# Reconfigurable Acceleration of Deep Learning Workloads with FPGA-Based Architectures in Edge and Embedded Systems

Lucas Oliveira[1,2], Camila Ferreira[3], Thiago Souza[2,4], Gulnaz Rati[1], and Mariana Costa[3,5]

[1] University of São Paulo, Brazil
gulnaz.zati@usp.br
[2] Federal University of Minas Gerais, Brazil
lucas.oliveira@ufmg.br
[3] Federal University of Pernambuco, Brazil
camila.ferreira@ufpe.br
[4] Federal University of Santa Catarina, Brazil
thiago.souza@ufsc.br
[5] State University of Campinas, Brazil
mariana.costa@unicamp.br

**Abstract.** As the proliferation of edge computing reshapes the landscape of artificial intelligence deployment, the demand for efficient, low-latency, and energy-conscious deep learning inference has never been more acute. Traditional acceleration platforms—such as CPUs, GPUs, and ASICs—face increasing challenges in meeting the unique constraints of edge environments, including limited power budgets, thermal envelopes, physical space, and the need for adaptive, context-aware operation. In this context, Field-Programmable Gate Arrays (FPGAs) have emerged as a compelling solution, offering a uniquely reconfigurable architecture that enables tailored hardware acceleration for diverse deep learning workloads. This survey provides a comprehensive and technically rigorous exploration of the design, deployment, and evaluation of scalable deep learning accelerators implemented on FPGAs, with a specific focus on edge and embedded intelligence.

We begin by categorizing FPGA-based architectures into distinct paradigms, including streaming dataflow, systolic arrays, overlay-based accelerators, coarse-grained reconfigurable architectures, and hybrid heterogeneous systems, highlighting their respective strengths, limitations, and suitability for different inference scenarios. The discussion proceeds to analyze the critical design methodologies used to transform high-level neural network descriptions into efficient, synthesizable hardware representations. These include high-level synthesis (HLS), tiling strategies, loop unrolling, memory scheduling, and automated compilation flows that align with the unique spatial and temporal characteristics of FPGA fabrics. Special emphasis is placed on the role of quantization, pruning, and precision scaling in reducing hardware complexity while preserving model accuracy.

We further examine runtime reconfiguration and adaptive execution techniques, where FPGA fabrics dynamically reshape their logic and compute

paths in response to changing workloads, power conditions, or functional requirements. This section delves into partial reconfiguration, adaptive precision, workload-aware scheduling, and emerging methods of using machine learning itself to optimize the reconfigurability of hardware at deployment time. Benchmarking methodologies are also discussed in detail, with a focus on multi-dimensional evaluation metrics—such as throughput, energy per inference, latency, model fidelity, and reconfiguration overhead—under realistic operational constraints. A diverse set of real-world case studies is presented, encompassing applications from autonomous drones and wearable health monitors to industrial vision systems and quantized AI inference engines, showcasing the breadth of FPGA applicability and the concrete engineering challenges involved.
Finally, the review synthesizes these findings to outline future research directions and systemic challenges, including the need for end-to-end toolchains, intelligent hardware-software co-design, support for dynamic and sparse models such as transformers, and the integration of FPGA fabrics into chiplet-based and 3D-stacked architectures. We argue that the evolution of FPGAs from peripheral accelerators to central elements of embedded AI systems marks a significant shift in how intelligence is architected and deployed at the edge. Through detailed technical analysis, practical case exploration, and a forward-looking perspective, this work provides a foundational reference for researchers, system architects, and developers aiming to harness the full potential of FPGAs for scalable, efficient, and adaptive deep learning at the intelligent edge.

**Keywords:** FPGA acceleration, deep learning inference, edge computing, embedded intelligence, hardware-aware neural networks, reconfigurable architectures, quantization, high-level synthesis, runtime adaptability, scalable AI systems, low-power AI, custom hardware design

## 1 Introduction

The proliferation of intelligent devices and the exponential growth in the deployment of artificial intelligence (AI) at the network edge have significantly transformed the computational landscape across industries [1]. Edge and embedded systems, which traditionally operated under stringent constraints in terms of power, latency, and thermal dissipation, are now expected to execute increasingly complex deep learning (DL) workloads in real-time. These workloads, which include tasks such as real-time object detection, speech recognition, anomaly detection, and predictive maintenance, are computationally intensive and memory-demanding. Conventional general-purpose processors (GPPs) and even specialized graphic processing units (GPUs) have exhibited limitations in meeting the tight energy-efficiency, latency, and form-factor requirements posed by edge environments [2]. As such, there has been a marked shift towards alternative compute paradigms that offer architectural flexibility, domain-specific customization, and improved performance-per-watt characteristics—among which

field-programmable gate arrays (FPGAs) have emerged as highly promising enablers. FPGAs offer a compelling solution to the unique demands of edge AI due to their inherent reconfigurability, parallelism, and adaptability [3]. Unlike fixed-function ASICs, FPGAs can be reprogrammed to tailor their architecture to specific workloads, enabling efficient mapping of deep neural networks (DNNs) with customized dataflows, quantization schemes, memory hierarchies, and compute pipelines. This reconfigurability is particularly advantageous in scenarios where model parameters, topologies, and deployment requirements evolve over time, or when system designers aim to deploy multiple heterogeneous models on the same hardware platform [4]. Furthermore, FPGAs can achieve significant throughput and latency benefits by exploiting fine-grained parallelism in DNN inference, including layer-wise and intra-layer parallelization, pipelining, and customized precision arithmetic. The intersection of scalable deep learning and FPGA-based acceleration introduces a multitude of technical challenges and opportunities. On one hand, the rapid growth in model size and complexity—exemplified by the rise of transformer-based architectures, graph neural networks, and large convolutional backbones—necessitates scalable hardware-software co-design methodologies that can distribute computation across multiple FPGA resources or operate within constrained on-chip memory. On the other hand, edge AI demands aggressive optimization of resource utilization, power consumption, and thermal profile, often in environments devoid of active cooling or power provisioning [5]. This has prompted extensive research into neural architecture compression techniques (e.g., pruning, quantization, knowledge distillation), memory bandwidth-aware designs, and novel interconnect topologies that enable efficient model execution at the edge [6]. This review aims to systematically explore the landscape of scalable deep learning architectures implemented through custom FPGA-based accelerators, with a particular focus on edge and embedded intelligence applications. We delve into the hardware design paradigms that have emerged to support state-of-the-art DNNs on FPGAs, including dataflow architectures, systolic arrays, domain-specific instruction sets, and overlay-based frameworks. Additionally, we investigate the co-design methodologies that jointly optimize neural network models and FPGA implementation to meet specific performance and efficiency targets [7]. Special attention is paid to the trade-offs between flexibility, performance, and scalability, as well as the integration of runtime reconfiguration and partial reprogramming to support dynamic workloads. Moreover, we contextualize the evolution of FPGA-based deep learning accelerators within the broader ecosystem of edge computing, emphasizing how advances in toolchains, high-level synthesis (HLS), model compression techniques, and open-source frameworks have democratized the development of FPGA solutions for AI developers. We also highlight emerging trends such as heterogeneous computing with FPGAs and CPUs/GPUs, the use of chiplet-based FPGA systems, and the role of domain-specific compilers and intermediate representations in automating the translation of DL models into efficient hardware. By surveying a diverse body of literature and recent innovations, this work seeks to provide both a historical perspective and a forward-

looking analysis of how custom FPGA-based acceleration is shaping the future of scalable edge intelligence [8]. In essence, this review offers a comprehensive synthesis of the challenges, innovations, and methodologies at the intersection of scalable DL architectures and FPGA-based acceleration. It serves as a foundational reference for researchers, practitioners, and system architects striving to harness the full potential of custom FPGA solutions for intelligent edge systems, and outlines promising directions for future work in designing resilient, efficient, and scalable AI systems under real-world deployment constraints [9].

## 2 Background and Theoretical Foundations

To rigorously explore the scalability of deep learning architectures via custom FPGA-based acceleration in edge and embedded systems, it is essential to establish a theoretical foundation that encompasses the core computational structures of deep neural networks (DNNs), the principles of hardware mapping, and the optimization of resource-constrained inference. This section formalizes the key mathematical underpinnings of deep learning models, discusses their computational complexity, and introduces the mapping of these models onto FPGA fabrics using hardware-aware performance models [10].

