Article

Not peer-reviewed version

Adaptive Dataflow and Precision Optimization for Deep Learning on Configurable Hardware Architectures

Gulnaz Rati *, Rafael Mendes , Aisha Noor

Posted Date: 8 October 2025

doi: 10.20944/preprints202510.0454.v1

Keywords: FPGA; neural network acceleration; deep learning; hardware acceleration; reconfigurable computing; systolic arrays; quantization; high-level synthesis (HLS); edge AI; energy-efficient inference; hardware-software co-design; custom precision arithmetic; dataflow architecture; embedded AI; AI hardware; compiler optimization; low-latency inference; domain-specific architecture; configurable logic; AI at the edge

 doi:10.20944/preprints202510.0454.v1

Article

Adaptive Dataflow and Precision Optimization for Deep Learning on Configurable Hardware Architectures

Gulnaz Rati [1,*], Rafael Mendes [2] and Aisha Noor [3]

[1] University of São Paulo, Brazil
[2] Federal University of Rio de Janeiro, Brazil
[3] University of Campinas, Brazil
* Correspondence: gulnaz.zati@usp.br

Abstract

As deep learning continues to revolutionize a wide range of domains—from computer vision and natural language processing to autonomous systems and edge computing—the demand for efficient, scalable, and domain-adaptable neural network acceleration has never been more critical. While Graphics Processing Units (GPUs) and Application-Specific Integrated Circuits (ASICs) have traditionally dominated the hardware landscape for both training and inference, Field-Programmable Gate Arrays (FPGAs) have recently gained significant traction due to their unique combination of reconfigurability, energy efficiency, and support for highly customized computation. This review presents a comprehensive and in-depth analysis of FPGA-based neural network accelerators, elucidating their architectural foundations, design methodologies, comparative performance characteristics, and deployment challenges in the context of modern machine learning workloads. We begin by examining the core motivations behind using FPGAs for deep learning, highlighting their suitability for low-latency, high-throughput inference, especially in power- and resource-constrained environments such as edge devices and embedded platforms. The ability to define custom data paths, implement novel numeric representations, and tailor memory hierarchies enables FPGAs to execute specialized models with high efficiency, often outperforming GPUs in terms of energy per operation. The review then delves into the major design patterns and architectural strategies employed in FPGA-based accelerators, including systolic arrays, streaming dataflows, loop unrolling, pipelining, and parallelism at various levels of the computation graph. State-of-the-art compilation frameworks and high-level synthesis tools such as Vitis AI, hls4ml, and FINN are discussed in detail, alongside recent advances in quantization, pruning, and model compression techniques that enhance the viability of FPGA deployment. A detailed comparison with GPU- and ASIC-based accelerators is presented, evaluating trade-offs across performance, flexibility, power efficiency, development complexity, and cost. Our findings suggest that FPGAs occupy a compelling middle ground between the general-purpose programmability of GPUs and the ultra-efficient specialization of ASICs, making them particularly well-suited for inference at the edge and in scenarios requiring frequent model updates or architectural experimentation. However, the adoption of FPGAs remains hindered by steep learning curves, toolchain immaturity, and limitations in dynamic runtime adaptability, resource utilization, and developer accessibility. To address these challenges, we survey emerging directions in FPGA research, including adaptive compute fabrics, hardware-software co-design automation, chiplet-based integration, support for dynamic workloads, and secure deployment frameworks. In conclusion, this review articulates the pivotal role that FPGAs can play in the future of AI acceleration. By bridging the gap between general-purpose and application-specific hardware, and by enabling fine-grained control over computation and memory, FPGA-based accelerators offer a highly versatile platform for deploying neural networks in increasingly diverse and demanding operational contexts. Through continued innovation in compiler technologies, hardware architectures, and cross-layer optimization methodologies, the FPGA ecosystem has the potential to evolve into a mainstream enabler of efficient, scalable, and adaptive machine learning systems.

 doi:10.20944/preprints202510.0454.v1

Keywords: FPGA; neural network acceleration; deep learning; hardware acceleration; reconfigurable computing; systolic arrays; quantization; high-level synthesis (HLS); edge AI; energy-efficient inference; hardware-software co-design; custom precision arithmetic; dataflow architecture; embedded AI; AI hardware; compiler optimization; low-latency inference; domain-specific architecture; configurable logic; AI at the edge

1. Introduction

The rapid proliferation of artificial intelligence (AI) and machine learning (ML) applications across diverse domains—ranging from computer vision, speech recognition, and autonomous systems to financial forecasting and medical diagnostics—has fueled a concomitant demand for high-performance, energy-efficient computing platforms capable of accelerating deep neural networks (DNNs). These models, particularly convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, are characterized by immense computational and memory requirements that challenge the capabilities of traditional general-purpose processors. As Moore's Law and Dennard scaling reach their practical limits, novel hardware solutions are imperative to sustain the momentum of AI innovation. Field-Programmable Gate Arrays (FPGAs) have emerged as a compelling platform for accelerating neural networks due to their unique blend of programmability, parallelism, energy efficiency, and low-latency processing. Unlike application-specific integrated circuits (ASICs), which offer high performance at the cost of flexibility, FPGAs provide a reconfigurable fabric that enables tailored architectural designs without incurring the non-recurring engineering (NRE) costs and time-to-market delays associated with ASIC development. Furthermore, in contrast to graphics processing units (GPUs), which excel at data-parallel workloads but suffer from high power consumption and memory bandwidth bottlenecks, FPGAs offer fine-grained control over hardware resources and the potential for deterministic performance, making them suitable for real-time and embedded inference applications. The suitability of FPGAs for DNN acceleration stems from several architectural features. First, the abundance of logic elements, digital signal processing (DSP) blocks, and block RAMs (BRAMs) facilitates the implementation of highly parallel and pipelined computation paths [1]. Second, the programmable interconnect fabric allows for customized dataflows, enabling optimizations such as weight reuse, data locality, and reduced off-chip memory access [2]. Third, modern high-level synthesis (HLS) tools and machine learning compilers (e.g., Xilinx Vitis AI, Intel OpenVINO, hls4ml, and TVM) have significantly lowered the barrier to entry for FPGA development, allowing researchers and practitioners to explore a wide design space with relative ease. Despite these advantages, the design of efficient FPGA-based neural network accelerators remains a challenging task that demands careful consideration of multiple, often conflicting, objectives [3]. These include maximizing throughput and energy efficiency while minimizing latency, resource utilization, and design complexity [4]. Design choices must account for model characteristics (e.g., layer types, precision requirements, sparsity), hardware constraints (e.g., logic and memory resources, I/O bandwidth, operating frequency), and deployment scenarios (e.g., edge vs [5]. cloud, static vs [6]. dynamic workloads). Consequently, a rich body of literature has emerged that explores architectural innovations, optimization techniques, and design methodologies to harness the full potential of FPGAs for DNN acceleration [7]. This review aims to provide a comprehensive and structured survey of FPGA-based neural network accelerators, highlighting key trends, architectural paradigms, and trade-offs in the design space [8]. We systematically categorize existing accelerator architectures based on their target network types, dataflow models, and level of customization. Additionally, we examine the impact of quantization, pruning, and model compression on FPGA implementation efficiency, and discuss emerging directions such as dynamic reconfiguration, hardware/software co-design, and support for evolving network architectures like transformers and spiking neural networks. Through this survey, we seek to inform

 doi:10.20944/preprints202510.0454.v1

researchers, engineers, and system designers of the state-of-the-art in FPGA-based DNN acceleration and provide insights into future research opportunities in this dynamic and interdisciplinary field.

2. Background and Mathematical Foundations of Neural Network Computations

In order to design efficient FPGA-based accelerators for neural networks, it is essential to first understand the mathematical operations that underpin modern deep learning models. Neural networks are, at their core, compositions of affine transformations and non-linear activation functions. These operations are computationally intensive and highly parallelizable, making them ideal candidates for hardware acceleration [9].

2.1. Fundamentals of Neural Network Layers

A neural network can be mathematically represented as a parameterized function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$, where $\boldsymbol{\theta}$ denotes the set of learnable parameters (weights and biases), n is the dimension of the input feature vector, and m is the dimension of the output [10]. The function $f_{\boldsymbol{\theta}}$ is typically composed of L sequential layers:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(\mathbf{x})$$

Each layer $f^{(l)}$ is generally defined by an affine transformation followed by a non-linear activation:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{a}^{(l)} = \phi^{(l)}(\mathbf{z}^{(l)})$$

where:

- $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix for layer l,
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector,
- $\phi^{(l)} : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is the activation function (e.g., ReLU, sigmoid, tanh),
- $\mathbf{a}^{(l)} \in \mathbb{R}^{n_l}$ is the output (activation) of layer l,
- $\mathbf{a}^{(0)} = \mathbf{x} \in \mathbb{R}^{n_0}$ is the input vector.

2.2. Convolutional Layers

A convolutional layer is a special case where the affine transformation is implemented via a discrete convolution operation. Given an input feature map $\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$, a convolutional kernel $\mathbf{K} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K_H \times K_W}$, and stride s, the output feature map $\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times H' \times W'}$ is computed as:

$$\mathbf{Y}_{c_{\text{out}},h,w} = \sum_{c_{\text{in}}=1}^{C_{\text{in}}} \sum_{i=1}^{K_H} \sum_{j=1}^{K_W} \mathbf{K}_{c_{\text{out}},c_{\text{in}},i,j} \cdot \mathbf{X}_{c_{\text{in}},s\cdot h+i,s\cdot w+j}$$

This operation can be reformulated as a generalized matrix multiplication (GEMM) through techniques such as im2col, which enables a more straightforward mapping to hardware primitives like multiply-accumulate (MAC) units.

