High-Performance FPGA Acceleration for Transformer-Based Models

Gulnaz Rati *, Rafael Costa , Lena Ishikawa

Posted Date: 15 July 2025

doi: 10.20944/preprints202507.1187.v1

Keywords: FPGA acceleration; foundation models; transformer neural networks; hardware-software co-design; reconfigurable computing; systolic arrays; quantization; dataflow architecture; deep learning hardware; high-level synthesis; AI inference; energy-efficient computation; model-hardware co-design; domain-specific architectures; edge AI deployment

 doi:10.20944/preprints202507.1187.v1

Article

High-Performance FPGA Acceleration for Transformer-Based Models

Gulnaz Rati [1,*], Rafael Costa [2] and Lena Ishikawa [3]

[1] University of São Paulo, Brazil
[2] Federal University of Rio Grande do Sul, Brazil
[3] Kyoto University, Japan
* Correspondence: gulnaz.zati@usp.br

Abstract

Foundation neural networks—large-scale, pre-trained models such as transformers—have rapidly emerged as the cornerstone of state-of-the-art artificial intelligence systems across natural language processing, vision, multi-modal understanding, and beyond. These models, characterized by billions of parameters and intricate computational graphs, demand unprecedented levels of compute, memory bandwidth, and energy efficiency, especially during inference at scale. While GPUs and TPUs have been the dominant hardware platforms supporting these models, their limitations in power efficiency, customization, and deterministic latency have motivated the exploration of alternative accelerators. Field-Programmable Gate Arrays (FPGAs) have recently gained significant attention as a viable solution due to their reconfigurability, fine-grained parallelism, and ability to tailor hardware directly to model-specific computations. However, deploying foundation models efficiently on FPGAs remains an enormously challenging task due to a multitude of hardware-software co-design complexities, memory hierarchy limitations, toolchain immaturity, and a lack of abstraction layers suited for the evolving model landscape. This paper provides a comprehensive and in-depth review of FPGA-based accelerators for foundation neural networks, systematically examining the current architectural techniques, optimization strategies, and deployment methodologies that have been proposed in the literature and industry. We begin by surveying the fundamental challenges in mapping high-dimensional tensor operations—especially those involved in attention mechanisms, normalization layers, and large-scale matrix multiplication—onto FPGA fabrics, and discuss techniques such as systolic arrays, dataflow execution, quantization, sparsity exploitation, and operator fusion that mitigate these challenges. We analyze representative accelerator architectures and demonstrate how different design trade-offs influence performance, power efficiency, and scalability across various FPGA platforms. Furthermore, we explore the critical role of compiler and toolchain ecosystems in enabling efficient model-to-hardware transformations, identifying the current bottlenecks and highlighting emerging frameworks aimed at closing the productivity gap. Beyond the state-of-the-art, we delve into unresolved technical barriers including on-chip memory constraints, dynamic sequence handling, design space exploration (DSE) complexity, and limitations in runtime adaptability. We also discuss how FPGAs can be integrated into larger heterogeneous systems, including hybrid FPGA-GPU architectures and cloud-based FPGA-as-a-Service platforms, to support full-scale deployment pipelines for foundation models. Particular attention is given to the emerging paradigm of model-hardware co-design, where foundation models are trained with explicit consideration of hardware constraints to maximize efficiency and deployability. Finally, we outline key future directions, including ultra-low-precision arithmetic, reconfigurable attention kernels, FPGA-friendly model architectures, and domain-specific compilers that may fundamentally reshape the design landscape of foundation model accelerators. Through this review, we aim to provide a detailed roadmap for researchers, engineers, and system architects seeking to harness the potential of FPGA platforms for foundation model inference and beyond. By bringing together insights from machine learning, hardware architecture, and systems engineering, we highlight not only the promise but also the rigorous interdisciplinary efforts required to make FPGA-based AI acceleration viable, scalable, and accessible in the era of ever-growing foundation models.

 doi:10.20944/preprints202507.1187.v1

Keywords: FPGA acceleration; foundation models; transformer neural networks; hardware-software co-design; reconfigurable computing; systolic arrays; quantization; dataflow architecture; deep learning hardware; high-level synthesis; AI inference; energy-efficient computation; model-hardware co-design; domain-specific architectures; edge AI deployment

1. Introduction

The rapid evolution of artificial intelligence (AI) and, more specifically, deep learning (DL) has led to unprecedented advancements across a broad spectrum of applications, including natural language processing, computer vision, robotics, and healthcare. At the core of these breakthroughs are deep neural networks (DNNs), particularly the increasingly popular class of models referred to as foundation models. These large-scale neural networks are trained on vast and diverse datasets and are capable of being fine-tuned or adapted to perform a wide variety of downstream tasks [1]. Examples such as OpenAI's GPT, Google's BERT, Meta's LLaMA, and DeepMind's Gopher have demonstrated the viability and effectiveness of this paradigm, where a single, massive model serves as a general-purpose backbone for numerous applications. However, the exceptional performance of foundation models comes at the cost of extreme computational and memory demands, presenting significant challenges for deployment in edge environments or power- and cost-constrained scenarios. Field-Programmable Gate Arrays (FPGAs) have emerged as a promising platform for accelerating DNNs due to their inherent flexibility, parallelism, reconfigurability, and energy efficiency [2]. Unlike fixed-function hardware such as Application-Specific Integrated Circuits (ASICs) or general-purpose processors such as GPUs and CPUs, FPGAs provide a balance between performance and programmability, making them especially attractive for research and development of novel neural network architectures. In particular, FPGAs allow for customized datapath designs, fine-grained control over memory hierarchies, and dynamic reconfiguration to accommodate changes in model structure, precision, or application-specific constraints. These capabilities are crucial in the context of foundation models, where the model size, layer diversity, and evolving architectural patterns necessitate adaptable hardware solutions. The growing scale and complexity of foundation models pose unique challenges to FPGA-based acceleration. Firstly, the memory footprint of these models often exceeds the on-chip memory resources of FPGAs, requiring sophisticated memory management strategies and efficient external memory interfacing. Secondly, foundation models exhibit diverse computational patterns, including attention mechanisms, large-scale matrix multiplications, and sparse or quantized operations [3]. These necessitate heterogeneous compute units and dynamic resource allocation, which must be carefully orchestrated on FPGA fabric. Thirdly, the demand for high-throughput and low-latency inference, especially in real-time or interactive settings, requires highly optimized dataflow architectures, parallelism exploitation, and pipelining strategies. Addressing these challenges while maintaining scalability and energy efficiency remains a central concern in the design of FPGA-based accelerators. Over the past decade, the research community has proposed a wide range of FPGA-based accelerators tailored for DNNs, with increasing attention now being paid to foundation models. These accelerators differ significantly in terms of architectural choices, programming models, compilation flows, memory hierarchies, and target applications [4]. Some leverage High-Level Synthesis (HLS) tools to improve design productivity, while others employ hand-tuned Register Transfer Level (RTL) implementations to maximize performance. Techniques such as weight pruning, quantization, low-rank approximation, and operator fusion have been extensively explored to reduce the computational burden and improve the fit of foundation models on FPGA platforms. Additionally, recent works have investigated the co-design of neural network models and hardware accelerators, where model architectures are explicitly shaped by hardware constraints to achieve better overall efficiency. This review aims to provide a comprehensive and critical survey of the state-of-the-art in FPGA-based accelerators for foundation neural networks [5]. We systematically analyze the architectural paradigms, optimization