### 2.1 Deep Neural Network Computational Model

A typical deep neural network can be described as a composition of parameterized transformations $f(\mathbf{x};\boldsymbol{\theta})$, where $\mathbf{x} \in \mathbb{R}^n$ is the input vector and $\boldsymbol{\theta} \in \mathbb{R}^p$ denotes the set of trainable parameters. For a network with $L$ layers, the function can be expressed as:

$$f(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\dots f_1(\mathbf{x})))$$

Each layer $f_\ell$, $\ell \in \{1, 2, \dots, L\}$, performs a linear operation followed by a non-linear activation, typically of the form:

$$f_\ell(\mathbf{h}_{\ell-1}) = \sigma_\ell(\mathbf{W}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell)$$

where $\mathbf{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix, $\mathbf{b}_\ell \in \mathbb{R}^{d_\ell}$ is the bias vector, and $\sigma_\ell(\cdot)$ denotes a non-linear activation function such as ReLU, GELU, or Sigmoid. The computational cost per layer, in terms of multiply-accumulate (MAC) operations, is:

$$\text{MACs}_\ell = d_\ell \cdot d_{\ell-1}$$

The total computational complexity $\mathcal{C}(f)$ of the network is then:

$$\mathcal{C}(f) = \sum_{\ell=1}^{L} d_\ell \cdot d_{\ell-1}$$

For convolutional neural networks (CNNs), this extends to:

$$\text{MACs}_\ell^{\text{conv}} = H_\ell \cdot W_\ell \cdot C_\ell \cdot K_\ell^2 \cdot C_{\ell-1}$$

where $H_\ell$ and $W_\ell$ are the output feature map height and width, $K_\ell$ is the kernel size, and $C_{\ell-1}, C_\ell$ are the input and output channels, respectively.

### 2.2 FPGA Resource Modeling and Mapping

The mapping of DNNs to FPGAs must consider limited computational resources, such as the number of available digital signal processing (DSP) slices $R_{\text{DSP}}$, block RAM (BRAM) $R_{\text{BRAM}}$, flip-flops $R_{\text{FF}}$, and lookup tables $R_{\text{LUT}}$. For an FPGA device $\mathcal{F}$, we define its resource vector as:

$$\mathbf{R}_\mathcal{F} = [R_{\text{DSP}}, R_{\text{BRAM}}, R_{\text{FF}}, R_{\text{LUT}}]$$

Let $\rho_\ell \in \mathbb{R}^4$ denote the resource footprint of layer $\ell$. The resource feasibility constraint requires:

$$\sum_{\ell=1}^{L} \rho_\ell \leq \mathbf{R}_\mathcal{F}$$

Hardware designers often implement layers using custom processing elements (PEs) with a dataflow model. If each PE processes a tile of size $T$, then latency per layer can be estimated by:

$$\mathcal{L}_\ell = \frac{\text{MACs}_\ell}{P \cdot f}$$

where $P$ is the number of PEs and $f$ is the FPGA operating frequency (in Hz) [11]. When pipelining is applied with an initiation interval $II$ and pipeline depth $D$, latency becomes:

$$\mathcal{L}_\ell^{\text{pipe}} = II \cdot N_\ell + D$$

where $N_\ell$ is the number of operations for the layer. Optimizing $\mathcal{L}_\ell^{\text{pipe}}$ under hardware constraints is a central objective in accelerator design.

### 2.3 Quantization and Memory Optimization

To meet strict energy and memory budgets in embedded environments, DNNs are quantized to lower-precision representations [12]. Suppose the original 32-bit floating-point weights $\mathbf{W}_\ell$ are quantized to $b$-bit fixed-point numbers:

$$\tilde{\mathbf{W}}_\ell = Q_b(\mathbf{W}_\ell) \quad \text{with} \quad Q_b(x) = \text{round}(x \cdot 2^{b-1}) \cdot 2^{-(b-1)}$$

Quantization reduces memory bandwidth and improves arithmetic intensity, allowing multiple weight elements to be stored per BRAM word [13]. The total memory required for a quantized model $\tilde{f}$ is:

$$\mathcal{M}(\tilde{f}) = \sum_{\ell=1}^{L} (d_\ell \cdot d_{\ell-1} \cdot b)$$

To optimize model deployment, we solve the following constrained minimization problem:

$$\min_{\tilde{f}} \quad \mathcal{L}(\tilde{f}(\mathbf{x}), \mathbf{y}) \quad \text{s.t.} \quad \mathcal{C}(\tilde{f}) \leq \mathcal{C}_{\max}, \quad \mathcal{M}(\tilde{f}) \leq \mathcal{M}_{\max}$$

where $\mathcal{L}$ denotes a loss function and $\mathcal{C}_{\max}, \mathcal{M}_{\max}$ are compute and memory budgets, respectively.

### 2.4 Scalability Metrics and Performance Boundaries

Scalability in the context of FPGA-based DNN acceleration refers to the ability to maintain or improve throughput, latency, and energy efficiency as network depth, input dimensionality, or dataset complexity increases. Let us define:

- Throughput $\mathcal{T} = \frac{N}{\sum_{\ell=1}^{L} \mathcal{L}_\ell}$, where $N$ is the number of inferences per second.
- Energy per inference $\mathcal{E} = \frac{P_{\text{avg}}}{\mathcal{T}}$, where $P_{\text{avg}}$ is the average power consumption [14].
- Resource efficiency $\eta = \frac{\mathcal{T}}{\sum_{\ell=1}^{L} \rho_\ell \cdot \mathbf{c}}$, where $\mathbf{c}$ is a cost vector for each resource type.

The challenge in scaling arises when $\mathcal{C}(f) \to \infty$ but $\mathbf{R}_{\mathcal{F}}$ remains fixed or scales sub-linearly [15]. Therefore, partitioning techniques, tiling strategies, model sparsification (where many weights $w_{i,j} \approx 0$), and activation gating are introduced to reduce compute density:

$$\text{Sparsity}_\ell = \frac{\|\mathbf{W}_\ell\|_0}{d_\ell \cdot d_{\ell-1}} \ll 1$$

Exploiting structured sparsity allows zero-skipping hardware primitives to be synthesized, further improving runtime efficiency [16].

### 2.5 Hardware-Software Co-Design Paradigm

Let $\mathcal{H}(\boldsymbol{\theta}, \mathbf{R})$ be the hardware realization of a model parameter set $\boldsymbol{\theta}$ under resource vector $\mathbf{R}$ [17]. Co-design then becomes a bi-level optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \mathcal{L}(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}) \\ \text{s.t.} \quad & \mathcal{H}(\boldsymbol{\theta}, \mathbf{R}) \in \mathcal{F}_{\text{feasible}} \end{aligned}$$

Here, $\mathcal{F}_{\text{feasible}}$ denotes the space of FPGA implementations realizable within timing, power, and area constraints. Recent advances in differentiable hardware mapping, reinforcement learning-based placement, and neural architecture search (NAS) on FPGAs attempt to approximate this co-design space in

a tractable fashion. In summary, the theoretical foundations underlying scalable deep learning acceleration on FPGAs involve a multifaceted integration of mathematical models of computation, hardware resource constraints, optimization strategies, and trade-off analysis. The interplay between these elements forms the basis upon which the next-generation edge intelligence systems can be designed, evaluated, and deployed [18].

## 3 Taxonomy of FPGA-Based Architectures for Deep Learning at the Edge

As deep learning continues to permeate edge computing scenarios—ranging from smart cameras and industrial IoT to autonomous vehicles and wearable health monitors—the demand for highly efficient and adaptable hardware accelerators becomes increasingly urgent. FPGAs, with their innate capacity for architectural customization and parallel execution, have catalyzed a proliferation of design methodologies tailored specifically for edge-based inference [19]. Over the past decade, various classes of FPGA-based architectures have emerged, each characterized by distinct trade-offs in flexibility, performance, power efficiency, and ease of deployment. To systematize this diversity, we propose a comprehensive taxonomy that categorizes these architectures into five major paradigms: (1) streaming-based dataflow architectures, (2) systolic array-based designs, (3) reconfigurable overlay-based accelerators, (4) coarse-grained reconfigurable architectures (CGRAs) on FPGA fabrics, and (5) hybrid heterogeneous systems integrating FPGAs with CPUs or GPUs. Each category encapsulates unique approaches to data movement, computation scheduling, memory reuse, and reconfigurability, and their adoption is often influenced by target application domains and deployment constraints. Streaming-based dataflow architectures represent one of the most prevalent and natural mappings of DNN workloads onto FPGAs. These designs aim to exploit spatial and temporal locality by establishing pipelined pathways through which data and weights are streamed in a controlled and predictable fashion [20]. Each processing element (PE) in a dataflow architecture is statically assigned to a specific computation stage, such as a convolution or activation unit, allowing for high-throughput processing with deterministic latency. These designs are particularly effective for fixed topologies and batch-size-insensitive execution, making them ideal for always-on inference in edge settings [21]. However, they often suffer from inflexibility when accommodating varying model structures or dynamic execution patterns [22]. On the other end of the spectrum are overlay-based designs, which abstract low-level FPGA reconfiguration details through programmable data paths and virtual instruction sets. These overlays enable rapid compilation, code reuse, and simplified model deployment, but typically incur an overhead in resource consumption and latency compared to fully customized RTL implementations. Systolic arrays, originally designed for matrix operations, have been reintroduced in modern FPGA accelerators due to their ability to harness fine-grained data reuse and controlled memory bandwidth [23]. Implemented as a grid of