2.3. Activation Functions

Non-linear activation functions are essential to introduce non-linearity into the model. Some common activation functions include:

- **Rectified Linear Unit (ReLU)**: $\phi(z) = \max(0, z)$
- **Sigmoid**: $\phi(z) = \frac{1}{1+e^{-z}}$
- **Hyperbolic Tangent**: $\phi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- **Softmax (for classification)**:

$$\phi(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \text{for } i = 1, \ldots, n$$

From a hardware perspective, the ReLU is particularly attractive due to its simplicity, requiring only a comparison and a multiplexer [11].

 doi:10.20944/preprints202510.0454.v1

2.4. Fully Connected Layers

Fully connected (dense) layers can be expressed as:

$$\mathbf{a}^{(l)} = \phi(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

This operation is a straightforward matrix-vector multiplication, which can be parallelized at various granularities depending on the available hardware resources. FPGAs can implement this using systolic arrays, parallel MAC units, or pipelined architectures.

2.5. Loss Functions and Backpropagation

The loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ quantifies the discrepancy between the predicted output $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$. Common loss functions include mean squared error (MSE) and categorical cross-entropy. For training, the backpropagation algorithm computes gradients of the loss with respect to each parameter $\boldsymbol{\theta}_i$ using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \cdot \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} \cdot \frac{\partial \mathbf{z}^{(L)}}{\partial \boldsymbol{\theta}_i}$$

While training is generally performed offline on GPUs or cloud-based infrastructures, FPGAs are increasingly being explored for on-device training and inference fine-tuning.

2.6. Quantization and Fixed-Point Arithmetic

To reduce memory footprint and power consumption, neural networks are often quantized from floating-point to fixed-point representations. Let $x \in \mathbb{R}$ be a real number and $\hat{x} \in \mathbb{Q}$ its quantized approximation. The quantization function $Q(x)$ can be defined as:

$$\hat{x} = Q(x) = \text{clip}\left(\text{round}\left(\frac{x}{s}\right), q_{\min}, q_{\max}\right) \cdot s$$

where $s \in \mathbb{R}$ is the scaling factor, and $q_{\min}, q_{\max}$ define the range of quantized values (e.g., for 8-bit integers, $q_{\min} = -128$, $q_{\max} = 127$) [12]. Fixed-point arithmetic is particularly well-suited for FPGAs due to its low hardware cost and predictable latency.

2.7. Sparsity and Pruning

Sparsity, the property of having a large number of zero-valued weights or activations, can be exploited to reduce the computational load:

$$\mathbf{y} = \mathbf{W}\mathbf{x}, \quad \text{with } W_{ij} = 0 \Rightarrow \text{skip MAC}$$

Let $\delta_{ij} \in \{0, 1\}$ be a binary mask denoting whether a weight is active:

$$\mathbf{y}_i = \sum_{j=1}^{n} \delta_{ij} \cdot W_{ij} \cdot x_j$$

Sparse representations can be efficiently mapped to custom FPGA architectures using zero-skipping techniques and compressed storage formats (e.g., CSR, CSC).

2.8. Parallelism and Dataflow Models

FPGAs allow the exploitation of various forms of parallelism in neural network computations:

- **Data-level parallelism (DLP)**: Simultaneous computation across input batches.
- **Model-level parallelism**: Distribution of model layers or channels across parallel units.
- **Instruction-level parallelism**: Pipelined operations at the arithmetic unit level [13].
- **Bit-level parallelism**: Use of bit-serial or mixed-precision computation.

 doi:10.20944/preprints202510.0454.v1

Dataflow models such as weight stationary (WS), output stationary (OS), and row stationary (RS) determine how data is reused and moved within the accelerator. Each has implications for on-chip buffer design, memory bandwidth usage, and throughput [14].

2.9. Summary

The mathematical operations discussed above form the computational backbone of modern deep learning models [15]. Understanding these operations in their algebraic and numerical context is a prerequisite for designing efficient FPGA-based accelerators [16]. The next sections will delve into architectural strategies, dataflow optimization, memory management, and real-world implementations that leverage these mathematical principles to achieve high-performance inference and training on reconfigurable hardware platforms [17].

3. Architectural Design of FPGA-Based Neural Network Accelerators

Designing high-performance neural network accelerators on FPGAs necessitates a meticulous exploration of the hardware architecture that supports massively parallel, low-latency computation. Unlike general-purpose processors, FPGAs offer a reconfigurable substrate composed of logic elements, block RAMs (BRAMs), digital signal processors (DSPs), and programmable interconnects, allowing for highly customized datapaths [18]. This section presents a comprehensive analysis of architectural strategies employed in FPGA-based neural network accelerators, organized around fundamental hardware design principles and components.

3.1. Computation Units: Multiply-Accumulate (MAC) Engines

At the core of virtually every neural network accelerator lies the multiply-accumulate (MAC) unit, responsible for performing operations of the form:

$$y = \sum_{i=1}^{n} w_i \cdot x_i$$

MAC units are implemented using FPGA DSP slices, which are optimized for fixed-point and floating-point multiplication and accumulation. A single MAC operation typically maps to:

$$\text{MAC}(x_i, w_i) = \text{DSP} \leftarrow (x_i \cdot w_i) + \text{acc}$$

Designers often instantiate arrays of MAC units to process multiple inputs and outputs concurrently [19]. For a layer with N_{in} inputs and N_{out} outputs, a fully parallel implementation would require $N_{\text{in}} \times N_{\text{out}}$ MAC units—often impractical due to resource constraints. Therefore, partial parallelism combined with time-multiplexing is commonly adopted [20].

3.2. Systolic Arrays

Systolic arrays provide a scalable architecture for implementing matrix multiplication, which underpins both fully connected and convolutional layers. A 2D systolic array comprises a grid of processing elements (PEs) that communicate in a pipelined fashion [21]. Each PE performs a MAC operation:

$$\text{PE}_{i,j} : y_{i,j} \leftarrow y_{i,j} + x_i \cdot w_j$$

The systolic architecture is advantageous due to its regular structure, local communication, and high throughput. The matrix multiplication $\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$, where $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{X} \in \mathbb{R}^{n \times p}$, can be computed over a $m \times p$ PE array with input and weight matrices streamed from the edges.

3.3. On-Chip Memory Hierarchy

Efficient memory management is critical for minimizing off-chip data movement and maximizing computational efficiency [22]. FPGA designs typically employ a multi-level memory hierarchy:

 doi:10.20944/preprints202510.0454.v1

- **Registers**: Hold temporary data within PEs (lowest latency) [23].
- **Block RAMs (BRAMs)**: Serve as scratchpads for activations, weights, and intermediate results.
- **UltraRAM (URAM)**: In large FPGAs, URAM provides higher capacity than BRAM [24].
- **Off-chip DRAM**: Stores full network parameters and batch inputs when on-chip resources are insufficient [25].

Let A_{on} and W_{on} be the sets of activations and weights stored on-chip. The goal is to maximize:

$$\frac{|A_{\text{on}}| + |W_{\text{on}}|}{|A_{\text{total}}| + |W_{\text{total}}|}$$

while adhering to capacity constraints:

$$|A_{\text{on}}| \cdot b_a + |W_{\text{on}}| \cdot b_w \leq M_{\text{BRAM}} + M_{\text{URAM}}$$

where b_a, b_w denote the bit widths of activations and weights, respectively [26].

3.4. Dataflow Models

The choice of dataflow model significantly impacts buffer reuse, interconnect traffic, and latency [27]. Common models include:

- **Weight Stationary (WS)**: Keep weights static in PEs while inputs and outputs move [28].

$$\text{Reuse: } w_{ij} \text{ used } N_{\text{in}} \text{ times}$$

- **Output Stationary (OS)**: Keep partial sums in registers, accumulating contributions from streamed weights and inputs.

$$y_j^{(k)} = y_j^{(k)} + w_{ij} \cdot x_i$$

- **Row Stationary (RS)**: Optimizes reuse of weights, inputs, and partial sums by tiling the computation.
- **No Local Reuse (NLR)**: Stream-based architectures that do not exploit data reuse but allow pipelining and high throughput for small models.

Let R_w, R_x, R_y be the number of times a weight, input, or output is reused in a given schedule. The goal is to maximize reuse:

$$\max\left(\alpha_w R_w + \alpha_x R_x + \alpha_y R_y\right)$$

subject to memory and PE bandwidth constraints [29].

3.5. Control Logic and Scheduling

Control logic orchestrates the flow of data and execution of operations across PEs. A static schedule assumes a fixed topology and order of operations, whereas a dynamic schedule supports control flow for models with conditional branches, loops, or varying layer configurations. For example, a control FSM (finite state machine) might be used to sequence the following phases:

1. Load weights from DRAM to BRAM.
2. Stream activations into the systolic array [30].
3. Accumulate results and apply activation.
4. Store output to BRAM or DRAM.

Designers aim to minimize idle cycles and maximize PE utilization:

$$\text{Utilization} = \frac{T_{\text{active}}}{T_{\text{total}}}$$

where T_{active} denotes the time PEs are performing valid MAC operations [31].

 doi:10.20944/preprints202510.0454.v1

3.6. Pipeline and Parallelism Strategies

Pipeline parallelism is exploited at multiple granularities:

- **Inter-layer pipelining**: Consecutive layers process different data samples in parallel [32].
- **Intra-layer pipelining**: Separate pipeline stages for fetch, compute, accumulate, and write-back.
- **Loop unrolling**: Explicit replication of hardware for inner loops of MAC operations.

Given a loop nest:

$$\texttt{for } i = 1 \texttt{ to } N \texttt{ do} \quad y += w_i \cdot x_i$$

an unrolling factor U yields a speedup of approximately:

$$\text{Speedup} \approx \min\left(U, \frac{N}{L}\right)$$

where L is the latency per MAC.

