A Holistic FPGA Architecture Exploration Framework for Deep Learning Acceleration

Jiadong Zhu
HKUST(GZ)

Dongsheng Zuo
HKUST(GZ)

Yuzhe Ma
HKUST(GZ)

Abstract

FPGAs have become a promising solution for accelerating deep learning (DL) workloads because of their inherent reconfigurability and heterogeneous architecture, which effectively handles specific computing tasks. Previous works have proposed various modifications to FPGA architectures for DL acceleration. However, they mainly focus on manual architecture designs, making it difficult to handle multiple scenarios and potentially limiting exploration of the search space. We propose a holistic automatic framework to explore FPGA architectures tailored for DL acceleration. By modifying and integrating CAD tools, we enable automated architecture generation and evaluation. This is combined with a multi-objective Tree-structured Parzen Estimator (TPE) algorithm to iterate the exploration process for finding optimal solutions. Experimental results show that the optimized architectures outperform all the baseline architectures in both delay and the area-delay product (ADP). Furthermore, our results achieve a 29.4% increase in hypervolume and an 89.5% reduction in average distance to reference set (ADRS).

ACM Reference Format:
Jiadong Zhu, Dongsheng Zuo, and Yuzhe Ma. 2025. A Holistic FPGA Architecture Exploration Framework for Deep Learning Acceleration. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3658617.3697698

1 Introduction

Deep learning (DL) has significantly boosted many applications, such as computer vision and natural language processing. To efficiently accelerate these complex algorithms, FPGAs have become popular computing platforms. The soft logic architecture in FPGA offers flexibility in functionality and precision to adapt to rapidly changing DL algorithms. FPGA's heterogeneous hard block architecture provides large-scale parallelism and acceleration for some specific high-density computing. Consequently, these architectural features allow FPGAs to achieve a unique balance between efficiency and programmability among various computing platforms. To maximize the potential of FPGAs in DL acceleration, it is essential to explore the FPGA architecture, which should meet the diverse demands of DL tasks, reduce delay, and minimize area.

ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01…$15.00
https://doi.org/10.1145/3658617.3697698

There has been some work paving the way for FPGA architecture exploration. COFFE 2 [1] is designed to model increasingly complex and heterogeneous FPGA architectures accurately. Verilog-To-Routing (VTR) project [2] is recognized as the leading open-source suite of CAD tools for FPGA architecture. Based on these tools, many previous works have focused on manually improving existing blocks or adding new ones. The Xilinx Versal [3] and Intel Stratix 10 NX [4, 5] are pioneering designs optimized for AI workloads by integrating hard blocks to enhance efficiency in high-density tensor and matrix operations. Arithmetic units are integrated into configurable logic blocks (CLBs) to enhance the efficiency of multiply-accumulate (MAC) operations [6, 7]. Zgheib *et al.* [8] re-evaluates the impact of various cluster parameters on FPGA performance. Enhancements in digital signal processors (DSP) blocks for DL workloads primarily aim to improve the performance of low-precision multiplications [9, 10]. Integrating in-memory compute capabilities into FPGA block RAMs (BRAMs) is explored through Compute-Capable Block RAMs [11] and CoMeFa [12]. Tensor Slice [13] incorporates processing elements (PEs) designed to support multiple tensor operations and precisions. It achieves a 1.63× increase in frequency and a 55% decrease in both area and routing wirelength on average when compared to the baseline. These advancements in previous works allow FPGAs to deliver significantly improved performance in DL tasks.

However, designing architectures manually makes it difficult to meet the requirements of different scenarios. For example, modifying a design to prioritize area, delay, or an optimal balance of both can be tedious and time-consuming. Moreover, the manual designs mentioned above mainly focus on specific parts of the FPGA architecture. Exploring the entire FPGA architecture involves a much larger design space, which makes manual design inefficient, especially when aiming to achieve solutions oriented toward various metrics in multi-objective optimization. When focusing on local architectures, those works' global architectures are often overlooked and designed based on experience. Therefore, algorithm-based automation is crucial for the exploration. Some works have employed algorithms for the design space exploration (DSE) of local FPGA architectures. In General Routing Block (GRB) [14], the simulated annealing (SA) algorithm is applied to explore the design space of FPGA routing architectures. GRAEBO [15] presents an approach for exploring general routing architecture in FPGAs using the Tree-Structured Parzen Estimator (TPE) method [16], outperforming existing architectures and those explored via SA. However, these implementations focus only on exploring the FPGA routing architecture and do not specifically optimize for DL workloads. They convert the multi-objective optimization problem into a single-objective one through simple weighted multiplication, which potentially leads to inferior results.

We propose a holistic exploration framework to efficiently explore the vast search space for global FPGA architecture designs targeting DL acceleration. Compared to designing architectures manually, our framework provides an automated process to efficiently generate a

set of complete FPGA architectures suitable for scenarios oriented toward various metrics. This framework implements a DSE algorithm based on Bayesian optimization (BO), suitable for optimizing high-cost problems such as the time-consuming exploration of FPGA architecture. The algorithm efficiently explores the discrete space of architecture parameters and utilizes hypervolume as the primary guide for the BO to explore and trade off the area and delay. In addition, constructing an FPGA architecture can be challenging for non-specialized researchers, so an automatic process that generates and evaluates the architecture would be ideal. By creating an FPGA architecture template that supports PE arrays for matrix operations and seamlessly integrating CAD tools into the automatic framework, we provide a workflow from parameterized FPGA architecture inputs to complete architecture generation and metric evaluation. Paired with the Koios benchmark suite [17] for DL workloads, it offers an efficient platform for DL-optimized FPGA architecture exploration.

