AxOTreeS: A Tree Search Approach to Synthesizing FPGA-based Approximate Operators

SIVA SATYENDRA SAHOO, Interuniversity Microelectronics Centre (IMEC), Belgium
SALIM ULLAH, cfaed, Technische Universität Dresden, Germany
AKASH KUMAR, cfaed, Technische Universität Dresden, Germany

Approximate computing (AxC) provides the scope for achieving disproportionate gains in a system's power, performance, and area (PPA) metrics by leveraging an application's inherent error-resilient behavior (BEHAV). Trading computational accuracy for performance gains makes AxC an attractive proposition for implementing computationally complex AI/ML-based applications on resource-constrained embedded systems. The growing diversity of application domains using AI/ML has also led to the increasing usage of FPGA-based embedded systems. However, implementing AxC for FPGAs has primarily been limited to the post-processing of ASIC-optimized approximate operators (AxOs). This approach usually involves selecting from a set of AxOs that have been optimized for a gate-based implementation in an ASIC. While such an approach does allow leveraging existing knowledge of ASIC-based AxO design, it limits the scope for considering the challenges and opportunities associated with FPGA's LUT-based computation structures. Similarly, the few works considering the LUT-based computing for AxO design use generic optimization approaches that do not allow integrating problem-specific prior knowledge—empirical and/or statistical. To this end, we propose a novel tree search-based approach to AxO synthesis for FPGAs. Specifically, we present a design methodology using Monte Carlo Tree Search (MCTS)-based search tree traversal that allows the designer to integrate statistical data, such as correlation, into the AxOs optimization. With the proposed methods, we report improvements over standard MCTS algorithm-based results as well as improved hypervolume for both operator-level and application-specific DSE, compared to state-of-the-art design methodologies.

CCS Concepts: • **Hardware → Circuit optimization**; *Hardware accelerators*; **Arithmetic and datapath circuits**; Software tools for EDA; • **Computing methodologies → Randomized search**.

Additional Key Words and Phrases: Approximate Computing, Arithmetic Operator Design, Circuit Synthesis, AI-based Exploration, Computer Arithmetic, Automated Hardware Design, Monte Carlo Tree Search

ACM Reference Format:
Siva Satyendra Sahoo, Salim Ullah, and Akash Kumar. 2023. *AxOTreeS*: A Tree Search Approach to Synthesizing FPGA-based Approximate Operators . *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2023), 25 pages. https://doi.org/10.1145/3609096

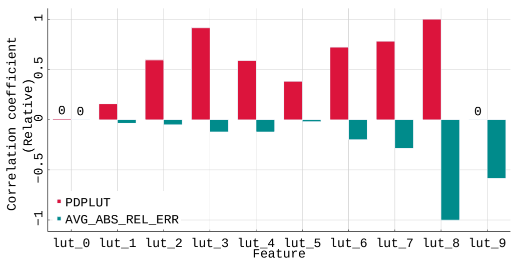
1 INTRODUCTION

Historically, embedded computing had primarily focused on the timeliness of systems rather than computational complexity. However, with the growing diversity of application domains using

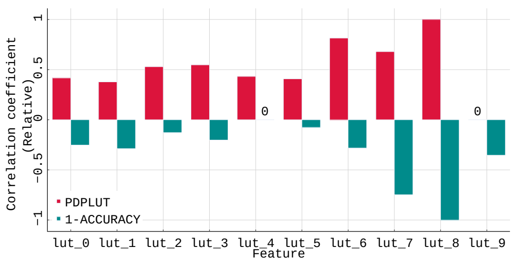
This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023

Authors' addresses: Siva Satyendra Sahoo, Interuniversity Microelectronics Centre (IMEC), Leuven, Belgium, Siva.Satyendra.Sahoo@imec.be; Salim Ullah, cfaed, Technische Universität Dresden , Dresden, Germany, salim.ullah@tu-dresden.de; Akash Kumar, cfaed, Technische Universität Dresden , Dresden, Germany, akash.kumar@tu-dresden.de.

1539-9087/2023/1-ART1 $15.00
https://doi.org/10.1145/3609096



(a) Signed 4x4 Multiplier



(b) LPF in ECG Peak Detection

Fig. 1. *AxOTreeS* Motivation

Artificial Intelligence (AI) and Machine Learning (ML), embedded computing increasingly demands executing complex inference engines on resource-constrained systems. Therefore, embedded systems deployed for edge processing typically need to provide better Power-Performance-Area (PPA) optimization while satisfying the applications' behavioral requirements. To this end, multiple ongoing research efforts, such as TinyML [31], aim to leverage the applications' behavioral tolerances to optimize the design choices at/across multiple layers of the system stack. Since inference engines primarily use Multiply-Accumulate (MAC) operations extensively, a lot of such optimization effort at the hardware layer has been focused on precision scaling of arithmetic operations [4]. It involves reducing the number of bits used for representing the operands (weights and/or features), thereby enabling the usage of lower bit-width operators with better PPA trade-offs. However, precision scaling is a generic approach and has limited scope, in terms of the number of possible bit-widths, for implementing application-specific optimizations at the hardware layer.

Consequently, there is an increasing focus on developing methods to exploit an application's output accuracy tolerances at a finer granularity. Approximate Computing (AxC) provides the scope for achieving disproportionate gains in a system's PPA by leveraging an application's inherent error-resilient behavior (BEHAV). AxC across multiple layers of the computation stack includes methods such as loop skipping and data scaling [14]. While precision scaling can also be considered a form of approximation, we will limit the term AxC to refer to methods that deliberately inject errors into the computation by avoiding some form of processing. In the context of arithmetic operations, AxC would, unlike low-precision operations, entail allowing some form of errors in the output of the arithmetic operation. Such Approximate Operators (AxOs) can be implemented for any bit-width operation and hence provide an additional Degree of Freedom (DoF) that can complement traditional precision scaling. Additionally, AxOs present a much broader scope for exploiting an application's error tolerance. However, the design of such application-specific AxO presents a highly complex Design Space Exploration (DSE) problem.

A major aspect of the DSE problem for AxOs involves the target hardware platform-specific optimizations. While most of the related works in AxO design are focused on Application-specific Integrated Circuits (ASICs), the static design nature of ASICs requires the implementation of an accurate operator alongside the approximate version. On the other hand, with Field Programmable Gate Arrays (FPGAs), different versions of the AxO can be implemented dynamically, including the accurate version. The reconfigurability of FPGA-based hardware platforms has resulted in their increasing usage in embedded systems, as they can cater to the diversity of application requirements more effectively. However, designing novel AxOs for FPGAs has primarily been limited to the post-processing of ASIC-optimized AxOs. While such an approach allows leveraging existing knowledge and libraries of ASIC-based AxO designs, it limits the scope for leveraging FPGA's Look-Up Table (LUT)-based computation structures [38, 40, 41]. A more effective approach to DSE for FPGA-based AxO requires novel operator models and related optimization methods than those used for ASICs. More recently, various AxO models have been proposed for enabling FPGA-specific optimizations [36]. Similarly, AI/ML-based DSE methods have also been proposed for synthesizing novel AxOs for FPGAs [15, 25, 36]. However, most of the proposed DSE approaches are based on generic search methods such as Genetic Algorithms (GA) and other randomized search algorithms that do not allow integrating prior knowledge from empirical and/or statistical analysis.

To further highlight the importance of performing various types of statistical analysis of AxOs and considering this knowledge in exploring feasible design points, we present an accuracy-performance correlation analysis of various 4×4 signed approximate multipliers. This analysis is based on the accurate signed multiplier implementation presented in [34], which utilizes 10 LUTs to generate the partial products of a 4×4 multiplier. Based on the AxOs implementation methodology of removing LUTs from an accurate implementation, as discussed in [36], we have utilized the 10 LUTs to implement 1024 different approximate designs. Fig. 1 shows the relative correlation coefficient between the utilization of each of the 10 LUTs and the PPA and BEHAV metrics. The LUTs, lut_xs correspond to those used in generating the partial products of the 4×4 multiplier. Fig. 1(a) and Fig. 1(b) show the results for operator-level evaluation and an application-specific analysis by utilizing the 1024 designs in the Low-pass Filter (LPF) implementation in Electrocardiogram (ECG) [19], respectively. As expected, utilizing any particular LUT shows a positive correlation with the PPA metric (PDPLUT[1]) and a negative correlation with the BEHAV (error) metric. Therefore, not using the lut_x in the implementation would cause more error (AVG_ABS_REL_ERR [2] for the operator and (1-Accuracy)[3] for ECG peak detection). However, the varying correlation metric indicates the relative importance of each LUT in contributing to PPA and BEHAV metrics. Similarly, the difference in the distribution of the correlation metrics in Fig. 1(a) and Fig. 1(b) indicates changing relative importance from an operator-level to an application-level dependency. State-of-the-art DSE approaches do not allow using such statistical patterns in the search methodology. To this end, we propose a novel search methodology that allows the effective integration of such LUT-wise significance in the DSE for FPGA-based AxOs. The related contributions include:

Contributions:

(1) We present a novel tree search-based approach to synthesizing new FPGA-based approximate operators. Specifically, we model the DSE for signed approximate multipliers as a Tree Traversal Problem (TTP) and use Monte Carlo Tree Search (MCTS) along with ML-based estimation for generating Pareto-front AxO designs under given PPA and BEHAV constraints.
(2) We present problem-specific adaptations to the more general MCTS approach. Specifically, we provide knobs to include statistical information regarding LUT usage in the TTP. We report up to 1.6x better hypervolume than using general MCTS for tightly constrained design problems.
(3) We present the results from the evaluation of the proposed methodology for both operator-level and application-specific DSE. We report improved Pareto-front designs than state-of-the-art approaches for both.

The rest of the article is organized as follows. Section 2 provides a brief overview of the background and related works pertaining to designing approximate operators. The problem statements, including the operator model used in the current work and the DSE problem formulation, are described in Section 3. Section 4 describes the proposed *AxOTreeS* methodology and includes a brief background of MCTS. The experimental evaluation of the proposed methods is presented

[1]Power Delay Product × LUT utilization
[2]Average Absolute Relative Error
[3]Classification error

in Section 5. Section 6 concludes the article with a summary and a brief discussion of the scope for related future work.

2 BACKGROUND AND RELATED WORKS

2.1 Approximate Computing

The paradigm of Approximate Computing has emerged as a viable solution to satisfy the ever-increasing computational and memory demands of modern applications ranging from data centers to mobile devices and embedded systems at the edge. A common attribute of most of these applications is their inherent error resilience to inaccuracies in the data representation and intermediate computations. The inherent error resilience of these applications enables them to produce multiple feasible answers instead of one golden answer. For example, an audio processing application on an embedded system can employ different encoders with various precisions to produce multiple acceptable quality outputs. AxC exploits this error resilience to trade the output accuracy of an application for performance gains. Recent related works have explored various approximation techniques covering all layers of the computation stack [1, 5, 13, 14, 32, 39, 44].