 doi:10.20944/preprints202507.1187.v1

techniques, design frameworks, and evaluation methodologies used in existing literature [6]. Our goal is to highlight key trends, identify open challenges, and provide insights into future directions in this rapidly evolving field [7]. To that end, we organize our review around several core themes: (1) an overview of foundation models and their computational characteristics; (2) a taxonomy of FPGA-based accelerator architectures; (3) optimization techniques tailored to foundation models; (4) design tools and automation frameworks; and (5) performance benchmarking and comparison. By synthesizing findings from recent advances and ongoing research, this review serves as a valuable reference for academics, practitioners, and system designers seeking to leverage FPGA technology for efficient deployment of foundation neural networks [8].

2. Computational Characteristics of Foundation Neural Networks

Foundation neural networks, often characterized by their massive parameter counts and deep architectural hierarchies, possess a range of computational characteristics that distinguish them from traditional deep learning models [9]. These characteristics are critical in understanding the challenges and opportunities involved in mapping such models onto FPGA-based accelerators [10]. In this section, we provide a rigorous mathematical analysis of the underlying operations, memory access patterns, and computational bottlenecks inherent in foundation models such as Transformers, large-scale language models (LLMs), and vision transformers (ViTs).

2.1. Model Size and Parameter Complexity

Let a foundation model be denoted by a function $f : \mathbb{R}^n \to \mathbb{R}^m$, parameterized by a set of weights $\boldsymbol{\theta} \in \mathbb{R}^P$, where P can range from hundreds of millions to hundreds of billions. For instance, GPT-3 has $P \approx 175 \times 10^9$ parameters [11]. These parameters are distributed across multiple layers, typically involving matrix multiplications, nonlinear activations, and normalization layers. Each layer l can be abstracted as a transformation:

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

where $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$, and $\phi^{(l)}$ is a nonlinear function (e.g., ReLU, GELU) [12]. The total number of floating-point operations (FLOPs) for a single forward pass is approximately:

$$\text{FLOPs}_{\text{total}} = \sum_{l=1}^{L} (2 \cdot d_l \cdot d_{l-1}),$$

excluding nonlinearities and assuming dense operations [13].

2.2. Transformer Architecture: A Computational Core

A majority of foundation models are based on the Transformer architecture, whose core component is the self-attention mechanism. Given an input sequence $\mathbf{X} \in \mathbb{R}^{T \times d}$, where T is the sequence length and d is the embedding dimension, the attention operation computes:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

where the query, key, and value matrices are defined as:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

with $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ [14]. The computational complexity of a single-head attention mechanism is:

$$\mathcal{O}(T \cdot d \cdot d_k + T^2 \cdot d_k + T \cdot d_k \cdot d_v),$$

 doi:10.20944/preprints202507.1187.v1

which becomes substantial for large T and d, especially in long-context applications such as document summarization and code generation. When extended to multi-head attention with H heads, the complexity scales linearly:

$$\text{FLOPs}_{\text{MHA}} = H \cdot \Big[\mathcal{O}(T \cdot d \cdot d_k) + \mathcal{O}(T^2 \cdot d_k) + \mathcal{O}(T \cdot d_k \cdot d_v)\Big],$$

followed by a projection:

$$\mathbf{Y} = \text{Concat}(\mathbf{h}_1, \ldots, \mathbf{h}_H)\mathbf{W}_O,$$

where $\mathbf{W}_O \in \mathbb{R}^{H \cdot d_v \times d}$ [15].

2.3. Memory Bandwidth and Data Movement

In FPGAs, the on-chip Block RAM (BRAM) and UltraRAM (URAM) resources are limited, typically supporting only a few megabytes of storage [16]. Given that model weights $\boldsymbol{\theta}$ for a single transformer layer may exceed hundreds of megabytes, memory bandwidth becomes a critical bottleneck. Let M denote the total size of parameters to be fetched from off-chip DRAM, and let B denote the available memory bandwidth in GB/s. Then the theoretical lower bound on data transfer time is:

$$t_{\text{mem}} \geq \frac{M}{B} \text{[17]}.$$

Moreover, frequent data movement between compute and memory units leads to increased latency and power consumption [18]. Thus, minimizing the data movement through techniques such as weight reuse, tiling, and caching is essential. Assuming a layer has an activation tensor of size $A \in \mathbb{R}^{T \times d}$, the access patterns can be modeled as:

$$\text{Read}(A) + \text{Read}(W) + \text{Write}(O),$$

where W is the weight matrix and O is the output tensor. The data reuse ratio r is defined as:

$$r = \frac{\text{Total Computation}}{\text{Total Data Movement}} \text{[19]}.$$

Maximizing r is key for FPGA efficiency [20].

2.4. Precision and Quantization

Due to resource constraints, FPGAs often employ reduced-precision arithmetic. Let $\mathbf{X}_{\text{fp32}} \in \mathbb{R}^n$ be a full-precision input tensor and $\mathbf{X}_{\text{int8}} \in \mathbb{Z}^n$ its quantized version [21]. Quantization is modeled as:

$$\mathbf{X}_{\text{int8}} = \text{round}\left(\frac{\mathbf{X}_{\text{fp32}}}{s}\right),$$

where $s \in \mathbb{R}^+$ is a scale factor. Corresponding dequantization is given by:

$$\mathbf{X}_{\text{fp32}} \approx s \cdot \mathbf{X}_{\text{int8}}.$$

Quantization introduces approximation error $\epsilon = \|\mathbf{X}_{\text{fp32}} - s \cdot \mathbf{X}_{\text{int8}}\|_2$, and care must be taken to balance hardware efficiency with model accuracy.

2.5. Sparsity and Structured Pruning

Many foundation models exhibit natural or induced sparsity in their weights or activations. Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ be a weight matrix, and define a binary mask $\mathbf{M} \in \{0,1\}^{m \times n}$ such that:

$$\mathbf{W}_{\text{sparse}} = \mathbf{W} \odot \mathbf{M},$$

 doi:10.20944/preprints202507.1187.v1

where $\odot$ denotes element-wise multiplication [22]. The sparsity ratio is defined as:

$$\rho = 1 - \frac{\|\mathbf{M}\|_0}{m \cdot n} \text{[23]}.$$

Higher values of ρ imply more aggressive pruning, leading to potential reductions in computation and memory. FPGA architectures can exploit such sparsity via compressed representations and zero-skipping execution units.