3.7. Parameterization and Design Space Exploration

Modern accelerators are often fully parameterized in terms of:

$$\{T_x, T_y, P_w, P_a, b_w, b_a\}$$

where:

- T_x, T_y: Tiling factors for input and output.
- P_w, P_a: Parallelism factors for weights and activations.
- b_w, b_a: Bit widths of weights and activations.

Exploring this design space involves a multi-objective optimization problem:

$$\max_{\theta \in \Theta} [\text{Throughput}(\theta),\ \text{Energy Efficiency}(\theta)] \quad \text{s.t [33]. Resource Usage}(\theta) \leq R_{\max}$$

Techniques such as analytical modeling, design space pruning, and evolutionary algorithms are employed to identify Pareto-optimal configurations.

3.8. Summary

The architectural design of FPGA-based neural network accelerators involves a rich interplay between algorithmic properties, hardware constraints, and system-level objectives. By leveraging fine-grained parallelism, reconfigurable datapaths, and customized memory hierarchies, FPGAs offer a unique platform for efficient deep learning inference. The next section will delve into optimization techniques, including quantization, pruning, and mixed-precision computation, that further enhance performance and resource efficiency in these architectures[34].

4. Optimization Techniques for FPGA-Based Neural Network Accelerators

The inherent flexibility of FPGAs makes them well-suited for implementing a wide array of architectural and algorithmic optimizations that are crucial for accelerating deep neural networks [35]. These optimizations are necessary not only to meet stringent constraints on power, area, and latency but also to fully exploit the potential of reconfigurable hardware in domains such as edge computing, embedded vision, and autonomous systems. In this section, we explore a range of key optimization strategies in extreme detail, beginning with numerical precision reduction techniques and proceeding through structural network simplification, dataflow enhancement, and hardware-aware neural architecture search [36]. One of the most impactful optimizations for FPGA-based neural network accelerators is numerical precision reduction through quantization. Unlike GPUs and CPUs, which are often optimized for IEEE 754 floating-point arithmetic, FPGAs allow designers to choose arbitrary bit widths and numeric representations, enabling a fine-grained trade-off between accuracy,

 doi:10.20944/preprints202510.0454.v1

resource utilization, and throughput [37]. Quantization involves converting the high-precision weights and activations of a neural network (typically 32-bit floating-point) into lower-precision fixed-point or integer representations. Let $w \in \mathbb{R}$ be a real-valued weight, and let $\hat{w} = Q(w) \in \mathbb{Z}$ be its quantized form. The quantization function Q can be linear (uniform quantization) or non-linear (logarithmic, k-means-based, etc.). Uniform affine quantization maps a continuous value to an integer as $\hat{w} = \text{round}(w/s)$, where s is the scaling factor. During inference, this is reversed via dequantization: $w \approx \hat{w} \cdot s$ [38]. On FPGAs, quantized operators reduce both computation time and memory bandwidth, allowing, for instance, an 8-bit MAC unit to consume one-fourth the resources of a 32-bit floating-point equivalent. More aggressive schemes, such as binary neural networks (BNNs) and ternary weight networks, push this concept further by restricting weights and activations to $\{-1, +1\}$ or $\{-1, 0, +1\}$, enabling the replacement of multipliers with XNOR or multiplexing logic. However, aggressive quantization introduces accuracy degradation, which must be compensated by retraining or fine-tuning [39]. FPGA accelerators often support mixed-precision designs, where critical layers (e.g., first and last layers) operate at higher bit widths (e.g., 16-bit) while intermediate layers use lower precision (e.g., 4- or 8-bit), thus balancing accuracy and efficiency. Bit-serial arithmetic and approximate computing techniques may also be employed to further reduce logic usage, although this increases control complexity and may introduce non-negligible latency overheads. Another major optimization technique is network pruning, which aims to eliminate redundant parameters and computations from neural networks [40]. By leveraging the observation that many weights in a trained network contribute negligibly to its final output, pruning algorithms generate sparse networks by zeroing out small-magnitude weights or removing entire neurons, channels, or filters. Mathematically, pruning can be viewed as applying a sparsity-inducing regularizer (e.g., ℓ_1 norm) during training, leading to a weight tensor $\mathbf{W}$ such that $\|\mathbf{W}\|_0 \ll \text{dense}(\mathbf{W})$. For example, structured pruning eliminates entire rows or columns of the weight matrix, preserving compatibility with parallel computation, whereas unstructured pruning leads to arbitrary sparsity patterns that require index bookkeeping and zero-skipping logic during execution. On FPGAs, sparsity is particularly advantageous, as unnecessary MAC operations can be completely omitted, reducing dynamic power and enabling higher clock frequencies due to lower switching activity. To efficiently map sparse networks onto FPGAs, accelerators may use compressed sparse row (CSR), compressed sparse column (CSC), or coordinate list (COO) representations, alongside custom decoders and address generators to fetch non-zero values and their indices [41]. However, the irregular access patterns introduced by unstructured sparsity can negate the performance benefits unless carefully managed [42]. Structured pruning, by contrast, allows entire computation blocks to be bypassed or removed at compile-time, resulting in more predictable and scalable performance. Pruning is often combined with quantization, a synergy known as quantization-aware pruning, which maximizes the reduction in memory footprint while minimizing accuracy loss [43]. Data reuse and locality-aware scheduling are additional critical optimizations for FPGAs, where the high cost of off-chip memory accesses necessitates minimizing redundant data transfers [44]. Convolutional neural networks, in particular, exhibit substantial input, weight, and output reuse, which can be exploited through loop tiling, loop interchange, and caching strategies [45]. For example, in a convolutional layer with kernel size $K \times K$, stride S, and input feature map of size $H \times W$, each weight element is reused $\left(\frac{H-K}{S}+1\right) \cdot \left(\frac{W-K}{S}+1\right)$ times, assuming no padding. Likewise, input pixels participate in multiple overlapping convolution windows [46]. An efficient accelerator must schedule operations such that these reused values remain on-chip as long as possible, residing in local registers or BRAMs. Tiling breaks down the input/output tensors into smaller blocks that fit within the available memory, allowing temporal reuse of weights and activations while avoiding off-chip memory thrashing. Loop interchange further helps by adjusting the order of nested loops to maximize reuse factors [47]. Designers often employ analytical models or performance simulators to derive optimal tiling factors and buffer sizes that balance latency, memory usage, and parallelism. Data reuse is especially effective when combined with output-stationary or row-stationary dataflows, which preserve partial sums in registers or local BRAMs throughout

 doi:10.20944/preprints202510.0454.v1

the computation, thereby minimizing intermediate data movement [48]. At a higher level, DMA engines and burst-mode memory access patterns are used to optimize DRAM transactions, and double buffering is adopted to overlap data transfer with computation [49]. Finally, a rapidly emerging field of optimization is hardware-aware neural architecture search (NAS), which aims to automatically discover network topologies that are co-optimized for accuracy and hardware efficiency [50]. Unlike traditional NAS that searches purely for accuracy under generic constraints (e.g., FLOPs), hardware-aware NAS explicitly incorporates metrics such as latency, LUT/DSP/BRAM usage, and energy into its search objective. Let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be the space of architectural candidates, and let $\mathcal{O} : \mathcal{A} \to \mathbb{R}^k$ be a vector-valued objective function capturing k design goals (e.g., accuracy, power, inference time). Then NAS solves:

$$\min_{a \in \mathcal{A}} \mathcal{L}(a) = \lambda_1 \cdot \text{Loss}(a) + \lambda_2 \cdot \text{Latency}(a) + \lambda_3 \cdot \text{Resources}(a)$$

where λ_i are weights reflecting design trade-offs. In practice, latency and resource usage must be estimated or measured on a target FPGA using cost models or hardware emulation. Search strategies such as reinforcement learning, evolutionary algorithms, or differentiable architecture search (DARTS) guide the exploration of this vast design space [51]. Importantly, the discovered architectures often feature non-standard operations (e.g., separable convolutions, group convolutions, channel shuffle) that can be challenging to implement efficiently on FPGAs unless the accelerator is highly modular and reconfigurable [52]. Therefore, a co-design methodology, where both the neural network and hardware architecture evolve in tandem, is critical to achieving optimal performance. In summary, optimization techniques for FPGA-based neural network accelerators span a diverse array of numerical, structural, and architectural domains. Quantization reduces arithmetic complexity, pruning eliminates redundancy, data reuse enhances memory efficiency, and NAS introduces automation and intelligence into the design loop [53]. Each of these techniques can be implemented at various granularities and may be interdependent, requiring careful integration into the accelerator design process. As neural networks grow in complexity and deployment moves toward resource-constrained environments, such optimizations are not merely desirable but essential for achieving real-time, energy-efficient inference on reconfigurable hardware.