tightly coupled PEs, systolic arrays execute multiply-accumulate operations in a rhythmic, clock-driven manner, minimizing external memory accesses and improving energy efficiency [24]. While they achieve remarkable efficiency for regular and dense matrix computations, their scalability is limited by the routing complexity and FPGA fabric interconnect bottlenecks, particularly when targeting large models under tight resource budgets. Coarse-grained reconfigurable architectures (CGRAs), when implemented atop or within FPGA fabrics, introduce an intermediate abstraction between fine-grained LUT-based designs and fully programmable processors [25]. CGRAs enable block-level reconfiguration of computing elements and interconnects, providing both the adaptability required for multiple DNN models and the performance efficiency of spatial parallelism. These architectures are increasingly explored for applications requiring frequent model updates or partial reconfiguration capabilities [26]. Finally, hybrid heterogeneous systems—comprising FPGAs in tandem with ARM cores, GPUs, or specialized NPUs—leverage workload partitioning to balance general-purpose flexibility and specialized acceleration [27]. These systems allow offloading of control-intensive or sparse operations to CPUs, while reserving dense linear algebra workloads for FPGA execution [28]. The orchestration of such systems requires sophisticated runtime schedulers and memory coherency protocols, but offers a compelling solution for multitask edge AI systems that must simultaneously handle vision, language, and control pipelines. Table 1 summarizes the primary characteristics, advantages, and limitations of these architecture classes [29]. It provides a high-level comparative analysis that can serve as a design guideline for system architects selecting appropriate acceleration strategies for diverse edge AI applications [30].

This architectural taxonomy highlights the multifaceted design space faced by practitioners in edge AI acceleration. While high-throughput streaming designs may be preferable in low-latency sensor applications, overlay-based approaches may be more appropriate for development environments where rapid prototyping, frequent model updates, and shorter time-to-market are crucial. Furthermore, the ability to mix architectural paradigms within a single FPGA or SoC platform opens up exciting opportunities for workload specialization, dynamic reconfiguration, and context-aware computing at the edge. The next generation of edge intelligence systems will likely leverage hybrid strategies that fluidly combine these architectural principles, guided by domain-specific optimizations, evolving ML model characteristics, and rapidly advancing FPGA toolchains.

## 4 Design Methodologies for Scalable FPGA-Based Deep Learning Acceleration

The effectiveness of deploying deep learning models on FPGA platforms, especially in edge and embedded environments, is intimately tied to the methodologies by which these models are translated into hardware [31]. The process spans from high-level model representation—typically authored in deep learning frameworks such as PyTorch or TensorFlow—to low-level register-transfer level (RTL)

**Table 1.** Taxonomy of FPGA-based deep learning accelerator architectures for edge and embedded systems.

| **Architecture Type** | **Computation Model** | **Strengths** | **Limitations** |
|---|---|---|---|
| Streaming-based Dataflow | Static pipelined dataflow graph mapped to spatially arranged PEs | High throughput, low latency, efficient BRAM/DSP usage, predictable timing | Rigid structure, difficult to adapt to model changes or dynamic control flow |
| Systolic Array | Rhythmic data movement with fixed local interconnects among PEs | Excellent for dense matrix operations, high data reuse, efficient MAC scheduling | Poor flexibility, challenging to scale for sparse or irregular models, complex placement/routing |
| Overlay-based Accelerator | Virtual instruction sets and reconfigurable interconnects abstracting RTL details | Fast deployment, software-friendly, adaptable to multiple DNNs | Overhead in latency and resource usage, suboptimal peak performance |
| CGRA on FPGA | Reconfigurable functional blocks with programmable interconnect mesh | Balanced trade-off between performance and flexibility, partial reconfiguration supported | Requires complex control logic, tool support still maturing |
| Hybrid Heterogeneous System | Partitioned workload execution across CPU/GPU/NPU + FPGA | Flexible model support, task-level parallelism, robust software stack | Synchronization complexity, high integration overhead, increased system cost |

code or intermediate abstractions that synthesize into bitstreams executable on FPGA fabrics. This transformation is not merely a mechanical conversion but a deeply involved optimization process that reconciles software-defined neural architectures with hardware-defined constraints. The key to scalable performance lies in the co-design and co-optimization of compute, memory, and data movement, governed by both algorithmic and architectural parameters. Such design methodologies must consider numerous variables, including model depth and width, layer diversity, quantization precision, memory bandwidth, tiling strategies, and spatial reuse patterns, all while constrained by FPGA resource budgets (e.g., DSPs, BRAM, LUTs). As these variables are interdependent and subject to rapid change across applications, methodologies must be modular, extensible, and automation-friendly to remain viable for real-world edge deployment scenarios. One of the central tenets in scalable design is the principle of tiling, wherein the neural network computation is decomposed into smaller sub-blocks or "tiles" that fit within the limited on-chip memory and compute resources of an FPGA [32]. This enables partial loading of model weights and activations into BRAM and reduces the reliance on costly off-chip DRAM access [33]. The tiling parameters—often denoted $T_m, T_n, T_k$ for matrix dimensions—are selected to optimize throughput under constraints defined by the roofline model, which relates achievable performance to memory bandwidth and compute throughput. Coupled with tiling is the notion of loop unrolling and pipelining, which converts sequential computations into parallel streams by replicating processing elements or scheduling operations in temporally staggered fashion. These hardware-level transformations are frequently guided by high-level synthesis (HLS) tools, such

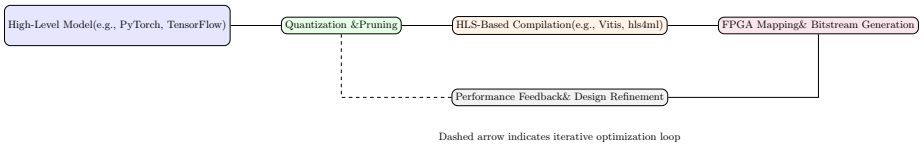


**Fig. 1.** Typical FPGA-based deep learning accelerator design flow. The process involves model quantization, hardware-aware compilation using HLS tools, resource mapping to the target FPGA platform, and iterative refinement based on deployment feedback.

as Xilinx Vitis HLS or Intel HLS Compiler, which allow design descriptions in C/C++ to be translated into synthesizable RTL. More advanced frameworks like hls4ml, FINN, and DNNWeaver enable direct ingestion of neural network descriptions and automate quantization, scheduling, and dataflow transformation. The optimality of a given schedule depends not only on performance but also on system-level metrics like energy efficiency, latency predictability, and adaptability to changing workloads—metrics that are difficult to express using traditional cost functions alone. To illustrate the typical stages of this design methodology, Figure 1 presents a high-level overview of the end-to-end flow, from model description to final deployment. This abstracted pipeline delineates how a trained model, possibly pruned and quantized, is processed by hardware-aware compilation flows that translate neural layers into efficient compute blocks mapped to FPGA resources. It also emphasizes the iterative feedback loop inherent in hardware-software co-design, where synthesis results guide further tuning of tiling, quantization, and architectural parameters. While early methodologies emphasized static compilation and fixed-function mapping, modern approaches increasingly incorporate dynamic reconfiguration, online profiling, and neural architecture search (NAS) to refine models post-deployment based on empirical performance data [34].

As FPGA technologies continue to evolve with higher logic density, embedded AI-specific IP blocks (such as AI Engines and hardened DSP slices), and tighter integration with CPUs and memory controllers, the design methodologies must also evolve toward increased automation and abstraction. There is growing emphasis on developing end-to-end toolchains that support not only static inference graphs but also dynamic execution models, on-device learning, and context-aware reconfiguration. Furthermore, the growing convergence between machine learning compilers (e.g., TVM, MLIR) and hardware synthesis tools is gradually dissolving the boundary between algorithm and architecture [35]. The ultimate goal is to enable hardware-aware model design at the same abstraction level as software, where FPGA-specific performance models, synthesis costs, and power estimations are natively embedded within the model development loop [36]. Such vertically integrated design methodologies are poised to unlock a new era of energy-proportional, scalable, and resilient intelligence across distributed edge ecosystems.