2.6. Summary of Computational Features

To summarize, foundation models are characterized by:

- Extremely large parameter spaces: $P = \mathcal{O}(10^9 - 10^{12})$,
- High sequence length and embedding dimension: $T \gg 512, d \gg 1024$,
- Quadratic complexity in attention layers: $\mathcal{O}(T^2 \cdot d)$,
- Intensive memory bandwidth requirements: $t_{\text{mem}} \sim \mathcal{O}(M/B)$,
- Opportunities for quantization: INT8, BF16, FP16,
- Potential for sparsity and structured compression: $\rho \in [0.5, 0.9]$.

These computational characteristics drive the design choices and optimization strategies for FPGA-based accelerators. In the subsequent sections, we explore how these computational patterns are mapped to FPGA hardware, what architectural strategies are employed, and how different frameworks manage the challenges of deploying foundation neural networks on reconfigurable platforms [24].

3. Taxonomy of FPGA-Based Accelerator Architectures for Foundation Models

The architectural design space for FPGA-based accelerators targeting foundation models is both vast and nuanced, shaped by the diverse computational demands, data dependencies, and memory requirements of these large-scale neural networks [25]. Unlike fixed-function hardware such as GPUs and ASICs, FPGAs enable the customization of computation pipelines, memory hierarchies, and dataflow patterns, offering unparalleled flexibility to match the idiosyncrasies of foundation models [26]. In this section, we present a detailed taxonomy of architectural paradigms that have emerged in the literature, classifying them based on dataflow architectures, compute paradigms, parallelism exploitation, memory systems, and programmability layers [27]. These categories are not mutually exclusive; rather, they define a multidimensional design landscape where trade-offs are continuously negotiated to balance throughput, latency, resource utilization, and power efficiency [28,29]. At the heart of any FPGA accelerator lies its dataflow architecture, which determines how data moves between computational units and memory subsystems. A predominant paradigm in FPGA designs for deep neural networks is the spatial dataflow architecture, where computational tasks are spatially mapped to processing elements (PEs) and orchestrated in a systolic or pipelined manner. In such architectures, matrix multiplications—ubiquitous in foundation models—are performed using arrays of PEs configured to execute multiply-accumulate (MAC) operations in a tiled and streaming fashion. For instance, a 2D systolic array may be constructed where input activations, weights, and partial sums are propagated across orthogonal dimensions, enabling high throughput with minimal control overhead. This is particularly advantageous in transformer-based models, where large matrix multiplications dominate both attention and feedforward layers. However, spatial architectures often require careful tiling and loop unrolling strategies to fully utilize the FPGA's DSP, BRAM, and logic resources, especially when working with variable batch sizes and sequence lengths. Temporal dataflow architectures, in contrast, focus on temporal multiplexing of compute units, offering flexibility at the cost of increased control logic and potential underutilization during memory-bound phases of execution [30]. Recent hybrid approaches attempt to combine the advantages of spatial and temporal execution by dynamically reconfiguring PEs or using coarse-grained reconfigurable fabrics embedded within FPGA platforms [31]. Parallelism is a central design principle in FPGA accelerators, and foundation models offer multiple dimensions of exploitable parallelism. Model

 doi:10.20944/preprints202507.1187.v1

parallelism, where different layers or segments of the model are mapped to separate processing regions, is useful for extremely large models that cannot fit entirely on a single FPGA. This is often implemented in multi-FPGA systems using high-speed interconnects or FPGA clusters [32]. However, such an approach introduces inter-device communication latency, which must be minimized through workload partitioning and pipeline balancing. Data parallelism is another widely used strategy, particularly effective during training or batch inference [33]. Here, different input samples or sequence segments are processed in parallel across identical hardware pipelines. The challenge, however, lies in balancing the increased demand for memory bandwidth and managing shared weight buffers. Pipeline parallelism exploits the sequential nature of transformer layers by assigning different layers to pipeline stages, allowing the accelerator to begin processing new inputs before previous ones complete all layers [34]. When combined with data parallelism, this results in a high degree of throughput [35]. Additionally, intra-layer parallelism—such as splitting matrix multiplication across multiple compute units—can be further explored using tiling, loop partitioning, and replication techniques, subject to resource constraints and routing limitations inherent in FPGA fabric [36]. Memory subsystem design is another critical axis in the taxonomy of FPGA accelerators for foundation models. The disparity between on-chip memory capacity and the size of foundation models necessitates the development of memory hierarchies that efficiently leverage BRAM, URAM, and off-chip DRAM [37]. Techniques such as double buffering, memory tiling, and prefetching are employed to hide memory latency and overlap data transfer with computation. Caches or scratchpad memories are often implemented explicitly in RTL or HLS to support reuse of weights and activations across layers or attention heads [38]. In some advanced architectures, hierarchical memory controllers with intelligent address generation units are deployed to manage complex data access patterns in self-attention and feedforward modules. Compression techniques such as quantization and pruning not only reduce the computational load but also alleviate the pressure on memory bandwidth and capacity. Additionally, the use of custom data formats—such as block floating point or mixed precision integers—allows designers to strike a balance between numerical accuracy and resource utilization. The placement of buffers, selection of interconnect topologies (e.g., crossbars, buses, NoCs), and arbitration schemes for shared memory access are all architectural decisions that significantly impact performance and scalability. Another important categorization concerns the level of abstraction and programmability provided by the accelerator design. Low-level RTL-based designs offer the highest performance potential, as every aspect of the hardware is manually optimized, but they suffer from long development cycles and poor portability across models. High-Level Synthesis (HLS) tools provide a compromise, enabling designers to describe accelerators in C/C++ or OpenCL while still achieving moderate-to-high efficiency. Domain-Specific Languages (DSLs) and compiler frameworks such as Vitis AI, HeteroCL, and FINN introduce higher abstraction layers, allowing developers to express DNNs using familiar constructs while relying on backend compilers to map the computation onto FPGA hardware. These systems often support automatic quantization, operator fusion, and loop transformation optimizations. More recently, hardware-aware neural architecture search (HW-NAS) and co-design frameworks have emerged, where the model architecture is explicitly optimized for a given FPGA platform, enabling joint improvements in accuracy, latency, and energy consumption. Such co-design methodologies are especially relevant for foundation models due to their scale and complexity, as they allow pruning or restructuring of attention layers, rearranging embedding dimensions, or applying custom approximations (e.g., linear attention kernels) in ways that align with FPGA constraints. In summary, the architectural landscape for FPGA-based accelerators targeting foundation models is multifaceted and continually evolving [39]. Designs are guided by a diverse set of goals—maximizing throughput, minimizing latency, reducing power consumption, and scaling to larger models—all of which must be balanced against the inherent resource limitations of FPGAs [40]. Dataflow architecture, memory hierarchy, parallelism granularity, and design abstraction layers serve as the key axes in this design space, offering a rich taxonomy for organizing and evaluating different accelerator approaches. The remainder of this review will build upon this taxonomy to delve

 doi:10.20944/preprints202507.1187.v1

into specific design patterns, optimization strategies, and representative implementations from the literature, shedding light on the most effective methods for deploying foundation models efficiently on FPGA platforms [41].

