Kratos: An FPGA Benchmark for Unrolled DNNs with Fine-Grained Sparsity and Mixed Precision

Xilai Dai, Yuzong Chen, Mohamed S. Abdelfattah
Department of Electrical and Computer Engineering, Cornell University
{xd44, yc2367, mohamed}@cornell.edu

***Abstract*—FPGAs offer a flexible platform for accelerating deep neural network (DNN) inference, particularly for non-uniform workloads featuring fine-grained unstructured sparsity and mixed arithmetic precision. To leverage these redundancies, an emerging approach involves partially or fully *unrolling* computations for each DNN layer. That way, parameter-level and bit-level ineffectual operations can be completely skipped, thus saving the associated area and power. Regardless, unrolled implementations scale poorly and limit the size of a DNN that can be unrolled on an FPGA. This motivates the investigation of new reconfigurable architectures to improve the efficiency of unrolled DNNs, while taking advantage of sparsity and mixed precision. To enable this, we present Kratos: a focused FPGA benchmark of unrolled DNN primitives with varying levels of sparsity and different arithmetic precisions. Our analysis reveals that unrolled DNNs can operate at very high frequencies, reaching the maximum frequency limit of an Arria 10 device. Additionally, we found that substantial area reductions can be achieved through fine-grained sparsity and low bit-width. We build on those results to tailor the FPGA fabric for unrolled DNNs through an architectural case study demonstrating ∼2× area reduction when using smaller LUT sizes within current FPGAs. This paves the way for further exploration of new programmable architectures that are purpose-built for sparse and low-precision unrolled DNNs. Our source code and benchmark are available on github.com/abdelfattah-lab/Kratos-benchmark.**

I. Introduction

Deep Neural Network (DNN) inference has become one of the most important compute workloads of our time, spanning many applications from image [1]–[5] and speech recognition [6] to natural language processing [7]–[9] and autonomous driving [10]–[13]. GPUs and custom ASIC chips currently dominate DNN inference, particularly because of their high compute capacity and memory bandwidth. This enables very efficient dense matrix multiplication on these platforms. However, it has been shown, time and again, that DNNs exhibit very high levels of *fine-grained* sparsity [14] and can tolerate low and mixed arithmetic precision [15]—two intrinsic properties that are challenging to accelerate on existing DNN accelerators. This begs the question of whether there are more suitable architectures for sparse and low-precision DNNs.

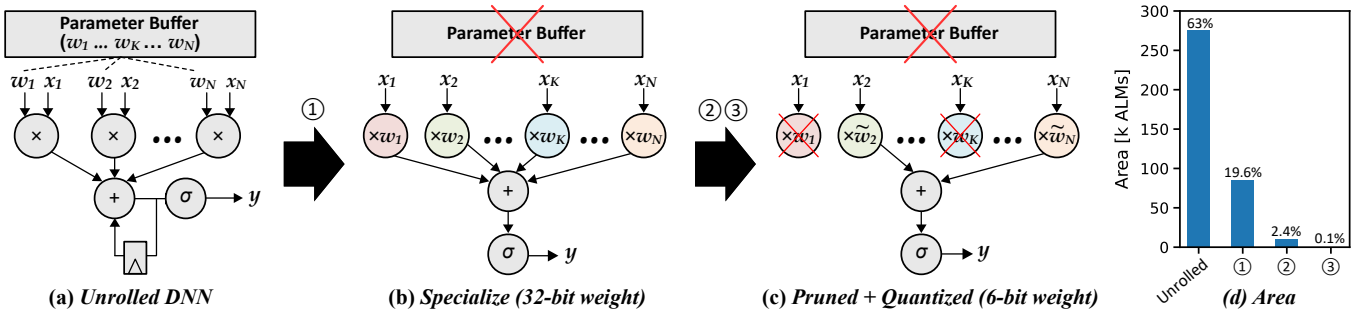
FPGAs provide an attractive acceleration platform because of their high flexibility and bit-level programmability. However, the reconfigurability overhead is generally very high, making FPGAs approximately an order of magnitude less efficient when compared to an ASIC implementation [16], [17]. Even though many innovative and sparse-aware DNN accelerator architectures were introduced on FPGAs [18]–[23], none gained enough traction to compete with current GPUs or ASICs. Nevertheless, there is an emerging *style* of DNN acceleration on FPGAs that holds promise. Specifically, **unrolled** DNN implementations, wherein a DNN accelerator contains partially or fully unrolled computation engines that are specialized for each DNN layer.

Fig. 1 shows a conceptual diagram of unrolled DNNs and the area of a 64×64 matrix multiplication on an Arria 10 GX 1150 FPGA [24]. Full unrolling means having a hardware multiply-accumulate (MAC) unit for each MAC operation in the matrix multiplication, as shown in Fig. 1(a). This naïve unrolling quickly utilizes most of the FPGA area (63%) as shown in Fig. 1(d). However, we consider unrolled DNN implementations that are specialized, pruned, and quantized. Specialization of MAC units means converting them to multiply with constant weight parameters, as shown in Fig. 1(b). This drastically reduces compute area (∼4×) by optimizing the MAC circuitry and by leveraging bit-level sparsity within the parameter values. Combining specialization with pruning and quantization, as shown in Fig. 1(c), further reduces area by ∼150× down to just 0.1% of the FPGA for 4096 *effective* FLOPs, making unrolled DNNs practical on current FPGAs. Indeed, there are a number of recent works that successfully leverage this implementation methodology of unrolled DNNs on FPGAs, especially for smaller DNNs with very high throughput requirements [25]–[31].