## 5 Runtime Reconfiguration and Adaptive Execution Strategies

One of the most profound advantages of FPGA technology in the context of edge-based deep learning acceleration is its capacity for runtime reconfiguration—a property that distinguishes FPGAs from fixed-function accelerators such as ASICs or even from traditional GPUs. While conventional compilation flows result in static execution pipelines, runtime reconfiguration allows FPGA-based systems to dynamically alter their architecture during operation, thereby adapting to changing workloads, performance constraints, or application contexts without requiring physical hardware modifications [37]. This adaptability is especially vital for edge computing platforms, which are frequently deployed in non-deterministic, multi-modal environments where operational conditions such as energy availability, thermal envelope, input data characteristics, or latency requirements can fluctuate over time [38]. Instead of maintaining a single monolithic bitstream that attempts to accommodate all possible execution scenarios—an approach that is both resource-hungry and inefficient—runtime reconfiguration allows the system to load specialized hardware configurations on demand, thereby achieving greater efficiency, lower latency, and improved energy proportionality. There are multiple levels at which reconfiguration may be employed [39]. At the most basic level, **partial dynamic reconfiguration (PDR)** allows specific regions of the FPGA fabric—typically known as reconfigurable partitions—to be updated independently of the rest of the design, without interrupting the overall system operation. This facilitates use cases such as switching between different neural network layers or operators (e.g., convolution, LSTM, transformer attention blocks) depending on current workload demands [40]. For instance, a surveillance drone may load a high-resolution object detector when in flight over populated areas, and switch to a low-power, low-resolution model during transit. Similarly, IoT devices may deploy lightweight anomaly detection networks under normal conditions, and reconfigure to more complex predictive models in the event of detected outliers [41]. The runtime controller, often implemented as a soft processor or a dedicated state machine, orchestrates the loading and unloading of bitstreams from local or external non-volatile storage [42]. Given that reconfiguration time is non-negligible—typically on the order of milliseconds depending on bitstream size and bandwidth—designers must carefully balance the overhead of reconfiguration against the gains in performance or energy efficiency [43]. Beyond PDR, **adaptive execution strategies** extend the notion of runtime flexibility to the algorithmic and software control levels [44]. These include techniques such as model sparsity gating, dynamic precision scaling, and workload-aware scheduling. For example, in precision scaling, input data statistics or confidence scores from intermediate layers may be used to adjust arithmetic precision during inference. A model might switch from 8-bit integer execution to 4-bit or binary mode when the input falls within well-understood statistical regimes, thereby reducing power consumption and increasing inference speed. FPGA implementations are well-suited for this due to their ability to instantiate multiple precision arithmetic units and route data dynamically

through them. Similarly, dynamic sparsity gating leverages runtime analysis of input activations or attention maps to skip unnecessary computations—such as zero-valued activations or low-importance tokens—thereby reducing memory access and compute cycles [45]. These mechanisms are particularly effective in transformer-based models and recurrent neural networks, where dynamic attention or temporal redundancy can be exploited. In hybrid systems, adaptive execution may also involve offloading decisions, where the FPGA collaborates with CPUs or NPUs to delegate portions of the workload based on real-time profiling. For example, an FPGA may perform initial feature extraction or convolutional front-end processing, and upon detecting increased load or thermal pressure, it may defer subsequent layers to an auxiliary processor. This requires tight integration of monitoring logic, task schedulers, and interconnect protocols such as AXI4, AMBA, or PCIe, and necessitates a carefully orchestrated software runtime. Furthermore, in systems with multiple FPGAs or chiplet-based fabrics, adaptive data placement and compute allocation across spatially distributed compute units introduces another layer of complexity. Load balancing, memory coherence, and latency sensitivity must all be considered within the reconfiguration logic to ensure uninterrupted inference and stable QoS metrics. In modern edge AI platforms, **online reconfigurability is increasingly guided by machine learning itself**. Reinforcement learning-based agents have been proposed to determine optimal reconfiguration policies, learning from environmental feedback such as power draw, inference latency, or accuracy degradation under quantized models [46]. This forms a virtuous loop where the FPGA not only accelerates deep learning models but also becomes a target for intelligent self-optimization [47]. In these frameworks, the control policy is treated as a meta-model trained either offline or in situ to maximize a reward function representing a composite of performance, energy, and resource usage [48]. The emerging field of **autonomous hardware management** builds on this idea, seeking to abstract away manual tuning and make FPGAs behave more like intelligent runtime-adaptive processors. By combining performance counters, thermal sensors, and logic usage monitors with predictive control algorithms, such systems are capable of forecasting bottlenecks or overloads and preemptively reconfiguring the fabric for better resilience and scalability. Despite these advancements, significant challenges remain [49]. The design and verification of reconfigurable hardware introduces substantial complexity, particularly with regard to bitstream compatibility, timing closure, and security [50]. Bitstream management requires trusted interfaces, version control, and secure boot mechanisms to avoid system corruption or malicious injection of reconfiguration payloads [51]. Additionally, not all FPGAs support fine-grained PDR, and the granularity of reconfigurable regions may limit the modularity of deployed models. Toolchain support is also still maturing; while some vendors provide basic partial reconfiguration frameworks, full support for seamless dynamic layer swapping and hardware virtualization remains an open area of research [52]. There is also a growing demand for standardized APIs and middleware that abstract reconfiguration details from application developers, enabling broader adoption without requiring deep hardware expertise [53].

Nevertheless, the capability for runtime reconfiguration and adaptive execution marks a paradigm shift in how FPGA-based deep learning systems are conceived and deployed. No longer constrained by static compilation and one-size-fits-all architectures, these systems are increasingly morphable, learning-aware, and environmentally responsive. In an edge computing world characterized by heterogeneity, uncertainty, and dynamism, such traits are not merely advantageous—they are essential [54]. The convergence of reconfigurable hardware and adaptive AI software will define the next generation of edge intelligence platforms, making them not only efficient and scalable, but also fundamentally alive to the changing conditions of their operational contexts.

## 6 Benchmarking and Performance Analysis of FPGA-Based Edge Accelerators

Benchmarking plays a pivotal role in the design and evaluation of FPGA-based deep learning accelerators, especially when deployed in edge and embedded systems that are characterized by resource-constrained environments and application-specific performance targets [55]. Unlike traditional computing platforms, where standardized benchmarking suites and homogeneous hardware architectures allow for relatively straightforward performance comparisons, FPGA-based systems operate across a highly heterogeneous and customizable design space [56]. Each accelerator may differ significantly in terms of hardware configuration, dataflow architecture, arithmetic precision, memory hierarchy, reconfiguration support, and toolchain flow, making apples-to-apples comparison non-trivial [57]. Moreover, edge deployments impose additional constraints that traditional cloud or server benchmarks do not capture—such as energy per inference, real-time latency bounds, thermal limits, and workload adaptivity [58]. As such, comprehensive and meaningful benchmarking must go beyond raw throughput or peak utilization metrics and instead adopt a holistic multi-dimensional approach that incorporates system-level metrics under realistic application conditions [59]. The first axis of evaluation in benchmarking FPGA-based accelerators is typically performance, often measured in terms of throughput (inferences per second), latency (time per inference), and effective MAC/s (multiply-accumulate operations per second). However, these metrics alone can be misleading without context [60]. Throughput may be artificially inflated by large batch sizes, which are infeasible for many edge applications that require real-time or near-instantaneous responses to streaming input. Latency measurements must differentiate between cold start latency, steady-state latency, and pipeline fill latency, each of which has different implications depending on the use case—e.g., voice activation in wearables versus object detection in autonomous drones. Furthermore, MAC/s values must be interpreted in light of arithmetic precision (e.g., INT8 vs FP32) and sparsity utilization (e.g., zero-skipping mechanisms), as lower-precision operations generally yield higher apparent compute throughput at the cost of reduced numerical accuracy. Therefore, performance metrics must be normalized appropriately—either by scaling with effective precision or by providing bit-