In summary, the contributions are as follows:

- We propose the first holistic automatic FPGA architecture optimization framework specifically designed for DL acceleration.
- We build an FPGA architecture template suitable for DL workloads and seamlessly integrate COFFE and VTR into the exploration framework.
- We implement a hypervolume-aware Bayesian optimization algorithm with a surrogate model and acquisition function tailored for multi-objective FPGA architecture exploration.
- Experimental results demonstrate the effectiveness of our framework on various DL benchmarks, outperforming all baseline architectures and optimization algorithms and obtaining near Pareto-optimal parameter combinations.

2 Preliminaries

2.1 FPGA Architecture

Modern FPGAs typically consist of I/Os, CLBs, DSPs, BRAMs, and other custom hard blocks like PE arrays, all connected via configurable interconnects. A CLB with I inputs comprises N basic logic elements (BLEs) along with local interconnect. Each BLE can implement a K-input logical function using a K-input lookup table (LUT), which can be divided into two LUTs with one less input each. A sparse crossbar with F_{clocal} population density is commonly used in the internal connections of each CLB. DSPs typically comprise multipliers and adders to execute MAC operations. BRAMs of a size S_{RAM} offer configurable width and depth. Routing channels have the same width, and the wire segments within them can vary in length.

2.2 Pareto Optimality

Definition 1 (Dominance) *When smaller values are better, an objective vector $f(x)$ dominates $f(x')$ (denoted as $f(x) \succeq f(x')$) if:*

$$\begin{aligned} &\forall i \in [1,n], f_i(x) \le f_i(x') \\ \text{and} \quad &\exists j \in [1,n], f_j(x) < f_j(x'). \end{aligned} \tag{1}$$

$f(x)$ strictly dominates $f(x')$ (denoted as $f(x) \succ f(x')$) if:

$$\forall i \in [1,n],\ f_i(x) < f_i(x'). \tag{2}$$

$f(x)$ is incomparable with $f(x')$ (denoted as $f(x) \parallel f(x')$) if neither $f(x) \succeq f(x')$ nor $f(x') \succeq f(x)$.

Definition 2 (Pareto Optimality) *A point x is Pareto-optimal if no other point x' exists such that $f(x')$ dominates $f(x)$.*

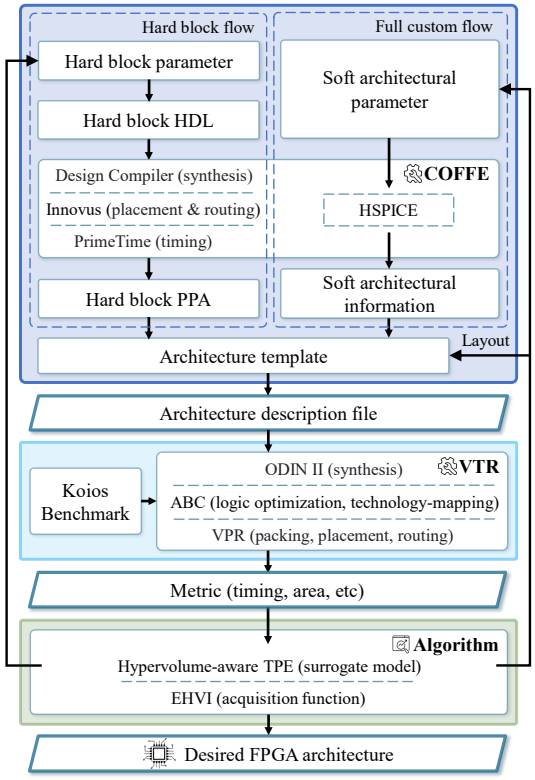

Figure 1: Overview of the proposed integrated FPGA architecture exploration framework.

Definition 3 (Non-domination Rank [18]) *The first front F_1 (Pareto front) with rank(1) contains solutions that are not dominated by any other one. The second front F_2 with rank(2) contains solutions that are only dominated by those in F_1. This process continues until all solutions are sorted into fronts with different non-domination ranks.*

3 Exploration Framework Overview

3.1 Proposed Exploration Flow

Our proposed framework is illustrated in Figure 1. Each architecture design is abstracted into two inputs: hard block design parameters and other soft architectural parameters not related to the heterogeneous part. These inputs are processed in an integrated CAD flow comprising COFFE 2 [1] and VTR 8 [2]. The COFFE part of the integrated flow generates a resulting architecture description file based on parameter inputs and an architecture template. This description file is then utilized by the VTR part along with a set of Verilog HDL benchmarks to evaluate the corresponding area and delay values. The hypervolume-aware TPE method models these designs based on their metrics and maximizes the acquisition function to determine the architecture designs for sampling. This process iterates until the desired optimized architecture is achieved.