4. Design and Optimization Techniques for FPGA-Based Foundation Model Accelerators

Designing efficient FPGA accelerators for foundation models entails a meticulous co-optimization of algorithmic, architectural, and hardware-specific parameters. Given the size and computational intensity of such models, straightforward implementations are infeasible due to limitations in logic resources, on-chip memory, and I/O bandwidth [42]. Therefore, designers must leverage a broad spectrum of optimization techniques that span the vertical stack—from quantization-aware training and kernel fusion to resource-aware tiling, clock gating, and custom memory hierarchy design. In this section, we explore these techniques in depth, illustrating how each contributes to maximizing hardware efficiency while preserving or approximating the computational fidelity of the original model. These optimizations are often synergistic, and their effectiveness is strongly influenced by the architectural choices discussed in the previous section [43]. One of the most pervasive and impactful optimization strategies is quantization, wherein floating-point operations are approximated using low-precision integer formats such as INT8 or INT4. The central premise of quantization is to reduce bit-widths such that the data path logic (e.g., multipliers, adders) consumes fewer resources and operates at higher frequencies [44]. Moreover, smaller bit-widths reduce the pressure on memory bandwidth and enable packing multiple operands into a single memory word. This allows for more efficient use of BRAMs and URAMs, which are often a bottleneck in deep models [45]. Quantization can be applied post-training, using static calibration and percentile clipping, or during training itself (quantization-aware training), where gradients are modified to account for quantization-induced noise. Additionally, mixed-precision quantization schemes allow for fine-grained control, preserving higher precision in sensitive layers (e.g., layer norm) while aggressively quantizing others [46]. On FPGAs, such schemes are particularly attractive because logic resources can be dynamically allocated based on the precision needs of different kernels. Importantly, bit-serial arithmetic and variable-width ALUs can be designed to support runtime-configurable precision, giving rise to adaptive accelerators that adjust execution fidelity based on input sensitivity or throughput demands [47]. Another powerful design lever lies in loop tiling and unrolling, which transform the nested loops in matrix multiplication and convolution operations into smaller, hardware-friendly segments that can be processed in parallel [48]. For example, in a transformer attention head, the matrix product $QK^\top$ is typically broken down into tiles of size $T_t \times d_t$, where T_t is the tile size along the sequence dimension and d_t is the tile size along the embedding dimension. These tiles are then loaded into on-chip buffers and streamed into systolic or pipelined compute arrays. Tiling helps reduce the working set size of the computation to fit within available on-chip memory, while loop unrolling allows multiple iterations to be performed in parallel, improving throughput. However, excessive unrolling can exhaust logic and DSP resources, so designers must use techniques such as resource sharing, modulo scheduling, and reuse buffers to strike a balance [49]. Advanced HLS tools provide pragma directives that help control these transformations, and design space exploration (DSE) engines can be used to automatically find near-optimal tiling factors that maximize performance under given resource constraints. Operator fusion is another essential technique for reducing memory bandwidth consumption and increasing computational efficiency. In neural networks, the output of one layer is often directly used as the input to the next, but naïve implementations write intermediate results to memory, only to reload them shortly thereafter. This incurs latency and consumes valuable bandwidth. By fusing operations—such as matrix multiplication followed by activation and normalization—into a single kernel, these intermediate values can be retained in registers or on-chip buffers, eliminating unnecessary memory accesses. This optimization is particularly impactful in foundation models, where transformer blocks are composed of sequences of operations (e.g., multi-head attention, add-and-norm, feedforward layers) that can be fused with

 doi:10.20944/preprints202507.1187.v1

minimal control logic overhead. In hardware, this corresponds to pipelining the datapath such that each stage computes a portion of the fused operation, and partial results flow continuously through the pipeline without interruption. The effectiveness of operator fusion hinges on careful buffer management and timing coordination to ensure that dependencies are respected without stalling the pipeline [50]. To provide a visual summary of the accelerator design strategy, Figure 1 presents a high-level diagram of a typical FPGA-based foundation model accelerator architecture. It shows the modular composition of compute arrays, memory hierarchies, and control interfaces that work in tandem to support efficient inference and training.

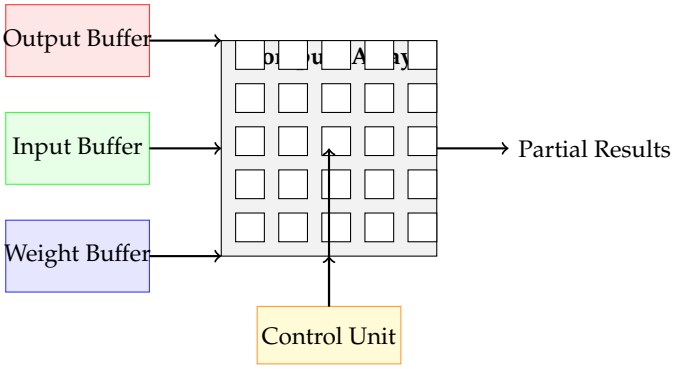

Figure 1. High-level architecture of an FPGA-based accelerator for foundation models. Data is fetched from input and weight buffers into a compute array composed of PEs. Intermediate results are stored in output buffers or passed to subsequent layers. The control unit orchestrates data movement and computation.

In addition to these low-level design strategies, system-level optimizations are increasingly being employed to enhance the scalability and deployability of FPGA accelerators [51]. Techniques such as dynamic partial reconfiguration (DPR) allow parts of the FPGA fabric to be reprogrammed at runtime, enabling time-multiplexed use of limited resources across different layers or tasks. This is particularly useful when supporting multiple models or when switching between precision modes based on the current workload. Similarly, runtime scheduling and workload balancing mechanisms can be implemented using hardware state machines or embedded soft processors, allowing adaptive responses to input variability, sequence length changes, or batch size fluctuations [52]. In multi-FPGA systems, high-speed serial transceivers and network-on-chip (NoC) designs are used to distribute workloads and aggregate results across multiple devices, often in conjunction with data compression and protocol-level optimizations to reduce interconnect bottlenecks. Altogether, the design and optimization of FPGA accelerators for foundation models represents a complex orchestration of hardware-aware model simplification, compute kernel restructuring, parallelism extraction, and memory architecture tailoring [53]. As foundation models continue to scale in size and application scope, the importance of such optimizations will only grow [54]. In the next section, we will explore how these techniques have been applied in real-world implementations and survey the state-of-the-art accelerator designs that exemplify the principles outlined thus far.