ops/s as an alternative measurement. Energy efficiency constitutes the second major axis of benchmarking and is perhaps the most critical for edge and mobile systems where battery life and thermal envelope are tightly constrained. Energy per inference, usually expressed in millijoules (mJ) or microjoules ($\mu$J) per sample, provides a direct indicator of the hardware-software stack's suitability for long-duration or always-on tasks [61]. Power profiling tools, such as Xilinx XPower Analyzer or Intel Power Analyzer, along with external measurements from current probes and power meters, are commonly used to assess dynamic power during inference. However, FPGA power usage is highly dependent on the specific region of the fabric being utilized, the clock frequency, and the activity factor of each computational block [62]. Consequently, fine-grained temporal profiling becomes necessary to identify power spikes during memory-intensive layers (e.g., fully connected layers or attention heads) or during partial reconfiguration events [63]. Additionally, thermal effects such as throttling, leakage, and ambient temperature variations can influence sustained performance and must be captured through long-duration testing under controlled or representative environmental conditions. Accuracy and model fidelity also play a central role in the benchmarking process, particularly when aggressive quantization or compression strategies are applied. A high-performance, low-latency accelerator is of limited use if the corresponding reduction in model accuracy renders the inference results unreliable. Therefore, it is standard practice to report not only top-1/top-5 accuracy scores but also domain-specific metrics such as mean average precision (mAP) for object detection, word error rate (WER) for speech recognition, or BLEU score for translation tasks. Crucially, these accuracy metrics must reflect post-deployment performance—i.e., measured after quantization, pruning, and mapping to the target hardware—with all hardware-induced artifacts (e.g., rounding, overflow, underflow, saturation) in place. Simulated accuracy derived from software emulation is insufficient and may overestimate real-world performance [64]. Some FPGA toolchains now provide bit-accurate emulators that model low-level effects of quantization, dataflow pipelining, and control path logic, allowing for more faithful pre-deployment accuracy estimation [65]. However, in-field validation using the actual bitstream on deployed hardware remains the gold standard for accuracy benchmarking [66]. Scalability and adaptability form the final set of critical benchmarking axes [67]. These relate not only to the accelerator's ability to handle increasing model sizes or input resolutions but also to its performance under dynamic workloads, multi-model inference, and real-time switching of model components via runtime reconfiguration. For instance, an edge gateway device may be required to run vision, audio, and anomaly detection models concurrently, each with distinct latency and throughput requirements. Benchmarking such scenarios involves evaluating multi-context execution, interleaved layer scheduling, and cache coherency across shared buffers [68]. Reconfigurable architectures must be evaluated on reconfiguration time, context-switch latency, and area overhead of static versus dynamic regions. Performance under partial reconfiguration must be profiled carefully, as bitstream loading delays can severely disrupt time-sensitive appli-

cations if not appropriately mitigated [69]. Moreover, adaptability metrics such as configuration agility (bitstreams per second), model footprint compression ratio, and performance degradation under load are increasingly becoming relevant for next-generation FPGA benchmarking. To facilitate standardized and reproducible benchmarking, several academic and industrial consortia have proposed FPGA-specific benchmarking frameworks. Examples include MLPerf Tiny, CHAI (Consortium for Hardware Acceleration of AI), and the FINN-Benchmark suite developed by Xilinx for low-bitwidth DNNs. These frameworks define reference models, input datasets, quantization parameters, and evaluation protocols, enabling fair comparison across research prototypes and commercial deployments. Nevertheless, adoption remains fragmented, and many FPGA acceleration papers report isolated metrics without complete system-level characterization [70]. It is therefore imperative that future benchmarking efforts in this field embrace open standards, community datasets, and comprehensive measurement methodologies that reflect the real-world operational context of edge AI systems. In summary, benchmarking FPGA-based deep learning accelerators is a multidimensional process that must span compute performance, energy efficiency, model accuracy, runtime adaptability, and system-level behavior under realistic operating conditions. A thorough and honest benchmarking methodology is not merely an academic exercise—it is a critical step toward guiding design trade-offs, validating research claims, and building trust in deployed systems [71]. As edge AI continues to proliferate into mission-critical domains such as healthcare, transportation, and autonomous robotics, rigorous performance analysis grounded in standardized, real-world-oriented benchmarks will become the cornerstone of credible and impactful FPGA-based accelerator research.

## 7 Case Studies of Real-World FPGA-Based Deep Learning Accelerators for Edge Intelligence

To ground the theoretical frameworks, architectural taxonomies, and benchmarking principles discussed thus far, it is essential to examine concrete case studies of FPGA-based deep learning accelerators deployed in real-world edge computing scenarios. These implementations, drawn from both industry and academia, not only demonstrate the practical viability of FPGAs as edge AI enablers but also shed light on the multifaceted engineering decisions, trade-offs, and design optimizations required to meet stringent operational constraints. From autonomous vehicles and industrial inspection systems to unmanned aerial vehicles (UAVs) and wearable medical devices, each application domain presents a unique combination of compute demand, latency tolerance, power envelope, environmental ruggedness, and upgradability [72]. By dissecting how specific accelerator architectures were conceived and realized in these domains, we gain valuable insight into the practical strategies that transform theoretical models into resilient, high-performance embedded AI systems. One representative case is Microsoft's Brainwave project, which initially targeted cloud inference using FPGAs but later expanded toward edge scenarios through Azure's IoT offerings.

Brainwave's architecture is notable for its use of statically configured, deeply pipelined dataflow designs synthesized from high-level descriptions using a custom toolchain that compiles directly from ML frameworks such as ONNX. Each layer of the deployed neural networks—often CNNs and RNNs—was mapped to a spatially distributed collection of processing elements optimized for 8-bit integer or 16-bit fixed-point operations, depending on precision sensitivity. Brainwave demonstrated a scalable design wherein multiple FPGAs were connected via high-speed interconnects, allowing data to flow across a multi-device topology without incurring costly memory offloading to host CPUs. In the context of edge deployment, this architectural modularity enabled task partitioning and redundancy, thereby supporting fault-tolerant inference in safety-critical applications such as traffic monitoring or drone navigation. An important insight from this case is that sustained throughput and predictability were prioritized over dynamic adaptability, reflecting the mission-critical nature of the workloads and the static model assumptions common in high-assurance edge domains. In contrast, Xilinx's FINN and FINN-R toolchains—developed for the deployment of quantized neural networks (QNNs) on FPGAs—focus on highly resource-efficient, ultra-low-power inference suitable for embedded edge platforms such as the Zynq UltraScale+ MPSoC family [73]. These designs often feature binary or ternary weights and activations, allowing inference logic to be implemented with minimal DSP usage and extensive LUT reuse. The case of deploying a binary CNN for traffic sign recognition on a ZCU102 board illustrates the effectiveness of such quantized architectures [74]. Here, the entire model was partitioned into hardware layers using a streaming dataflow approach, with dedicated line buffers and activation functions implemented in on-chip BRAM [75]. The architecture employed layer-specific tiling and loop unrolling parameters to balance memory bandwidth and logic utilization. Despite the severe quantization, the model maintained competitive classification accuracy on the GTSRB dataset while consuming less than 2W total system power [76]. This demonstrates how judicious co-optimization of algorithmic precision, memory scheduling, and hardware configuration can yield real-time, power-efficient inference suitable for battery-operated platforms like smart cameras or autonomous robots. Another instructive case is the deployment of YOLOv3-based object detection models on Intel's Stratix 10 and Agilex FPGAs for real-time industrial defect inspection. These models require high-resolution image inputs and fast frame processing to maintain production line throughput. Engineers adopted a heterogeneous architecture, where the initial convolutional layers—accounting for the bulk of compute—were offloaded to an FPGA fabric configured as a systolic array of MAC engines, while non-convolutional post-processing stages ran on the integrated hard ARM cores. The use of Intel's OpenVINO toolkit allowed model conversion and performance tuning across different quantization levels [77]. FPGA-specific kernels were compiled via the Intel HLS Compiler, with customized pipeline stages to align with the memory hierarchy and data layout of the device. The system achieved sub-20ms latency per frame while maintaining over 85% mean average precision, demonstrating that even compu-