3.2 Architecture Template

As the basis of our architecture generation and exploration, an architecture template is established first. The template includes columns of CLBs, DSPs, PE arrays, and BRAMs, with I/Os positioned along the FPGA perimeter. Each CLB comprises several BLEs, each containing a fracturable LUT and two flip-flops. The complex DSP

blocks employed are Intel Agilex-like [19, 20], supporting fixed-point and floating-point precisions. BRAM blocks have registered inputs and outputs and support both true and simple dual-port modes. The PE arrays, equipped as additional hard blocks in the template, are similar to the tensor slice [13]. We reproduce the tensor slice's support for int8 and int16 precisions, as well as for matrix-matrix and matrix-vector multiplication in tensor mode, which are common precisions and computations in similar PE arrays. The primary difference between the implementation of this array and the original tensor slice lies in the PE's MAC portion. Our implementation employs the Schoolbook decomposition method [21] to split 16-bit multiplication, which requires four fewer 8-bit adders than the tensor slice for 16-bit multiplication. After implementing the new PEs, they are verified for functional correctness.

Additionally, the template employs unidirectional routing, utilizing wire segments of lengths 4 and 16, denoted as $L4$ and $L16$. SBs utilize a custom switching pattern identical to the architecture used to evaluate the tensor slice [13]. CBs' input flexibility F_{cin} and output flexibility F_{cout} are set to 0.15 and 0.1, as COFFE's defaults.

3.3 CAD Flow Integration

3.3.1 COFFE Flow for Characterization. COFFE supports a heterogeneous flow, incorporating both a hard block flow for characterizing hard blocks and a full custom flow for characterizing the remaining FPGA architecture, including CLBs and interconnects. Thus, architecture parameters are categorized into two parts accordingly.

Specifically, in the hard block flow for generating hard block power, performance, and area (PPA), RTL synthesis is conducted through the Design Compiler [22], followed by placement and routing (P&R) with Innovus [23] and static timing analysis (STA) using PrimeTime [24]. As detailed in Figure 2, some hard blocks support multiple modes with different delays. For example, suppose a hard block supports both int and float precision modes. In that case, we measure the respective maximum delay for each mode in HDL and record these values in the corresponding sections of the architecture template file. Since VTR requires the minimum-width transistor area (MWTA) for subsequent processing, the Innovus results need to be converted accordingly.

In the full custom flow, HSPICE [25] simulates CLBs and interconnects. When describing CLBs, internal interconnects must be considered in addition to area and delay, which is unnecessary for hard blocks as they are treated as black boxes. The flow also describes the routing wires, switch blocks (SBs), and connection blocks (CBs) that connect various blocks. Then, the generated hard block PPA, soft architectural information, and the layout parameters are input into the architecture template. This forms a resulting FPGA architecture description file for each iteration, specifying the block types and quantities, interconnects, and layout.

3.3.2 VTR Flow for Evaluation. In the VTR flow, the architecture description file and Verilog HDL benchmarks are used as the input. The benchmarks used are chosen from Koios benchmark suite [17], which is designed explicitly for DL and ideal for exploring FPGA architectures tailored to DL workloads. As listed in Table 1, they cover int8 and int16 precisions and include dense operations like matrix-matrix and matrix-vector multiplication. ODIN II synthesizes the Verilog HDL benchmark into a netlist, ABC performs technology-independent logic optimization and maps the circuit into LUTs. Then,

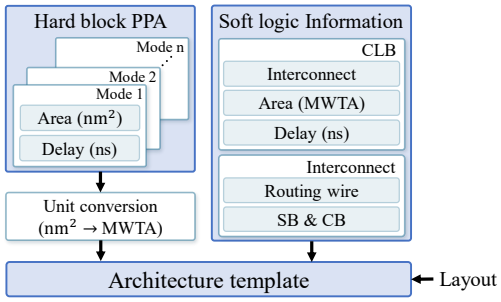

Figure 2: Detailed architecture file generation flow.

Table 1: DL benchmarks used for architecture evaluation

Benchmark	Precision	Array Mode	Description
attention_layer	int16	mat-vec	Self-attention layer
conv_layer	int16	mat-mat	Convolution layer
lstm	int16	mat-vec	LSTM layer
tpu	int8	mat-mat	Google's TPU v1 like
fcl	int8	mat-mat	Fully connected layer

VPR packs the netlist into more coarse-grained logic blocks, performs P&R, and generates evaluation reports with metrics.

It is worth noting that VTR flow automatically maps combinational arithmetic operators such as multiplication and addition but does not support the automatic mapping of custom hard blocks. It requires the manual instantiation of hard blocks in benchmark designs [26]. Consequently, a pre-built library of hard block HDL descriptions and a corresponding benchmark library that instantiates those hard blocks are created. Each benchmark in the library contains two versions that instantiate PE arrays of sizes 4×4 and 8×8. These libraries ensure the integration of custom hard blocks into the flow and enable the automation of the exploration process.