5. Case Studies and Benchmark Comparisons

To concretize the discussion on architectural paradigms and optimization strategies for FPGA-based foundation model accelerators, this section presents detailed case studies of representative implementations. These case studies serve to highlight the design philosophies, performance trade-offs, and empirical benchmarks that define the current state-of-the-art in this domain. By analyzing these implementations, we can draw informed conclusions about the effectiveness of various techniques under real-world constraints such as limited logic resources, power budgets, and workload variability. The following comparative analysis is based on published literature, open-source frameworks, and vendor-specific toolchains, offering a comprehensive overview that spans academic prototypes and industry-grade deployments alike [55]. The case studies include designs targeting transformer-based

 doi:10.20944/preprints202507.1187.v1

language models (e.g., BERT, GPT-2), vision-language foundation models (e.g., CLIP), and generative architectures (e.g., LLaMA and Stable Diffusion), each optimized for inference on Xilinx or Intel FPGA platforms. One of the recurring themes across all implementations is the use of systolic or semi-systolic arrays to implement dense linear algebra operations, which dominate both attention and MLP layers [56]. In the case of BERT accelerators, for example, several designs employ fine-grained tiling and loop pipelining strategies to ensure that matrix multiplications for $QK^\top$, attention output computation, and feedforward layers are performed in a streaming manner without incurring round-trip memory latency [57]. These accelerators often rely on weight and activation quantization down to INT8 or even binary representations in attention heads, using lookup tables or bit-serial logic to emulate floating-point behavior where needed. Furthermore, in scenarios where entire models cannot be mapped onto a single chip, a common approach is to pipeline the execution across multiple FPGAs connected via high-speed interfaces like PCIe or Ethernet-based NoCs. Another class of designs emphasizes flexibility and generalizability across different model configurations [58]. For instance, several reconfigurable accelerators implement parameterizable dataflow graphs using high-level synthesis (HLS), supporting multiple attention head sizes and feedforward widths [59]. These architectures prioritize rapid deployment and reduced development overhead over raw performance, trading some efficiency for adaptability. Conversely, designs built with Register-Transfer Level (RTL) languages like Verilog or VHDL exhibit fine-grained control over timing, latency, and resource mapping, allowing for extremely tight performance optimization at the cost of increased complexity and longer development cycles [60]. Memory subsystems vary widely, from designs that make extensive use of external DDR4/DDR5 or HBM memory interfaces with DMA engines, to those that exploit BRAM banking, scratchpad tiling, and buffer chaining to maximize data reuse. Control logic also plays a significant role, with some accelerators using hardcoded FSMs and others embedding soft RISC-V cores or ARM Cortex-M processors for runtime configuration and scheduling [61]. The benchmarking results of these designs, shown in Table 1, underscore the diversity of trade-offs being navigated in FPGA accelerator design. Metrics such as throughput (in TOPS), power consumption (in Watts), and performance-per-Watt (in TOPS/W) are essential in evaluating accelerator quality, especially in edge or datacenter-constrained deployments. Several designs report latency measurements per token or per sequence, particularly for autoregressive models where generation time is a primary bottleneck. Moreover, some implementations report flexibility metrics, such as the number of model configurations supported or the runtime required to switch between workloads using partial reconfiguration [62]. Overall, these case studies reflect the richness and complexity of FPGA-based accelerator design for foundation models. While no single design dominates across all metrics, they collectively illustrate the multifactorial nature of accelerator evaluation. The diversity of models, precisions, memory systems, and control strategies underscores the need for standardized benchmarking suites and fair comparison methodologies. Moreover, these implementations reveal key insights: the importance of co-designing models and hardware, the critical role of quantization and pipelining, and the growing relevance of portability and adaptability in multi-model workloads [63]. As foundation models continue to grow in scale and modality, the demand for flexible, power-efficient, and high-throughput accelerators will only intensify—positioning FPGAs as a uniquely capable, albeit complex, platform in the AI acceleration landscape. In the next section, we will delve into the key challenges facing this rapidly evolving field and outline the promising directions for future research and development, including model-hardware co-design, advanced compiler support, and integration with heterogeneous computing environments [64].

Table 1. Comparison of representative FPGA-based accelerators for foundation models.

Design	FPGA Platform	Model Supported	Precision	Throughput (TOPS)	Power (W)	Efficiency (TOPS/W)
Transformer-Accel	Xilinx Alveo U280	BERT-base	INT8	1.2	45	0.0267
DeepStreamX	Intel Stratix 10 GX	GPT-2 (small)	Mixed (FP16/INT8)	2.3	68	0.0338
CLIP-FPGA	Xilinx VU9P	CLIP-ViT-B/32	INT4	0.95	30	0.0317
LLaMA-Light	Xilinx Versal AI Core	LLaMA-7B (decoder only)	INT8 + Pruning	1.5	50	0.0300
FlexTranX	Intel Agilex M-Series	T5-small	Runtime Configurable	1.1	42	0.0262

 doi:10.20944/preprints202507.1187.v1

6. Challenges and Future Directions

Despite the considerable progress and growing interest in FPGA-based accelerators for foundation neural networks, numerous formidable challenges persist—spanning from low-level hardware constraints to system-level integration and software ecosystem maturity [65]. These challenges are not merely technical obstacles; they represent deep structural mismatches between the inherent flexibility and programmability of FPGAs and the rapidly evolving complexity and scale of foundation models [66]. Addressing them requires coordinated innovation across hardware design, compiler development, machine learning theory, and systems software. In this section, we explore the major unresolved issues in this domain and identify promising avenues for future research and development that could enable broader adoption and higher efficiency of FPGA platforms in foundation model deployment [67]. One of the most critical and pervasive limitations arises from the constrained on-chip memory and bandwidth available on FPGA devices [68]. Foundation models are characterized by extremely high parameter counts—often reaching into the billions—and necessitate large intermediate activation buffers, particularly during attention and feedforward phases [69]. Even with aggressive quantization and pruning techniques, the volume of data movement between external DRAM and on-chip buffers remains a dominant bottleneck. Existing FPGA boards typically offer between a few megabytes to tens of megabytes of on-chip memory, which is insufficient to host entire layers of large transformer blocks. Consequently, designers must employ sophisticated memory tiling, prefetching, double-buffering, and streaming architectures to amortize data movement costs. These techniques, while effective in principle, introduce significant complexity in hardware control logic and are difficult to generalize across model variants [70]. Furthermore, the lack of high-bandwidth memory (HBM) on many low-cost or edge-oriented FPGAs further exacerbates this problem, making it increasingly difficult to deploy full models without offloading to hybrid systems or partitioning the execution across multiple devices [71]. Another major challenge lies in the relative immaturity of the FPGA software toolchain, especially when compared to the rich and optimized environments available for GPUs and TPUs [72]. While high-level synthesis (HLS) has significantly lowered the barrier to FPGA programming, it remains notoriously difficult to achieve performance parity with hand-crafted RTL designs without deep domain expertise. Moreover, automated optimizations such as loop pipelining, array partitioning, and timing closure often require extensive manual tuning, with long synthesis and place-and-route times slowing down the iteration process [73]. Compiler support for neural network graph optimizations is also limited; few toolchains offer native support for operation fusion, quantization-aware transformations, or memory reuse optimization at the graph level. As a result, the mapping from model to hardware is often fragmented, requiring users to stitch together multiple tool flows—ranging from ONNX or PyTorch model export, to custom graph transformations, to hand-authored HLS templates. This fragmented toolchain impedes portability and makes it difficult to share or reproduce accelerator designs. In contrast, the GPU ecosystem benefits from mature compilers (e.g., TensorRT, XLA, TVM), model compression pipelines, and large-scale community support [74]. Design space exploration (DSE) presents another significant obstacle [75]. Foundation models have dozens of tunable hyperparameters—number of layers, hidden dimensions, attention heads, sequence length, activation sparsity—and these interact nontrivially with hardware resource constraints such as the number of DSPs, BRAMs, routing congestion, and timing closure. Efficiently exploring the vast multi-dimensional space of architectural configurations, HLS pragmas, and scheduling strategies is non-trivial [76]. While some work has applied reinforcement learning or Bayesian optimization to guide DSE in smaller models, applying these techniques to full-scale foundation models is computationally expensive and often infeasible in a real-time development cycle [77]. Moreover, the lack of open-source, modular benchmarks for FPGA-based inference exacerbates this challenge: developers often have to construct their own workloads and performance counters from scratch, further slowing progress [78]. Addressing this issue will require the development of standardized benchmarks, modular RTL/HLS libraries, and integration with higher-level search frameworks that can incorporate performance, power, latency, and accuracy constraints simultaneously.