A key advantage of unrolled DNNs on FPGAs is the proportional reduction in circuit area and efficiency gains from all forms of redundancy. Conventional DNN accelerators, including GPUs, achieve only ∼15% performance/watt improvement from 50% *structured* sparsity, even with dedicated hardware support [32], far short of the expected 2×. FPGA's bit-level reconfigurability accelerates fine-grained and unstructured sparsity effectively. However, capacity limits exist: a 4-MFLOP DNN can be fully unrolled on an Arria 10 GX 1150 FPGA, as shown in Fig. 1, but this is suitable only for small DNNs, highlighting the need for FPGA architectural exploration to enhance efficiency.

Open-source CAD and architecture exploration frameworks like VTR [33] enable us to examine architectural tradeoffs, including LUT sizes, interconnection flexibility, and new hard blocks. We can prototype new programmable devices based on FPGAs, specifically designed to accelerate unrolled DNNs. This motivates our work on **Kratos**[1]: a benchmark suite for unrolled DNNs with unstructured sparsity and mixed precision.

[1]Kratos personifies strength in Greek mythology. Our benchmark leverages the **strength** of FPGAs, specifically bit-level programmability, to accelerate sparse DNNs.



(a) ***Unrolled DNN*** **(b)** ***Specialize (32-bit weight)*** **(c)** ***Pruned + Quantized (6-bit weight)*** ***(d) Area***

Fig. 1: Diagram of unrolled DNNs and the area of a 64×64 matrix multiplication on an FPGA. Naïve unrolling quickly utilizes most of the FPGA area (63%), but specialization ①, pruning ②, and quantization ③ reduce area by 600× down to just 0.1% of the FPGA for 4096 effective FLOPs.

More specifically, we make the following contributions:

1) We introduce the Kratos benchmark. A circuit benchmark suite of unrolled convolutional and general matrix multiplication (GEMM) DNN layers with different levels of fine-grained sparsity and numerical precisions.
2) Unlike other FPGA benchmarks, our SystemVerilog code is human-readable, parameterized, and extensible, in addition to being compatible with both commercial (Quartus Prime) and academic (VTR) CAD flows.
3) We present area and delay characteristics of our benchmark, showing that fully-unrolled DNNs can far exceed the clock network limitations on Arria 10 FPGAs. We also observe *linear* improvements in efficiency with respect to higher sparsity and lower bitwidth.
4) We perform an FPGA architectural case study investigating the most efficient LUT size for unrolled DNNs. We show ∼2× FPGA area reduction by tailoring FPGA logic blocks for unrolled DNNs, paving the way for the investigation of new purpose-built devices for sparse and low-precision DNN acceleration.

While existing FPGA benchmarks provide valuable insights for general-purpose applications, they fall short in addressing the specific needs of unrolled DNNs with fine-grained sparsity and mixed precision. Kratos fills this gap by offering a specialized benchmark suite and related tools that enable architectural exploration and optimization for unrolled DNNs. This is crucial for designing next-generation programable accelerators that can fully leverage the potential of unrolled DNNs, achieving higher throughput and efficiency compared to traditional dense execution models.

II. Related Work

Unrolled DNNs follow the synchronous dataflow design paradigm, where DNN layers are partially or fully unrolled on an FPGA to match throughput between layers [34], [35]. This approach efficiently implements binary/ternary DNNs [30], high-throughput data analysis [31], and anomaly detection [28]. Recent work focuses on tailoring fully unrolled DNNs to FPGAs using LUT primitives, low arithmetic precision, and high unstructured sparsity [25]–[27], achieving high efficiency compared to traditional DNN accelerators [19].

FPGA Benchmark circuits have commonly been used to guide the architecture exploration of FPGAs [36]. Traditionally, a variety of benchmarks from different domains are used [37] to maintain the general-purpose nature of FPGAs. Recently, some DNN-focused benchmarks have addressed the need for domain-specific FPGA fabrics for DNN acceleration. Koios [38] includes DNN accelerator circuits with varied implementations, and Roorda et al. [39] released a flexible, auto-generated DNN benchmark suite for new DSP architecture investigation. **Kratos** focuses on (1) unrolled DNN implementations, (2) unstructured sparsity and mixed precision, and (3) enhancing FPGA logic and routing architecture.

DNN-Optimized FPGAs have been proposed by improving logic blocks for low-precision DNNs [40], [41], enhancing DSPs with more low-precision computation [42], [43], or augmenting BRAMs with compute capabilities [44]–[47]. We aim to use **Kratos** for investigating optimized logic block and routing architectures to create new domain-specific programmable devices for enhanced unrolled DNN performance.

TABLE I: The Kratos Benchmarks.

Kernel	Unrolling Factor	Input / Cycle	Weight Duplicate	Output / Cycle
gemmt	row-parallel	$1 \times n$	–	$1 \times p$
gemmt	fully-unrolled	$m \times n$	$m\times$	$n \times p$
gemms	row-parallel	$1 \times n$	–	$1 \times p$
conv1d	pixelwise	$F_w \times 1 \times I_c$	–	$1 \times 1 \times O_c$
conv1d	fully-unrolled	$I_W \times 1 \times I_c$	$O_w\times$	$O_w \times 1 \times O_c$
conv2d	pixelwise	$F_w \times F_h \times I_c$	–	$1 \times 1 \times O_c$
conv2d	row-parallel	$I_W \times F_h \times I_c$	$O_w\times$	$O_w \times 1 \times O_c$
conv2d	fully-unrolled	$I_W \times I_h \times I_c$	$O_wO_h\times$	$O_w \times O_h \times O_c$

[1] All kernels accept user-defined sparsity $\in [0, 1]$ and precision $\in \mathbb{N}_{>0}$.
[2] gemmt = multiply-add tree implementation of GEMM.
[3] gemms = weight-stationary systolic implementation of GEMM.