4 Multi-objective FPGA Architecture Search

4.1 Design Space Definition

An FPGA architecture design space is constructed and shown in Table 2. Parameters are categorized into CLB, PE array, BRAM, routing, and layout. The first three categories are more front-end oriented, focusing on FPGA component design and functionality. The latter two categories are related to back-end processing, which deals with physical implementation and interconnects.

The parameters N, K, I, F_{clocal}, S_{RAM}, and the FPGA aspect ratio Asp are restricted to the most common options used in previous architectures [8, 13, 17, 27–29], as these ranges have been proven to perform well. The delay and area of the 20Kb and 40Kb BRAM blocks are derived by interpolating between the values obtained from running COFFE on 16 Kb, 32 Kb, and 64 Kb BRAMs. The PE array has two size configurations S_{array}: 4×4 and 8×8. The parameter R_l represents the proportion of the wire segments that are $L16$. There are two layout strategies explored in this framework: spatial and clustered, as shown in Figure 3. In the spatial layout, different types of blocks are dispersed across the FPGA, whereas in the clustered layout, blocks of the same type are grouped within a defined region. Additionally, a linear scaling method is employed to adjust the block ratio in layout to optimize FPGA block utilization for given benchmarks. This ensures high utilization to make the FPGA more compact, thereby reducing the delay and area. Specifically, it starts

Figure 3: Layout strategies to be explored. Blocks extend vertically in columns and repeat horizontally based on the strategy.

with a manual design from a previous work [13]:

$$D_{est_i} = C_{og_i} \times U_i, \tag{3}$$

where D_{est_i} is each block type's estimated demanded column number, C_{og_i} is each block type's original column number, and U_i denotes their measured original utilization. The estimates are then adjusted to ensure the sum of the new column numbers is close to the original:

$$C_{new_i} = \frac{\sum C_{og}}{\sum D_{est}} \times D_{est_i}, \tag{4}$$

where C_{new_i} is each block type's new column number. In this case, the original column numbers of CLB, DSP, and PE array are 11, 4, 3, and 1, respectively. Their measured average utilizations in the selected benchmarks are 0.16, 0.007, 0.32, and 0.97. By using Equation (3) and Equation (4), the new column numbers are 9, 1, 5, and 5, respectively. The effectiveness of this optimization will be verified in Section 5.2. In addition, since the height of each block varies while the overall architecture design remains rectangular, extending vertically in VTR's auto-layout mode may cause misalignment, resulting in gaps between columns of blocks and the top of the design. The parameter *Fill* determines whether these gaps should be filled with CLBs.

Table 2: Selected FPGA architecture parameters

Type	Parameter	Description	Range of values
CLB	N	BLEs per CLB	6, 8, 10, 12
	K	LUT inputs	5, 6
	I	CLB inputs	32: 68: 4
	F_{clocal}	crossbar density	0.25, 0.5
PE array	S_{array}	size of PE arrays	4×4, 8×8
BRAM	S_{RAM}	size of BRAMs	16Kb, 20Kb, 32Kb, 40Kb
Routing	R_l	wire segment ratio	0.1, 0.15, 0.2
Layout	*Layout*	layout strategy	spatial, clustered
	Fill	whether fill gaps	0, 1
	Asp	aspect ratio	0.5, 1, 2

* The values are either listed individually or start : end : stride.

4.2 Search Space Reduction

The size of the search space greatly affects exploration efficiency. Besides constraining the parameter selection range, utilizing domain knowledge to prune the search space is crucial. If domain knowledge suggests a likely optimal parameter value, further exploration can be avoided, thus reducing the exploration space and speeding up the process. A summarizing formula $I = \frac{K}{2} \times (N+1)$ is proposed by Ahmed *et al.* [30] to infer I through N and K. For fractured LUTs, the parameters become $K' = K - 1$ and $N' = N \times 2$. Consequently, the formula transforms into $I = \frac{K'}{2} \times (N'+1) = \frac{K-1}{2} \times (N \times 2 + 1)$.

Subsequent research finds that adding approximately 5 to the calculated I value typically yields better results [8]. From an area-routability perspective, since depopulated crossbars are used, it is best to avoid using all the inputs, as full usage can result in routing difficulties. Hence, adding a few inputs is generally a better solution. To better match the parameter selection of F_{clocal}, each CLB's inputs are divided into four groups, so I should be approximated to the nearest multiple of 4. As a result, in this framework,

$$I = 4 \left\lfloor \frac{\left\lceil \frac{K-1}{2} \times (2N+1) + 5 \right\rceil + 2}{4} \right\rfloor. \tag{5}$$

By avoiding independent exploration of I, the search space in Table 2 is reduced from 46080 architecture designs to 4608 ones.

4.3 Pareto-front Search

Our algorithm is based on Bayesian optimization, which uses a surrogate model to simulate the objective function and an acquisition function to select sampling points. Common BO methods include those based on Gaussian Process (GP) regression [31] and TPE [16, 32]. GP models the function $p(y|x)$ by assuming a multivariate normal distribution over the search space. While effective for continuous variables, it may struggle with discrete ones due to its smoothness assumption.