tationally intensive detection models can be made to run efficiently on FPGAs without offloading to discrete GPUs [78]. However, this case also highlighted the challenge of achieving timing closure and memory interface optimization in designs involving multiple asynchronous data paths, necessitating extensive iteration and floorplanning expertise. The deployment of edge AI on autonomous drones presents a unique use case with extreme constraints in terms of size, weight, power (SWaP), and latency [79]. One such case involved integrating an FPGA-based feature extraction accelerator onto a UAV flight control board powered by a Xilinx Zynq-7000 SoC. The model in question was a streamlined version of MobileNet, with depthwise separable convolutions approximated using integer arithmetic. Here, the emphasis was on minimizing inference latency to enable obstacle avoidance and target tracking within a 30ms control loop [80]. The FPGA implementation consisted of custom PE clusters configured via a dynamic scheduling FSM, which adjusted the compute flow based on UAV state and sensor data volume [81]. Partial reconfiguration support allowed the flight control logic and the CNN inference engine to co-exist within the same device, enabling both hard real-time control and machine vision with minimal interference [82]. Despite limited computational resources, the platform achieved 10 FPS inference at under 5W system power, validating the efficacy of FPGA-based edge AI in severely resource-constrained, latency-sensitive embedded environments. In wearable health monitoring systems, FPGAs have also proven useful for continuous biosignal processing with machine learning classifiers [83]. A notable case involved an arrhythmia detection pipeline implemented on an Intel MAX 10 FPGA, where input ECG signals were preprocessed using digital filters and then passed through a compact LSTM-based classifier trained offline and quantized to 8-bit weights. Given the continuous nature of input and the need for sub-millisecond response times in life-critical settings, the system was designed to support overlapping window execution and re-entrant processing. The classifier's memory and computation were tiled to fit within the device's limited BRAM and DSP blocks, and data augmentation techniques were applied to stabilize quantized model accuracy. Real-world deployment tests showed successful heartbeat classification with over 95% sensitivity and specificity, while operating under 100mW power—enabling week-long use on a single charge [84]. This case underscores the suitability of FPGAs for biomedical edge AI, where deterministic execution, long battery life, and privacy-preserving local computation are paramount. Collectively, these case studies illustrate the breadth and depth of FPGA applications in edge deep learning, and more importantly, the diversity of strategies that must be adopted depending on the specific deployment scenario [85]. No one-size-fits-all architecture exists; rather, success hinges on the careful alignment of algorithmic design, hardware resource management, system integration, and application-level constraints. From ultra-low-power binary networks for embedded vision to high-throughput, reconfigurable detection pipelines in industrial automation, FPGAs have demonstrated their unmatched versatility and effectiveness [86]. As tools, compilers, and IP libraries mature, and as community best practices continue to crystallize through case-driven design,

the accessibility and adoption of FPGA-based acceleration for intelligent edge systems are poised to grow rapidly, unlocking new frontiers in high-performance, efficient, and trustworthy embedded AI deployment.

## 8 Conclusion and Future Directions

The convergence of deep learning and reconfigurable hardware has opened a transformative pathway for intelligent computing at the edge, where energy efficiency, latency sensitivity, and adaptability are paramount. In this context, FPGAs have emerged not merely as alternatives to CPUs and GPUs but as fundamentally distinct platforms capable of tailoring computational structures to the intricate needs of modern AI workloads. This review has traversed the expansive design landscape of FPGA-based deep learning accelerators—from architectural taxonomies and design methodologies to dynamic runtime reconfiguration, benchmarking paradigms, and real-world case studies—each revealing a critical dimension of how scalability, efficiency, and versatility are architected into edge systems. The essential theme that emerges is not just one of performance gains or hardware-software co-optimization, but a broader rethinking of how AI can be spatially, temporally, and contextually aligned with the environments in which it operates. As deep learning models grow more complex and edge applications become more diverse and demanding, the role of FPGAs will likely deepen, evolving beyond fixed-function acceleration to become adaptive computational substrates embedded within our physical world.

Yet, the full potential of FPGA-based acceleration remains constrained by several persistent challenges that future research and development must systematically address. Toolchain maturity remains one of the most critical bottlenecks. While high-level synthesis tools, domain-specific compilers, and ML integration frameworks have significantly lowered the barrier to FPGA deployment, the learning curve, debugging complexity, and limited abstraction layers still impede mass adoption. There is a pressing need for unified software stacks that allow developers to target FPGAs using familiar ML abstractions while automatically generating optimized hardware implementations that respect performance, power, and area constraints. Such tools should not only support quantization-aware training and pruning-aware mapping, but also integrate hardware profiling feedback in an iterative loop akin to traditional compiler optimizations. The emergence of frameworks like MLIR, TVM-VTA, hls4ml, and Xilinx Vitis AI points in this direction, but widespread interoperability, platform-independence, and ease-of-use remain aspirational. A future in which FPGA acceleration is accessible to every ML developer—without requiring hardware design expertise—will hinge on whether the compiler and runtime ecosystems mature into a unified, end-to-end deployment paradigm.

Another area demanding deeper exploration is the integration of reconfigurability and learning, not merely as separate capabilities but as tightly coupled properties of adaptive AI systems. Current FPGA-based platforms allow dynamic reconfiguration and runtime control, but these mechanisms are often

orchestrated via static rules or pre-specified policies. The frontier lies in building intelligent FPGA platforms that can reason about their own configuration states, observe environmental variables such as power availability or workload diversity, and adjust their hardware behavior using learned strategies. This introduces the exciting possibility of deploying control models—based on reinforcement learning, Bayesian optimization, or neuro-symbolic reasoning—that guide FPGA configuration decisions in real-time, turning the FPGA into a self-optimizing organism. In such systems, the hardware becomes an active participant in the AI lifecycle, not merely executing neural computations but shaping how, when, and with what precision those computations are carried out. Embedding this intelligence into the FPGA's runtime control fabric could enable dramatic leaps in adaptability, efficiency, and robustness—particularly in scenarios involving edge swarms, autonomous robotics, or fault-tolerant sensor networks.

Moreover, as deep learning itself evolves—with the rise of foundation models, multi-modal pipelines, and sparse dynamic computation graphs—the architectural templates used in current FPGA accelerators will need to undergo substantial transformation. Traditional CNN and MLP-based pipelines, while still important, are being supplemented and, in many cases, supplanted by transformer architectures, graph neural networks, and hybrid neuro-symbolic systems. These models pose new challenges: irregular memory access patterns, high temporal dependencies, dynamic activation paths, and massive parameter footprints that strain the resource budgets of embedded FPGAs. Addressing these will require new forms of architectural innovation: dynamically reconfigurable attention engines, logic-level support for sparse matrix multiplications, near-memory computation for intermediate activations, and the integration of domain-specific IP blocks such as AI Engines or custom vector processors within the FPGA fabric. Equally important is the evolution of interconnect and memory subsystems, which must support efficient pipelining and buffering across multiple compute regions, enabling fine-grained partitioning of large models across reconfigurable hardware instances. This also implicates the broader system design—where FPGAs must be tightly integrated with CPUs, NPUs, and potentially neuromorphic accelerators to form hybrid processing substrates capable of executing complex workloads with real-time constraints.

In the long term, the emergence of chiplet-based and 3D-stacked FPGA technologies will further reshape the landscape, allowing modular integration of reconfigurable logic, analog computing blocks, in-memory compute arrays, and high-bandwidth interconnect fabrics. These advances promise to overcome many of the spatial and power delivery limitations of monolithic FPGAs, enabling highly specialized, edge-optimized AI platforms that can adapt at both the functional and structural levels. Combined with trends in approximate computing, event-driven execution, and federated inference, such developments point toward a vision in which edge intelligence is no longer a constrained derivative of cloud-based AI but an autonomous, first-class computational paradigm in its own right. FPGAs, with their inherent morphability, will play a central role in

this vision—not only as accelerators but as cognitive substrates capable of hosting AI that evolves, adapts, and endures across time, tasks, and environments.

In conclusion, the intersection of deep learning and FPGA-based acceleration represents a frontier of immense opportunity and complexity. It is a domain where architectural creativity, algorithmic ingenuity, and system-level pragmatism must coexist, constantly balancing the trade-offs between flexibility, performance, and cost. As the demand for ubiquitous intelligence continues to grow, and as edge systems take on increasingly vital roles in our infrastructure, industry, and daily lives, the strategic deployment of custom, reconfigurable accelerators will become essential. FPGAs are not a panacea, but they are uniquely positioned to embody a new computational ethos—one that is adaptive, efficient, and intimately aware of its physical and contextual constraints. Through continued advances in tools, design methodologies, and application-driven innovation, FPGA-based AI systems will no longer be niche or exotic. They will be foundational pillars of the next generation of embedded intelligence.

## References

1. Raúl Rojas. Stochastic Networks. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 371–387. Springer, Berlin, Heidelberg, 1996.
2. J Arnold, A Caldow, and K Harman. A reconfigurable 100 Mchip/s spread spectrum receiver. In *2003 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL II, PROCEEDINGS: SPEECH II; INDUSTRY TECHNOLOGY TRACKS; DESIGN & IMPLEMENTATION OF SIGNAL PROCESSING SYSTEMS; NEURAL NETWORKS FOR SIGNAL PROCESSING*, International Conference on Acoustics Speech and Signal Processing ICASSP, pages 445–448, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2003. IEEE.
3. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.
4. Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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*, FPGA '17, pages 65–74, New York, NY, USA, February 2017. Association for Computing Machinery.
5. Ricardo Lent. A Cognitive Network Controller Based on Spiking Neurons. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2018.
6. Yassine Zniyed, Thanh Phuong Nguyen, et al. Enhanced network compression through tensor decompositions and pruning. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):4358–4370, 2024.
7. Hualiang Zhuang and Kay Soon Low. Real Time Runway Detection in Satellite Images Using Multi-Channel PCNN. In *PROCEEDINGS OF THE 2014 9TH IEEE CONFERENCE ON INDUSTRIAL ELECTRONICS AND APPLICATIONS (ICIEA)*, IEEE Conference on Industrial Electronics and Applications, pages 253+, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014. IEEE.
8. Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, June 2020.

9. Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size, 2016.
10. Kimberly Robinson, Andrew Schorr, and David Smith. NASA's Space Launch System: Opportunities for Small Satellites to Deep Space Destinations.
11. Manual Grana, Miguel A Veganzons, and Borja Ayerdi. Hyperspectral remote sensing scenes. *Grupo de Inteligencia Computacional*, 2020.
12. R. L. Davidson and C. P. Bridges. Error Resilient GPU Accelerated Image Processing for Space Applications. *IEEE Transactions on Parallel and Distributed Systems*, 29(9):1990–2003, September 2018.
13. Xin Zhang, Wenjing Wang, Chaopin Bai, Yueqiang Sun, Shichen Jiang, Zhihao Yang, Qiang Chen, Lichang Zhang, Liguo Zhang, Zhiliang Zhang, Ziting Wang, and Shuai Zhang. An Analysis of the Effect of Hall Thruster Plumes on Surface Charging of a Complex Spacecraft Structure. *APPLIED SCIENCES-BASEL*, 14(6), March 2024.
14. Andrew Ekblad, Trupti Mahendrakar, Ryan White, Markus Wilde, Isaac Silver, and Brooke Wheeler. Resource-constrained FPGA Design for Satellite Component Feature Extraction. In *2023 IEEE AEROSPACE CONFERENCE*, IEEE Aerospace Conference Proceedings, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2023. IEEE.
15. Raúl Rojas. The Backpropagation Algorithm. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 149–182. Springer, Berlin, Heidelberg, 1996.
16. Eric P. Kim and Naresh R. Shanbhag. Soft N-Modular Redundancy. *IEEE Transactions on Computers*, 61(3):323–336, March 2012.
17. Jinghua Wang, Yue Wang, Shiwen Xing, Bingyi Li, Yizhuang Xie, Ping Nie, and Kun Tang. A New Reconfigurable Methodology to Implement the 8-bit Linear Quantization for Real-time SAR. In *2016 16TH INTERNATIONAL SYMPOSIUM ON COMMUNICATIONS AND INFORMATION TECHNOLOGIES (ISCIT)*, pages 171–174, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2016. IEEE.
18. Dominika Przewlocka, Mateusz Wasala, Hubert Szolc, Krzysztof Blachut, and Tomasz Kryjak. Optimisation of a Siamese Neural Network for Real-Time Energy Efficient Object Tracking. volume 12334, pages 151–163. 2020.
19. Lin Li, Shengbing Zhang, and Juan Wu. Efficient Object Detection Framework and Hardware Architecture for Remote Sensing Images. *REMOTE SENSING*, 11(20), October 2019.
20. Tara A. Estlin, Benjamin J. Bornstein, Daniel M. Gaines, Robert C. Anderson, David R. Thompson, Michael Burl, Rebecca Castaño, and Michele Judd. AEGIS Automated Science Targeting for the MER Opportunity Rover. *ACM Trans. Intell. Syst. Technol.*, 3(3):50:1–50:19, May 2012.
21. Rahul Ratnakumar and Satyasai Jagannath Nanda. A high speed roller dung beetles clustering algorithm and its architecture for real-time image segmentation. *APPLIED INTELLIGENCE*, 51(7):4682–4713, July 2021.
22. Intel Movidius. Intel® Movidius™ Myriad™ 2 Vision Processing Unit 4GB - Product Specifications. https://www.intel.com/content/www/us/en/products/sku/122461/intel-movidius-myriad-2-vision-processing-unit-4gb/specifications.html, 2020.
23. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

24. Bingyi Zhang, Rajgopal Kannan, Viktor Prasanna, and Carl Busart. Accurate, Low-latency, Efficient SAR Automatic Target Recognition on FPGA. In *2022 32ND INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, FPL*, International Conference on Field Programmable Logic and Applications, pages 1–8, 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2022. IEEE COMPUTER SOC.
25. Javier Echanobe, Raul Finker, and Ines del Campo. A Divide-and-Conquer Strategie for FPGA Implementations of Large MLP-based Classifiers. In *2015 INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN)*, IEEE International Joint Conference on Neural Networks (IJCNN), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2015. IEEE.
26. European Space Agency ESA. Sentinel 2 Products. https://sentiwiki.copernicus.eu/web/s2-products, 2025.
27. Luca Zulberti, Matteo Monopoli, Pietro Nannipieri, Silvia Moranti, Geert Thys, and Luca Fanucci. Efficient Coarse-Grained Reconfigurable Array architecture for machine learning applications in space using DARE65T library platform. *Microprocessors and Microsystems*, 113:105142, March 2025.
28. Raúl Rojas. Threshold Logic. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 29–53. Springer, Berlin, Heidelberg, 1996.
29. Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation, December 2017.
30. Shun Yan, Zhengyan Liu, Yun Wang, Chenglong Zeng, Qiang Liu, Bowen Cheng, and Ray C. C. Cheung. An FPGA-based MobileNet Accelerator Considering Network Structure Characteristics. In *2021 31ST INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS (FPL 2021)*, International Conference on Field Programmable Logic and Applications, pages 17–23, 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2021. IEEE COMPUTER SOC.
31. Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation, July 2018.
32. Michael Petry, Patrick Gest, Andreas Koch, Max Ghiglione, and Martin Werner. Accelerated Deep-Learning inference on FPGAs in the Space Domain. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, CF '23, pages 222–228, New York, NY, USA, August 2023. Association for Computing Machinery.
33. Shriharsha Koila, Goutham G. D. Simha, Muralidhar Kulkarni, and U. Sripati. FPGA Implementation of a BCH Codec for Free Space Optical Communication System. In *2014 INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI)*, pages 1822–1826, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014. IEEE.
34. Romen Neris, Adrian Rodriguez, Raul Guerra, Sebastian Lopez, and Roberto Sarmiento. FPGA-Based Implementation of a CNN Architecture for the On-Board Processing of Very High-Resolution Remote Sensing Images. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING*, 15:3740–3750, 2022.
35. Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

36. Janet L. Barth. Space and Atmospheric Environments: From Low Earth Orbits to Deep Space. In Jacob I. Kleiman and Zelina Iskanderova, editors, *Protection of Materials and Structures from Space Environment*, pages 7–29, Dordrecht, 2003. Springer Netherlands.
37. Serena Curzel, Michele Fiorito, Patricia Lopez Cueva, Tiago Jorge, Thanassis Tsiodras, and Fabrizio Ferrandi. Exploration of Synthesis Methods from Simulink Models to FPGA for Aerospace Applications Invited Paper. In *PROCEEDINGS OF THE 20TH ACM INTERNATIONAL CONFERENCE ON COMPUTING FRONTIERS 2023, CF 2023*, pages 243–249, 1601 Broadway, 10th Floor, NEW YORK, NY, UNITED STATES, 2023. ASSOC COMPUTING MACHINERY.
38. Jie Li, Chuanlun Zhang, Wenxuan Yang, Heng Li, Xiaoyan Wang, Chuanjun Zhao, Shuangli Du, and Yiguang Liu. FPGA-Based Low-Bit and Lightweight Fast Light Field Depth Estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–14, 2025.
39. Donghyuk Kim, Sanghyun Jeong, and Joo-Young Kim. Agamotto: A Performance Optimization Framework for CNN Accelerator With Row Stationary Dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(6):2487–2496, June 2023.
40. Edgar Lemaire, Matthieu Moretti, Lionel Daniel, Benoit Miramond, Philippe Millet, Frederic Feresin, and Sebastien Bilavarn. An FPGA-based Hybrid Neural Network accelerator for embedded satellite image classification. In *2020 IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS)*, IEEE International Symposium on Circuits and Systems, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2020. IEEE.
41. George Lentaris, Ioannis Stamoulias, Dimitrios Soudris, and Manolis Lourakis. HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(8):1563–1577, August 2016.
42. David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–4, April 2019.
43. J Lyke. Reconfigurable systems: A generalization of reconfigurable computational strategies for space systems. In *2002 IEEE AEROSPACE CONFERENCE PROCEEDINGS, VOLS 1-7*, IEEE Aerospace Conference Proceedings, pages 1935–1950, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2002. IEEE.
44. David Miranda. 2020 NASA technology taxonomy. Technical report, 2020.
45. YZ Han, Z Ruan, and JG Han. On fuzzily-processed-BP-network-based dual-mode control with realization on on-line reconfigurable FPGA. In MS Zhao and ZZ Shi, editors, *PROCEEDINGS OF THE 2005 INTERNATIONAL CONFERENCE ON NEURAL NETWORKS AND BRAIN, VOLS 1-3*, pages 841–846, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2005. IEEE.
46. Martin Daněk, Leoš Kafka, Lukáš Kohout, Jaroslav Sýkora, and Roman Bartosiński. The LEON3 Processor. In Martin Daněk, Leoš Kafka, Lukáš Kohout, Jaroslav Sýkora, and Roman Bartosinski, editors, *UTLEON3: Exploring Fine-Grain Multi-Threading in FPGAs*, pages 9–14. Springer, New York, NY, 2013.
47. Weison Lin, Yajun Zhu, and Tughrul Arslan. DycSe: A Low-Power, Dynamic Reconfiguration Column Streaming-Based Convolution Engine for Resource-Aware Edge AI Accelerators. *JOURNAL OF LOW POWER ELECTRONICS AND APPLICATIONS*, 13(1), March 2023.