III. Benchmark Description

A. Kernels

The Kratos benchmark contains 8 kernels as shown in Table I. These kernels implement two main DNN operations: GEMM and convolution, which are heavily used by a wide range of DNNs. The GEMM operation is used by the fully-connected layer which is ubiquitous in many DNNs such as long short-term memory [6] and transformers [7], while the convolution operation dominates convolutional neural networks [1]. Since Kratos focuses on unrolled DNNs, weights are embedded into circuit connections like LUTs rather than memory. For instance, during multiplication, the input goes directly into LUTs, producing the output without needing to access a multiplier or load weights from BRAM.

The GEMM dataflow is shown in Fig. 2(a), where the input matrix $x^{m \times n}$ is multiplied by the unrolled weight matrix $w^{n \times p}$ to generate the output matrix $y^{m \times p}$. Kratos contains two types of hardware implementations for GEMM that use multiply-adder tree (*gemmt*) and weight-stationary systolic array (*gemms*) as shown in Fig. 3(a) and (b), respectively. The datapath of our design is heavily pipelined by inserting registers between every stage of multiplication or addition. For convolution, Kratos contains 1-D convolution (*conv1d*) and 2-D convolution (*conv2d*) implemented using the multiply-adder tree. Fig. 2(b) shows the dataflow of conv2d, where the $I_W \times I_h \times I_c$ input feature map is convolved with the $F_w \times F_h \times I_c \times O_c$ filter matrix to generate the $O_w \times O_h \times O_c$ output feature map. The conv1d kernel has a similar dataflow as conv2d except that $I_h = O_h = F_h = 1$. Using multiply-adder trees allows pruning leaves of zero weights while the traditional systolic array still needs structural registers to keep systolic and thus leads to low resource efficiency.

B. Input Unrolling Factors

All weights are fully unrolled in the Kratos kernels to take full advantage of parameter-level sparsity. An important design consideration is the input unrolling factor—this quantifies the portion of the input tensor that can be processed simultaneously, and directly affects the resulting throughput. Kratos supports three input unrolling factors as illustrated in Fig. 2 and described below. The different color boxes indicate the number of elements processed in one cycle. This visual representation helps to understand the efficiency gains achieved through our approach.

Pixelwise: This unrolling factor is applicable to convolution. As shown in Fig. 2(b), the pixelwise unrolling generates one pixel along all channels of the output feature map in parallel.

Row-Parallel: This unrolling factor is applicable to both GEMM and convolution. The row-parallel unrolling generates one row of the output matrix for GEMM, and one row along all channels of the output feature map for convolution in parallel as shown in Fig. 2(a) and (b), respectively.

Fully-Unrolled: For fully-unrolled GEMM, the whole output matrix can be generated in one shot as shown in Fig. 2(a). The fully-unrolled 1-D convolution is the same as the row-parallel 1-D convolution since $O_h = 1$. For fully-unrolled 2-D convolution, the entire input feature is processed simultaneously to obtain the whole output feature map in one shot as shown in Fig. 2(b).

The unrolling factor impacts hardware design and resource utilization. For instance, the row-parallel *gemmt* implementation broadcasts one row of the input matrix to the unrolled weight, generating one row of the output matrix per cycle. Input and weight duplication can improve throughput by processing more inputs in parallel. In fully unrolled implementations, the entire input matrix is processed simultaneously, obtaining the whole output matrix in 1 cycle, with a throughput of $m \times n \times p$ operations. However, for *gemms*, this results in diminishing returns due to the systolic propagation penalty [48]. For convolution, weight duplication is necessary for row-parallel and fully-unrolled implementations. We use BRAM for pixelwise unrolling and a shift-register network

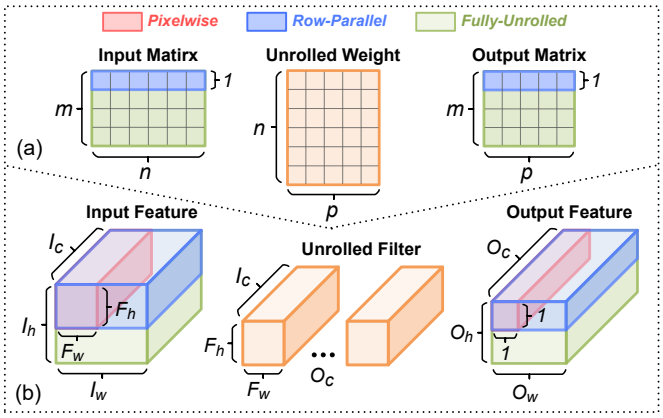


Fig. 2: Dataflow of (a) GEMM and (b) convolution for different input unrolling factors: pixelwise, row-parallel, and fully-unrolled. The weight/filter is always fully unrolled

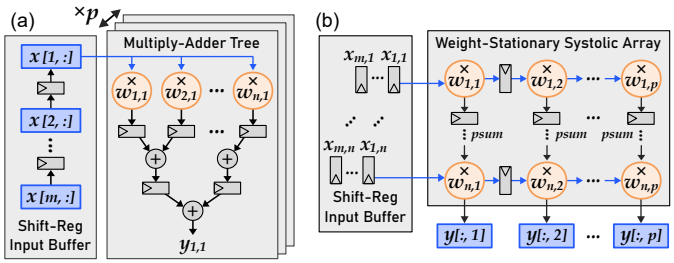


Fig. 3: Hardware implementation of GEMM: (a) multiply-adder tree and (b) weight-stationary systolic array.

for row-parallel and fully-unrolled kernels to ensure sufficient input bandwidth.