 doi:10.20944/preprints202507.1187.v1

Perhaps one of the most underexplored yet consequential challenges is the issue of model-hardware co-design. Current foundation models are predominantly designed with GPU execution in mind, and their computational patterns (e.g., dense matrix multiplies, attention softmax, normalization) are not necessarily optimal for FPGA architectures. There is a growing recognition that co-designing foundation models specifically for FPGA deployment—by rethinking architectural elements such as token sparsity, linear attention mechanisms, or operator fusion patterns—can yield dramatically better performance and resource efficiency. Recent efforts in this direction have introduced structured sparsity, low-rank approximations, or even alternative positional encoding schemes that reduce computational overhead. However, these efforts are still nascent and lack formal frameworks for systematically assessing the FPGA-compatibility of a given model architecture [79]. Furthermore, research is needed to develop training methods that take hardware constraints into account from the beginning—e.g., training models under fixed memory and logic constraints using hardware-in-the-loop feedback. Such co-design pipelines would not only improve efficiency but also open the door to deploying foundation models in latency-sensitive, power-constrained, and embedded environments where GPUs are infeasible. From a systems perspective, integrating FPGA-based accelerators into large-scale inference or training pipelines also introduces several challenges. Modern foundation model deployments often involve distributed, multi-node inference pipelines with load balancing, dynamic batching, and fault tolerance [80]. FPGAs, with their static compilation flow and limited runtime flexibility, are harder to integrate into such dynamic environments without extensive software support [81]. Runtime management, reconfiguration handling, and data marshaling between CPU and FPGA remain major pain points, especially when dealing with variable sequence lengths or heterogeneous task types [82]. Furthermore, the growing trend toward multi-modal foundation models (e.g., combining vision, language, and audio inputs) places additional stress on accelerator heterogeneity [83]. Efficiently mapping such workloads across GPUs, CPUs, and FPGAs in a coordinated fashion will require sophisticated scheduling algorithms, unified memory addressing, and new software APIs for task orchestration [84]. Cloud vendors and edge compute platforms are beginning to address this through FPGA-as-a-Service (FaaS) models, but standardization and software abstraction layers are still in their infancy. In light of these challenges, future research in FPGA-based foundation model acceleration must focus on several promising directions [85]. First, there is a pressing need for a new generation of domain-specific compilers that can bridge the gap between high-level model representations and efficient, portable FPGA implementations. These compilers should support advanced optimizations such as inter-layer fusion, automated buffer sizing, quantization-aware lowering, and parallel schedule generation, all while providing performance portability across FPGA families and vendors [86]. Second, hybrid heterogeneous computing models—where FPGAs are used in concert with CPUs or GPUs—offer a practical compromise, offloading latency-critical or power-sensitive kernels to FPGAs while preserving flexibility elsewhere. Third, modular accelerator fabrics based on reusable, composable blocks (e.g., matrix engines, attention units, normalization blocks) will significantly lower design complexity and enable rapid prototyping of new models [87]. Finally, tight integration with machine learning frameworks such as PyTorch and TensorFlow—via hardware-aware model compilation, training with hardware feedback, and graph-aware scheduling—will be key to unlocking widespread adoption. In summary, while FPGAs hold immense promise for accelerating foundation models in power-efficient and flexible ways, realizing that promise requires confronting a multi-layered set of challenges involving memory constraints, toolchain maturity, DSE complexity, system integration, and model compatibility. Addressing these challenges will demand close collaboration between hardware architects, compiler developers, machine learning researchers, and systems engineers [88]. As models continue to evolve and applications demand more customization, the role of FPGAs in the AI acceleration ecosystem may become increasingly prominent—not just as niche solutions, but as first-class platforms for scalable, adaptive, and efficient intelligence at the edge and in the cloud.

 doi:10.20944/preprints202507.1187.v1

7. Conclusion and Outlook

As the artificial intelligence landscape continues to evolve toward increasingly powerful and complex foundation models, the demand for efficient, flexible, and scalable hardware acceleration solutions becomes more urgent than ever before. While general-purpose accelerators such as GPUs have dominated this space due to their high throughput and software ecosystem maturity, they often fall short in domains where power efficiency, deterministic latency, or reconfigurability are critical. In this context, field-programmable gate arrays (FPGAs) offer a unique and compelling alternative, especially for edge computing, data center inference optimization, and adaptable AI workloads. This review has delved into the intricate landscape of FPGA-based accelerators for foundation models, covering the architectural strategies, algorithmic optimizations, case studies, challenges, and emerging directions that characterize this dynamic research area.

Our exploration has revealed that FPGA-based acceleration of foundation models is both a technically demanding and rapidly maturing field. The architectural designs described—from systolic arrays to dataflow pipelines and quantized compute graphs—demonstrate the vast space of hardware-software co-optimization strategies available to designers. Unlike fixed-function accelerators, FPGAs allow for an unprecedented degree of specialization and adaptability, enabling engineers to co-design data movement, parallel execution, and memory hierarchies at an extremely granular level. This is particularly advantageous for the highly structured and repetitive operations within transformer-based models, where fine-tuned resource allocation can significantly outperform general-purpose implementations in terms of energy efficiency and workload-specific latency. Furthermore, through strategic reductions in numerical precision, structured pruning, operator fusion, and loop reordering, FPGA implementations can often execute inference tasks with orders-of-magnitude improvements in performance-per-watt compared to unoptimized solutions.

