure shows the estimation utilizing all the trained coefficients. As shown by the results, PR models with only 2 and 3 coefficients (C2 and C3 respectively) produce large deviations from the corresponding approximate results. Due to the reduced number of coefficients, these models behave like accurate multipliers. However, PR models having more than three coefficients produce high R^2 values and efficiently represent the approximate multiplier. For example, for the actual average relative error (1.36) and maximum absolute error (8001), a C4-based PR model produces 1.34 and 5012 for these error metrics, respectively. Further, increasing the number of coefficients beyond 6 has no impact on the PR-model's accuracy for *mul8s_1KR3*. The fine-grained control on utilizing an only appropriate number of coefficients also helps in reducing the complexity of ML-models for estimating applications' behavioral accuracy.

To predict the impact of various *DoFs* on an application's output accuracy, we have used a multi-layer perceptron (MLP)-based model. The model is used to predict accuracy of achieving noise-removal with random configurations of cross-layer approximation *DoFs*, compared to that obtained by a *golden* configuration. The model is trained and tested by providing an input dataset of 2000 configurations. These configurations contain various uniformly distributed combinations of the considered *DoFs*. Our model utilizes randomly selected 80% configurations of the input dataset for training the model. The remaining 20% of the input dataset is employed for testing the trained model. Further, the model uses 20% of the training dataset for validation during the training phase. For this work, we have modeled a 3×3

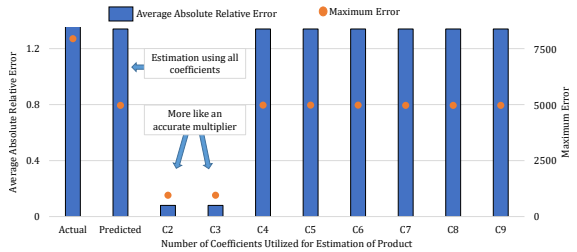


Fig. 7: Error analysis of mul8s_1KR3 approximate multiplier from [6]

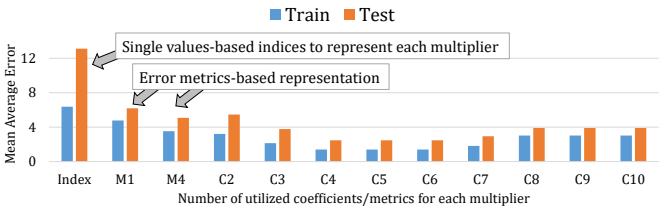


Fig. 8: Mean average error (MAE) of training and testing an MLP for a 3×3 kernel-based Gaussian Image Smoothing filter utilizing different number of coefficients for multipliers from [6] and [11].

kernel-based Gaussian Image Smoothing filter. To represent the nine multipliers in the convolution kernel of the application, we have used *index*-, *error metrics*-, and polynomial regression coefficients-based methods. In the *index*-based method, each multiplier is assigned a unique random value for its identification. In the *error metrics*-based method, we have utilized each multiplier's four statistical error metrics (maximum absolute error, average relative error, error probability and mean squared error) for its identification. We refer to this configuration as $M4$ in our experiments. We have also experimented with using only one single statistical error metric, referred to as $M1$, for the identification of multipliers in the MLP model. Such single values-based identification is used in [12], which utilizes the weighted mean error distance (WMED) of a multiplier for its identification. However, our application-level error analysis reveals that *PR coefficients*-based representation of a multiplier performs better than other techniques.