Among the various computation layers for approximation, the architecture and circuit layers have acquired the most significant attention for resource-constrained embedded systems. On the architecture level, employing reduced precision arithmetic and storage has remained one of the most commonly utilized techniques [6, 42, 44]. Similarly, using inaccurate (approximate) computational units is one of the most effective techniques on the circuit layer. As MAC is one of the primary operations in ML applications (inherently error-tolerant), most related works have focused on proposing various approximate implementations for adders and multipliers [8, 9, 16, 18, 21, 24, 30, 33, 35, 38, 43]. These techniques mainly rely on either truncating parts of computation or utilizing inaccurate computations to introduce deliberate approximations for performance gains. For example, the works presented in [30, 43] have utilized multiple sub-adders to truncate the long carry propagation chain in ASIC-optimized larger adders. These designs utilize multiple previous bits to predict the input carry for each sub-adder. Similarly, the authors of [24] have utilized different carry and sum prediction techniques to present a set of FPGA-optimized approximate adders.

Considering the higher computational complexity of multiplication operations, most related works have proposed various approximate architectures for both ASIC- and FPGA-optimized architectures. For instance, the authors of [9, 21] have utilized various truncation techniques to produce M-bit output for an $\mathrm{M} \times \mathrm{M}$ ASIC-optimized multiplier. Similarly, some works, such as [8], truncate the input operands to employ a smaller multiplier to implement a larger multiplier for ASIC-based systems. Other works, such as [10, 26], perform functional approximation to implement ASIC-optimized inaccurate 2×2 multipliers. The 2×2 multipliers are then utilized to implement larger multipliers. For example, the authors of [16, 18] have utilized Cartesian Genetic Programming (CGP) to present libraries of ASIC-optimized approximate adders and multipliers. In their proposed technique, they utilize a set of accurate implementations of an operator and perform multiple iterations of the CGP to generate corresponding approximate circuits. For this purpose, the accurate circuits are represented using a string of integers, and a worst-case error-based objective function is used to generate various approximate versions of a circuit. Considering the architectural specifications of FPGAs, the works presented in [33, 35, 38] have utilized the LUTs and carry chains of Xilinx FPGAs to propose various approximate multiplier architectures. The authors of [35] have presented a 4×4 approximate multiplier to implement higher-order multipliers. Similarly, the work presented in [33] utilizes three different designs of FPGA-optimized approximate 4×4 multipliers to implement a higher-order approximate multipliers library. However, this work does not provide an intelligent DSE mechanism for identifying design points that provide better accuracy-performance

Table 1. Comparing related works

Related Work	EvoApprox [16, 18]	autoAx [15]	ApproxFPGA [23]	[29]	[38]	AppAxO [36]	SyFAxO-GeN [25]	*AxOTreeS*
LUT-level Optimization	✗	✗	✗	✗	✓	✓	✓	✓
Application-specific Design Search	✗	✓	✓	✗	✗	✓	✗	✓
Automated Search	✓	✓	✗	✓	✗	✓	✓	✓
ML-based Estimation	✗	✓	✓	✗	✗	✓	✓	✓
Directed Search	✗	✗	✗	✗	✗	✗	✗	✓

trade-offs. The authors of [38] have utilized radix-4 Booth's algorithm to present an approximate signed multiplier implementation. In their proposed implementation, the authors initially focus on efficiently utilizing the 6-input LUTs and the associated carry chains to propose a resource-efficient implementation of the signed multiplier. Furthermore, to reduce the critical path delay and dynamic power consumption of the implementation, the authors employ truncation of selected partial produce bits in each partial product row. To improve the output accuracy of the approximate multiplier, the authors include functional approximation by selectively changing the functionality of a few LUTs in each partial product row. However, such manual optimization techniques are time-consuming for designing and implementing an approximate operator with different specified accuracy-performance constraints.

2.2 DSE for Approximate Operators

Some recent works, such as [16, 18, 33], provide libraries consisting of hundreds of approximate versions of an operator with varying accuracy-performance trade-offs. Therefore, in many cases, it is necessary to identify feasible operator implementations that can satisfy an application's overall accuracy-performance constraints. Table 1 provides an overview of some of the related frameworks for performing design space exploration of approximate operators. Each row in the table specifies a certain aspect or method used in the DSE methodology for each of these frameworks. For example, the work presented in [18] utilizes a worst-error-based objective to identify ASIC-optimized feasible design points. Another similar work proposed in [29] focuses on identifying the largest sub-circuit that can be removed from an accurate implementation to produce an approximate version that complies with the provided accuracy constraints. For this purpose, this work formulates sub-circuit identification as a binary tree exploration problem. It employs various techniques to assign weights to all nodes in the original netlist to compute the accuracy impact of removing a set of nodes. However, these works do not include the generation of approximate operators while considering application-level accuracy and performance constraints. Furthermore, these works do not employ ML-based fast estimators to predict the accuracy and performance metrics of the generated design points, and therefore may not be feasible for accelerator-level DSE that involves long characterization times. The authors of [17] have used CGP to design ASIC-optimized unsigned approximate multipliers for Artificial Neural Networks (ANNs). However, this work has not considered the performance metrics, such as dynamic power consumption, while generating the new designs. The work presented in [15] utilizes the approximate operators from [18] to present a framework for application-specific DSE for approximate adders. Similarly, the authors of [23] employ various ML models to identify feasible ASIC-based operators for implementing them on FPGA-based systems. In this work, the various ML models are initially trained using a dataset including the performance values of implementing ASIC operators (from [18]) for FPGAs. In the testing and estimation phase, the authors use the trained models to infer the FPGA-based performance values for a given hardware description of the operator. However, this work, along with [15, 16, 18], does not focus on performing any FPGA-specific LUT-level optimizations to improve the implementation of the operator. They rely on the library of ASIC-optimized operators as the power set of operators that can be implemented on FPGAs. This limits the scope of application-specific optimizations that be explored during DSE. Further, limiting the exploration to the library of existing implementations, typically a few hundred designs, does not warrant ML-based estimators and can be evaluated exhaustively.

The authors of [38] have used GA to identify feasible operator implementations (accurate and approximate) for a Gaussian image smoothing application. However, this work focuses on identifying the combination of feasible operators from a comparatively smaller set of approximate implementations and does not consider the PPA metrics, such as power and the resulting accelerator's logic delay, during DSE. The framework presented in [36] leverages various ML models and GA-based multi-objective optimization technique to implement application-specific FPGA-optimized approximate versions of accurate operators. The operator model proposed in [36] uses a binary string to address the LUTs in the accurate implementation of an operator, and the corresponding approximate versions of the operator are generated by removing targeted LUTs. The ML models employed in this framework act as surrogate functions to estimate the accuracy-performance metrics for a given approximate operator configuration. An improved model that integrates two multiplier algorithms, Booth and Baugh-Wooley, has also been presented recently [37]. However, the proposed approach uses only the implicit search directions, derived during the metaheuristics-based randomized search of GA, to find novel approximate implementations. The work presented in [25] has used Generative Adversarial Networks (GANs) to identify operator configurations satisfying provided accuracy-performance constraints. However, the proposed approach uses a fairly open-loop approach where the newly explored designs are not used for improving the subsequent search.
Summary: As shown in Table 1, in general, the works related to DSE for FPGA-based AxOs can be categorized broadly into – 1) the ones that follow the selection approach which involves leveraging ASIC-optimized logic [15, 23] and, 2) the ones that perform LUT-level optimization for synthesizing novel AxOs that may differ logically from ASIC optimized ones [25, 33, 36, 38]. The FPGA-specific synthesis approaches can further be categorized into works that use handcrafted optimizations resulting in one or two novel designs [38] and those using automated search methods to generate a library of operators [25, 33, 36]. Works such as *AppAxO* [36] and *SyFAxO-GeN* [25] essentially aim to automate the approach proposed in [38]. This involves identifying the top power consuming LUTs and selecting a subset of the LUTs for removal from the optimized accurate implementation while providing the best PPA-BEHAV trade-offs. They, along with [15, 23], use ML-based PPA and BEHAV metrics estimation and, rely on the search algorithm to decipher the operator/application-specific PPA-error trade-offs provided by each LUT, in order to find the *optimal* approximate configuration (LUT usage). They do not provide any methods of explicitly directing the related search towards design points with better trade-offs. The search directions are implicitly derived from the iterative search algorithm such as GA [36], random sampling [15], or sampling with generative models [25]. The characterization data of the sampled approximation configurations is limited to just designing regression models for predicting PPA and BEHAV metrics. Since the iterative search algorithm determines the PPA-BEHAV trade-offs from predictive models, that are primarily designed for fitness evaluation of each approximate configuration, *we posit that the search process can be improved with information derived from statistical/empirical analysis of design characterization data and using the resulting LUT-wise significance* . To this end, we propose *AxOTreeS*, a methodology that allows the integration of the results of such analyses.

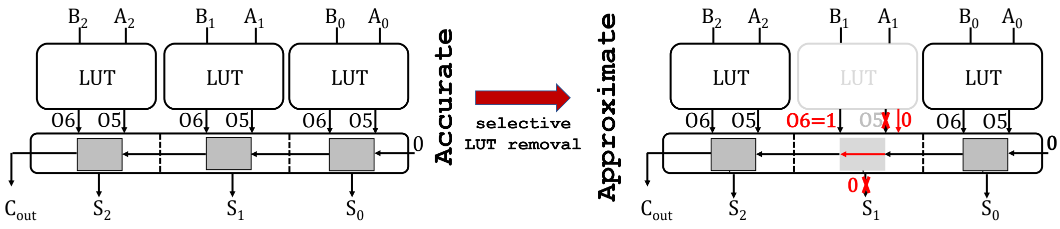


Fig. 2. Approximating a $3 - bit$ unsigned adder design. Selective removal of LUTs results in approximate implementations of the AxOs. The unused LUTs can be utilized by other components of the designs.

3 PROBLEM FORMULATION

3.1 Operator Model

For the current article, we use an operator model similar to that proposed in [36] and also used in [25]. Accordingly, any FPGA-based arithmetic operator can be represented by an 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 or not and L represents the total number of LUTs of the accurate implementation that may be removed to implement approximation. Removing a LUT from the accurate implementation means that the target LUT does not receive any inputs and does not contribute to the computations. So, the accurate implementation can be represented as $O_{Ac}(1, 1, ..., 1)$. For instance, the accurate implementation of the 3-bit unsigned adder, shown in Fig. 2, is represented by the tuple (1,1,1). Similarly, $O = \{O_i\}$ represents the set of all possible implementations of the operator. Therefore, the set O for the adder shown in the figure is { (0,0,0), (0,0,1), (0,1,0),(0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}. The approximate implementation in the figure corresponds to the tuple (1,0,1). In this operator modeling and implementation technique, removing a LUT will always result in a 0 value assigned to the output bit. For example, for approximate adder configuration (1,0,1), output bit S1 is assigned a constant value of 0. Furthermore, the associated carry chain element of a LUT removed from the computation is employed only to forward the preceding carry-out signal to the following carry chain element. An arbitrary operator/application's behavior can be abstracted by a function $\mathcal{S}$. So, the operator/application output for a set of inputs can be outlined as shown in Eq. (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 Eq. (2). Similar to $\mathcal{S}$, the functions $\mathcal{H}_W, \mathcal{H}_U$ and $\mathcal{H}_C$ abstract the physical characterization-based estimates for power dissipation, LUT utilization, and Critical Path Delay (CPD) of any arbitrary approximate operator O_i respectively.