In contrast, TPE transforms the modeling of $p(y \mid x)$ into modeling $p(x \mid y)$ using Bayes' Theorem. It partitions observations into a good set D_l and a bad set D_g with a quantile ratio γ. Then, it uses the Parzen window to estimate $p(x \mid y)$ with a good density function $l(x)$ and a bad density function $g(x)$. It is a non-parametric method that does not make assumptions about the global data distribution, which allows it to handle both continuous and discrete search spaces effectively. Since the FPGA architecture parameters to be optimized here are all discrete, TPE is an appropriate choice.

However, in this multi-objective optimization problem involving both the area and delay metrics, BO methods using acquisition functions like expected improvement (EI) typically transform a multi-objective problem into a single-objective one through weighted addition or a product. This hinders effective exploration and trade-offs among objectives, leading to poor performance in the hypervolume [33] and the average distance to reference set (ADRS) [34]. The hypervolume measures the volume between a Pareto frontier and a reference point in the objective space. For example, in Figure 4, assuming Y^* is the Pareto frontier and the upper right corner of the rectangle is the reference point, the upper right shaded area represents the hypervolume. ADRS measures the proximity of an approximated Pareto-optimal set to the reference Pareto-optimal set. Given a reference Pareto-optimal set $\mathcal{R}$ and an approximated Pareto-optimal set $\mathcal{P}$, ADRS can be calculated as follows:

$$\text{ADRS}(\mathcal{R}, \mathcal{P}) = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{p} \in \mathcal{P}} \min_{\mathbf{r} \in \mathcal{R}} d(\mathbf{r}, \mathbf{p}), \tag{6}$$

where d is the Euclidean distance function.

To improve efficiency, we introduce hypervolume awareness [35, 36] and dominance to this framework and demonstrate its effectiveness in exploring the area-timing space of various FPGA architecture designs. Consequently, density functions are formed as follows:

$$p(x \mid y) = \begin{cases} l(x) & \text{if } (y \succ Y^*) \cup (y \parallel Y^*) \\ g(x) & \text{if } Y^* \succeq y \end{cases} \quad p(y \succ Y^* \cup y \parallel Y^*) = \gamma \tag{7}$$

where Y^* is a set of points to split the observed architecture design set D according to γ. The dominance relationship between the

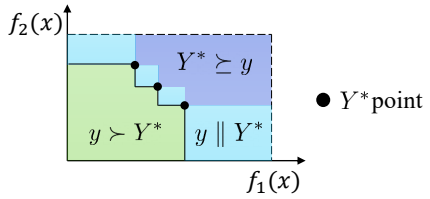

Figure 4: In a dual-objective case, colors are different according to the dominance relationship between y and Y^*

Algorithm 1 Design Set Split

Require: $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n_t)}, y^{(n_t)})\}$: observed FPGA architecture designs, $n_t \in \mathbb{N}$: number of iterations; $\gamma \in (0, 1)$: quantile ratio
Ensure: D_l: good observed designs; D_g: bad observed designs
1: $D_l \leftarrow \{\}$
2: Sort the observed designs based on their non-domination rank
3: **for** $i \leftarrow 1$ to $|D|$ **do**
4: **if** $|D_l| + \left|D_{\text{rank}(i)}\right| > \lfloor \gamma\,|D| \rfloor$ **then**
5: **break**
6: **end if**
7: $D_l \leftarrow D_l \cup D_{\text{rank}(i)}$
8: **end for**
9: **if** $|D_l| < \lfloor \gamma\,|D| \rfloor$ **then**
10: **for** $j \leftarrow 1$ to $(\lfloor \gamma\,|D| \rfloor - |D_l|)$ **do**
11: $x^* \leftarrow \arg\max_{x \in D \setminus D_l} [\text{HV}(D_l \cup \{x, f(x)\}) - \text{HV}(D_l)]$
12: $D_l \leftarrow D_l \cup \{x^*, f(x^*)\}$
13: **end for**
14: **end if**
15: $D_g \leftarrow D \setminus D_l$

two-dimensional y and Y^* is shown in Figure 4, where $f_1(x)$ represents area and $f_2(x)$ represents delay. The observed design set split is detailed in Algorithm 1, where y in D are the area and delay corresponding to x, and HV means the calculated hypervolume. After sorting, designs are added to D_l rank by rank until the number of designs in D_l plus those at the next rank exceeds the threshold (Line 7). Then, if D_l is insufficient, the algorithm ensures the target design number by greedily selecting architecture designs that maximize the hypervolume from the next rank's observation (Line 11).

In terms of the acquisition function, the expected hypervolume improvement (EHVI) [35–37] is introduced to maximize the hypervolume of the approximated Pareto front. It calculates the expected increase in hypervolume after adding a new y to the current Y^*:

$$\begin{aligned} \text{EHVI}_{Y^*}(x) &= \int_L \left(\text{HV}\left(Y^* \cup \{y\}\right) - \text{HV}\left(Y^*\right)\right) p(y \mid x) dy \\ &= \int_L \left(\text{HV}\left(Y^* \cup \{y\}\right) - \text{HV}\left(Y^*\right)\right) \frac{p(x \mid y) p(y)}{p(x)} dy, \end{aligned} \tag{8}$$

where $L = \{y \mid y \succ Y^* \cup y \parallel Y^*\}$, which corresponds to the green and cyan areas in Figure 4 in a dual-objective problem, and the latter equation is obtained by Bayes' Theorem. Thus, $\gamma = p(y \in L)$, and the denominator of Equation (8) is derived as: $p(x) = \gamma l(x) + (1-\gamma) g(x)$, and the numerator is: $l(x) \int_L \left(\text{HV}\left(Y^* \cup \{y\}\right) - \text{HV}\left(Y^*\right)\right) p(y) dy$.