Fig. 8 compares the training and testing accuracy of the utilized MLP model. The $M1$-based representation shows the mean squared error-based identification. The *C2 — C10* shows the number of utilized *PR* coefficients to represent each multiplier in the training and testing datasets. The index-based method produces the highest mean average errors (MAE) for both the training and the testing phases, 6.37 and 13.12, respectively. Clearly, the model cannot identify the correlation between multipliers' indices and the impact of utilized approximate multipliers on the output quality degradation. The $M1$- and $M4$-based techniques produce lower MAE values than the index-based method. For example, $M4$-based representation produces 3.5 and 5.0 MAE values for training and testing datasets, respectively. However, the utilization of PR coefficients-based representation improves the MLP model's accuracy significantly. For example, the $C4$-based representation (four PR coefficients) generates only 1.4 and 2.4 MAE values for training and testing datasets, respectively. However, increasing the number of coefficients to represent a multiplier directly impacts the total number of trainable parameters of the MLP model. For an MLP model with a high number of trainable parameters, a large dataset is required to train the model. For the given dataset, the *C7*- to *C10*-based representation of multipliers result in reducing the output accuracy of the model due to insufficient training data.

To further evaluate the performance of the MLP model, we have used the *fidelity* metric [12]. The fidelity metric denotes the relationship $(=, <, >)$ between the input configurations and their corresponding actual and predicted values. Fig. 9 shows the percentage fidelity of the MLP model on training and testing sets using various representations for the multipliers. As previously observed in Fig. 8 for MAE analysis, the index-based method produces the minimum fidelity. The *C4* to *C6* based representations of multipliers produce

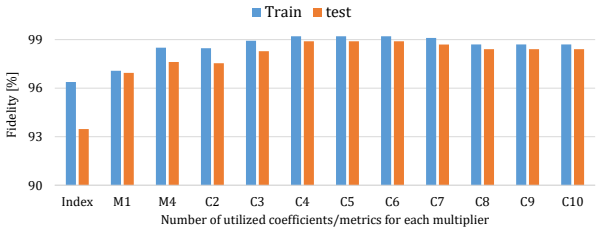
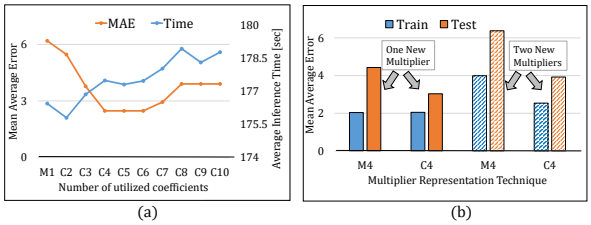


Fig. 9: Fidelity of training and testing an MLP for a 3×3 kernel-based Gaussian Image Smoothing filter utilizing different number of coefficients for multipliers from [6] and [11].



(a) (b)

Fig. 10: Efficacy of MLP-based estimation of a Gaussian Image Smoothing filter (a) Impact of utilizing different number of coefficients on the MAE and average estimation time (b) Estimation accuracy on test data having new unseen multipliers

the best training and testing fidelity, 99.2% and 98.9%, respectively.

To understand the relationship between the number of PR coefficients and the corresponding inference time of the MLP model, Fig. 10(a) compares the MAE and average inference time for 1000 iterations of the testing dataset. On average, the MAE decreases, and the inference time increases by utilizing more coefficients to represent multipliers in the MLP model. For the given MLP model and the testing dataset, *C4* (employing 4 *PR* coefficients to show a multiplier) produces the best result in terms of low MAE and inference time.

The utilization of PR coefficients to represent multipliers enables the MLP model to correlate approximate multipliers and their impact on an application's output accuracy. This correlation enables the MLP model to predict an application's output quality for new multipliers (not included in the training and validation datasets). Fig. 10(b) shows the training and testing MAEs of an MLP model for two cases. In the first experiment, the testing dataset configurations also include a multiplier that is not present in any configuration of the training dataset. In the second experiment, the testing dataset has two such multipliers that are not available to any configuration of the training dataset. As shown by the results, the PR coefficients-based representation of the approximate multipliers has enabled the MLP model to produce high output accuracy even for unseen new multipliers (with up to 98.4% fidelity).

C. Accelerator Performance Estimation

Similar to the behavioral analysis, we implemented MLP-based models for predicting the performance of accelerators. Fig. 11 shows the results from the experiments for a 2D convolution accelerator. The bar-graphs in the figure show the prediction accuracy, in terms of fidelity of results, for four different metrics– Power Delay Product (PDP), Latency (the number of clock cycles needed for the 2D convolution over an image), the FPGA's LUT utilization and the accelerator's power dissipation. 1000 designs were used for training the MLP models and 200 design points were used for the testing.