C. CAD for FPGA Architecture Exploration

One of the main motivations of this work is to evaluate existing FPGA architectures and explore new optimized architectures for unrolled DNNs. To achieve this, Kratos is designed to be compatible with both the commercial Intel Quartus Prime and the open-source VTR [33] flow.

Creating a VTR-compatible benchmark has long been a labor-intensive process due to the limited Verilog syntax coverage of VTR's Odin II synthesis front-end [49]. However, Odin II provides efficient partial technology mapping for balancing soft logic and hard blocks of a target FPGA architecture. Recently, VTR has integrated Yosys [50], an open-source synthesis tool with extensive Verilog-2005 and SystemVerilog support such as the "generate" statement. The new VTR synthesis front-end using a combination of Yosys for synthesis and Odin II for partial mapping [51] significantly reduces the efforts of handling unsupported Verilog syntax. Hence, the Kratos benchmark uses this newly released VTR flow.

D. Benchmark Workflow

Unlike many previous FPGA benchmarks [37], [38] that provide a fixed Verilog design for every kernel, Kratos provides Python scripts to automatically generate the top-level SystemVerilog module given user-provided design parameters specified in a Python dictionary. These modules contain the pre-implemented kernels we mentioned above and the

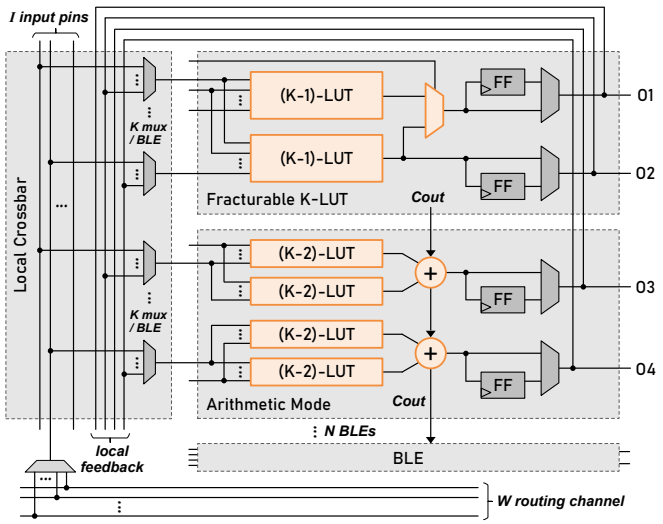


Fig. 4: Logic block diagram of the baseline FPGA for VTR architectural exploration.

embedded weights for synthesize. Thanks to the enhanced synthesis front-end of VTR, our SystemVerilog is human-readable, parameterized, performance-optimized, and extensible. After generating the hardware description, the Python scripts generate the necessary flow scripts to run either Intel Quartus Prime or VTR and report the performance and area results.

All kernels in the Kratos benchmark are parameterized by the dimensions of inputs and weights, as well as sparsity and precision. Sparsity specifies the percentage of zero elements in the weight tensor, and precision specifies the data width of inputs and weights. To simulate unstructured sparsity, we generate the weight matrix with the desired amount of non-zero elements and randomly shuffle their location. For precision, Kratos allows any integer data type, but it can be easily extended to support other data formats by changing the hardware description of the MAC unit. In addition, all kernels are functionally verified by simulating kernels running on random weights and inputs with Modelsim and comparing results with ground truth To facilitate large-scale design space exploration, Kratos tools set also provides a batch job script that allows users to define multiple sizes, precisions, and sparsities, and it will launch flows for different combinations.

IV. Evaluation Methodology

A. Experimental Setup

To quantify the efficiency of existing FPGA architectures for unrolled DNNs, we use the Intel Quartus Prime Software Version 22.3 and Arria 10 GX 1150 when running all Kratos benchmarks. To conduct FPGA architectural exploration, we use a customized version of VTR[2] As a sanity check to verify the successful parsing of our benchmark through VTR, we compare the resource utilization reported by VTR and Quartus for all kernels and observe $\pm 10\%$ variation on average.

The baseline FPGA for our VTR experiments has a Stratix-IV-like architecture using 40 nm technology and is available in the official VTR release. The logic block (LB) diagram of this architecture is shown in Fig. 4, which contains $I = 52$ input pins and a default of $N = 10$ basic logic elements (BLEs). Each BLE contains a LUT with size $K = 6$ and the two outputs can be optionally registered. The BLE can also operate in the fracturable LUT mode where each 6-LUT can be fractured into two 5-LUTs, or the arithmetic mode where the two hard adders receive inputs from four 4-LUTs. To facilitate architectural exploration, we develop a Python-based architecture file generator to automatically modify different LB parameters. The area and delay of the modified LB are extracted from COFFE 2.0 [52] and scaled to 40 nm technology. During VTR routing, we set the default router option to perform a binary search to find the minimum routing channel width W required to route the circuit.

[2]Our fork from the VTR main branch includes better SystemVerilog support, an option to manually specify the top module, and several bug fixes.

TABLE II: Kratos Design Space for Evaluation.