Combining the numerator and denominator:

Algorithm 2 Hypervolume-aware Tree-structured Parzen Estimator

Require: $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n_t)}, y^{(n_t)})\}$: observed FPGA architecture designs; $n_t \in \mathbb{N}$: number of iterations; $n_c \in \mathbb{N}$: number of candidate designs per iteration; $\gamma \in (0, 1)$: quantile ratio
Ensure: Pareto optimal architecture designs in D
1: **for** $i \leftarrow 1$ to n_t **do**
2: $D_l, D_g \leftarrow \text{DSS}(D, \gamma)$ ▷ Algorithm 1
3: Construct $l(x)$ and $g(x)$ ▷ Equation (7)
4: $C \leftarrow \left\{x^{(i,j)} \sim l(x) \mid j = 1, \ldots, n_c\right\}$
5: $x^* \leftarrow \text{argmax}_{x \in C} \text{EHVI}_{Y^*}(x)$ ▷ Equation (9)
6: $D \leftarrow D \cup \{(x^*, f(x^*))\}$ ▷ $f(x^*)$ is from the CAD flow
7: **end for**

$$\begin{aligned} \text{EHVI}_{Y^*}(x) &= \frac{\int_L \left(\text{HV}\left(Y^* \cup \{y\}\right) - \text{HV}\left(Y^*\right)\right) p(y) dy}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}} \\ &\propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1}. \end{aligned} \tag{9}$$

As shown in Equation (9), the algorithm samples the architecture design x^* by maximizing $\frac{l(x)}{g(x)}$. Then, we derive Algorithm 2. In each iteration, the observed FPGA architecture designs are split (Line 2). And two density functions are constructed for good and bad designs, respectively (Line 3). Subsequently, several candidate designs are sampled from $l(x)$ (Line 4), and the estimated optimal design x^* is selected from candidates by maximizing the EHVI value (Line 5). At the end of each iteration, the sampled design is evaluated through the integrated CAD flow, and the observation is updated accordingly (Line 6).

5 Experimental Results

5.1 Setup

The proposed framework is implemented on a Linux platform with a 2.0GHz Intel Xeon Gold 6338 CPU with 1024GB of memory and an NVIDIA RTX 4090 GPU. COFFE 2 [1] and VTR 8.1 [2] are used in the integrated CAD flow. SPICE simulations in COFFE use 22nm libraries from the Predictive Technology Model [38]. The standard cell library used by the Design Compiler in COFFE is the NanGate 45nm Open Cell Library [39]. Scaling factors are applied to scale down the PPA values of hard blocks from 45 nm to 22 nm [40]. The exponents of delay and area in the cost function of COFFE are set to 2 and 1, respectively, to emphasize performance more. The channel width is fixed at 300.

A dataset comprising 4608 architecture designs is collected as described in Section 4. The proposed method is compared against manual designs and other DSE algorithms, including SA, GP-based BO, and TPE-based BO. Each algorithm runs for 50 iterations, where GP-based BO, TPE-based BO, and our framework perform five uniform random initializations followed by 45 formal optimization iterations.

5.2 Performance Comparison

Considering that a DL-specific FPGA architecture should be optimized to handle a variety of DL workloads, this case study employs the geometric mean of the area and critical path delay values obtained from the benchmarks in Table 1. This approach fairly captures the PPA improvements and declines across each benchmark.

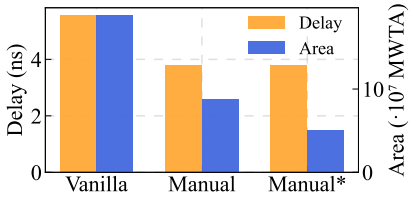

Figure 5: Comparing delay and area of FPGA architecture designs with different blocks and block ratios.

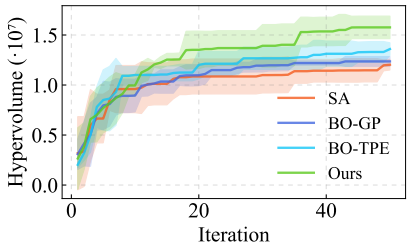

Figure 6: Optimization trajectories for different methods illustrate the mean hypervolume ± standard deviation.

First, we compare the geometric mean metrics of FPGA architecture designs with different blocks and block ratios to verify the necessity of adding PE arrays and optimizing block ratios. In Figure 5, "vanilla" refers to the architecture without PE arrays, "manual" denotes the architecture coming from the previous work [13] equipped with PE arrays, and "manual*" represents the "manual" design using the block ratio calculated in Section 4.1. It shows that PE arrays significantly reduce area and critical path delay. Additionally, adjusting the block ratio further decreases the area and delay. The area reduction resulting from adjusting the block ratio is primarily due to a 2.8× reduction in routing area, making the FPGA more compact and routing easier. Since "manual*" performs best in both area and delay, we mainly use it as the manually designed baseline for comparison in subsequent experiments.