The accuracy on training and testing datasets is reported for two approaches – *IDX*, where each multiplier is represented by an index value and *EXP*, an expanded representation, where we tested different combinations of DoFs for determining the dimensions of the corresponding MLP model. TABLE I shows the resulting dimensions for each model, categorized as accelerator- and multiplier-level dimensions. The set of dimensions for the *EXP*-based model was chosen based on its testing accuracy. It can be noted from the table that the model for latency has

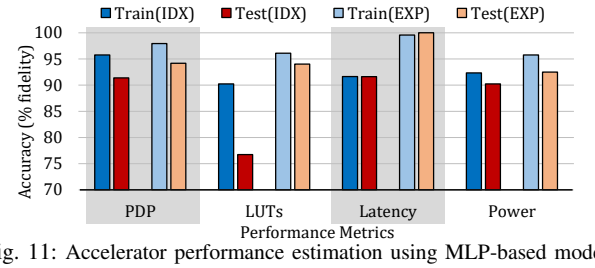


Fig. 11: Accelerator performance estimation using MLP-based model

TABLE I
MLP DIMENSIONS FOR ACCELERATOR PERFORMANCE MODELING

Performance metric→	PDP	LUTs	Latency	Power Dissipation
Accelerator Dimensions	Image Size, Stridelength, Downsampling	Image Size, Stridelength, Downsampling	Image Size	Image Size, Stridelength, Downsampling
Multiplier Dimensions	Critical Path Delay, Total Power Dissipation	LUT Utilization	–	Signal Power, Logic Power

the least parameters with the highest prediction accuracy. Since latency depends primarily on the image size, multiplier-specific parameters were not used in the model. It can be observed that using the *EXP*-based approach leads to higher accuracy for both training and testing datasets.

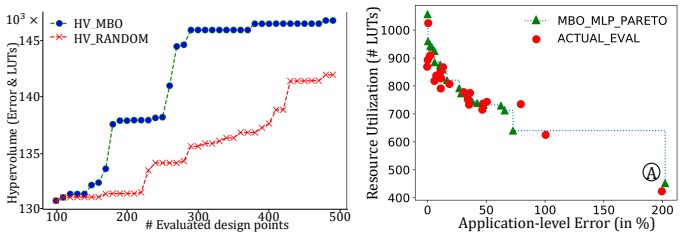
D. DSE Performance

The *directed* search capability of Bayesian Optimization is especially beneficial for problems with costly fitness function evaluation—similar to the *actual* estimation of accelerator performance. Although we provide a faster ML-based approach for the same, the search capability of the optimization algorithm determines the quality of results. Fig. 12(a) compares the progress of a similar search for application-level error and LUT utilization trade-offs, using MBO and random search. The plot shows the hypervolume obtained with an increasing number of design point evaluations. Both methods use the ML-based estimation of application-level error and LUT utilization. It can be observed that using MBO obtains better quality results much faster. The data in the figure was obtained from an optimization run that evaluates 10 new samples in each iteration, selected from 50 random samples that are generated by the Acquisition Function.

Fig. 12(b) shows the results from the analysis of the design points obtained from MBO-based DSE. In this search for Error-LUT trade-offs, the MBO-based search using MLP models generated 23 Pareto-front design points, shown as MBO_MLP_PARETO in the figure. It must be noted that only one among those 23 points, shown as Ⓐ, used a configuration where all the 9 multipliers (for a 3×3 window) were of the same type. This reinforces our motivation of the need for searching among the vast number of possible multiplier permutations in an application. Further, 3 points had a stride length of 2 compared to 1 for the others, and 12 points had downsampling enabled. Similarly, image scaling values 3, 2, and 1 were observed for 2, 19, and 2 design points, respectively. These results further demonstrate the need for cross-layer exploration across multiple types of *DoFs*. We evaluated the 23 points with actual hardware synthesis, and the resulting points are plotted in Fig. 12(b) (ACTUAL_EVAL). It can be observed that the *true* points are close to those obtained using the MLP-based model.