5. Case Studies and Benchmarking of FPGA-Based Neural Network Accelerators

To concretely demonstrate the practical relevance and performance trade-offs of FPGA-based neural network accelerators, it is instructive to examine a variety of case studies drawn from the literature and industrial implementations. These case studies highlight diverse application domains—such as image classification, object detection, natural language processing, and anomaly detection—and reveal the impact of architectural decisions, optimization techniques, and hardware resource allocation on critical metrics including throughput, latency, energy efficiency, and logic utilization. Despite the diversity of platforms and neural network workloads, certain architectural themes and performance trends consistently emerge, providing valuable insights for future accelerator design [54]. Consider, for example, the Xilinx Deep Learning Processing Unit (DPU), a commercial FPGA IP core designed for high-throughput CNN inference [55]. The DPU adopts a deeply pipelined, VLIW-style architecture that features parallel MAC engines, configurable weight/activation bit widths, and multiple compute units operating in parallel. In representative configurations targeting the Zynq UltraScale+ MPSoC platform, the DPU achieves performance levels of over 200 GOPS (giga-operations per second) while sustaining a power envelope below 5W. Performance is achieved by integrating multiple optimization techniques described in previous sections: quantization-aware training (INT8 inference), aggressive loop tiling, a unified on-chip memory for fast data reuse, and a compiler-assisted instruction scheduling engine that statically analyzes data dependencies. Importantly, the DPU is tailored for specific workload classes such as MobileNet and ResNet, and it incorporates prefetching and caching strategies to mitigate DRAM access latency. Academic designs such as the MIT Eyeriss architecture also illustrate key trade-offs. Eyeriss introduced the concept of row-stationary dataflow, which prioritizes on-chip

 doi:10.20944/preprints202510.0454.v1

reuse of partial sums and weights over raw throughput. This design choice leads to a more balanced use of resources, favoring energy efficiency and flexibility. Eyeriss maps each processing element in a 2D array to a spatially-aware data reuse pattern, which, combined with on-chip global buffers and a hierarchical NoC (network-on-chip), allows it to dynamically adjust to different layers of a CNN [56]. While its peak performance may be lower than purely throughput-optimized designs, its energy per inference is significantly reduced, making it ideal for embedded and mobile environments. To provide a generalized abstraction of these types of FPGA accelerators, we can visualize a parameterized hardware architecture that illustrates the interaction between computation units, memory hierarchies, and control infrastructure. Figure 1 presents a simplified but representative model.

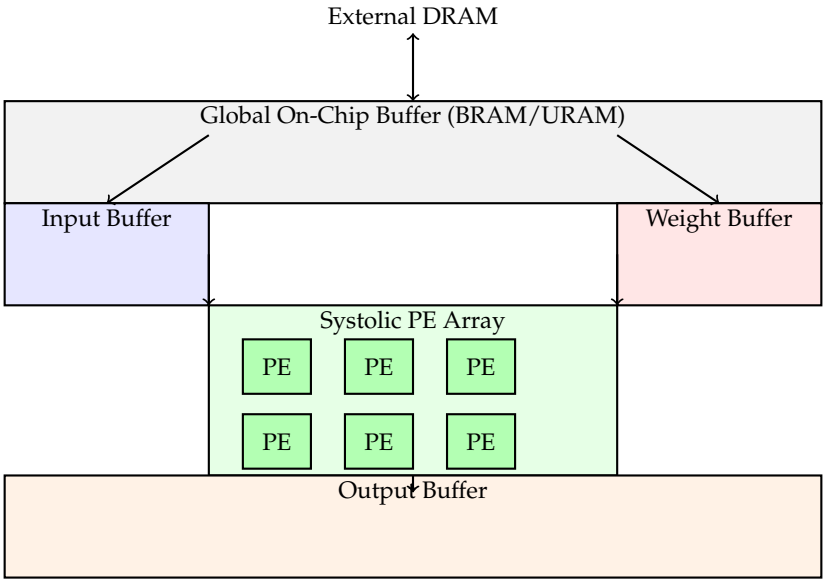

Figure 1. Simplified block diagram of an FPGA-based neural network accelerator architecture [57]. Key components include on-chip buffers, a systolic PE array, and memory interfaces optimized for high throughput and low latency.

This architecture comprises several major subsystems [58]. At the top lies the global on-chip buffer, typically composed of BRAM or URAM blocks, responsible for staging data transferred from external DRAM [59]. Input activations and weights are partitioned and routed to dedicated local buffers, which in turn feed a 2D systolic array of processing elements. Each PE performs a multiply-accumulate operation on incoming streams of activations and weights, either broadcasting or shifting data through the array depending on the dataflow strategy. Outputs are accumulated either within the array or in a dedicated accumulator buffer and then passed to the output buffer, which stores the final results or intermediate feature maps before offloading to DRAM [60]. Throughout the system, control logic orchestrates data transfers, synchronization, and instruction sequencing, while DMA engines ensure burst transfers and memory coalescing. Benchmarking results across various platforms and networks underscore the performance variability introduced by hardware configuration, optimization depth, and workload characteristics. For instance, on an Intel Stratix 10 FPGA, a hand-optimized design implementing ResNet-50 with INT8 quantization and pipelined convolution blocks achieved over 500 GOPS at 250 MHz, with resource utilization peaking at 80% of available DSPs and 60% of BRAM [61]. In contrast, a design targeting a smaller Artix-7 FPGA implementing a binarized neural network (BNN) reported inference latencies under 1 ms with power consumption under 1 W, albeit with lower classification accuracy [62]. Such comparisons emphasize the importance of matching design objectives to application constraints, and they highlight the tunability of FPGAs as a unique advantage over fixed-architecture ASICs or GPUs [63]. To enable fair benchmarking, standardized datasets (e.g., ImageNet, CIFAR-10), reference networks (e.g., AlexNet, VGG, MobileNet), and open-source design flows (e.g., FINN, hls4ml, VTA) are increasingly employed by the FPGA research community [64]. Metrics such

 doi:10.20944/preprints202510.0454.v1

as GOPS/W (operations per watt), inference per second (IPS), resource utilization breakdown (LUTs, FFs, DSPs, BRAMs), and end-to-end latency are typically reported. However, care must be taken in comparing results due to differences in precision, batch size, clock frequency, and toolchain efficiency [65]. Despite these challenges, case studies remain an essential part of evaluating and advancing the state of FPGA-based neural network acceleration, serving as both proof of concept and validation of architectural innovation.

6. Comparison with GPU and ASIC Accelerators

While FPGAs have emerged as a powerful and flexible platform for accelerating deep neural networks, they coexist in a broader ecosystem of specialized hardware that includes graphics processing units (GPUs) and application-specific integrated circuits (ASICs). Each of these platforms exhibits distinct strengths and limitations in terms of performance, energy efficiency, programmability, scalability, and cost, and the decision to adopt a particular solution depends heavily on the target application domain, deployment context, and workload characteristics. This section provides an in-depth comparative analysis of FPGAs versus GPUs and ASICs, examining architectural differences, design trade-offs, toolchain maturity, and real-world deployment considerations [66]. At a high level, the most prominent advantage of GPUs lies in their extremely high throughput, enabled by hundreds to thousands of parallel cores and a SIMT (single instruction, multiple threads) execution model. GPUs excel in batch processing of large datasets, especially in training scenarios, due to their optimized floating-point arithmetic units and fast global memory bandwidth. In contrast, ASICs represent the pinnacle of specialization; they are designed for a single application or a narrow range of tasks and can achieve unmatched energy efficiency and throughput-per-watt by eliminating all unnecessary logic, tailoring every aspect of the microarchitecture to the target workload. FPGAs occupy the space between these two extremes: more flexible and reprogrammable than ASICs, yet more energy-efficient and application-customizable than GPUs. Their unique value proposition lies in their ability to implement application-specific parallelism, custom data paths, and non-standard precision formats, while still allowing post-fabrication reconfiguration. Table 1 summarizes the comparative characteristics of FPGAs, GPUs, and ASICs across multiple dimensions relevant to neural network acceleration [67].

As shown in Table 1, the trade-off between performance and flexibility is central to the choice among FPGA, GPU, and ASIC accelerators. FPGAs offer a compelling balance for edge deployment scenarios where inference must be performed under tight power and latency budgets, and where workloads may evolve post-deployment. Unlike ASICs, which cannot be reconfigured once manufactured, FPGAs allow for iterative updates, making them suitable for prototyping, low-volume deployment, and field-reprogrammable systems such as autonomous drones, smart cameras, or portable medical devices [68]. They are also advantageous in situations where custom dataflows or exotic network topologies (e.g., transformer variants or sparse CNNs) must be mapped efficiently, something that GPUs often struggle with due to rigid memory hierarchies and SIMT constraints. However, FPGAs also come with notable challenges. Their programming model is more complex and less accessible than that of GPUs, often requiring domain-specific hardware knowledge or the use of high-level synthesis (HLS) tools that may not generate optimal hardware by default [69]. Despite significant advances in design automation (e.g., vendor platforms such as Xilinx Vitis AI and Intel OpenVINO), the development cycle remains longer than for GPU-based inference. Moreover, FPGAs tend to have lower clock frequencies than GPUs or ASICs and may suffer from underutilization if the accelerator is not properly tailored to the specific workload [70]. On the other hand, ASICs, while representing the gold standard in terms of energy and throughput, require enormous up-front investment in design, verification, and silicon fabrication, which limits their use to high-volume deployments such as smartphone SoCs, data center TPUs, or automotive-grade ADAS chips. In conclusion, while GPUs remain dominant in large-scale deep learning training and ASICs deliver unmatched efficiency in mass-produced products, FPGAs offer a unique middle ground that blends flexibility, efficiency, and customizability. Their ability to support custom numeric formats, layer-specific acceleration strategies, and dataflow-aware

 doi:10.20944/preprints202510.0454.v1

architectures make them an essential tool in the hardware acceleration toolbox, particularly as the field shifts toward edge intelligence, heterogeneous computing, and rapid deployment of new model variants. Understanding the detailed trade-offs among these platforms is therefore critical for system architects aiming to deploy robust, efficient, and scalable AI systems.

Table 1. Comparison of FPGA, GPU, and ASIC Accelerators for Neural Networks.