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

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

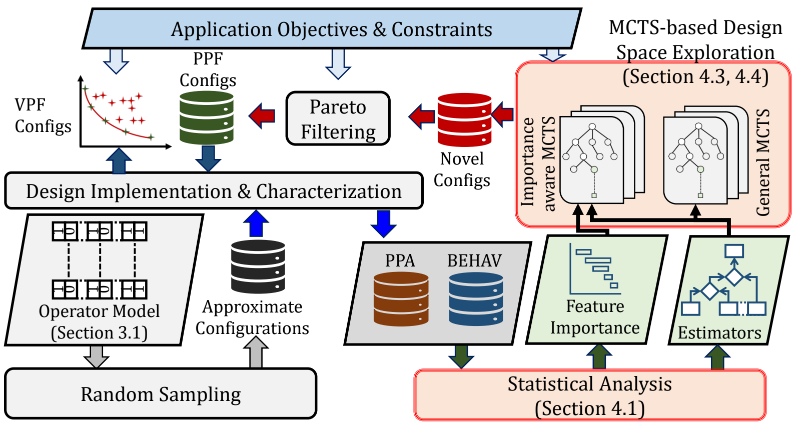

Application Objectives & Constraints
MCTS-based Design Space Exploration (Section 4.3, 4.4)
PPF Configs
VPF Configs
Pareto Filtering
Novel Configs
Importance aware MCTS
General MCTS
Design Implementation & Characterization
Operator Model (Section 3.1)
Approximate Configurations
PPA
BEHAV
Feature Importance
Estimators
Random Sampling
Statistical Analysis (Section 4.1)

Fig. 3. *AxOTreeS* Methodology

3.2 Design Space Exploration

The constrained search problem, with behavioral accuracy (BEHAV) and/or PPA constraints, is shown in Eq. (3), where B_{MAX} and P_{MAX} refer to the BEHAV (error) and PPA metric constraints respectively. Eq. (3) represents a typical multi-objective optimization problem with the goal of finding design solutions that minimize two objectives. A variety of methods are adopted by different optimization tools to achieve the minimization of more than one metric. They include but are not limited to, minimizing a weighted sum of the objectives, solving multiple problems that constrain one objective and minimize the other, minimizing for one objective and then using the solution to minimize the other, etc. For the current article, we use hypervolume metrics obtained by the PPA and BEHAV metric of each design point as the single metric that is used to drive the optimization. It must be noted that the same DSE methodology to search for feasible Pareto-optimal solutions may deploy any of these methods of multi-objective optimization. While Eq. (3) represents the traditional approach opted for this optimization problem, we propose to include LUT-wise significance in the optimization. Therefore, the corresponding objective is shown in Eq. (4), where the set $\mathcal{F}_{\mathcal{L}}$ encodes the importance for each LUT being explored in the design, in terms of its impact on the PPA and BEHAV metrics.

$$\begin{aligned} &\underset{O_i \in O}{\text{minimize}}(BEHAV_{O_i}, PPA_{O_i}) \\ &s.t.\ BEHAV_{O_i} \leq B_{MAX} \ \ and \ \ PPA_{O_i} \leq P_{MAX} \end{aligned} \tag{3}$$

$$\underset{O_i \in O}{\text{minimize}}((BEHAV_{O_i}, PPA_{O_i}) \mid \mathcal{F}_{\mathcal{L}}) \tag{4}$$

4 *AXOTREES*

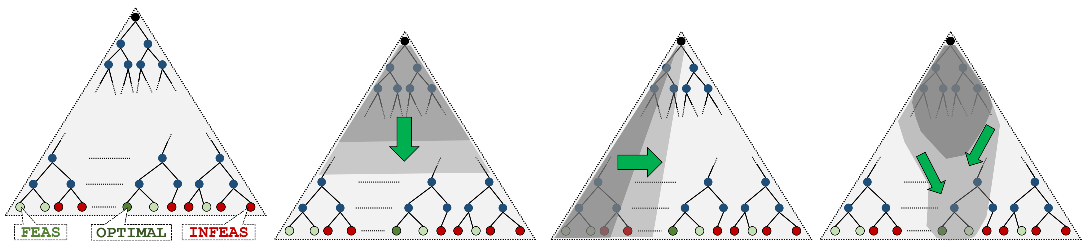
The various aspects of *AxOTreeS* are shown in Fig. 3. Similar colored arrows indicate the information/data flow for each sub-process in the methodology. For instance, *Random Sampling* uses the operator model to generate the approximate configurations that serve as the training set for the models. The designs corresponding to the selected approximate configurations are then implemented, both as standalone operators and as part of applications/accelerators and characterized to generate the PPA and BEHAV data. *Statistical Analysis* involves using the characterization data for generating the ML-based fitness function estimators and LUT-wise significance metrics such as correlation coefficients and feature importance in the ML-based regression models. The ML-based estimators and the LUT-wise significance metrics are used by the MCTS-based DSE, along with the application-specific PPA and BEHAV constraints, to provide a set of Pareto-front design points. This Pareto-front, referred to as Pseudo Pareto-front (PPF), is based on the metrics predicted by the regression models, and the corresponding designs are then characterized to generate the Validated Pareto-front (VPF) approximate design configurations. The primary contributions, as highlighted in the figure, include modeling the DSE for FPGA-based AxOs as a Tree Traversal Problem (TTP) and enabling integrating LUT-wise significance into the search algorithm. Each of these aspects of *AxOTreeS* is explained next.

4.1 Statistical Analysis

4.1.1 Estimator design. The statistical estimation of LUT-wise significance metrics and ML-based estimator design form an integral part of the proposed *AxOTreeS* methodology. While the proposed MCTS-based search methods (described in Sections 4.3 and 4.4) allow the integration of prior knowledge, the requisite problem-specific information is obtained during the statistical analysis. For this purpose, we characterized ~ 2000 randomly generated approximate signed 8×8 multiplier designs with the Baugh-Wooley multiplier's operator model. The characterized dataset was used within an AutoML [22] tool to explore various types of ML-based regression models. The models with the lowest Root Mean Squared Error (RMSE) errors were chosen for the estimator modeling. While training the selected estimators, both RMSE and R2 scores across the training and testing datasets were considered. Trained models with similar R2 scores for both the training and testing datasets were used in the methodology. Mathematically, the ML-based estimators can be abstracted as approximators of true characterization, as shown in Eq. (5). Each PPA and BEHAV metric can be abstracted as a predictive function, characterized by the training dataset and the ML algorithm. Therefore, each of the functions (abstracting physical characterization) in Eq. (2) can be abstracted into its ML-based counterpart as shown in Eq. (5).

$$\begin{aligned}
PPA\ Metric:\quad & PPA_{O_i} \approx \hat{P}_{ML}(O_i)\\
BEHAV\ Metric:\quad & BEHAV_{O_i} \approx \hat{B}_{ML}(O_i)\\
So:\quad & \\
\mathcal{H}_W(O_i, Inputs) &\approx \hat{\mathcal{H}}_W(O_i, \mathcal{W}_{ML}(TRAIN))\\
\mathcal{H}_U(O_i) &\approx \hat{\mathcal{H}}_U(O_i, \mathcal{U}_{ML}(TRAIN))\\
\mathcal{H}_C(O_i) &\approx \hat{\mathcal{H}}_C(O_i, C_{ML}(TRAIN))\\
\mathcal{H}_W(O_i, Inputs) \times \mathcal{H}_C(O_i) &\approx \hat{PDP}(O_i, PDP_{ML}(TRAIN))\\
\mathcal{H}_W(O_i, Inputs) \times \mathcal{H}_C(O_i) \times \mathcal{H}_U(O_i) &\approx \hat{PDPLUT}(O_i, PDPLUT_{ML}(TRAIN))
\end{aligned} \tag{5}$$

4.1.2 LUT-wise significance . The trained ML models for the PPA and BEHAV metric provide fast *surrogates* of the fitness function and have been used extensively in DSE problem solving both for AxOs and in general. However, in *AxOTreeS* we use the ML models additionally for extracting LUT-wise significance metrics. We compute the global SHapley Additive exPlanations (SHAP) values for each LUT and normalize the values across the features to obtain each LUT's relative importance in determining the PPA and BEHAV metric. SHAP [12] is a mathematical method, based on game theory, to provide explanations for each prediction (local) and to compute the importance of each feature for predictions (global). SHAP is being used extensively in explainable AI and can be used in AI-based Electronic Design Automation (EDA) to improve related search methods.



(a) Search tree (b) Breadth First Search (c) Depth First Search (d) Heuristics-based Search

Fig. 4. Tree Traversal methods

Additionally, we also compute to correlation coefficient between the LUTs and the metrics from true characterization data. The SHAP values and the correlation coefficient can be used in *AxOTreeS* to improve the MCTS-based search algorithm. Eq. (6) and (7) show two ways of generating the overall LUT-wise significance metrics, $\mathcal{F}_{\mathcal{L}}$, that is used in the optimization problem shown in Eq. (2). It involves using either the maxima or a weighted sum of the PPA, $f_{l(PPA)}$, and BEHAV, $f_{l(BEHAV)}$, metrics for each LUT l.

$$\begin{aligned} \mathcal{F}max_l &: max(f_{l(PPA)}, f_{l(BEHAV)}) \\ \mathcal{F}_{\mathcal{L}} &= \{\mathcal{F}max_l\} \, \forall l \in \{0, 1, ..., L-1\} \end{aligned} \tag{6}$$

$$\begin{aligned} \mathcal{F}wsum_l &: w_{PPA} \times f_{l(PPA)} + (1 - w_{PPA}) \times f_{l(BEHAV)} \\ \mathcal{F}_{\mathcal{L}} &= \{\mathcal{F}wsum_l\} \, \forall l \in \{0, 1, ..., L-1\} \end{aligned} \tag{7}$$

4.2 Monte Carlo Tree Search

A Tree Traversal Problem (TTP) usually involves modeling the problem as a sequence of decisions and searching for the leaf node that fulfills the problem's objectives. So, while each leaf node encodes a set of choices for all the decisions, every non-leaf node encodes a subset of such choices. As shown in Fig. 4(a), each of the leaf nodes can be categorized as feasible (FEAS) or infeasible (INFEAS) depending upon whether it satisfies the problem-specific constraints. While finding any FEAS leaf node is sufficient for a Constraint Satisfaction Problem (CSP), minimization problems also search for the OPTIMAL leaf node, the best among the FEAS nodes. Exhaustive search approaches such as Breadth First Search (BFS) and Depth First Search (DFS), as shown in Fig. 4(b) and Fig. 4(c) respectively, look to expand all the leaf nodes iteratively while storing a portion of the search tree in memory. The arrows in the figure indicate the general direction in which the nodes of the search tree are expanded. Although exhaustive search methods guarantee finding the optimal solution, such methods are too costly for large design problems. For instance, as seen in Fig. 4(b), the BFS approach needs to store all the nodes in the tree, except the last level, in order to start expanding even the first leaf node.