Kernel	Unroll Factor[1]	Size[1]	Input Dim[2]	Weight Dim[3]	Output Dim[4]
gemmt	RP	S	32×32	32×32	32×32
gemmt	RP	L	128×128	128×128	128×128
gemmt	FU	S	16×16	16×16	16×16
gemmt	FU	L	32×32	32×32	32×32
gemms	RP	S	16×16	16×16	16×16
gemms	RP	L	128×128	128×128	128×128
conv1d	PW	S	$32 \times 1 \times 64$	3×3	$30 \times 30 \times 64$
conv1d	PW	L	$32 \times 1 \times 64$	3×3	$30 \times 30 \times 128$
conv1d	FU	S	$32 \times 1 \times 8$	3×3	$30 \times 30 \times 8$
conv1d	FU	L	$32 \times 1 \times 16$	3×3	$30 \times 30 \times 16$
conv2d	PW	S	$25 \times 25 \times 32$	3×3	$23 \times 23 \times 64$
conv2d	PW	L	$25 \times 25 \times 64$	3×3	$23 \times 23 \times 64$
conv2d	RP	S	$8 \times 8 \times 8$	3×3	$6 \times 6 \times 8$
conv2d	RP	L	$8 \times 8 \times 16$	3×3	$6 \times 6 \times 16$
conv2d	FU	S	$8 \times 8 \times 4$	3×3	$6 \times 6 \times 4$
conv2d	FU	L	$8 \times 8 \times 8$	3×3	$6 \times 6 \times 8$

[1] PW: pixelwise. RP: row-parallel. FU: fully-unrolled. S: small. L: large.
[2] Format: $m \times n$ for GEMM, $I_W \times I_h \times I_c$ for convolution.
[3] Format: $n \times p$ for GEMM, $F_W \times F_h$ for convolution.
[4] Format: $p \times k$ for GEMM, $O_W \times O_h \times O_c$ for convolution.

B. Design Space

The Kratos benchmark enables large design space exploration by allowing users to specify arbitrary kernel sizes, as well as sparsity and precision. For our experiments, we use the set of kernel sizes as shown in Table II, which contains two size variants (small and large) for all kernels. The convolution kernels have a stride of 1 without padding. For every kernel size, we evaluate 10 evenly spaced sparsity from 0 to 0.9, and 4 data precision (1-bit, 2-bit, 4-bit, 8-bit). Note that the kernel sizes are chosen to ensure that they can pass the placement and routing under the lowest sparsity level (i.e., no sparsity) and the highest data precision (8-bit).

V. Experimental Results

In this section, we present area and frequency trends of Kratos benchmark circuits to highlight the effect of sparsity and precision. In addition, we present a proof-of-concept architectural exploration case study to investigate the LUT size for unrolled DNNs, and the potential area savings compared to current general-purpose FPGAs.

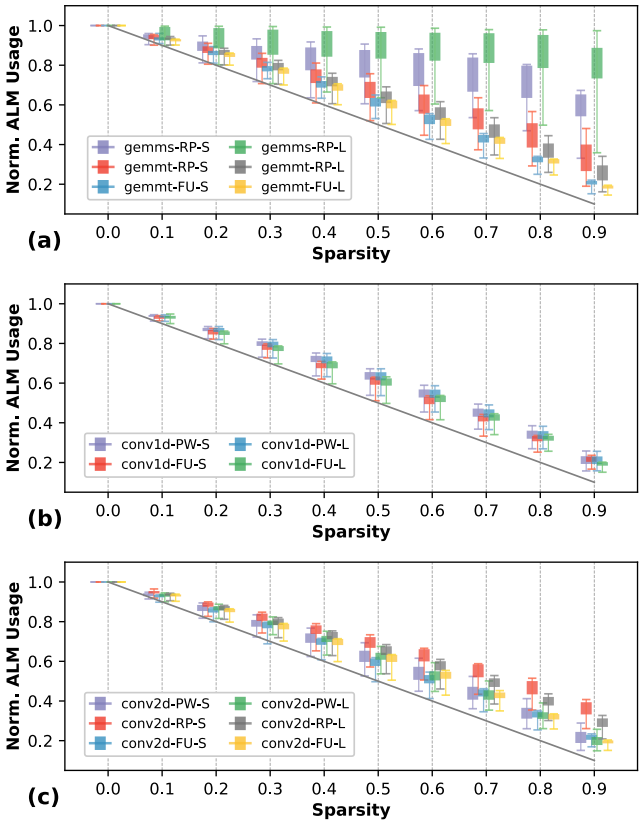


Fig. 5: Normalized ALM utilization on Arria 10 vs. sparsity for (a) GEMM, (b) conv1d, and (c) conv2d kernels. The solid black line highlights the ideal trend where the ALM utilization linearly decreases with higher sparsity.

A. Area and Frequency Trends on Arria 10

Resource utilization vs. sparsity. Fig. 5 shows the normalized adaptive logic module (ALM) utilization vs. sparsity for different Kratos kernels on Arria 10 GX 1150. The error bars indicate the range of ALM utilization under different precisions, with the interquartile range marked by filled rectangles. Most kernels exhibit a near-ideal linear reduction in ALM utilization with increased sparsity, demonstrating the effectiveness of FPGAs in accelerating unrolled DNNs. The row-parallel *gemms* deviates from this trend; at 0.9 sparsity, its ALM utilization is reduced by only 46% and 31% for small and large designs, respectively. This is due to *gemms*' structured datapath with delay registers between processing elements, which hampers optimization of zero-weight MAC units. Conversely, the multiply-adder tree implementation prunes zero branches entirely during synthesis, eliminating the need for LUTs and registers as sparsity increases.