Attribute	FPGA	GPU	ASIC
Performance (Throughput)	Medium to high (100–1000+ GOPS), depending on optimization; tailored parallelism	Very high (1–10 TOPS), especially for training and large batch inference	Extremely high (10–100+ TOPS), with optimal architectural tuning
Energy Efficiency (GOPS/W)	High, especially with quantized inference and sparse networks	Moderate; high dynamic power due to general-purpose execution	Very high; optimized for minimal switching and maximal reuse
Flexibility/Reconfigurability	Fully reprogrammable post-deployment; supports custom dataflows and precision	Fixed architecture but programmable kernels; supports general DL frameworks	Fixed hardware; no post-fabrication flexibility
Latency (Single Inference)	Very low, especially for pipelined or low-batch designs	Moderate to high; optimized for batch throughput rather than single-instance latency	Very low; fixed pipelines enable constant and deterministic latency
Precision Support	Arbitrary precision (e.g., INT4, INT8, binary); designer-controlled	Typically FP16, FP32, INT8; limited custom precision	Fixed or limited configurable precision (e.g., INT8, binary)
Development Complexity	High; requires RTL/HLS expertise or domain-specific compilers	Low to moderate; supported by mature CUDA/OpenCL toolchains	Very high; requires full silicon design, validation, and tape-out
Toolchain Maturity	Improving; vendor tools (Vitis, Vivado HLS) and open frameworks (FINN, hls4ml)	Highly mature; full ecosystem support (TensorFlow, PyTorch, cuDNN)	Proprietary; limited reuse and vendor-dependent
Deployment Cost	Moderate; lower than ASICs, higher than commodity GPUs	Low to moderate; depends on GPU tier and volume	Very high; amortized only at scale (high NRE)
Time to Market	Short to moderate; faster than ASICs due to post-fabrication programmability	Very short; off-the-shelf availability	Very long; includes full VLSI flow and fabrication delays
Use Case Fit	Ideal for edge, embedded, and latency-sensitive inference; adaptable workloads	Best for data center training and high-throughput inference	Optimal for high-volume, power-constrained, and dedicated-use cases

7. Challenges and Future Directions in FPGA-Based Neural Network Acceleration

Despite the growing maturity and demonstrated success of FPGA-based neural network accelerators, the domain continues to grapple with numerous technical, practical, and systemic challenges [71]. These challenges span from low-level hardware limitations to high-level software abstraction barriers and deployment ecosystem gaps. Overcoming these issues is essential not only for scaling current FPGA-based solutions to meet the increasingly complex demands of modern AI workloads,

 doi:10.20944/preprints202510.0454.v1

but also for expanding their accessibility to a broader community of developers, researchers, and system integrators. At the same time, the field presents an abundance of future opportunities—from tighter hardware-software co-design methodologies to the integration of emerging memory technologies and adaptive runtime systems—which collectively point toward a vibrant and innovative trajectory for the next generation of FPGA-accelerated machine learning systems [72]. One of the most persistent challenges in this space is the difficulty of achieving both high performance and ease of programmability. Traditional hardware design using register-transfer level (RTL) languages such as Verilog or VHDL is both time-consuming and error-prone, requiring deep expertise in digital logic design, timing analysis, and FPGA-specific constraints such as clock domain crossing, floorplanning, and resource budgeting [73]. High-Level Synthesis (HLS) has emerged as a promising alternative, allowing designers to describe functionality in C/C++ or OpenCL, but current HLS tools often suffer from suboptimal code generation, opaque optimization decisions, and steep learning curves when tuning performance-critical constructs such as loop unrolling, memory partitioning, and pipelining. Furthermore, machine learning models themselves are increasingly authored in high-level frameworks like PyTorch or TensorFlow, creating a semantic gap between network designers and hardware architects [74]. Bridging this divide requires a new generation of compiler and synthesis tools that can automatically map high-level neural network descriptions into efficient hardware implementations, with minimal manual intervention. Projects such as hls4ml, FINN, and VTA (Versatile Tensor Accelerator) are steps in this direction, but the general problem of automated, high-performance hardware generation from abstract model specifications remains unsolved. Resource limitations and hardware constraints also pose substantial hurdles. Unlike ASICs, which can be engineered to include precisely the resources needed for a given neural network workload, FPGAs must work within fixed budgets of logic elements, DSP blocks, on-chip memory (BRAM/URAM), and I/O bandwidth. For large-scale models such as BERT, GPT, or even deep convolutional networks with extensive parameter counts, naïve mappings onto FPGAs can easily exceed available resources, resulting in the need for aggressive quantization, pruning, or off-chip data transfers that reduce computational efficiency [75]. Moreover, managing memory hierarchy on FPGAs remains a complex task; the lack of hardware-managed caches necessitates explicit control over buffer sizing, data reuse, and access scheduling. Inefficient memory use not only limits performance but can also cause severe timing closure issues, as routing congestion and excessive fanout delay signal propagation across the FPGA fabric. The development of new memory-centric accelerator architectures, possibly leveraging high-bandwidth memory (HBM), 3D stacked memory, or near-memory compute primitives, represents a promising direction to address this bottleneck. Another pressing issue lies in the lack of robust support for dynamic workloads [76]. Most FPGA accelerators for neural networks are statically scheduled and optimized for fixed model architectures and dimensions [77]. This inflexibility can become a serious limitation in real-world deployment scenarios where models evolve, inputs vary in shape or size, or multiple applications share the same compute platform. Dynamically reconfigurable FPGAs and partial reconfiguration offer a path toward addressing this issue, but current toolchains for partial reconfiguration are immature, slow, and incompatible with rapid model iteration cycles [78]. Runtime-adaptive architectures that support multiplexed execution, elastic data paths, and configurable compute fabrics may alleviate these limitations, but they introduce additional complexity in control logic, verification, and resource sharing [79]. A future trend may involve hybrid solutions in which FPGAs act as dynamic co-processors working alongside general-purpose processors or ASICs, orchestrated by intelligent scheduling software that performs real-time workload profiling and accelerator tuning. Security and reliability are additional challenges that are increasingly important as FPGAs are deployed in edge environments with limited physical and cyber protections. Unlike ASICs or secure enclaves within CPUs/GPUs, FPGAs are often susceptible to side-channel attacks, bitstream manipulation, and denial-of-service conditions caused by hostile reconfiguration or logic overuse. Ensuring secure deployment involves encryption of bitstreams, attestation mechanisms, and possibly runtime monitoring of logic behavior, all of which add overhead and require standardized support from FPGA vendors and tool providers [80]. In

 doi:10.20944/preprints202510.0454.v1

mission-critical applications such as autonomous driving, healthcare diagnostics, and aerospace, the lack of fault tolerance and error recovery mechanisms in FPGA-based accelerators presents a potential liability [81]. Introducing redundancy, error-correcting codes, and self-healing fabrics may provide avenues to address these concerns, albeit at a cost in area and power [82]. Looking forward, the co-evolution of machine learning models and FPGA architectures is likely to define the next era of research. New model paradigms such as spiking neural networks, neuromorphic computing, and transformer-inspired lightweight attention mechanisms will push the boundaries of what FPGA-based accelerators must support [83]. These models may offer novel sparsity patterns, temporal computation styles, or graph-based connectivity that diverge significantly from the grid-based convolutional and MLP patterns commonly optimized today [84]. Adapting FPGA architectures to efficiently support such models will likely require novel interconnect topologies, more granular control over compute sequencing, and the exploration of non-Von Neumann paradigms such as in-memory computing. At the same time, advances in chiplet-based design and heterogeneous integration may allow FPGAs to be embedded more closely with other accelerators and memory devices on a shared interposer, enabling high-bandwidth, low-latency communication across system components [85]. In summary, while FPGA-based neural network accelerators have made significant strides in recent years, particularly in embedded and low-latency inference scenarios, they face a constellation of challenges that span hardware design, software tooling, resource constraints, and security. Addressing these challenges will require coordinated efforts from the broader computing research community, FPGA vendors, compiler developers, and machine learning practitioners [86]. By pushing forward on these fronts, the field can unlock the full potential of FPGAs as reconfigurable, efficient, and versatile platforms for next-generation AI [87].

8. Conclusions

The landscape of deep learning acceleration is increasingly shaped by the dual imperatives of performance and adaptability, and within this evolving context, FPGA-based neural network accelerators have emerged as a compelling and versatile solution. Their inherent reconfigurability, energy efficiency, and suitability for custom precision and dataflow optimizations position FPGAs as a uniquely powerful platform at the intersection of software-defined flexibility and hardware-level specialization. This review has examined the architectural foundations, design strategies, comparative performance metrics, and development ecosystems surrounding FPGA-based accelerators, offering a panoramic view of their capabilities, limitations, and trajectory.

Central to the appeal of FPGAs is their capacity for fine-grained control over computation. Unlike GPUs, which enforce a rigid SIMT model, or ASICs, which are locked into pre-defined execution pipelines, FPGAs enable architects to create deeply pipelined, parallel, and task-specialized hardware units that are precisely tailored to the structure and requirements of individual neural networks. This ability to co-optimize data movement, arithmetic precision, memory hierarchy, and control logic leads to highly efficient accelerators that outperform general-purpose hardware on specific workloads, particularly in low-batch, latency-sensitive, and edge-deployed inference applications. The support for customized low-precision formats—ranging from INT8 down to binary representations—not only reduces the hardware footprint and memory bandwidth requirements but also enables higher degrees of parallelism, directly improving throughput without sacrificing accuracy in many cases.

Moreover, the reprogrammable nature of FPGAs introduces an unparalleled level of post-deployment adaptability. As machine learning models evolve rapidly, and as new architectural innovations such as transformer variants, attention-based mechanisms, and sparsity-aware layers become more prevalent, FPGA accelerators can be updated via bitstream reconfiguration to accommodate these changes without the need for new silicon. This characteristic makes FPGAs highly suitable for prototyping, iterative model deployment, and field upgrades in long-lifecycle systems such as autonomous robots, IoT devices, and embedded medical platforms. In contrast, ASIC-based

 doi:10.20944/preprints202510.0454.v1

accelerators, while extremely efficient, are fundamentally limited by their static design and long development cycles, rendering them inflexible to evolving standards or emerging model innovations.