Alternatively, heuristics-based search methods aim to reach the optimal leaf node by expanding as few nodes of the search tree as possible. As shown in Fig. 4(d), such methods aim to use information regarding each expanded node as a guide to moving towards the OPTIMAL leaf node. For instance, A-star search [7] involves designing some problem-specific heuristics to traverse the search tree more intelligently than expanding every possible node. Monte Carlo Tree Search (MCTS) is one such heuristics-based search method where the problem-specific heuristics are derived from random simulations of pending decisions. So, the quality of each non-leaf node is estimated from the quality

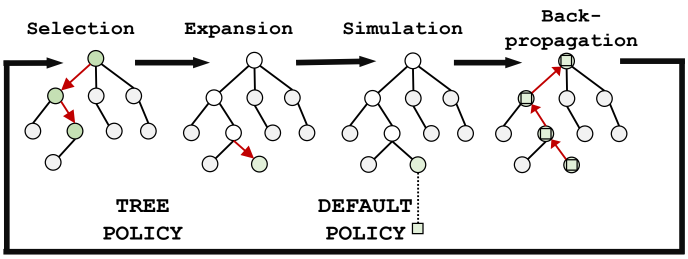


Fig. 5. MCTS stages [3]

of the leaf node obtained by such Monte Carlo simulations over the duration of the search. MCTS includes the following processes, as shown in Fig. 5:

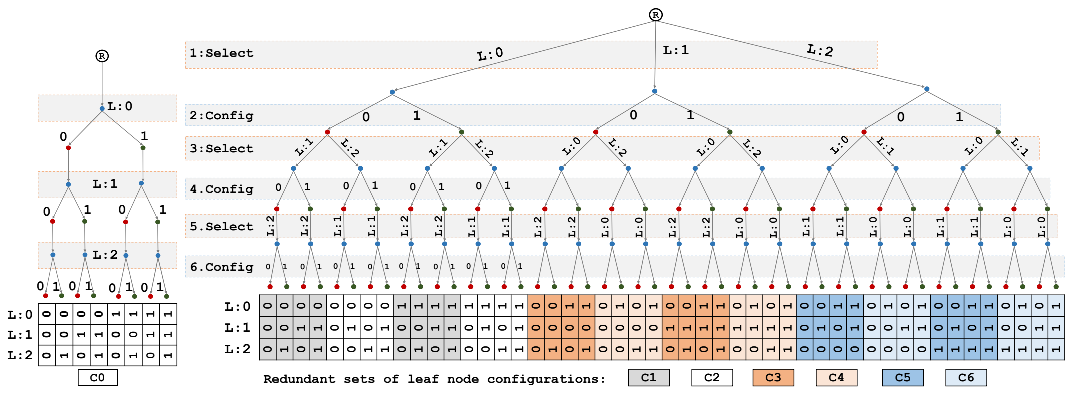
- *Tree policy*: It involves *selecting* the appropriate non-leaf node to perform the Monte Carlo simulations from. It encodes the consideration of the trade-offs between exploration and exploitation to determine the best node to expand.
- *Expansion*: It refers to the process of adding a new node based selected by the Tree policy.
- *Default policy*: It refers to the *rollout* or Monte Carlo simulation from any expanded node to a leaf node.
- *Backpropagation*: It entails estimating the quality of the leaf node resulting from a rollout and updating the rewards for nodes upstream from the rollout node.

Although fairly recent, MCTS-based search optimization is already being used in EDA problems [28]. However, most of such approaches are limited to using either standard problem-agnostic MCTS or introducing some form of parallelism into the rollouts. In contrast, we focus on improving the guided search from existing characterization data, and the *AxOTreeS* approach can the combined with complementary approaches such as parallel MCTS [11, 27].

4.3 MCTS-based DSE

4.3.1 Modelling DSE as a TTP. Based on the operator model for the approximation of the 3-bit unsigned adder shown in Fig. 2, Fig. 6(a) shows the set of all possible approximate LUT configurations. It includes the accurate configuration (111) and the 7 approximate configurations. In the tree, the sequence of decisions from the root node is as follows: *Configure* LUT:0 $\longrightarrow$ for each of the decision (1$\longrightarrow$ use, 0$\longrightarrow$ remove), *Configure* LUT:1 $\longrightarrow$ *Configure* LUT:2 for each configuration decision of LUT:1. While for the small design, we could list out all the possible configurations, for a larger design, for example, the 36 configurable LUTs in a signed 8×8 multiplier, it would be infeasible to do so. Therefore, within a given computational budget, we would be able to expand the search tree only partially. It must be noted that for any arbitrary non-leaf node (say *Nx*) the sub-tree downstream to it denotes the exploration of other decisions given the decision already encoded in *Nx*. Therefore, in a partially expanded search tree the LUT that we decide to configure first, say LUT:0, would also denote the maximum exploration of design decisions for each configuration of LUT:0.

Since we do not have any prior information regarding the optimal order in which the LUTs should be configured, we include the decision regarding the same, as part of the tree search. Consequently, Fig. 6(b) shows the completely expanded search tree for the 3-bit adder with this decision tree model. As seen in the figure, the design decisions in the search tree alternate between selecting the LUT to configure next and what configuration to use for the selected LUT. As can be seen from the figure, it results in a much larger search tree and the number of leaf nodes is 6 times that of the original search tree. The redundant configurations are shown in different colors in the table at the bottom of the figure. It must be noted that this approach of using the *Select* as an



(a) *Config* (b) *Select & Config*

Fig. 6. Decision trees for approximating a 3-bit unsigned adder implementation (a) The decision includes how to *Configure* each LUT and follows a fixed/randomized order among the LUTs (b) The decisions include *Selecting* the LUT to be configured next and *Configuring* the LUT to be used/removed in the approximate operator implmentation

Algorithm 1. MCTS [3]

```
1:  function MCTS-SEARCH($s_0$)
2:  create root node $n_0$ with state $s_0$
3:  loop
4:      while within computational budget do
5:          $n_l \leftarrow TreePolicy(n_0)$
6:          $r_l \leftarrow DefaultPolicy(s(n_l))$
7:          $Backpropagate(n_l, r_l)$
8:      end while
9:  end loop
10: return $a(BestChild(n_0))$
```

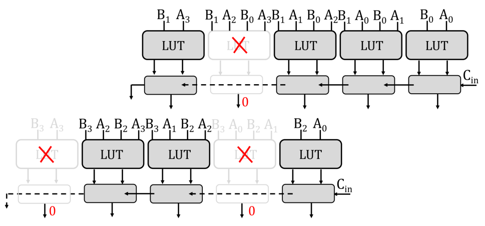


Fig. 7. Approximate partial product generation in signed 4x4 multiplier

additional design decision results in a much larger search tree. However, as we shall show from the experiments, following a fixed order (similar to the search tree in Fig. 6(a)) can result in degraded search results for larger designs.

4.3.2 General MCTS-based DSE. The general MCTS algorithm is shown in Algorithm 1. We use the accurate implementation of the signed 4×4 multiplier, presented in [35], to demonstrate the various steps and different types of MCTS methods discussed in this article. Fig. 7 shows an approximate version of the partial product generator used in the same design. In the configuration shown in the figure, 3 of the 10 LUTs in the accurate version are configured to be not used, as depicted by the shaded LUTs. Similarly, any approximate configuration would be generated by a different combination of the usage of the 10 LUTs, a total of 1024 designs. Fig. 8 shows the search tree generated from the first 30 iterations of using the general MCTS algorithm in the search for corresponding approximate configurations.

In MCTS, for each iteration, the algorithm starts from the root node, denoting the state where no decisions regarding the usage of the LUTs have been finalized. Fig. 8(a) shows the search tree after the first 10 iterations. These 10 iterations are used to expand one node for each of the 10 possible LUTs that can be used to select the first LUT to configure. The rectangular boxes show the leaf node generated from each iteration, as a result of the rollout from the newly expanded node in that iteration. The red and green color indicates configurations as infeasible and feasible

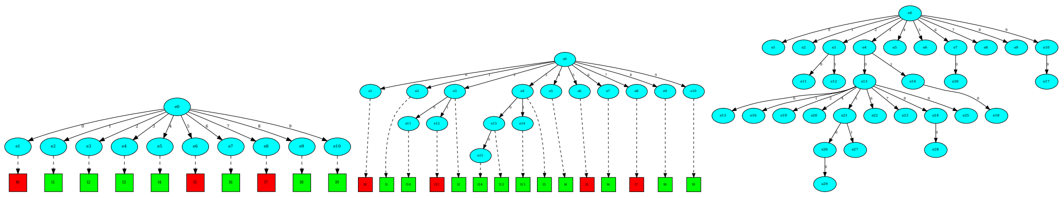
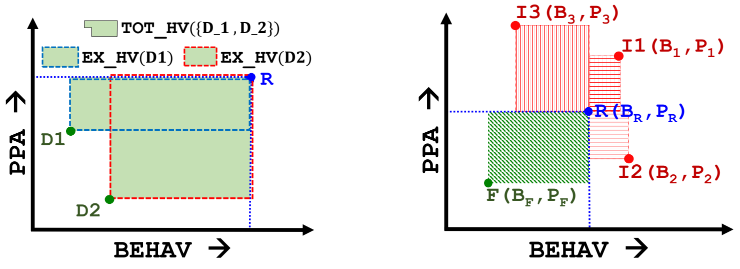

(a) # Iterations: 10 (b) # Iterations: 15 (c) # Iterations: 30

Fig. 8. Progression of the *flatMCTS*-based search tree



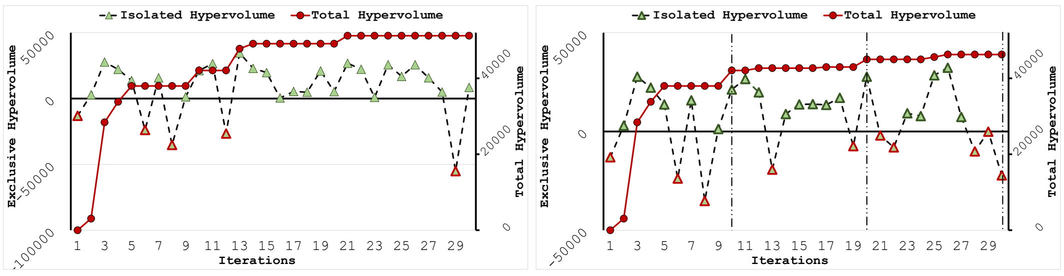
(a) Hypervolume (b) Reward Model

Fig. 9. Reward model used in MCTS-based DSE of *AxOTreeS*. (a) Hypervolume measures: Total hypervolume (TOV_HV) and Exclusive hypervolume (EX_HV) (b) Reward/Penalty for feasible and infeasible solutions

respectively, based on the PPA and BEHAV constraints of the problem. For a given leaf node, which denotes a complete configuration of the 10 LUTs, ML-based estimators are used to estimate the PPA and BEHAV metrics [4]. Based on the estimated metrics, a reward is assigned to the leaf node which is then backpropagated to all the nodes in the path from the root to the newly expanded node, as shown in Fig. 5. The reward assigned to a leaf node should ideally encode the information regarding the feasibility of the configuration of the leaf node as well as a measure of the *goodness* of the configuration, in order to direct the search towards the OPTIMAL leaf node.

Fig. 9 shows the use of hypervolume metrics in formulating the reward for each leaf node of the search tree. Hypervolume is one of the more widely used metrics in multi-objective optimization and can be used in two contexts during DSE— for evaluating each design point in isolation, and for evaluating a set of design points. Fig. 9(a) shows the measure of hypervolume in each of these contexts. It plots two design points, $D1$ and $D2$, in terms of their resulting BEHAV and PPA metrics (for any arbitrary BEHAV and PPA measure). The *exclusive hypervolume* of each of these points is shown as rectangular areas with a dashed outline. It is quantified by the area swept by the design point and a reference point R. For more objectives (> 2), it would be measured by the hypervolume swept by the design point and a reference point. In isolation, the exclusive hypervolume of a design point denotes the quality of the design point in terms of minimizing both design metrics. For a set of design points, say $\{D1, D2\}$, Fig. 9(a) also shows the *total hypervolume* of the set as a six-sided polygon determined by the area swept by each of the design points in the set. It can be noted from the figure that computing the total hypervolume does not duplicate the overlapping exclusive hypervolume and therefore is not impacted by any newly discovered point lying within the existing hypervolume.