Resource utilization vs. bit-width. As we decrease bitwidth, area decreases super-linearly as shown in Fig. 6. This is expected because multipliers scale quadratically with bit-width while adders scale linearly and control circuitry remains constant. Further inspection of Fig. 6 reveals comparative area savings trends from higher sparsity and lower bit-width. For instance, when we inspect the 8-bit *conv2d-FU-L* plot, reducing the precision to 4-bits leads to a 2.9-fold decrease in area. Achieving a similar reduction in area for an 8-bit

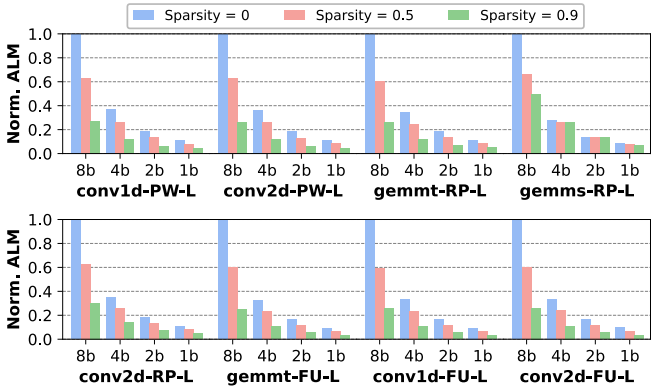


Fig. 6: Normalized ALM utilization on Arria 10 vs. precision under different sparsity levels.

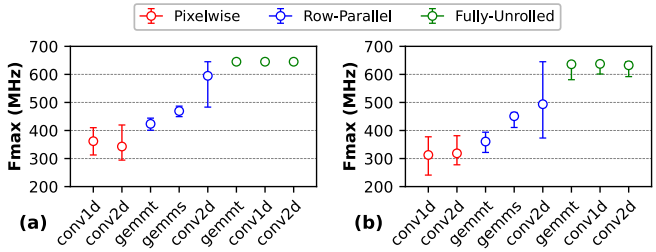


Fig. 7: Frequency ranges of (a) small and (b) large Kratos circuits under different sparsity and precision on Arria 10. The circle on each error bar marks the average.

implementation would require high sparsity levels, ranging from 80% to 90%. Recent research is beginning to explore how pruning compares with quantization in terms of accuracy [53]. Together with our hardware efficiency results, this opens the door for more extensive studies on accuracy-efficiency tradeoffs for pruning, quantization, and their combination.

Critical Path Delay. Fig. 7 shows the frequency ranges of small and large Kratos kernels on Arria 10. There is a clear trend in favor of higher unrolling factors, with our heavily-pipelined fully-unrolled designs reaching the maximum frequency supported on Arria 10. In this case, the unrestricted fmax can reach 1GHz by Quartus timeing report, and the restricted fmax can reach over 600Mhz. Conversely, the row-parallel and pixelwise implementations suffer from higher critical path delays within the control and buffering circuitry but are still capable of reaching frequencies 300–600 MHz. The high speeds attainable with unrolled DNNs, combined with their direct area savings from fine-grained sparsity and reduced bit-width motivate further investigation of new FPGA architectures to enable larger DNN deployments.

B. Architectural Exploration Case Study

Using the baseline architecture described in Section IV-A, we conduct a case study to find the optimal LUT size for unrolled DNNs. We evaluate four LB architectures whose LUT sizes K vary from 3 to 6. For each different LUT size, we determine the corresponding number of LB input pins I with $N = 10$ basic logic elements based on the empirical equation $I = \frac{K}{2}(N+1)$ from prior work [54]. We use VTR to evaluate the four architectures on three kernels *gemmt-RP-S*, *conv1d-PW-S*, *conv2d-PW-S* from Table II as these designs balance silicon footprint and data throughput. While fully unrolled

TABLE III: Resource utilization, silicon area, and performance under different LUT sizes, sparsity, and precision.

Sparsity	K [1]	Precision = 8-bit									Precision = 4-bit								
		gemmt-RP-S			conv1d-PW-S			conv2d-PW-S			gemmt-RP-S			conv1d-PW-S			conv2d-PW-S		
		kLBs	Area (mm^2)	Fmax (MHz)	kLBs	Area (mm^2)	Fmax (MHz)	kLBs	Area (mm^2)	Fmax (MHz)	kLBs	Area (mm^2)	Fmax (MHz)	kLBs	Area (mm^2)	Fmax (MHz)	kLBs	Area (mm^2)	Fmax (MHz)
0%	3	1.13	**1.88**	93.1	13.4	**22.4**	44.0	20.4	**34.0**	38.6	0.97	**1.62**	**177.9**	11.3	**18.9**	113.6	17.1	**28.4**	87.7
	4	1.06	2.18	**124.7**	12.8	26.2	45.5	19.1	39.3	**41.9**	0.97	2.0	170.4	11.3	23.3	116.4	17.0	35.0	79.1
	5	1.06	2.67	102.0	12.8	32.2	**53.5**	19.2	48.3	40.0	0.97	2.45	172.0	11.3	28.5	**118.9**	17.0	42.9	81.4
	6	1.06	3.62	114.5	12.8	43.7	46.5	19.2	65.6	37.7	0.97	3.32	167.8	11.4	38.9	118.9	17.1	58.5	**93.4**
50%	3	0.66	**1.11**	**146.9**	8.05	**13.4**	72.6	12.4	**20.6**	**49.6**	0.5	**0.83**	**179.8**	6.01	**10.0**	118.7	9.07	**15.1**	**92.9**
	4	0.63	1.29	143.8	7.58	15.6	**73.7**	11.5	23.6	46.5	0.5	1.02	179.8	6.01	12.3	106.0	9.06	18.6	87.2
	5	0.63	1.59	144.9	7.57	19.1	65.4	11.4	28.8	48.2	0.5	1.25	172.3	6.01	15.1	106.8	9.06	22.8	88.6
	6	0.63	2.15	135.6	7.57	25.9	70.0	11.4	39.0	48.6	0.5	1.7	173.4	6.01	20.6	**119.5**	9.06	31.0	73.8
90%	3	0.21	**0.35**	**169.4**	2.74	**4.56**	100.1	4.5	**7.48**	76.9	0.1	**0.17**	179.9	1.25	**2.09**	124.0	2.02	**3.37**	106.0
	4	0.21	0.43	165.8	2.58	5.31	109.1	4.13	8.48	**99.0**	0.1	0.21	179.1	1.24	2.55	**134.4**	1.93	3.96	105.3
	5	0.21	0.52	166.5	2.58	6.51	**125.3**	4.08	10.3	86.1	0.1	0.26	**183.5**	1.24	3.12	118.0	1.92	4.83	103.7
	6	0.21	0.7	165.9	2.58	8.83	101.1	4.05	13.8	92.4	0.1	0.35	181.6	1.24	4.24	132.4	1.92	6.56	**108.7**