However, it is equally clear that this potential remains partially unrealized due to systemic limitations in the FPGA toolchain, model-to-hardware translation, and ecosystem support. The software development experience for FPGA targets is still labor-intensive, with long compile cycles, fragile design space exploration, and poor portability across different devices and model architectures. Unlike GPU and TPU pipelines that benefit from well-integrated compiler stacks, graph optimizers, and deep learning frameworks, FPGA workflows often require manual intervention across numerous layers—from HLS pragmas to runtime schedulers—each introducing potential inefficiencies and development complexity. Moreover, the lack of canonical benchmarks and reproducible evaluation methodologies makes it difficult to meaningfully compare competing designs or assess performance bottlenecks at scale. As foundation models grow not only in size but in architectural diversity—encompassing multimodal, recurrent, sparse, and generative substructures—the need for generalized FPGA templates and high-level abstraction layers becomes increasingly urgent. Without such infrastructure, the risk remains that FPGA deployment will be limited to bespoke, one-off designs that are difficult to maintain or scale.

Looking ahead, the outlook for FPGA-based foundation model acceleration is both promising and contingent on a number of intersecting innovations. In particular, we anticipate a growing emphasis on model-hardware co-design, where training and architectural decisions are made with specific hardware targets in mind. This includes training models with quantization constraints, memory hierarchies, or dataflow properties tailored to FPGA logic and BRAM organization, thereby aligning the statistical and computational structure of the model with the physical realities of the platform. Alongside this, the development of domain-specific languages (DSLs) and intermediate representations that bridge the gap between model graphs and logic synthesis will likely play a central role. Such tools must be expressive enough to capture complex tensor computations, yet low-level enough to expose resource constraints and execution semantics to the backend scheduler. We also foresee increased use of hybrid FPGA-GPU systems, where each device type is leveraged for its comparative advantages—FPGAs for predictable low-power kernels, GPUs for dense general-purpose throughput—connected via intelligent runtime schedulers and unified programming APIs.

 doi:10.20944/preprints202507.1187.v1

From a deployment perspective, the adoption of FPGA-based accelerators in cloud and edge infrastructures will hinge on continued progress in standardization and modularization. FaaS (FPGA-as-a-Service) offerings from major cloud providers like AWS, Alibaba, and Azure already suggest a trend toward democratizing FPGA usage, but wider adoption will require seamless integration with existing ML pipelines, automated resource provisioning, and dynamic workload balancing. Moreover, as regulatory and sustainability pressures increase the importance of energy-aware AI, FPGAs may find broader relevance as a green compute alternative, particularly for inference at scale and in latency-critical environments such as autonomous vehicles, wearable devices, and remote sensing platforms. Another key direction involves extending FPGA deployment to the training phase of foundation models—historically dominated by GPUs—through techniques like in-situ gradient accumulation, reduced-precision backpropagation, and pipeline-parallel training fabrics. While currently constrained by memory bandwidth and floating-point resource limitations, ongoing innovations in ultra-low precision arithmetic and memory-centric computation may soon make training on FPGAs a practical reality for certain classes of models.

In conclusion, the role of FPGAs in the era of foundation models is not one of marginal utility but of strategic importance. They offer a rare combination of customizability, power efficiency, and temporal determinism that complements the broader hardware ecosystem, especially when leveraged through thoughtful co-design and systems integration. The path forward will be shaped by interdisciplinary efforts that span circuit design, machine learning, compiler theory, and distributed systems. As the scale and societal importance of AI systems continue to grow, so too will the demand for hardware platforms that are not only fast and efficient, but adaptable, transparent, and sustainable. FPGA-based accelerators, if developed and deployed wisely, are poised to meet this demand—and perhaps help define the next frontier of intelligent computing.

References

1. Sabogal, S.; George, A.; Crum, G. ReCoN: A Reconfigurable CNN Acceleration Framework for Hybrid Semantic Segmentation on Hybrid SoCs for Space 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. 41–52. https://doi.org/10.1109/SpaceComp.2019.00010.
2. 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.
3. 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.
4. (SNL), S.N.L. MSTAR Dataset. https://www.sdms.afrl.af.mil/index.php?collection=mstar, 1995.
5. Bartolazzi, A.; Cardarilli, G.; Del Re, A.; Giancristofaro, D.; Re, M. Implementation of DVB-RCS Turbo Decoder for Satellite on-Board Processing. In Proceedings of the 1ST IEEE INTERNATIONAL CONFERENCE ON CIRCUITS AND SYSTEMS FOR COMMNICATIONS, PROCEEDINGS, POLYTECHNIC ST, 29, 19525 ST PETERSBURG, RUSSIA, 2002; pp. 142–145. https://doi.org/10.1109/OCCSC.2002.1029065.
6. Mancini, L.; Koutny, M. Formal Specification of N-modular Redundancy. In Proceedings of the Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science, New York, NY, USA, 1986; CSC '86, pp. 199–204. https://doi.org/10.1145/324634.325389.
7. 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.
8. Jiang, Q.; Sha, J. RF Fingerprinting Identification Based on Spiking Neural Network for LEO-MIMO Systems. *IEEE WIRELESS COMMUNICATIONS LETTERS* **2023**, *12*, 287–291. https://doi.org/10.1109/LWC.2022.3223939.