[4]It should be noted that actual design implementation and characterization can be used instead of ML-based estimation. For the signed 4×4 multiplier, we characterized all the 1024 configurations and use that table to determine the PPA and BEHAV metrics for each leaf node.



(a) *flatMCTS* (b) *descMCTS*

Fig. 10. Progression of the hypervolume for the General MCTS-based search trees

Fig. 9(b) shows the reward model used in the current article. We use the *exclusive hypervolume* of the leaf node design as the reward metric. The point $R(B_R, P_R)$ denotes the constraints imposed by $BEHAV_{MAX}$ and PPA_{MAX}. So, for any feasible design point, $F(B_F, P_F)$, the area swept between F and R denotes the exclusive hypervolume. For infeasible design points, similar to $I1(B_1, P_1)$, $I2(B_2, P_2)$, and $I3(B_3, P_3)$, we use the negative of the area swept between R and the infeasible point as the reward. Therefore, the sign of the reward serves as the indicator of the feasibility of the design point. Similarly, the magnitude of the reward value serves as the indicator of the *goodness* of the design point. Together, the sign and the magnitude should direct the search toward leaf nodes that are feasible and have higher exclusive hypervolume.

Fig. 8(b) shows the search tree expanded after the next 5 iterations (11 to 15) along with all the 15 leaf nodes. As can be seen, 5 new nodes are expanded. Since all possible decisions at the root node (which LUT to select for deciding its configuration) have been expanded in the first 10 iterations. The decision regarding which note to expand is based on the Tree policy. Fig. 8(c) shows only the expanded modes of the search tree after 30 iterations. It contains 30 expanded nodes (excluding the root node) and the tree is expanded asymmetrically (neither BFS nor DFS). Fig. 10(a) shows the progression of the hypervolume values from the leaf nodes across the 30 iterations. The negative values denote the exclusive hypervolume of the infeasible points (3 within the first 10 iterations). The total hypervolume denotes the Pareto-front hypervolume of the leaf node designs accumulated after each iteration.

As shown in Fig. 8, in the progression of the search tree in the MCTS algorithm described above, the traversal in each iteration starts at the root node. We shall refer to this algorithm as the *flatMCTS* method. Alternatively, Fig. 11 shows another approach where the 30 iterations are evenly split into 3 stages of 10 iterations each. The first stage is similar to that in *flatMCTS*. However, after the first stage, the best child node from the root node is selected as the root node for all iterations in the second stage. This process is repeated after each stage. The three sub-figures in Fig. 11 show the 10 expanded nodes and the leaf nodes for each stage. This process is similar to making decisions for each move in a game where the next move is based on the current *game-state* instead of the initial stage of the game where no moves have been made, which is equivalent to the *flatMCTS* approach. We shall refer to this method as the descending MCTS (*descMCTS*). Fig. 10(b) shows the progression of exclusive and total hypervolume of designs for *descMCTS* method with the 3 stages demarcated by dashed vertical lines.

4.4 Problem-specific adaptations

The *flatMCTS* and *descMCTS* methods described above represent standard MCTS algorithms and do not encode any problem-specific adaptations. They are similar to general game-playing agents that learn the requisite skill for each game through trial and error only. In this section, we propose

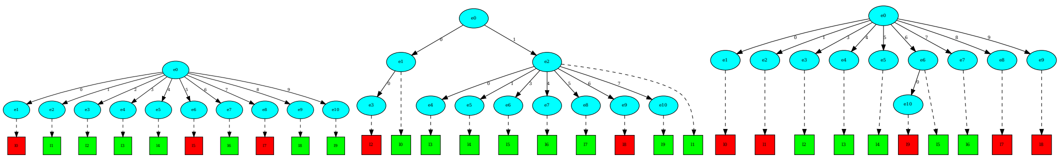

(a) # Iterations: 10 (b) # Iterations: 20 (c) # Iterations: 30

Fig. 11. Progression of the *descMCTS*-based search tree

some problem-specific adaptations to the MCTS-based search for FPGA-based AxOs. The proposed adaptations stem from the drawbacks observed from the experiments with the standard MCTS methods as discussed next.

4.4.1 Enabling deeper search tree expansion. In both the general MCTS methods, all the possible child nodes are expanded before exploring the sub-tree associated with any single child node. This approach does not have much of an adverse impact in case of fewer possible design decision choices, for example for the 0/1 choices for the *Configure LUT* decisions. However, for a larger number of choices, this translates to most of the leaf nodes being generated from a longer rollout. For instance, in the case of the signed 8×8 multiplier with 36 configurable LUTs, all 36 choices of the *Select LUT* for configuration decision have to be expanded before moving to the next level of decision making. Similarly, every time the search method encounters a node where the next decision is *Select LUT*, all the possible child nodes are expanded first before exploring further design decisions for any single child node. As a result, most of the leaf nodes are generated with a majority of the design decisions being taken by randomization rather than by learned heuristics.

In order to alleviate this issue, we use two potential solution approaches. The first one involves limiting the maximum number of child nodes for the *Select LUT* decisions. While this limits the exploration of all the design decision choices at the higher levels (closer to the root node) we leverage the fact that with the proposed modeling there are multiple paths from the root node to any leaf node configuration, as was shown in Fig. 6(b). In fact, Fig. 6(a) and 6(b) represent the corner cases of this method as we can vary the maximum child nodes from 1 to the number of LUTs that are being configured in the design. Additionally, we also explore the method of pre-configuring a subset of the LUTs to 0s or 1s prior to the MCTS-based search. This is equivalent to starting the game from a non-initial state and helps in enabling deeper searches in the tree.

4.4.2 Using LUT-specific statistical information. Both methods/modifications described above allow the use of LUT-specific prior knowledge in the tree search. Specifically, the order in which we explore the choices for the *Select LUT* decision determines the impact of which LUT's configuration is being explored more in the search tree. We use a combination of the PPA- and BEHAV-related feature importance (correlation coefficients or SHAP values) metric of the LUTs to generate this ordering. The combination could be the maximum of the PPA and BEHAV metrics or a weighted sum of the same, as was shown in Eqs. (6), (7). It must be noted that using such problem-specific adaptations introduces additional hyperparameters in the DSE methodology. While determining the appropriate configuration for these hyperparameters is beyond the scope of this article, we evaluate the impact of the variations of the hyperparameters in the next section. Also, given sufficient computation resources, sweeping across these hyperparameters can provide approximate configurations that would potentially result in better-quality of solutions than those obtained through random sampling.

Table 2. ML-based estimator design results

Design	Operator: Signed 8x8 Mult				Application: ECG				Application: GAUSS			
Metric, Model	PDPLUT, Neural Network		AVG_ABS_ERR, CatBoost		PDPLUT, LinReg		1-Accuracy, XGBoost		PDPLUT, LinReg		AVG_PSNR_RED, CatBoost	
Metric	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST
MSE	21306485	34957179	0.00216	0.15873	8.57E+12	8.54E+12	1.00E-05	0.00051	2.3E+14	2.65E+14	0.07485	0.21495
MAE	3531	4465	0.03504	0.27673	2321377	2308061	0.00097	0.01156	11781306	12819703	0.17331	0.28081
R2	0.9936	0.98986	0.99998	0.99857	0.82811	0.83909	0.99698	0.89285	0.9765	0.9726	0.98819	0.96595

Table 3. Top-5 ranked LUTs based on each metric

Rank	CorrPPA	CorrBEHAV	MaxScaledCORR	SumScaledCorr	ShapPPA	ShapBEHAV	MaxRelSHAP	SumRelSHAP
1	lut_25	lut_34	lut_25	lut_25	lut_25	lut_34	lut_25	lut_34
2	lut_29	lut_35	lut_29	lut_29	lut_32	lut_35	lut_34	lut_25
3	lut_34	lut_33	lut_34	lut_31	lut_29	lut_33	lut_32	lut_33
4	lut_33	lut_26	lut_33	lut_24	lut_30	lut_26	lut_29	lut_35
5	lut_31	lut_32	lut_31	lut_28	lut_28	lut_32	lut_30	lut_32

5 EXPERIMENTS AND RESULTS

5.1 Experiment Setup

For the statistical analysis, the approximate design configurations of signed 8×8 multiplier for training were generated, from the data provided in [36]. These configurations are implemented in VHDL and synthesized for the $7VX330T$ device of the Virtex-7 family using Xilinx Vivado 19.2. The synthesis and implementation of each configuration involved multiple executions where we updated the critical path constraint according to the previously achieved critical path slack to obtain highly precise CPD and dynamic power consumption values for each design. The dynamic power is computed by recording the dynamic switching activity for all possible input combinations of the multiplier configurations. For this purpose, we have used Vivado Simulator and Power Analyzer tools. The MCTS and DSE methods are implemented in Python, utilizing packages such as PyGMO [2] and Scikit [20] among others.

5.2 Statistical Analysis

5.2.1 Estimator Design. Table 2 shows the ML models used for the PPA and BEHAV metrics for both operator-level and application-specific DSE. As mentioned earlier, we use the signed 8×8 multiplier as a test case to evaluate the proposed methods. Similarly, the ECG peak detection (employing 1D convolution for LPF in an accelerator) and gaussian image smoothing (using a line buffer-based 2D convolution accelerator) were used for evaluating application-specific DSE. It must be noted that we use the two applications using the signed 8×8 multiplier only to test the methods and the accelerator designs are not contributions of the current article. As mentioned before, AutoML [22] was used to select the specific estimators. It can be noted that the PPA estimators report large regression error values. This can be attributed to the PDPLUT being the product of three different PPA metrics and therefore has large values. We train the models with the hyperparameters provided by the AutoML tool and the only input into the modeling involved efforts to attain similar R2 scores for the training and testing dataset along with reduced RMSE values for both.