That said, the review has also identified significant challenges that continue to hinder the widespread adoption of FPGA-based accelerators in commercial and research applications. Chief among these are the steep learning curve associated with hardware development, the complexity of current toolchains, and the lack of seamless integration with mainstream deep learning frameworks. Although recent progress in high-level synthesis (HLS), domain-specific compilers, and open-source initiatives such as FINN, hls4ml, and VTA has narrowed this gap, the overall hardware design process remains more fragmented and less accessible compared to the well-established GPU programming ecosystem. Furthermore, issues related to timing closure, resource fragmentation, and memory management demand considerable hardware design expertise, limiting the speed and scalability with which new accelerators can be brought to deployment.

From a systems perspective, limitations in bandwidth, on-chip memory capacity, and dynamic reconfigurability continue to constrain the types of workloads that FPGAs can efficiently support. Large-scale transformer models, multi-tenant inference services, and training workloads with highly variable memory access patterns remain better served by GPU or ASIC platforms—at least for now. However, as the FPGA ecosystem begins to incorporate technologies such as high-bandwidth memory (HBM), chiplet integration, and runtime reconfigurable fabrics, it is plausible that many of these barriers will be gradually mitigated. The rise of heterogeneous computing platforms that combine CPUs, GPUs, FPGAs, and domain-specific accelerators under a unified programming interface also points toward a future in which FPGAs function not in isolation but as dynamically deployable components in an intelligent orchestration framework.

In light of the detailed analysis presented in this review, it is evident that the future of FPGA-based neural network acceleration is both promising and complex. The key to unlocking their full potential lies not only in advancing FPGA hardware itself but also in rethinking the broader compiler toolchains, abstraction layers, and co-design methodologies that underpin their integration into the machine learning development lifecycle. There is a pressing need for more expressive programming models, automated optimization flows, and flexible deployment frameworks that allow software engineers to harness the benefits of FPGA acceleration without requiring deep hardware expertise. Additionally, stronger standardization, open tooling, and ecosystem collaboration will be essential to reduce fragmentation and foster innovation.

Ultimately, as AI systems continue to permeate domains as diverse as healthcare, finance, robotics, and scientific discovery, the demand for efficient, scalable, and adaptable inference hardware will only intensify. FPGA-based neural network accelerators—by virtue of their unique combination of performance, efficiency, and programmability—are well-positioned to play a critical role in meeting this demand. Whether as standalone inference engines, co-processors in heterogeneous computing stacks, or adaptable edge platforms in the Internet of Things, FPGAs offer a path toward deploying advanced neural networks in environments that demand more than what general-purpose processors can deliver. Realizing this vision, however, will require sustained research, cross-disciplinary collaboration, and continued evolution in both hardware and software—a challenge that, if met, could significantly reshape the future of intelligent computation.

References

1. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the CVPR09, 2009.
2. Lillicrap, T.P.; Santoro, A.; Marris, L.; Akerman, C.J.; Hinton, G. Backpropagation and the Brain. *Nature Reviews Neuroscience* **2020**, *21*, 335–346. https://doi.org/10.1038/s41583-020-0277-3.
3. Pritt, M.; Chern, G. Satellite image classification with deep learning. In Proceedings of the 2017 IEEE applied imagery pattern recognition workshop (AIPR). IEEE, 2017, pp. 1–7.
4. Hall, C.F. Pioneer 10 and Pioneer 11. *Science* **1975**, *188*, 445–446. https://doi.org/10.1126/science.188.4187.445.

 doi:10.20944/preprints202510.0454.v1

5. Liu, P.; Xing, K.; He, W.; Zhang, Z.; Deng, W. Research on On-Orbit SEU Characterization of the Signal Processing Platform. In Proceedings of the PROCEEDINGS OF THE 2016 INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND ENGINEERING APPLICATIONS; Davis, H.; Fang, ZG.; Ke, JF., Eds., 29 AVENUE LAVMIERE, PARIS, 75019, FRANCE, 2016; Vol. 63, *ACSR-Advances in Comptuer Science Research*, pp. 335–340.
6. Cheng, G.; Zhou, P.; Han, J. Learning Rotation-Invariant Convolutional Neural Networks for Object Detection in VHR Optical Remote Sensing Images. *IEEE Transactions on Geoscience and Remote Sensing* **2016**, *54*, 7405–7415. https://doi.org/10.1109/TGRS.2016.2601622.
7. Férésin, F.; Kervennic, E.; Bobichon, Y.; Lemaire, E.; Abderrahmane, N.; Bahl, G.; Grenet, I.; Moretti, M.; Benguigui, M. In Space Image Processing Using AI Embedded on System on Module: Example of OPS-SAT Cloud Segmentation, 2021. https://doi.org/10.5281/zenodo.5574960.
8. Andjelkovic, M.; Chen, J.; Simevski, A.; Stamenkovic, Z.; Krstic, M.; Kraemer, R. A Review of Particle Detectors for Space-Borne Self-Adaptive Fault-Tolerant Systems. In Proceedings of the 2020 IEEE East-West Design & Test Symposium (EWDTS), 2020, pp. 1–8. https://doi.org/10.1109/EWDTS50664.2020.9225138.
9. Li, Z.; Wang, L.; Dou, Q.; Tang, Y.; Guo, S.; Zhou, H.; Lu, W. ACCDSE: A Design Space Exploration Framework for Convolutional Neural Network Accelerator. In Proceedings of the COMPUTER ENGINEERING AND TECHNOLOGY, NCCET 2017; Xu, W.; Xiao, L.; Li, J.; Zhang, C.; Zhu, Z., Eds., HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2018; Vol. 600, *Communications in Computer and Information Science*, pp. 22–34. https://doi.org/10.1007/978-981-10-7844-6_3.
10. Chitra, K.; Vennila, C. RETRACTED: A Novel Patch Selection Technique in ANN B-Spline Bayesian Hyperprior Interpolation VLSI Architecture Using Fuzzy Logic for Highspeed Satellite Image Processing (Retracted Article). *JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING* **2021**, *12*, 6491–6504. https://doi.org/10.1007/s12652-020-02264-9.
11. Martins, L.A.; Viel, F.; Seman, L.O.; Bezerra, E.A.; Zeferino, C.A. A Real-Time SVM-based Hardware Accelerator for Hyperspectral Images Classification in FPGA. *MICROPROCESSORS AND MICROSYSTEMS* **2024**, *104*. https://doi.org/10.1016/j.micpro.2023.104998.
12. Blott, M.; Preußer, T.B.; Fraser, N.J.; Gambardella, G.; O'brien, K.; Umuroglu, Y.; Leeser, M.; Vissers, K. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Transactions on Reconfigurable Technology and Systems* **2018**, *11*, 16:1–16:23. https://doi.org/10.1145/3242897.
13. Zhao, Y.; Lv, Y.; Li, C. Hardware Acceleration of Satellite Remote Sensing Image Object Detection Based on Channel Pruning. *APPLIED SCIENCES-BASEL* **2023**, *13*. https://doi.org/10.3390/app131810111.
14. Navarro, J.E.; Samuelsson, A.; Gingsjö, H.; Barendt, J.; Dunne, A.; Buckley, L.; Reisis, D.; Kyriakos, A.; Papatheofanous, E.A.; Bezaitis, C.; et al. High-Performance Compute Board - A Fault-Tolerant Module for On-Boards Vision Processing, 2021. https://doi.org/10.5281/zenodo.5521624.
15. Heiner, J.; Sellers, B.; Wirthlin, M.; Kalb, J. FPGA Partial Reconfiguration via Configuration Scrubbing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, 2009, pp. 99–104. https://doi.org/10.1109/FPL.2009.5272543.
16. Kim, J.H.; Kim, Y.; Cho, D.H.; Kim, S.M. On-Orbit AI: Cloud Detection Technique for Resource-Limited Nanosatellite. *International Journal of Aeronautical and Space Sciences* **2024**. https://doi.org/10.1007/s42405-024-00865-8.
17. Lentaris, G.; Stamoulias, I.; Soudris, D.; Lourakis, M. HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars. *IEEE Transactions on Circuits and Systems for Video Technology* **2016**, *26*, 1563–1577. https://doi.org/10.1109/TCSVT.2015.2452781.
18. Liu, C.; Wang, C.; Luo, J. Large-Scale Deep Learning Framework on FPGA for Fingerprint-Based Indoor Localization. *IEEE ACCESS* **2020**, *8*, 65609–65617. https://doi.org/10.1109/ACCESS.2020.2985162.
19. Gomperts, A.; Ukil, A.; Zurfluh, F. Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS* **2011**, *7*, 78–89. https://doi.org/10.1109/TII.2010.2085006.
20. Mellempudi, N.; Kundu, A.; Mudigere, D.; Das, D.; Kaul, B.; Dubey, P. Ternary Neural Networks with Fine-Grained Quantization, 2017, [arXiv:cs/1705.01462]. https://doi.org/10.48550/arXiv.1705.01462.
21. Ying, Z.; Xuan, C.; Zhai, Y.; Sun, B.; Li, J.; Deng, W.; Mai, C.; Wang, F.; Labati, R.D.; Piuri, V.; et al. TAI-SARNET: Deep Transferred Atrous-Inception CNN for Small Samples SAR ATR. *Sensors* **2020**, *20*, 1724. https://doi.org/10.3390/s20061724.
22. Apicella, A.; Donnarumma, F.; Isgrò, F.; Prevete, R. A Survey on Modern Trainable Activation Functions. *Neural Networks* **2021**, *138*, 14–32. https://doi.org/10.1016/j.neunet.2021.01.026.