48. Philippe Reiter, Philipp Karagiannakis, Murray Ireland, Steve Greenland, and Louise Crockett. FPGA acceleration of a quantized neural network for remote-sensed cloud detection. In *7th International Workshop on On-Board Payload Data Compression*, Virtual, September 2020.
49. Wolfgang Klimesch. The frequency architecture of brain and brain body oscillations: An analysis. *European Journal of Neuroscience*, 48(7):2431–2453, 2018.
50. Hualiang Zhuang, Kay-Soon Low, and Wei-Yun Yau. A multiplier-less GA optimized pulsed neural network for satellite image analysis using a FPGA. In *ICIEA 2008: 3RD IEEE CONFERENCE ON INDUSTRIAL ELECTRONICS AND APPLICATIONS, PROCEEDINGS, VOLS 1-3*, IEEE Conference on Industrial Electronics and Applications, pages 302+, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2008. IEEE.
51. Pansoo Kim and Deock-Gil Oh. Study of Clock Synchronization for High Speed Satellite Communication Transmission. In SI Ao, C Douglas, WS Grundfest, and J Burgstone, editors, *WORLD CONGRESS ON ENGINEERING AND COMPUTER SCIENCE, WCECS 2013, VOL II*, volume Ao, of *Lecture Notes in Engineering and Computer Science*, pages 750–754, UNIT 1, 1-F, 37-39 HUNG TO ROAD, KWUN TONG, HONG KONG, 00000, PEOPLES R CHINA, 2013. INT ASSOC ENGINEERS-IAENG.
52. S.E. Kerns, B.D. Shafer, L.R. Rockett, J.S. Pridmore, D.F. Berndt, N. van Vonno, and F.E. Barber. The design of radiation-hardened ICs for space: A compendium of approaches. *Proceedings of the IEEE*, 76(11):1470–1509, November 1988.
53. S.R. Hedberg. AI coming of age: NASA uses AI for autonomous space exploration. *IEEE Expert*, 12(3):13–15, May 1997.
54. Yuan Liu, Yi Shen, Zhao-wei Sun, and Lei Xing. Task Scheduling Algorithm of FPGA for On-Board Reconfigurable Coprocessor. In *2013 INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE (ICCSAI 2013)*, pages 207–214, 439 DUKE STREET, LANCASTER, PA 17602-4967 USA, 2013. DESTECH PUBLICATIONS, INC.
55. Cao Yang and Li Ming. Far Field Demonstration Experiment of Fine Tracking System for Satellite-to-Ground Optical Communication. *CHINA COMMUNICATIONS*, 7(3):139–145, July 2010.
56. Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, July 2019.
57. Mihai Coca and Mihai Datcu. FPGA Accelerator for Meta-Recognition Anomaly Detection: Case of Burned Area Detection. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING*, 16:5247–5259, 2023.
58. Zhihao Zhang, Gaoming Du, Zhenmin Li, Qinran Kang, Wenyao Zhao, and Xiaolei Wang. An energy-efficient dehazing neural network accelerator based on E$$^2$$AOD-Net. *Journal of Real-Time Image Processing*, 21(6):197, November 2024.
59. Aksel S. Danielsen, Tor Arne Johansen, and Joseph L. Garrett. Self-Organizing Maps for Clustering Hyperspectral Images On-Board a CubeSat. *Remote Sensing*, 13(20):4174, January 2021.
60. AI Vitis. Vitis AI user guide (UG1414 v2. 0). *AMD Xilinx*, 2022.
61. Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

62. Christopher Wilson, Sebastian Sabogal, Alan George, and Ann Gordon-Ross. Hybrid, adaptive, and reconfigurable fault tolerance. In *2017 IEEE Aerospace Conference*, pages 1–11, March 2017.
63. Raúl Rojas. Associative Networks. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 309–334. Springer, Berlin, Heidelberg, 1996.
64. ESAS. National academies of sciences, engineering, and medicine. 2018. Thriving on our changing planet: A decadal strategy for earth observation from space, 2017.
65. Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A Survey of Optimization Methods From a Machine Learning Perspective. *IEEE Transactions on Cybernetics*, 50(8):3668–3681, August 2020.
66. David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
67. Raúl Rojas. One and Two Layered Networks. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 123–148. Springer, Berlin, Heidelberg, 1996.
68. Olivier Grasset, MK Dougherty, A Coustenis, EJ Bunce, C Erd, D Titov, M Blanc, A Coates, P Drossart, LN Fletcher, et al. Jupiter icy moons explorer (juice): An esa mission to orbit ganymede and to characterise the jupiter system. *Planetary and Space Science*, 78:1–21, 2013.
69. Alan D. George and Christopher M. Wilson. Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites. *Proceedings of the IEEE*, 106(3):458–470, March 2018.
70. Raúl Rojas. Genetic Algorithms. In Raúl Rojas, editor, *Neural Networks: A Systematic Introduction*, pages 427–448. Springer, Berlin, Heidelberg, 1996.
71. Sparsh Mittal. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, 32(4):1109–1139, February 2020.
72. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.
73. Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proceedings of the IEEE*, 105(10):1865–1883, October 2017.
74. Michael Wirthlin. High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond. *Proceedings of the IEEE*, 103(3):379–389, March 2015.
75. Trupti Mahendrakar, Steven Holmberg, Andrew Ekblad, Emma Conti, Ryan T. White, Markus Wilde, and Isaac Silver. Autonomous Rendezvous with Non-cooperative Target Objects with Swarm Chasers and Observers, January 2023.
76. Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, December 1997.
77. Chunming Jiang and Yilei Zhang. A Noise-Based Novel Strategy for Faster SNN Training. *Neural Computation*, 35(9):1593–1608, August 2023.
78. Cunguang Zhang, Hongxun Jiang, Riwei Pan, Haiheng Cao, and Mingliang Zhou. High-Throughput, Resource-Efficient Multi-Dimensional Parallel Architecture for Space-Borne Sea-Land Segmentation. *JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS*, 30(2), February 2021.
79. Ali Syahputra Nasution, Bayu Satya Adhitama, Fadillah Halim Rasyidi, Adi Aufarachman Putra Bambang Dwi, Muhammad Thufaili Imdad, Nugroho Widi Jatmiko, Suhermanto, Hidayat Gunawan, A. Hadi Syafruddin, and Denny Darlis. Development of the NOAA-19 Satellite Data Receiver Ingest System based on FPGA.

In *2022 INTERNATIONAL CONFERENCE ON RADAR, ANTENNA, MICROWAVE, ELECTRONICS, AND TELECOMMUNICATIONS (ICRAMET): EMERGING SCIENCE AND INDUSTRIAL INNOVATION IN ELECTRONICS AND TELECOMMUNICATION*, pages 132–137, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2022. IEEE.

80. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks, March 2019.
81. Parth Shah, Bhavesh Soni, Mohammad Waris, Rajiv Kumaran, Sanjeev Mehta, and Arup Roy Chowdhury. Generic and Programmable Timing Generator for CCD Detectors. In *2014 INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI)*, pages 1845–1851, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014. IEEE.
82. D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, New York, May 2002.
83. Woo-Joong Kim and Chan-Hyun Youn. Cooperative Scheduling Schemes for Explainable DNN Acceleration in Satellite Image Analysis and Retraining. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 33(7):1605–1618, July 2022.
84. Erik Kulu. Small Launchers - 2023 Industry Survey and Market Analysis. *th International Astronautical Congress*, 2023.
85. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
86. M. Abramovici and C.E. Stroud. BIST-based test and diagnosis of FPGA logic blocks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):159–172, February 2001.