[1] For $K = 3, 4, 5, 6$, the maximum channel widths required to route all designs are $W = 102, 96, 90, 90$, which gives a tiles area of $1664um^2, 2053um^2, 2520um^2, 3420um^2$ from COFFE [52] after normalizing to 40 nm technology.

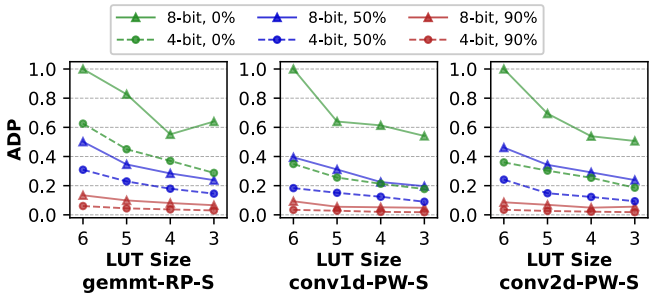


Fig. 8: Normalized area-delay product (ADP) for the Kratos circuits in Table III.

designs offer the best clock frequency, practical constraints and the need to run multiple experiments necessitated choosing smaller, more manageable designs in this initial exploration.

For every architecture, we extract the maximum routing channel width W reported by VTR that can fit all designs, which is then passed to COFFE [52] to compute the LB area (including routing) with a given (K, I, W). The total silicon area of a kernel is then calculated by multiplying the LB utilization and the LB area. All reported results are averaged over 3 runs using different random seeds

The experiment results summarized in Table III show potential savings of $\sim 2\times$ when reducing the LUT size from 6 (default in most Intel FPGAs) to 3. This comes with a 10–20% degradation in critical path delay for 8-bit kernels, whereas a small improvement is observed for most 4-bit designs. We hypothesize that smaller 4-bit MAC units are more likely to fit within a single logic block, even with $K = 3$, compared to 8-bit counterparts. When optimizing the FPGA device for area-delay product, Fig. 8 favors the smallest LUT size ($K = 3$) except for one circuit (*gemmt-RP-S* with 8-bit precision and no sparsity). This strongly indicates the superiority of smaller LUTs for unrolled DNN implementations.

VI. Conclusions and Future Work

Motivated by the efficiency advantages of unrolled DNNs, we created a benchmark suite to enable the architectural exploration of new programmable hardware devices for accelerating unrolled DNNs. Our empirical analysis shows that unrolled DNNs on FPGA can run at very high speed, can can significantly benefit from improvements in efficiency with fine-grained sparsity and reduced arithmetic precision — two properties that are not easily attainable with conventional DNN accelerators. Furthermore, we performed an architectural case study to reveal $\sim 2\times$ possible area savings from exploring the optimal LUT size of contemporary FPGA architectures to suit unrolled DNNs better.

While we can't optimize a general-purpose FPGA solely for unrolled DNNs, future work can explore integrating specialized bit-programmable fabrics within general-purpose FPGAs or creating new bit-programmable devices specifically for unrolled DNNs. One goal of Kratos is to inspire research on new programmable architectures that are much more efficient (e.g., $10-100\times$) than current FPGAs, maintaining linear and quadratic efficiency scaling with sparsity and low precision, respectively. Although the size of unrolled DNNs that can fit on current FPGAs is small, future works can explore algorithmic optimizations such as weight sharing and time-domain multiplexing to drastically increase the capacity of unrolled DNNs that can fit on the target bit-programmable device. For example, with weight sharing [55], there can be one large unrolled layer that is shared throughout the DNN, and a small accelerator for "adapter" layers that run much slower. Although time-domain multiplexing (investigated by Tabula Inc.) has been proven challenging for general-purpose FPGAs, it could be a really good fit for programmable devices targeting unrolled DNNs with a much simpler CAD flow due to domain specialization, and can achieve multiplicative efficiency on mapping larger unrolled DNNs. We believe that the Kratos benchmark is the first and valuable step to begin investigating unrolled DNNs on programmable architectures, which is a promising research direction because it addresses an open problem in DNN research: how to fully leverage fine-grained unstructured sparsity and mixed precision effectively—something that current accelerators cannot handle.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arxiv preprint arxiv:1704.04861*, 2017.

[4] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arxiv preprint arxiv:2010.11929*, 2020.

[6] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network," *arxiv preprint arxiv:1808.03314*, 2018.