 doi:10.20944/preprints202510.0454.v1

23. Li, J.; Zhang, C.; Yang, W.; Li, H.; Wang, X.; Zhao, C.; Du, S.; Liu, Y. FPGA-Based Low-Bit and Lightweight Fast Light Field Depth Estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2025**, pp. 1–14. https://doi.org/10.1109/TVLSI.2024.3496751.
24. Coca, M.; Datcu, M. FPGA Accelerator for Meta-Recognition Anomaly Detection: Case of Burned Area Detection. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING* **2023**, *16*, 5247–5259. https://doi.org/10.1109/JSTARS.2023.3273309.
25. Liu, Y.; Zhu, M.; Wang, J.; Guo, X.; Yang, Y.; Wang, J. Multi-Scale Deep Neural Network Based on Dilated Convolution for Spacecraft Image Segmentation. *Sensors* **2022**, *22*, 4222. https://doi.org/10.3390/s22114222.
26. Davoli, F.; Kourogiorgas, C.; Marchese, M.; Panagopoulos, A.; Patrone, F. Small Satellites and CubeSats: Survey of Structures, Architectures, and Protocols. *International Journal of Satellite Communications and Networking* **2019**, *37*, 343–359. https://doi.org/10.1002/sat.1277.
27. Mazouz, A.; Bridges, C.P. Automated CNN Back-Propagation Pipeline Generation for FPGA Online Training. *Journal of Real-Time Image Processing* **2021**, *18*, 2583–2599. https://doi.org/10.1007/s11554-021-01147-2.
28. Cui, C.; Ge, F.; Li, Z.; Yue, X.; Zhou, F.; Wu, N. Design and Implementation of OpenCL-Based FPGA Accelerator for YOLOv2. In Proceedings of the 2021 IEEE 21st International Conference on Communication Technology (ICCT), 2021, pp. 1004–1007. https://doi.org/10.1109/ICCT52962.2021.9657856.
29. Abramovici, M.; Emmert, J.; Stroud, C. Roving STARs: An Integrated Approach to on-Line Testing, Diagnosis, and Fault Tolerance for FPGAs in Adaptive Computing Systems. In Proceedings of the Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001, 2001, pp. 73–92. https://doi.org/10.1109/EH.2001.937949.
30. Assim, A.; Reaz, M.B.I.; Ibrahimy, M.I.; Ismail, A.F.; Choong, F.; Mohd-Yasin, F. An AI Based Self-Moderated Smart-Home. *INFORMACIJE MIDEM-JOURNAL OF MICROELECTRONICS ELECTRONIC COMPONENTS AND MATERIALS* **2006**, *36*, 91–94.
31. Xiang-Zhi, Z.; Ai-Bing, Z.; Yi-Bing, G.; Chao, L.; Wen-Jing, W.; Zheng, T.; Ling-Gao, K.; Yue-Qiang, S. Research on retarding potential analyzer aboard China seismo-electromagnetic satellite. *ACTA PHYSICA SINICA* **2017**, *66*. https://doi.org/10.7498/aps.66.079401.
32. Isik, M.; Paul, A.; Varshika, M.L.; Das, A. A Design Methodology for Fault-Tolerant Computing Using Astrocyte Neural Networks. In Proceedings of the Proceedings of the 19th ACM International Conference on Computing Frontiers, New York, NY, USA, 2022; CF '22, pp. 169–172. https://doi.org/10.1145/3528416.3530232.
33. Furano, G.; Meoni, G.; Dunne, A.; Moloney, D.; Ferlet-Cavrois, V.; Tavoularis, A.; Byrne, J.; Buckley, L.; Psarakis, M.; Voss, K.O.; et al. Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities. *IEEE Aerospace and Electronic Systems Magazine* **2020**, *35*, 44–56. https://doi.org/10.1109/MAES.2020.3008468.
34. Zniyed, Y.; Nguyen, T.P.; et al. Enhanced network compression through tensor decompositions and pruning. *IEEE Transactions on Neural Networks and Learning Systems* **2024**, *36*, 4358–4370.
35. Jallad, A.H.M.; Mohammed, L.B. Hardware Support Vector Machine (SVM) for Satellite On-Board Applications. In Proceedings of the 2014 NASA/ESA CONFERENCE ON ADAPTIVE HARDWARE AND SYSTEMS (AHS), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014; NASA/ESA Conference on Adaptive Hardware and Systems, pp. 256–261.
36. Racca, G.D.; Laureijs, R.; Stagnaro, L.; Salvignol, J.C.; Alvarez, J.L.; Criado, G.S.; Venancio, L.G.; Short, A.; Strada, P.; Boenke, T.; et al. The Euclid Mission Design. 2016, p. 99040O, [arXiv:astro-ph/1610.05508]. https://doi.org/10.1117/12.2230762.
37. Kerns, S.; Shafer, B.; Rockett, L.; Pridmore, J.; Berndt, D.; van Vonno, N.; Barber, F. The Design of Radiation-Hardened ICs for Space: A Compendium of Approaches. *Proceedings of the IEEE* **1988**, *76*, 1470–1509. https://doi.org/10.1109/5.90115.
38. Ngo, D.; Harris, M. A Reliable Infrastructure Based on COTS Technology for Affordable Space Application. In Proceedings of the 2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542), 2001, Vol. 5, pp. 2435–2441 vol.5. https://doi.org/10.1109/AERO.2001.931203.
39. Cheng, G.; Han, J. A Survey on Object Detection in Optical Remote Sensing Images. *ISPRS Journal of Photogrammetry and Remote Sensing* **2016**, *117*, 11–28. https://doi.org/10.1016/j.isprsjprs.2016.03.014.
40. Kumar, N.; Prakash, C.; Satashia, S.N.; Kumar, V.; Parikh, K.S. Efficient Implementation of Low Density Parity Check Codes for Satellite Ground Terminals. In Proceedings of the 2014 INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014; pp. 689–695.

 doi:10.20944/preprints202510.0454.v1

41. Murphy, J.; Ward, J.E.; Namee, B.M. Machine Learning in Space: A Review of Machine Learning Algorithms and Hardware for Space Applications⋆ **2021**.
42. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and ¡0.5MB Model Size, 2016, [arXiv:cs.CV/1602.07360].
43. Spiller, D.; Carbone, A.; Latorre, F.; Curti, F. Hardware-in-the-Loop Simulations of Remote Sensing Disaster Monitoring Systems with Real-Time on-Board Computation. In Proceedings of the 2022 IEEE INTERNATIONAL CONFERENCE ON METROLOGY FOR EXTENDED REALITY, ARTIFICIAL INTELLIGENCE AND NEURAL ENGINEERING (METROXRAINE), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2022; pp. 731–736. https://doi.org/10.1109/MetroXRAINE54828.2022.9967688.
44. Franconi, N.; Cook, T.; Wilson, C.; George, A.D. Comparison of Multi-Phase Power Converters and Power Delivery Networks for Next-Generation Space Architectures. In Proceedings of the 2023 IEEE AEROSPACE CONFERENCE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2023; IEEE Aerospace Conference Proceedings. https://doi.org/10.1109/AERO55745.2023.10115579.
45. Przewlocka, D.; Wasala, M.; Szolc, H.; Blachut, K.; Kryjak, T. Optimisation of a Siamese Neural Network for Real-Time Energy Efficient Object Tracking; 2020; Vol. 12334, pp. 151–163, [arXiv:cs, eess/2007.00491]. https://doi.org/10.1007/978-3-030-59006-2_14.
46. Yanguas-Gil, A.; Mane, A.; Elam, J.W.; Wang, F.; Severa, W.; Daram, A.R.; Kudithipudi, D. The Insect Brain as a Model System for Low Power Electronics and Edge Processing Applications. In Proceedings of the 2019 IEEE SPACE COMPUTING CONFERENCE (SCC), 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2019; pp. 60–66. https://doi.org/10.1109/SpaceComp.2019.00012.
47. Rojas, R. The Backpropagation Algorithm. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 149–182. https://doi.org/10.1007/978-3-642-61068-4_7.
48. Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Shi, L. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience* **2018**, *12*. https://doi.org/10.3389/fnins.2018.00331.
49. Mazouz, A.E.; Nguyen, V.T. Online Continual Streaming Learning for Embedded Space Applications. *JOURNAL OF REAL-TIME IMAGE PROCESSING* **2024**, *21*. https://doi.org/10.1007/s11554-024-01438-4.
50. Potsdam, ISPRS. 2d Semantic Labeling Dataset. *Accessed: Apr* **2018**.
51. Zhang, L.; Zhang, L.; Du, B. Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art. *IEEE Geoscience and Remote Sensing Magazine* **2016**, *4*, 22–40. https://doi.org/10.1109/MGRS.2016.2540798.
52. Alom, M.Z.; Taha, T.M.; Yakopcic, C.; Westberg, S.; Sidike, P.; Nasrin, M.S.; Van Esesn, B.C.; Awwal, A.A.S.; Asari, V.K. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches, 2018, [arXiv:cs/1803.01164].
53. Koren.; Su. Reliability Analysis of N-Modular Redundancy Systems with Intermittent and Permanent Faults. *IEEE Transactions on Computers* **1979**, *C-28*, 514–520. https://doi.org/10.1109/TC.1979.1675397.
54. Reed, I.; Yu, X. Adaptive Multiple-Band CFAR Detection of an Optical Pattern with Unknown Spectral Distribution. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **1990**, *38*, 1760–1770. https://doi.org/10.1109/29.60107.
55. Edmonds, L.; Barnes, C.; Scheick, L.; Aeronautics, U.S.N.; Administration, S.; Laboratory (U.S.), J.P. *An Introduction to Space Radiation Effects on Microelectronics*; JPL Publication, Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2000.
56. Li, Y.; Yao, B.; Peng, Y. FPGA-based Large-scale Remote Sensing Image ROI Extraction for On-orbit Ship Detection. In Proceedings of the 2022 IEEE INTERNATIONAL INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE (I2MTC 2022), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2022; IEEE Instrumentation and Measurement Technology Conference. https://doi.org/10.1109/I2MTC48687.2022.9806614.
57. Mellier, Y.; Racca, G.; Laureijs, R. Unveiling the Dark Universe with the Euclid Space Mission. In Proceedings of the 42nd COSPAR Scientific Assembly, 2018, Vol. 42, pp. E1.16–2–18.
58. Li, B.; Peng, X.; Wang, Z.; Xu, J.; Feng, D. AOD-Net: All-in-One Dehazing Network. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 4780–4788. https://doi.org/10.1109/ICCV.2017.511.
59. JannisWolf. JannisWolf/Fpga_bnn_accelerator, 2021.
60. Martins, L.A.; Sborz, G.A.M.; Viel, F.; Zeferino, C.A. An SVM-based Hardware Accelerator for Onboard Classification of Hyperspectral Images. In Proceedings of the Proceedings of the 32nd Symposium on