Algorithm experiments are conducted three times on SA, GP-base BO, TPE-based BO, and our proposed framework. The mean hypervolume trajectories are represented by a solid line, with the standard deviation shown as the surrounding shadow in Figure 6. As iterations increase, our hypervolume value steadily grows more than those of other algorithms. The hypervolume and ADRS result comparisons for FPGA manual architecture designs and those generated by algorithms are presented in Figure 7, where the hypervolume and ADRS of the algorithms are derived from the Pareto frontiers fused from three experiments. Our framework consistently shows superior performance, achieving 29.4% higher hypervolume and 89.5% lower ADRS than the second-best results. These results demonstrate that our algorithm efficiently explores the discrete multi-objective search space by avoiding the smoothness assumption and incorporating multi-objective concepts into the optimization process.

Manual architecture designs and the Pareto frontiers of different algorithms are illustrated in Figure 8, with area and delay values presented after applying the geometric mean. The true Pareto frontier is obtained from the dataset designs. This figure intuitively highlights the limitations of manual designs in addressing multiple objectives, as each corresponds to only one area-delay point. In contrast, our framework can generate a set of architecture designs that are closest

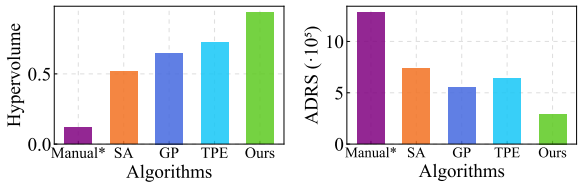

Figure 7: Comparison of different methods based on normalized hypervolume and ADRS.

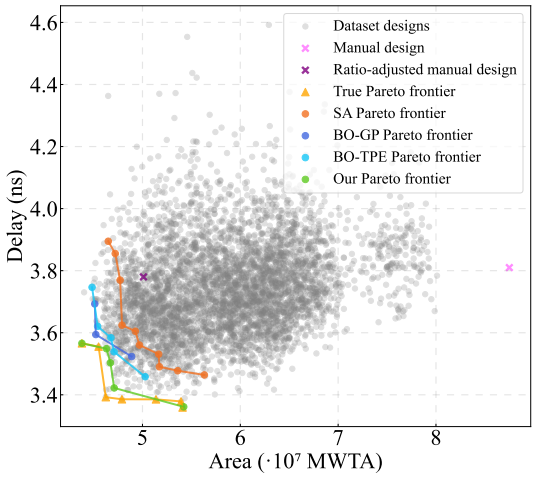

Figure 8: Manual architecture designs and Pareto frontiers of different algorithms.

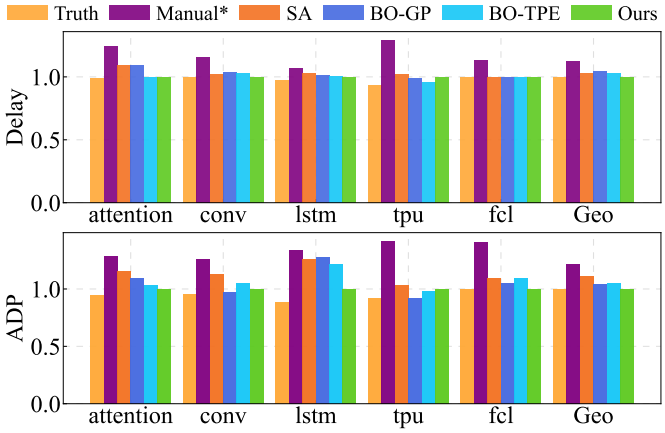

Figure 9: Comparison of different methods across various benchmarks based on delay and ADP.

to the true Pareto frontier for multiple scenarios compared to all the manual and algorithmic baselines.

Considering two typical scenarios, minimizing delay and area-delay product (ADP), the bar graphs for these geometric mean values are displayed on the far right of the two subgraphs in Figure 9. Our method consistently delivers the best results, whether aiming for minimum delay or ADP. It reduces delay by 12.8% and ADP by 21.4% compared to the manual design with adjusted block ratio, and it outperforms all algorithm baselines in both delay and ADP.

When substituting these manual designs and Pareto frontiers estimated by algorithms back into each benchmark for further analysis,

as illustrated in the rest of Figure 9, our framework's performance remains competitive across different benchmarks, achieving the best results in 7 out of 10 cases. If the performance needs to be biased toward certain benchmarks, the weight of each benchmark's PPA values in the mean calculation can be adjusted accordingly.

6 Conclusion

This work introduces an innovative framework for exploring FPGA architectures to accelerate DL workloads. By combining the hypervolume-aware TPE method with the integrated COFFE and VTR toolsets, the framework efficiently explores the FPGA architecture design space, automatically generating outstanding designs applicable to multiple scenarios in the design space. This enhances design efficiency and yields significant performance gains. Experimental validation across various DL benchmarks demonstrates notable improvements in both area and delay metrics, outperforming baseline manual designs and optimization algorithms. Future research will focus on expanding the framework application scenarios to include exploring multi-die FPGAs and co-designing network architecture and FPGA architecture.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 62204066) and the Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013).