[7] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *North American Chapter of the Association for Computational Linguistics (ACL)*, 2019.

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[10] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[11] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arxiv preprint arxiv:1804.02767*, 2018.

[12] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," 2021. [Online]. Available: https://arxiv.org/abs/2107.08430

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *European Conference on Computer Vision (ECCV)*, 2015.

[14] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 1135–1143.

[15] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[16] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[17] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, dec 2018.

[18] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. B. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, 2017, p. 75–84.

[19] M. S. Abdelfattah, D. Han, A. Bitar, R. DiCecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling, and G. R. Chiu, "Dla: Compiler and fpga overlay for neural network inference acceleration," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2018, pp. 411–4117.

[20] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 599–615.

[21] J. Meng, S. K. Venkataramanaiah, C. Zhou, P. Hansen, P. N. Whatmough, and J. sun Seo, "Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access," *International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 9–16, 2021.

[22] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[23] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. chen Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019.

[24] I. Corp., "Intel® Arria® 10 FPGA and SoC FPGA," https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html, 2023, accessed: 10/03/2023.

[25] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "High-throughput dnn inference with logicnets," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 238–238.

[26] E. Wang, J. J. Davis, P. K. Cheung, and G. A. Constantinides, "Lutnet: Learning fpga configurations for highly efficient neural network inference," *IEEE Transactions on Computers*, vol. 69, no. 12, pp. 1795–1808, dec 2020.

[27] E. Wang, J. J. Davis, G.-I. Stavrou, P. Y. K. Cheung, G. A. Constantinides, and M. Abdelfattah, "Logic shrinkage: Learned fpga netlist sparsity for efficient neural network inference," p. 101–111, 2022.

[28] B. Lou, D. Boland, and P. Leong, "Fsead: A composable fpga-based streaming ensemble anomaly detection library," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, jun 2023.

[29] M. Nazemi, A. Fayyazi, A. Esmaili, A. Khare, S. N. Shahsavani, and M. Pedram, "Nullanet tiny: Ultra-low-latency dnn inference through fixed-function combinational logic," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 266–267.

[30] S. Tridgell, M. Kumm, M. Hardieck, D. Boland, D. Moss, P. Zipf, and P. H. W. Leong, "Unrolling ternary neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 4, oct 2019.

[31] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[32] J. Pool, A. Sawarkar, and J. Rodge, "Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt," https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt, Jul 2021.

[33] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "Vtr 8: High performance cad and customizable fpga architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, 2020.

[34] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 40–47.

[35] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, p. 65–74.

[36] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The vtr project: Architecture and cad for fpgas from verilog to routing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2012, p. 77–86.

[37] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic cad," *International Conference on Field Programmable Logic and Applications*, pp. 1–8, 2013.

[38] A. Arora, A. Boutros, D. Rauch, A. Rajen, A. Borda, S. Damghani, S. Mehta, S. Kate, P. Patel, K. B. Kent, V. Betz, and L. K. John, "Koios: A deep learning benchmark suite for fpga architecture and cad research," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, sep 2021, pp. 355–362.

[39] E. Roorda, S. Rasoulinezhad, P. H. W. Leong, and S. J. E. Wilton, "Fpga architecture exploration for dnn acceleration," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, may 2022.

[40] S. Rasoulinezhad, Siddhartha, H. Zhou, L. Wang, D. Boland, and P. H. W. Leong, "Luxor: An fpga logic cell architecture for effi-

cient compressor tree implementations," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Association for Computing Machinery, 2020, p. 161–171.

[41] 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 *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, p. 94–103.

[42] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced dsp blocks for low-precision deep learning on fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 35–357.

[43] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. Leong, "Pir-dsp: An fpga dsp block architecture for multi-precision deep neural networks," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 35–44.

[44] Y. Chen and M. S. Abdelfattah, "BRAMAC: Compute-in-BRAM Architectures for Multiply-Accumulate on FPGAs," in *31st IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[45] A. Arora, T. Anand, A. Borda, R. Sehgal, B. Hanindhito, J. Kulkarni, and L. K. John, "CoMeFa: Compute-in-Memory Blocks for FPGAs," in *30th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022.

[46] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. R. Iyer, and R. Das, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," in *29th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[47] Y. Chen, J. Dotzel, and M. S. Abdelfattah, "M4BRAM: Mixed-Precision Matrix-Matrix Multiplication in FPGA Block RAMs," in *International Conference on Field Programmable Technology (ICFPT)*, 2023.

[48] C. Eckert, A. K. Subramaniyan, X. Wang, C. Augustine, R. Iyer, and R. Das, "Eidetic: An in-memory matrix multiplication accelerator for neural networks," *IEEE Transactions on Computers*, vol. 72, no. 6, pp. 1539–1553, 2023.

[49] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii - an open-source verilog hdl synthesis tool for cad research," *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–156, 2010.

[50] C. Wolf, "Yosys Open SYnthesis Suite," 2012. [Online]. Available: https://yosyshq.net/yosys/

[51] S. A. Damghani and K. B. Kent, "Yosys+odin-ii: The odin-ii partial mapper with yosys coarse-grained netlists in vtr," *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022.

[52] 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.

[53] A. Kuzmin, M. Nagel, M. van Baalen, A. Behboodi, and T. Blankevoort, "Pruning vs quantization: Which is better?" 2023. [Online]. Available: https://arxiv.org/abs/2307.02973

[54] E. Ahmed and J. S. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 288–298, 2000.

[55] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2019. [Online]. Available: https://arxiv.org/abs/1909.11942