 doi:10.20944/preprints202510.0454.v1

Integrated Circuits and Systems Design, New York, NY, USA, 2019; SBCCI '19, pp. 1–6. https://doi.org/10.1145/3338852.3339869.

61. Kim, W.J.; Youn, C.H. Cooperative Scheduling Schemes for Explainable DNN Acceleration in Satellite Image Analysis and Retraining. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS* **2022**, *33*, 1605–1618. https://doi.org/10.1109/TPDS.2021.3122454.
62. Di Mascio, S.; Menicucci, A.; Gill, E.; Monteleone, C. Extending the NOEL-V Platform with a RISC-V Vector Processor for Space Applications. *JOURNAL OF AEROSPACE INFORMATION SYSTEMS* **2023**. https://doi.org/10.2514/1.I011097.
63. Ekblad, A.; Mahendrakar, T.; White, R.; Wilde, M.; Silver, I.; Wheeler, B. Resource-Constrained FPGA Design for Satellite Component Feature Extraction. In Proceedings of the 2023 IEEE AEROSPACE CONFERENCE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2023; IEEE Aerospace Conference Proceedings. https://doi.org/10.1109/AERO55745.2023.10115681.
64. Daram, A.R.; Kudithipudi, D.; Yanguas-Gil, A. Task-Based Neuromodulation Architecture for Lifelong Learning. In Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED), 2019, pp. 191–197. https://doi.org/10.1109/ISQED.2019.8697362.
65. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proceedings of the Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015; Navab, N.; Hornegger, J.; Wells, W.M.; Frangi, A.F., Eds., Cham, 2015; pp. 234–241. https://doi.org/10.1007/978-3-319-24574-4_28.
66. Cheng, G.; Han, J.; Lu, X. Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proceedings of the IEEE* **2017**, *105*, 1865–1883. https://doi.org/10.1109/JPROC.2017.2675998.
67. El-Darymli, K.; McGuire, P.; Power, D.; Moloney, C.R. Target Detection in Synthetic Aperture Radar Imagery: A State-of-the-Art Survey. *Journal of Applied Remote Sensing* **2013**, *7*, 071598. https://doi.org/10.1117/1.JRS.7.071598.
68. Yan, S.; Liu, Z.; Wang, Y.; Zeng, C.; Liu, Q.; Cheng, B.; Cheung, R.C.C. An FPGA-based MobileNet Accelerator Considering Network Structure Characteristics. In Proceedings of the 2021 31ST INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS (FPL 2021), 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2021; International Conference on Field Programmable Logic and Applications, pp. 17–23.
69. Perrotin, M.; Conquet, E.; Delange, J.; Tsiodras, T. TASTE: An Open-Source Tool-Chain for Embedded System and Software Development. In Proceedings of the Embedded Real Time Software and Systems (ERTS2012), Toulouse, France, 2012.
70. Sharma, A.; Unnikrishnan, E.; Ravichandran, V.; Valarmathi, N. Development of CCSDS Proximity-1 Protocol for ISRO's Extraterrestrial Missions. In Proceedings of the 2014 INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014; pp. 2813–2819.
71. Pappalardo, A. Xilinx/Brevitas. Zenodo, 2023.
72. Carmeli, G.; Ben-Moshe, B. AI-Based Real-Time Star Tracker. *ELECTRONICS* **2023**, *12*. https://doi.org/10.3390/electronics12092084.
73. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020, [arXiv:cs.CV/2004.10934].
74. Luo, Q.; Cheng, X.; Zhou, Z. FPGA Design and Implementation of Carrier Synchronization for DVB-S2 Demodulators. In Proceedings of the ASICON 2007: 2007 7TH INTERNATIONAL CONFERENCE ON ASIC, VOLS 1 AND 2, PROCEEDINGS; Tang, TA.; Li, W., Eds., 345 E 47TH ST, NEW YORK, NY 10017 USA, 2007; pp. 846–849. https://doi.org/10.1109/ICASIC.2007.4415763.
75. Gough, MP.; Buckley, AM.; Carozzi, T.; Beloff, N. Experimental Studies of Wave-Particle Interactions in Space Using Particle Correlators: Results and Future Developments. In Proceedings of the FUTURE TRENDS AND NEEDS IN SCIENCE AND ENGINEERING FOR PLASMA PHYSICS IN SPACE; Lui, ATY.; Treumann, RA., Eds., THE BOULEVARD, LANGFORD LANE,, KIDLINGTON OX5 1GB, OXFORD, ENGLAND, 2003; Vol. 32, *ADVANCES IN SPACE RESEARCH-SERIES*, pp. 407–416. https://doi.org/10.1016/S0273-1177(03)90281-X.
76. Ortiz, F.; Lagunas, E.; Martins, W.; Dinh, T.; Skatchkovsky, N.; Simeone, O.; Rajendran, B.; Navarro, T.; Chatzinotas, S. Towards the Application of Neuromorphic Computing to Satellite Communications. In Proceedings of the 39th International Communications Satellite Systems Conference (ICSSC 2022), 2022, Vol. 2022, pp. 91–97. https://doi.org/10.1049/icp.2023.1367.

 doi:10.20944/preprints202510.0454.v1

77. Stivaktakis, R.; Tsagkatakis, G.; Moraes, B.; Abdalla, F.; Starck, J.L.; Tsakalides, P. Convolutional Neural Networks for Spectroscopic Redshift Estimation on Euclid Data. *IEEE Transactions on Big Data* **2020**, *6*, 460–476. https://doi.org/10.1109/TBDATA.2019.2934475.
78. Rojas, R. Fast Learning Algorithms. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 183–225. https://doi.org/10.1007/978-3-642-61068-4_8.
79. Karakizi, C.; Karantzalos, K.; Vakalopoulou, M.; Antoniou, G. Detailed Land Cover Mapping from Multitemporal Landsat-8 Data of Different Cloud Cover. *Remote Sensing* **2018**, *10*, 1214. https://doi.org/10.3390/rs10081214.
80. Johnson, B. Crowdsourced Mapping, 2016. https://doi.org/10.24432/C56315.
81. Gankidi, P.R.; Thangavelautham, J. FPGA Architecture for Deep Learning and Its Application to Planetary Robotics. In Proceedings of the 2017 IEEE AEROSPACE CONFERENCE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2017; IEEE Aerospace Conference Proceedings.
82. Munshi, A. The OpenCL Specification. In Proceedings of the 2009 IEEE Hot Chips 21 Symposium (HCS), 2009, pp. 1–314. https://doi.org/10.1109/HOTCHIPS.2009.7478342.
83. Kraft, M.; Walas, K.; Ptak, B.; Bidzinski, M.; Stezala, K.; Pieczynski, D. INTEGRATION OF HETEROGENEOUS COMPUTATIONAL PLATFORM-BASED, AI-CAPABLE PLANETARY ROVER USING ROS 2. In Proceedings of the IGARSS 2023 - 2023 IEEE INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2023; IEEE International Symposium on Geoscience and Remote Sensing IGARSS, pp. 2014–2017. https://doi.org/10.1109/IGARSS52108.2023.10281823.
84. Liu, Y.; Shen, Y.; Sun, Z.w.; Xing, L. Task Scheduling Algorithm of FPGA for On-Board Reconfigurable Coprocessor. In Proceedings of the 2013 INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE (ICCSAI 2013), 439 DUKE STREET, LANCASTER, PA 17602-4967 USA, 2013; pp. 207–214.
85. Pitsis, G.; Tsagkatakis, G.; Kozanitis, C.; Kalomoiris, I.; Ioannou, A.; Dollas, A.; Katevenis, M.G.H.; Tsakalides, P. Efficient Convolutional Neural Network Weight Compression for Space Data Classification on Multi-fpga Platforms. In Proceedings of the 2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2019; International Conference on Acoustics Speech and Signal Processing ICASSP, pp. 3917–3921.
86. Yang, C.; Ming, L. Far Field Demonstration Experiment of Fine Tracking System for Satellite-to-Ground Optical Communication. *CHINA COMMUNICATIONS* **2010**, *7*, 139–145.
87. Estlin, T.A.; Bornstein, B.J.; Gaines, D.M.; Anderson, R.C.; Thompson, D.R.; Burl, M.; Castaño, R.; Judd, M. AEGIS Automated Science Targeting for the MER Opportunity Rover. *ACM Trans. Intell. Syst. Technol.* **2012**, *3*, 50:1–50:19. https://doi.org/10.1145/2168752.2168764.