5.2.2 Feature Importance. In addition to generating the ML-based estimators, we used the characterization data to generate LUT-wise significance metrics. The dataset was used directly to generate the correlation coefficients between each LUT's usage and the PPA and BEHAV metrics, similar to that shown in Fig. 1. Further, we used the trained ML models to generate the global SHAP values

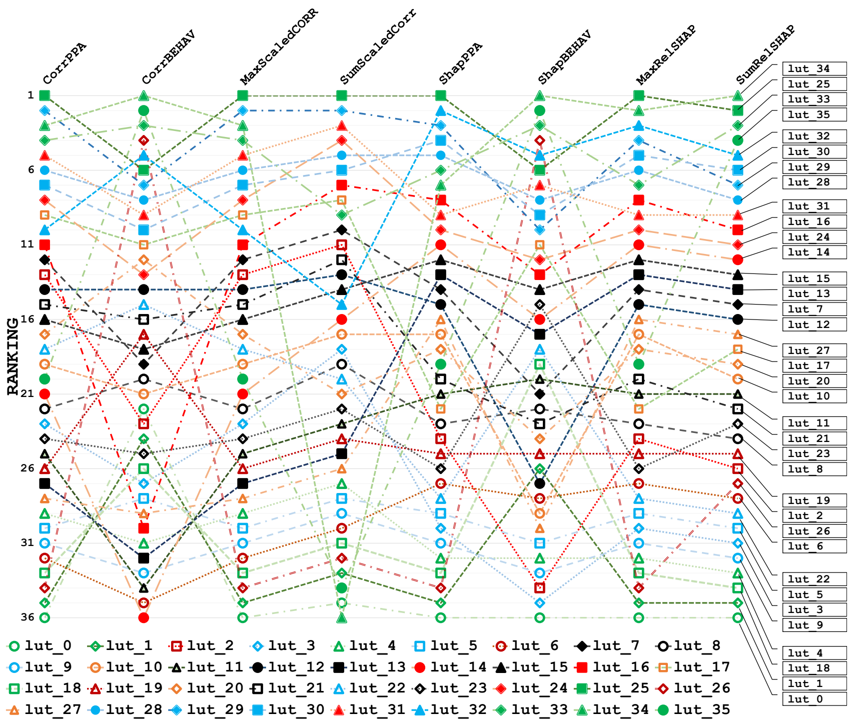


Fig. 12. Ranking LUTs based on LUT-wise significance metrics

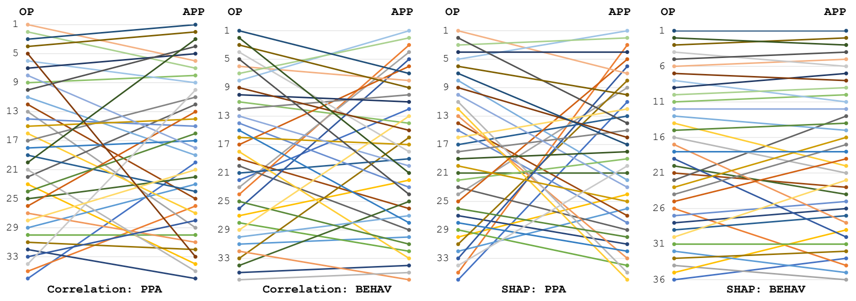
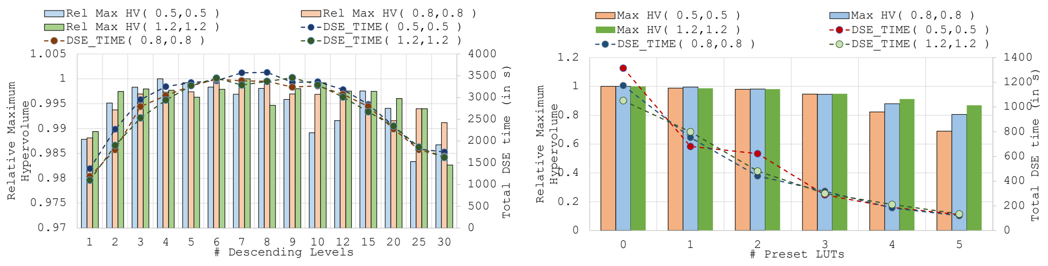


Fig. 13. Comparing LUT-wise significance for operator and application level

for each LUT. These LUT-specific statistics were used to rank the LUTs and resulting the ranking data was used in different scenarios. Fig. 12 shows the *bump-chart* of the LUTs ranking under different criteria such as correlation coefficient with PPA (*CorrPPA*) and BEHAV (*CorrBEHAV*), the maxima of both (*MaxScaledCorr*), the sum of scaled correlation values (scaled between 0 and 1), (*SumScaledCorr*) and similar metrics with SHAP values. As can be seen from the figure, from data corresponding to the signed 8×8 multiplier's operator-level metrics, the importance of LUTs varies widely between PPA and BEHAV metrics. Table 3 lists the top-5 ranked LUTs based on each criteria. It can be noted that LUTs 25 and 29, and, 34,35, and 33 rank highly in PPA- and BEHAV-related statistics respectively. The slope-chart shown in Fig. 13 compares the ranking of LUTs, using the same statistical LUT-wise significance metric, for the operator- and application-level PPA and BEHAV metrics. The ECG data were used for the application-specific (APP) rankings. Only the SHAP values for BEHAV show some consistency among the top-ranked LUTs. The wide variation shown in the figure also motivates the need for the application-specific AxO design.



(a) *flatMCTS* vs. *descMCTS* (b) Presetting LUT usage to 1

Fig. 14. Comparison of maximum hypervolume (relative) and DSE execution times for different variations of the proposed *AxOTreeS* methods in the search for approximate signed 4×4 multipliers. The figures show the comparison of results for three different problems, one for each set constraint scaling factor (0.5,0.5), (0.8,0.8), (1.2,1.2) (a) Relative hypervolume w.r.t. the maximum hypervolume reported across different numbers of descending levels for each problem. (b) Relative hypervolume w.r.t. the maximum hypervolume reported across different numbers preset LUTs for each problem.

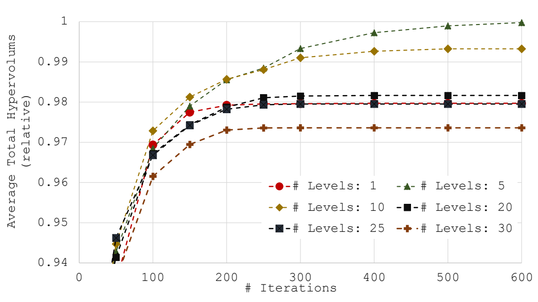
5.3 MCTS Analysis

MCTS, similar to other randomized algorithms, uses random sampling. Therefore, to provide a better evaluation of the effectiveness of MCTS and the proposed methods, we executed the DSE search for signed 4×4 approximate multipliers 50 times for 600 iterations each. Since it provides a smaller problem size (10 LUTs compared to 36 in 8×8 multiplier), we could perform multiple executions of the related search. Further, we used a scaling factor of the maximum PPA and BEHAV values in the training dataset, as constraints to the optimization problem. In the rest of the discussion, the scaling factors are often shown as tuples comprising PPA and BEHAV scaling factors.

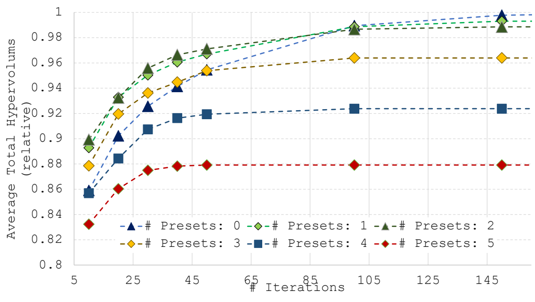
For instance Fig. 14(a) compares the maximum hypervolume obtained by *flatMCTS* and *descMCTS* under a varying number of descending levels for the constrained searches with scaling factors of 0.5x, 0.8x, and 1.2x for the signed 4×4 multiplier. The secondary vertical axis also reports the total elapsed time for the 50 runs of 600 iterations each. The data for the number of descending levels equal to 1 denotes the *flatMCTS*. As evident from the figure, using more levels results in increasing maximum hypervolume obtained during the search. This can be attributed to the deeper searches possible with *descMCTS*. However, at very high values, the quality of the resulting design points starts to reduce. This is due to very few simulations being used to determine the best child for the next stage. For instance, with 5 levels, 120 simulations are used in each stage. However, only 20 simulations are used to determine the best child in the case of 30 levels. Fig. 14(b) shows the effect of presetting some LUTs, based on the ordering as per *SumScaledCorr*. As can be seen from the figure, having more LUTs preset reduces the random sampling required during the rollout and hence requires less time. However, except for a few number of LUTs being preset, the hypervolume reported from the search reduces considerably.

Fig. 15 shows the effect of the number of levels and the presets on the progress of the search method for constraints with a scaling factor of 1.2x for both PPA and BEHAV. As seen in Fig. 15(a), the search with 5 levels improves the hypervolume even after 500 iterations while the one with 30 levels flattens out after around 300 iterations. Similarly, having preset LUTs results in the search beginning with good quality designs (better hypervolume) but cannot find better designs in further iterations as the preset LUTs can curtail the exploitable design space. For instance, having 3 preset LUTs begins with a better hypervolume than with zero, but flattens out after around 100 iterations.

Fig. 16 shows the impact of using the proposed problem-specific adaptations in the DSE for signed 8×8 approximate multipliers. Fig. 16(a) shows the relative hypervolume (compared to maximum)



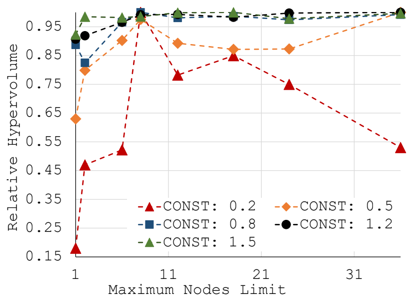
(a) *flatMCTS* vs. *descMCTS*



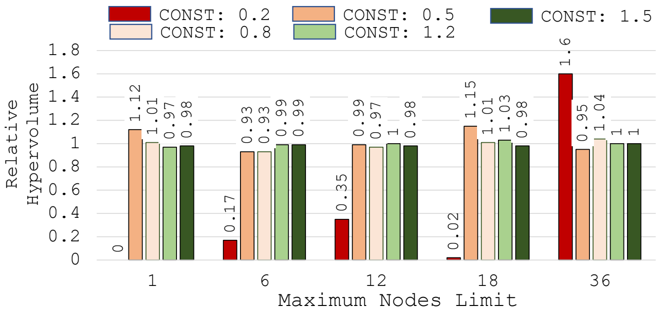
(b) Presetting LUT usage to 1

Fig. 15. Comparison of the progress of the DSE in terms of the hypervolume (relative) for variations of the proposed *AxOTreeS* methods in the search for approximate signed 4×4 multipliers. (a) The values are relative to the maximum final total hypervolume obtained across the DSE runs with varying numbers of descending levels. (b) The values are relative to the maximum final total hypervolume obtained across the DSE runs with varying numbers of preset LUTs.

as the limit on the maximum possible child nodes for *Select LUT* decisions is varied from 1 to 36 in the *flatMCTS* approach. The bound of 36 essentially translates to General MCTS. The line plots correspond to varying constraint scaling values (equal for both PPA and BEHAV). As seen in the figure, the best hypervolume is usually obtained at lower bounds on the expansion, specifically for tighter constraints. Also, the reduced quality of hypervolume for the upper-bound of 1, shows the limitations of not using the *Select LUT* decision as part of the search tree. Similarly, Fig. 16(b) shows the relative hypervolume when using the *MaxRelSHAP* as the ranking metric, while using different expansion limits for varying constraints. The relative hypervolume is compared to that using *no ordering/ranking* information. As shown in the figure, oven even for *flatMCTS* with no limits (up to 36 expandable nodes for *Select LUT*), using the ordering information improves the hypervolume (up to 1.6x), specifically for tighter constraints (0.2x). Therefore the proposed methods of integrating LUT-wise significance information in the search can improve the DSE results significantly.



(a) *flatMCTS* with bounded expansion



(b) Search using LUT importance information

Fig. 16. Comparison of relative hypervolume for proposed methods of limiting the number of expansion nodes for *Select LUT* decision and using ordering/ranking information in the search. The results correspond to the DSE for signed 8 × 8 approximate multipliers. (a) Changing maximum (relative) hypervolume with changing upper-bound on the number of child node expansion. The values are relative to the maximum hypervolume reported for each problem (with a different constraint scaling factor for PPA and BEHAV) across a varying number of maximum child nodes (b) Relative hypervolume in an importance-aware search compared to one without using any LUT-wise significance metrics. The values are relative to the hypervolume reported with a DSE search not using any LUT-wise significance metric, for each problem.