VI. CONCLUSION

Approximate computing presents an attractive path for achieving low-cost execution for a large variety of applications. Further, a cross-layer approach allows the design of application-specific approximations with more availability of *DoFs*. In this article, we present *CLAppED*, a framework for enabling such a design approach. The proposed behavioral analysis allows the designer to create application-specific models for exploring existing arithmetic operators (with up to 98.9%) and predicting new unseen ones' efficacy (with up to 98.42% fidelity). Further, the fast accelerator performance estimation methods and the MBO-based DSE methodology enable the search for high-quality designs.



(a) DSE progress (b) Pareto analysis

Fig. 12: (a) Comparison of the quality of results (hypervolume) during DSE for trade-offs between application-level error and LUTs utilization using MBO and randomized search (b) Analysis of the Pareto points obtained from MBO-based DSE

REFERENCES

[1] A. Yazdanbakhsh, et al. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE D&T*, 34(2):60–68, 2017.

[2] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), March 2016.

[3] V. K. Chippa, et al. Scalable effort hardware design. *IEEE TVLSI*, 22(9):2004–2016, 2014.

[4] P. Düben, et al. Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications. In *2015 DATE*, pages 764–769, 2015.

[5] S. Ullah, et al. SMApproxlib: Library of FPGA-based approximate multipliers. In *2018 DAC*, pages 1–6. IEEE, 2018.

[6] V. Mrazek, et al. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *DATE, 2017*, pages 258–261, March 2017.

[7] I. Goiri, et al. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. *SIGPLAN Not.*, 50(4):383–397, March 2015.

[8] S. Ullah, et al. L2L: A highly accurate log_2_lead quantization of pre-trained neural networks. In *DATE, 2020*, pages 979–982, 2020.

[9] V. K. Chippa, et al. Analysis and characterization of inherent application resilience for approximate computing. In *DAC, 2013*, pages 1–9, 2013.

[10] S. De, et al. Approximation trade offs in an image-based control system. In *DATE*, pages 1680–1685, 2020.

[11] S. Ullah, et al. Area-optimized accurate and approximate softcore signed multiplier architectures. *IEEE Transactions on Computers*, April 2020.

[12] V. Mrazek, et al. AutoAx: An Automatic Design Space Exploration and Circuit Building Methodology Utilizing Libraries of Approximate Components. In *DAC 2019*. ACM, 2019.

[13] M. Shafique, et al. Invited: Cross-layer approximate computing: From logic to architectures. In *DAC*, pages 1–6, 2016.

[14] C. De la Parra, et al. Improving approximate neural networks for perception tasks through specialized optimization. *Future Generation Computer Systems*, 113:597 – 606, 2020.

[15] Y. Fan, et al. AxDNN: Towards the Cross-Layer Design of Approximate DNNs. ASPDAC '19, page 317–322, New York, NY, USA, 2019. Association for Computing Machinery.

[16] M. Hanif, et al. X-dnns: Systematic cross-layer approximations for energy-efficient deep neural networks. *Journal of Low Power Electronics*, 14:520–534, 12 2018.

[17] J. Liang, et al. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013.

[18] M. Bruestel et al. Accounting for systematic errors in approximate computing. In *DATE*, pages 298–301, 2017.

[19] F. J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[20] Xilinx. UG1270: Vivado HLS Optimization Methodology Guide, April 2018.

[21] Hung-Yi Liu et al. On learning-based methods for design-space exploration with high-level synthesis. In *DAC*, pages 1–7, 2013.

[22] A. Mehrabi, et al. Prospector: Synthesizing Efficient Accelerators via Statistical Learning. In *DATE*, pages 151–156, 2020.

[23] J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pages 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[24] M. Abadi, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[25] F. Biscani et al. esa/pagmo2: pagmo 2.9, August 2018.