References

[1] S. Yazdanshenas and V. Betz, "Coffe 2: Automatic modelling and optimization of complex and heterogeneous fpga architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–27, 2019.

[2] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker *et al.*, "Vtr 8: High-performance cad and customizable fpga architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–55, 2020.

[3] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versaltm architecture," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 84–93.

[4] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, and M. Langhammer, "Beyond peak performance: Comparing the real performance of ai-optimized fpgas and gpus," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2020, pp. 10–19.

[5] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 nx architecture and applications," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 57–67.

[6] A. Boutros, M. Eldafrawy, S. Yazdanshenas, and V. Betz, "Math doesn't have to be hard: Logic block architectures to enhance low-precision multiply-accumulate on fpgas," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 94–103.

[7] M. Eldafrawy, A. Boutros, S. Yazdanshenas, and V. Betz, "Fpga logic block architectures for efficient deep learning inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 3, pp. 1–34, 2020.

[8] G. Zgheib and P. Ienne, "Evaluating fpga clusters under wide ranges of design parameters," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8.

[9] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced dsp blocks for low-precision deep learning on fpgas," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 35–357.

[10] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. Leong, "Pir-dsp: An fpga dsp block architecture for multi-precision deep neural networks," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 35–44.

[11] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. Iyer, and R. Das, "Compute-capable block rams for efficient deep learning acceleration on fpgas," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 88–96.

[12] A. Arora, A. Bhamburkar, A. Borda, T. Anand, R. Sehgal, B. Hanindhito, P.-E. Gaillardon, J. Kulkarni, and L. K. John, "Comefa: Deploying compute-in-memory on fpgas for deep learning acceleration," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 16, no. 3, pp. 1–34, 2023.

[13] A. Arora, M. Ghosh, S. Mehta, V. Betz, and L. K. John, "Tensor slices: Fpga building blocks for the deep learning era," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 4, pp. 1–34, 2022.

[14] J. Qian, Y. Shen, K. Shi, H. Zhou, and L. Wang, "General routing architecture modelling and exploration for modern fpgas," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2021, pp. 1–9.

[15] S. Zheng, J. Qian, H. Zhou, and L. Wang, "Graebo: Fpga general routing architecture exploration via bayesian optimization," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2022, pp. 282–286.

[16] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 24, 2011.

[17] A. Arora, A. Boutros, S. A. Damghani, K. Mathur, V. Mohanty, T. Anand, M. A. Elgammal, K. B. Kent, V. Betz, and L. K. John, "Koios 2.0: Open-source deep learning benchmarks for fpga architecture and cad research," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[19] Intel, "Intel agilex fpgas and socs," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/fpga/agilex.html

[20] A. Arora, "Dsp slice," https://github.com/samidhm/DSP_Slice.

[21] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "Impress: Large integer multiplication expression rewriting for fpga hls," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10.

[22] Synopsys, Inc., "Design Compiler." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[23] Cadence Design Systems, Inc., "Innovus Implementation System." [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html

[24] Synopsys, Inc., "PrimeTime Static Timing Analysis." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html

[25] ——, "PrimeSim HSPICE." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html

[26] B. Grady and J. H. Anderson, "Synthesizable heterogeneous fpga fabrics," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 222–229.

[27] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell *et al.*, "The stratixπ routing and logic architecture," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2003, pp. 12–20.

[28] J. Greene, S. Kaptanoglu, W. Feng, V. Hecht, J. Landry, F. Li, A. Krouglyanskiy, M. Morosan, and V. Pevzner, "A 65nm flash-based fpga fabric optimized for low cost and power," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2011, pp. 87–96.

[29] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, and I. Ganusov, "Architectural enhancements in intel® agilex™ fpgas," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020, pp. 140–149.

[30] E. Ahmed and J. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2000, pp. 3–12.

[31] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[32] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference on Machine Learning (ICML)*. PMLR, 2013, pp. 115–123.

[33] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," in *Evolutionary Multi-Criterion Optimization (EMO)*. Springer, 2007, pp. 862–876.

[34] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, and L. Salomon, "Performance indicators in multiobjective optimization," *European Journal of Operational Research*, vol. 292, no. 2, pp. 397–422, 2021.

[35] A. Shah and Z. Ghahramani, "Pareto frontier learning with expensive correlated objectives," in *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1919–1927.

[36] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2020, pp. 533–541.

[37] M. T. Emmerich, A. H. Deutz, and J. W. Klinkenberg, "Hypervolume-based expected improvement: Monotonicity properties and exact computation," in *IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 2147–2154.

[38] Arizona State University, "Predictive technology model," 2012. [Online]. Available: http://ptm.asu.edu/

[39] Nangate Inc., "Open Cell Library v2008_10 SP1," 2008. [Online]. Available: http://www.nangate.com/openlibrary/

[40] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration, the VLSI Journal*, vol. 58, pp. 74–81, 2017.