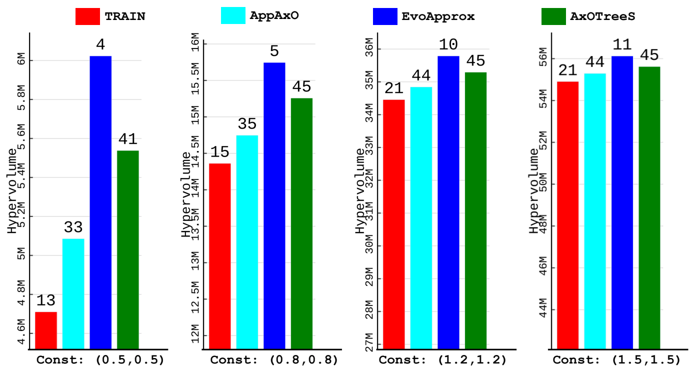


Fig. 17. Comparing the Pareto-front hypervolume of signed 8 × 8 multipliers generated with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

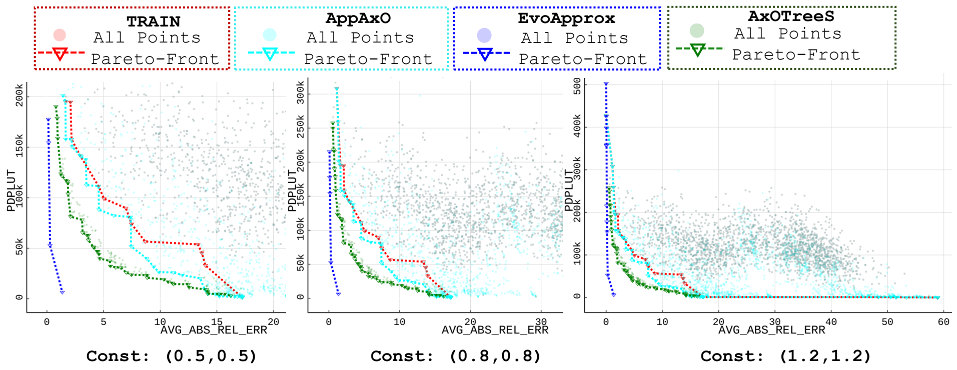


Fig. 18. Comparing the separate Pareto-fronts of signed 8 × 8 multipliers generated with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

5.4 Comparing with State-of-the-art

5.4.1 Operator-level DSE. Fig. 17 shows the comparison of the Pareto-front hypervolume (total hypervolume) of the signed 8×8 multiplier AxOs reported by *AxOTreeS* , that in the training dataset (TRAIN) and those reported in *AppAxO* [36] and *EvoApprox* [16]. Each set of bar plots refers to the solutions generated for a problem with the B_{MAX} and P_{MAX} in Eq. (3) set by the constraint scaling factors (*Const*) and the maximum PPA and BEHAV value in the training dataset. By varying the constraint scaling, we can evaluate the efficacy of the DSE method for different types of problems. Better results at lower values of *Const* signify a method's ability to find design points that minimize both objectives considerably. Similarly, the performance of the method for loosely-constrained problems shows the ability of the method to exploit large tolerances to error in generating low-cost implementations. As shown in Fig. 17, with *AxOTreeS* , we report improved hypervolume than TRAIN and *AppAxO* across all problems. 161 new approximate operators were characterized to obtain the results for *AxOTreeS*. Fig. 18 shows the separate Pareto-fronts obtained with each method for the different problems. As evident from the figure, both *AppAxO* and *AxOTreeS* obtain additional design points by increasing the AVG_ABS_REL_ERR, obtained by removing additional LUTs from the implementation, resulting in lower PDPLUT. While *EvoApprox* results in a better Pareto-front, it is limited to a fixed library of designs and cannot generate more designs to leverage higher error tolerances. As was reported in *AppAxo*, the current operator model suffers from the lack of the

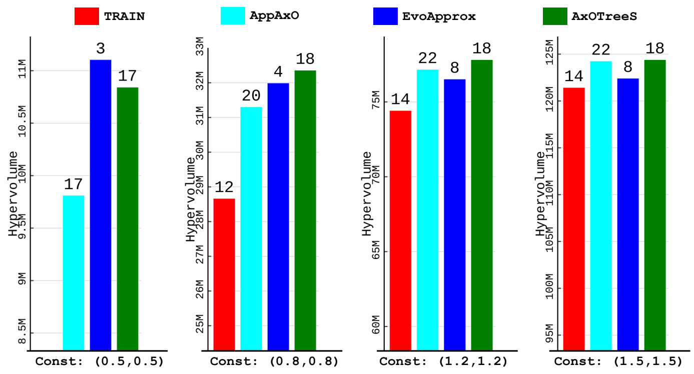


Fig. 19. Comparing the Pareto-front hypervolume of signed 8×8 multipliers generated for ECG peak detection with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

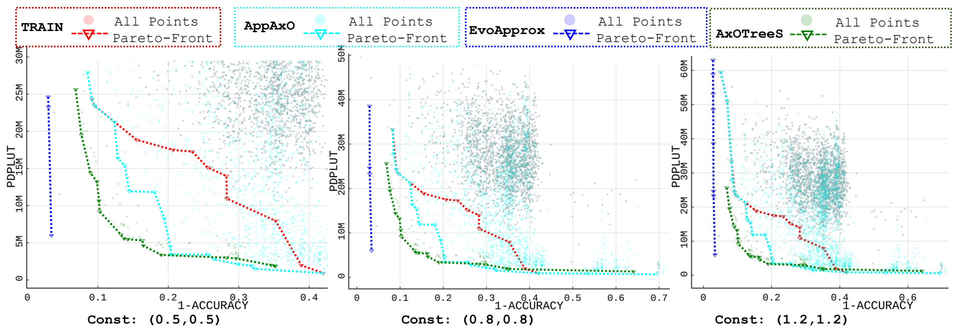


Fig. 20. Comparing the separate Pareto-fronts of signed 8 × 8 multipliers generated for ECG peak detection with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

LUT's *init configuration* as a DoF and hence performs poorly compared to *EvoApprox*. However, the limited DoF is still sufficient to extract beneficial PPA-BEHAV trade-offs for application-specific DSE primarily by leveraging loosely constrained behavioral accuracy requirements.

5.4.2 Application-specific DSE. Similar to Fig. 17 for operator-level DSE, Fig. 19 and Fig. 21 show the comparison of the separate Pareto-front hypervolume for application-specific DSE for ECG and GAUSS respectively. The results for these are derived from 97 and 69 new approximate configurations of signed 8 × 8 multipliers respectively for ECG and GAUSS. For ECG, where AxOs were used in the LPF of peak detection, *EvoApprox* shows better results only for the tightly constrained problem–*Const:(0.5,0.5)*. For all other cases, LUT-level optimization-based approaches, *AppAxO* and *AxOTreeS*, perform better, with *AxOTreeS* showing higher hypervolume than *AppAxO* across all cases. As seen in the Pareto-fronts in Fig. 20, for loosely-constrained problems, *AxOTreeS* generates new design points at the cost of behavioral accuracy.

In the case of Gaussian smoothing, as seen in Fig. 21, *AxOTreeS* reports considerably higher hypervolume than *EvoApprox* across all problems. Only two design points of *EvoApprox* are present on the separate Pareto-front estimations, as seen in Fig. 22. Fig. 23 shows the combined Pareto-front design points, where the design points across the methods are collected together and the contribution of points from each method is shown. As seen in the figure, all points of *EvoApprox* are subsumed by the LUT-level optimization methods, *AppAxO* and *AxOTreeS*, and hence do not show on the combined Pareto-fronts. The application-specific DSE results highlight the importance

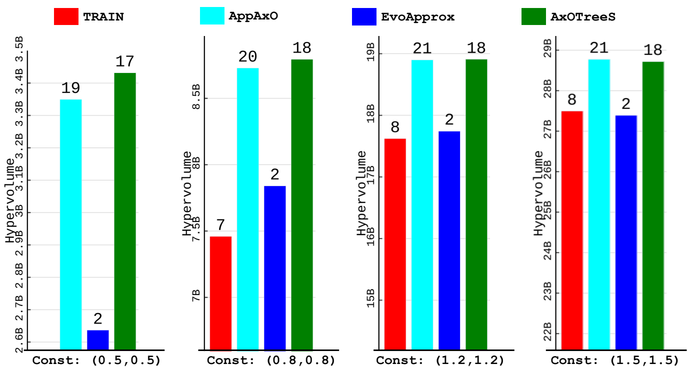


Fig. 21. Comparing the Pareto-front hypervolume of signed 8×8 multipliers generated for Gaussian Smoothing with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

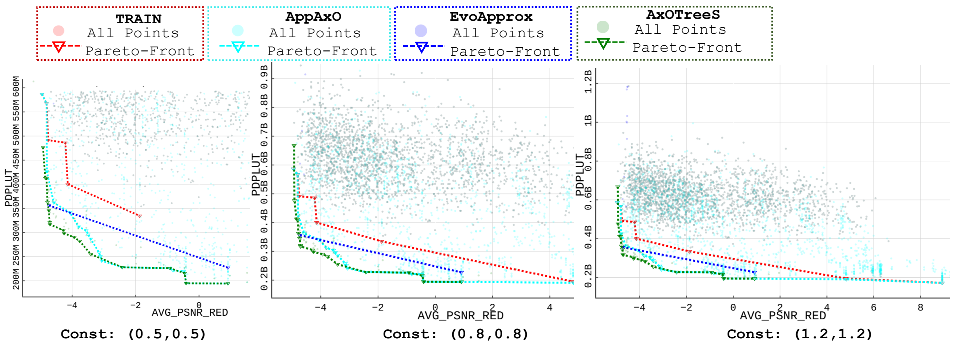


Fig. 22. Comparing the separate Pareto-fronts of signed 8 × 8 multipliers generated for Gaussian Smoothing with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

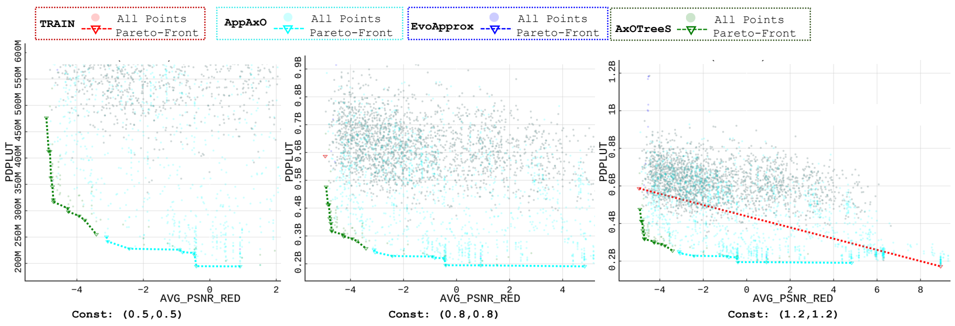


Fig. 23. Comparing the combined Pareto-fronts of signed 8 × 8 multipliers generated for Gaussian Smoothing with *AxOTreeS* to that in the training set and related works. *Const: (x,x)* refers to the constraints set to *x* times the maximum PPA and BEHAV metric in the TRAIN dataset.

of a platform-aware design methodology to extract the most usable benefits from approximate computing. A methodology that can generate novel designs tailored for an application/use case would always be more beneficial than selecting just from already existing design points.

6 CONCLUSION

The current article proposes a methodology for leveraging the statistical analysis of design characterization data for improving the efficacy of the DSE for FPGA-based approximate computer arithmetic. The proposed approach improves upon the traditional usage of characterization data just for the design of regression models to predict the PPA and BEHAV metrics during iterative optimization searches. The proposed methodology uses MCTS as the optimization algorithm of choice. Although MCTS is primarily envisaged as a problem-agnostic optimization algorithm, *AxOTreeS* introduces problem-specific adaptations to MCTS. The LUT-wise significance results, derived from the statistical analysis of the characterization data, drive these adaptations and, as reported in the article, result in improved quality of designs from the corresponding DSE. We also report improved quality of results than those reported in related works for both operator-level and application-specific DSE, especially for more loosely constrained optimization problems. It must be noted that, although tested with a specific operator model, the contributions of the article are orthogonal to the operator model and its complexity. The problem-agnostic nature of generic MCTS and the proposed statistical analysis methods make the approach in *AxOTreeS* complimentary to any operator model that allows automated synthesis from an approximate configuration representation. However, the proposed methods also introduce multiple hyperparameters such as the depth of the *descMCTS* method, the number of LUTs to be preset, the upper bound of the expansion stage for the choice of the next LUT to configure, etc. Similar to other metaheuristic optimization methods, finding the appropriate combination of hyperparameters poses a challenge.

The current article focuses primarily on the following aspects - adapting MCTS for problem-specific improvements and using the characterization data more effectively in the DSE. Further research related to the first aspect could involve developing heuristics for deciding the appropriate set of hyperparameters for a problem. Similarly, research into the second aspect may include using the recent advances in AI/ML methods such as generative AI, explainable AI[5], and other data analysis methods to extract additional knowledge from the characterization data. A rather straightforward extension of the current work may entail initializing a partial search tree for MCTS from the already existing characterization data. Further, parallel MCTS can be implemented for the current problem statement.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Sara Achour and Martin C Rinard. 2015. Approximate computation with outlier detection in topaz. *Acm Sigplan Notices* 50, 10 (2015), 711–730.

[2] Francesco Biscani and Dario Izzo. 2020. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software* 5, 53 (2020), 2338. https://doi.org/10.21105/joss.02338

[3] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[4] Vincent Camus, Christian Enz, and Marian Verhelst. 2019. Survey of Precision-Scalable Multiply-Accumulate Units for Neural-Network Processing. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 57–61. https://doi.org/10.1109/AICAS.2019.8771610

[5] Vinay Kumar Chippa, Debabrata Mohapatra, Kaushik Roy, Srimat T Chakradhar, and Anand Raghunathan. 2014. Scalable effort hardware design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 9 (2014), 2004–2016.

[5]SHAP analysis used here forms a part of explainable AI

[6] Siddharth Gupta, Salim Ullah, Kapil Ahuja, Aruna Tiwari, and Akash Kumar. 2020. Align: A highly accurate adaptive layerwise log_2_lead quantization of pre-trained neural networks. *IEEE Access* 8 (2020), 118899–118911.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[8] Soheil Hashemi, R Iris Bahar, and Sherief Reda. 2015. DRUM: A dynamic range unbiased multiplier for approximate applications. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 418–425.

[9] Hou-Jen Ko and Shen-Fu Hsiao. 2011. Design and Application of Faithfully Rounded and Truncated Multipliers With Combined Deletion, Reduction, Truncation, and Rounding. *IEEE Transactions on Circuits and Systems II: Express Briefs* 58, 5 (2011), 304–308. https://doi.org/10.1109/TCSII.2011.2148970

[10] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. 2011. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *2011 24th Internatioal Conference on VLSI Design*. 346–351. https://doi.org/10.1109/VLSID.2011.51

[11] Anji Liu, Yitao Liang, Ji Liu, Guy Van den Broeck, and Jianshu Chen. 2020. On effective parallelization of monte carlo tree search. *arXiv preprint arXiv:2006.08785* (2020).

[12] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[13] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. 2015. Doppelgänger: a cache for approximate computing. In *Proceedings of the 48th International Symposium on Microarchitecture*. 50–61.

[14] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* 48, 4, Article 62 (March 2016), 33 pages. https://doi.org/10.1145/2893356

[15] Vojtech Mrazek, Muhammad Abdullah Hanif, Zdenek Vasicek, Lukas Sekanina, and Muhammad Shafique. 2019. 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* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 123, 6 pages. https://doi.org/10.1145/3316781.3317781

[16] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. 2017. 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*. 258–261. https://doi.org/10.23919/DATE.2017.7926993

[17] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. 2016. Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks. In *Proceedings of the 35th International Conference on Computer-Aided Design* (Austin, Texas) *(ICCAD '16)*. Association for Computing Machinery, New York, NY, USA, Article 81, 7 pages. https://doi.org/10.1145/2966986.2967021

[18] Vojtech Mrazek, Lukas Sekanina, and Zdenek Vasicek. 2020. Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10, 4 (2020), 406–418. https://doi.org/10.1109/JETCAS.2020.3032495

[19] Jiapu Pan and Willis J. Tompkins. 1985. A Real-Time QRS Detection Algorithm. *IEEE Transactions on Biomedical Engineering* BME-32, 3 (1985), 230–236. https://doi.org/10.1109/TBME.1985.325532

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[21] Nicola Petra, Davide De Caro, Valeria Garofalo, Ettore Napoli, and Antonio G. M. Strollo. 2010. Truncated Binary Multipliers With Variable Correction and Minimum Mean Square Error. *IEEE Transactions on Circuits and Systems I: Regular Papers* 57, 6 (2010), 1312–1325. https://doi.org/10.1109/TCSI.2009.2033536

[22] Aleksandra Płońska and Piotr Płoński. 2021. MLJAR: State-of-the-art Automated Machine Learning Framework for Tabular Data. Version 0.10.3. https://github.com/mljar/mljar-supervised

[23] Bharath Srinivas Prabakaran, Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, and Muhammad Shafique. 2020. ApproxFPGAs: Embracing ASIC-Based Approximate Arithmetic Components for FPGA-Based Systems. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. https://doi.org/10.1109/DAC18072.2020.9218533

[24] Bharath Srinivas Prabakaran, Semeen Rehman, Muhammad Abdullah Hanif, Salim Ullah, Ghazal Mazaheri, Akash Kumar, and Muhammad Shafique. 2018. DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 917–920. https://doi.org/10.23919/DATE.2018.8342140

[25] Rohit Ranjan, Salim Ullah, Siva Satyendra Sahoo, and Akash Kumar. 2023. SyFAxO-GeN: Synthesizing FPGA-Based Approximate Operators with Generative Networks. In *Proceedings of the 28th Asia and South Pacific Design Automation*

Conference (Tokyo, Japan) *(ASPDAC '23)*. Association for Computing Machinery, New York, NY, USA, 402–409. https://doi.org/10.1145/3566097.3567891

[26] Semeen Rehman, Walaa El-Harouni, Muhammad Shafique, Akash Kumar, Jorg Henkel, and Jörg Henkel. 2016. Architectural-space exploration of approximate multipliers. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/2966986.2967005

[27] Kamil Rocki and Reiji Suda. 2011. Large-Scale Parallel Monte Carlo Tree Search on GPU. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 2034–2037. https://doi.org/10.1109/IPDPS.2011.370

[28] Siva Satyendra Sahoo and Akash Kumar. 2021. Using Monte Carlo Tree Search for EDA – A Case-study with Designing Cross-layer Reliability for Heterogeneous Embedded Systems. In *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*. 1–6. https://doi.org/10.1109/VLSI-SoC53125.2021.9606987

[29] Ilaria Scarabottolo, Giovanni Ansaloni, and Laura Pozzi. 2018. Circuit carving: A methodology for the design of approximate hardware. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 545–550. https://doi.org/10.23919/DATE.2018.8342067

[30] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. 2015. A Low Latency Generic Accuracy Configurable Adder. In *Proceedings of the 52nd Annual Design Automation Conference* (San Francisco, California) *(DAC '15)*. Association for Computing Machinery, New York, NY, USA, Article 86, 6 pages. https://doi.org/10.1145/2744769.2744778

[31] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann. 2021. TinyML: Current Progress, Research Challenges, and Future Roadmap. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1303–1306. https://doi.org/10.1109/DAC18074.2021.9586232

[32] Salim Ullah and Akash Kumar. 2023. *Introduction: Approximate Arithmetic Circuit Architectures for FPGA-based Systems*. Springer International Publishing, Cham, 1–26. https://doi.org/10.1007/978-3-031-21294-9_1

[33] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. 2018. SMApproxlib: Library of FPGA-Based Approximate Multipliers. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 157, 6 pages. https://doi.org/10.1145/3195970.3196115

[34] Salim Ullah, Tuan Duy Anh Nguyen, and Akash Kumar. 2020. Energy-efficient low-latency signed multiplier for FPGA-based hardware accelerators. *IEEE Embedded Systems Letters* 13, 2 (2020), 41–44.

[35] Salim Ullah, Semeen Rehman, Muhammad Shafique, and Akash Kumar. 2021. High-Performance Accurate and Approximate Multipliers for FPGA-based Hardware Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/TCAD.2021.3056337

[36] Salim Ullah, Siva Satyendra Sahoo, Nemath Ahmed, Debabrata Chaudhury, and Akash Kumar. 2022. AppAxO: Designing Application-specific Approximate Operators for FPGA-based Embedded Systems. *ACM Transactions on Embedded Computing Systems (TECS)* (2022).

[37] 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* (Knoxville, TN, USA) *(GLSVLSI '23)*. Association for Computing Machinery, New York, NY, USA, 671–677. https://doi.org/10.1145/3583781.3590222

[38] Salim Ullah, Hendrik Schmidl, Siva Satyendra Sahoo, Semeen Rehman, and Akash Kumar. 2021. Area-Optimized Accurate and Approximate Softcore Signed Multiplier Architectures. *IEEE Trans. Comput.* 70, 3 (2021), 384–392. https://doi.org/10.1109/TC.2020.2988404

[39] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2015. Computing approximately, and efficiently. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 748–751.

[40] E. George Walters. 2014. Partial-product generation and addition for multiplication in FPGAs with 6-input LUTs. In *2014 48th Asilomar Conference on Signals, Systems and Computers*. 1247–1251. https://doi.org/10.1109/ACSSC.2014.7094659

[41] E. George Walters. 2016. Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs. *Computers* 5, 4 (2016). https://doi.org/10.3390/computers5040020

[42] Shibo Wang and Pankaj Kanwar. 2019. BFloat16: The secret to high performance on Cloud TPUs. *Google Cloud Blog* (2019).

[43] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. 2013. On reconfiguration-oriented approximate adder design and its application. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 48–54. https://doi.org/10.1109/ICCAD.2013.6691096

[44] Shihui Yin, Gaurav Srivastava, Shreyas K Venkataramanaiah, Chaitali Chakrabarti, Visar Berisha, and Jae-sun Seo. 2017. Minimizing area and energy of deep learning hardware design using collective low precision and structured compression. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 1907–1911.