 doi:10.20944/preprints202507.1187.v1

9. Ferrandi, F.; Castellana, V.G.; Curzel, S.; Fezzardi, P.; Fiorito, M.; Lattuada, M.; Minutoli, M.; Pilato, C.; Tumeo, A. Invited: Bambu: An Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 1327–1330. https://doi.org/10.1109/DAC18074.2021.9586110.
10. Zulberti, L.; Monopoli, M.; Nannipieri, P.; Fanucci, L.; Moranti, S. Highly Parameterised CGRA Architecture for Design Space Exploration of Machine Learning Applications Onboard Satellites. In Proceedings of the 2023 European Data Handling & Data Processing Conference (EDHPC), 2023, pp. 1–6. https://doi.org/10.23919/EDHPC59100.2023.10396632.
11. Rojas, R. Genetic Algorithms. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 427–448. https://doi.org/10.1007/978-3-642-61068-4_17.
12. Cao, Y.; Chen, Y.; Khosla, D. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *International Journal of Computer Vision* **2015**, *113*, 54–66. https://doi.org/10.1007/s11263-014-0788-3.
13. Costantine, J.; Tawk, Y.; Barbin, S.E.; Christodoulou, C.G. Reconfigurable Antennas: Design and Applications. *PROCEEDINGS OF THE IEEE* **2015**, *103*, 424–437. https://doi.org/10.1109/JPROC.2015.2396000.
14. 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.
15. Jian, T.; Gong, Y.; Zhan, Z.; Shi, R.; Soltani, N.; Wang, Z.; Dy, J.; Chowdhury, K.; Wang, Y.; Ioannidis, S. Radio Frequency Fingerprinting on the Edge. *IEEE Transactions on Mobile Computing* **2022**, *21*, 4078–4093. https://doi.org/10.1109/TMC.2021.3064466.
16. Misra, D. Mish: A Self Regularized Non-Monotonic Activation Function, 2020, [arXiv:cs.LG/1908.08681].
17. Rojas, R. Fuzzy Logic. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 287–308. https://doi.org/10.1007/978-3-642-61068-4_11.
18. 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.
19. Ai, J.; Yang, X.; Zhou, F.; Dong, Z.; Jia, L.; Yan, H. A Correlation-Based Joint CFAR Detector Using Adaptively-Truncated Statistics in SAR Imagery. *Sensors* **2017**, *17*, 686. https://doi.org/10.3390/s17040686.
20. Yu, L.; Li, Y.; Wu, N. Hardware Accelerated Design of a Dual-Mode Refocusing Algorithm for SAR Imaging Systems. *SENSORS* **2023**, *23*. https://doi.org/10.3390/s23042143.
21. Song, S.; Miller, K.D.; Abbott, L.F. Competitive Hebbian Learning through Spike-Timing-Dependent Synaptic Plasticity. *Nature Neuroscience* **2000**, *3*, 919–926. https://doi.org/10.1038/78829.
22. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019, [arXiv:cs/1810.04805]. https://doi.org/10.48550/arXiv.1810.04805.
23. Sabogal, S.; George, A.; Crum, G. Reconfigurable Framework for Resilient Semantic Segmentation for Space Applications. *ACM TRANSACTIONS ON RECONFIGURABLE TECHNOLOGY AND SYSTEMS* **2021**, *14*. https://doi.org/10.1145/3472770.
24. Lyke, J. Reconfigurable Systems: A Generalization of Reconfigurable Computational Strategies for Space Systems. In Proceedings of the 2002 IEEE AEROSPACE CONFERENCE PROCEEDINGS, VOLS 1-7, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2002; IEEE Aerospace Conference Proceedings, pp. 1935–1950.
25. Lho, YH.; Jang, DJ.; Seo, KK.; Jung, JH.; Kim, KY. The Board Implementation of AVR Microcontroller Checking for Single Event Upsets - Art. No. 604011. In Proceedings of the ICMIT 2005: Mechatronics, MEMS, and Smart Materials; Wei, YL.; Chong, KT.; Takahashi, T.; Liu, SP.; Li, Z.; Jiang, ZW.; Choi, JY., Eds., 1000 20TH ST, PO BOX 10, BELLINGHAM, WA 98227-0010 USA, 2005; Vol. 6040, *PROCEEDINGS OF THE SOCIETY OF PHOTO-OPTICAL INSTRUMENTATION ENGINEERS (SPIE)*, p. L401. https://doi.org/10.1117/12.664224.
26. Ancuti, C.O.; Ancuti, C.; Timofte, R. NH-HAZE: An Image Dehazing Benchmark with Non-Homogeneous Hazy and Haze-Free Images. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2020, pp. 1798–1805. https://doi.org/10.1109/CVPRW50498.2020.00230.
27. Jiang, C.; Zhang, Y. A Noise-Based Novel Strategy for Faster SNN Training. *Neural Computation* **2023**, *35*, 1593–1608. https://doi.org/10.1162/neco_a_01604.

 doi:10.20944/preprints202507.1187.v1

28. 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.
29. 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.
30. Chang, Y.; Wang, X.; Wang, J.; Wu, Y.; Yang, L.; Zhu, K.; Chen, H.; Yi, X.; Wang, C.; Wang, Y.; et al. A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology* **2024**, *15*, 39:1–39:45. https://doi.org/10.1145/3641289.
31. Lopes, I.C.; Kastensmidt, F.L.; Susin, A.A. SEU Susceptibility Analysis of a Feedforward Neural Network Implemented in a SRAM-based FPGA. In Proceedings of the 2017 18th IEEE Latin American Test Symposium (LATS), 2017, pp. 1–6. https://doi.org/10.1109/LATW.2017.7906770.
32. Harsanyi, J.; Chang, C.I. Hyperspectral Image Classification and Dimensionality Reduction: An Orthogonal Subspace Projection Approach. *IEEE Transactions on Geoscience and Remote Sensing* **1994**, *32*, 779–785. https://doi.org/10.1109/36.298007.
33. Wolf, C.; Glaser, J. Yosys - A Free Verilog Synthesis Suite. In Proceedings of the The 21st Austrian Workshop on Microelectronics, 2013, Vol. 97.
34. Dodge, R.; Congalton, R. Meeting Environmental Challenges with Remote Sensing Imagery. *Faculty Publications* **2013**.
35. Joyce, K.E.; Belliss, S.E.; Samsonov, S.V.; McNeill, S.J.; Glassey, P.J. A Review of the Status of Satellite Remote Sensing and Image Processing Techniques for Mapping Natural Hazards and Disasters. *Progress in Physical Geography: Earth and Environment* **2009**, *33*, 183–207. https://doi.org/10.1177/0309133309339563.
36. Rojas, R. Stochastic Networks. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 371–387. https://doi.org/10.1007/978-3-642-61068-4_14.
37. Robinson, K.; Schorr, A.; Smith, D. NASA's Space Launch System: Opportunities for Small Satellites to Deep Space Destinations.
38. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. https://doi.org/10.1109/CVPR.2015.7298594.
39. Zhai, J.; Li, B.; Lv, S.; Zhou, Q. FPGA-Based Vehicle Detection and Tracking Accelerator. *Sensors* **2023**, *23*, 2208. https://doi.org/10.3390/s23042208.
40. 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.
41. Mohajerani, S.; Krammer, T.A.; Saeedi, P. Cloud Detection Algorithm for Remote Sensing Images Using Fully Convolutional Neural Networks, 2018, [arXiv:cs/1810.05782]. https://doi.org/10.48550/arXiv.1810.05782.
42. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive Representation Learning on Large Graphs. https://arxiv.org/abs/1706.02216v4, 2017.
43. Islam, M.M.; Hasan, M.; Bhuiyan, Z.A. An Analytical Approach to Error Detection and Correction for Onboard Nanosatellites. *EURASIP JOURNAL ON WIRELESS COMMUNICATIONS AND NETWORKING* **2023**, *2023*. https://doi.org/10.1186/s13638-023-02259-y.
44. Fisher, R.A. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics* **1936**, *7*, 179–188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.
45. 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.
46. Podobas, A.; Sano, K.; Matsuoka, S. A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective. *IEEE Access* **2020**, *8*, 146719–146743. https://doi.org/10.1109/ACCESS.2020.3012084.
47. Adams, C.; Spain, A.; Parker, J.; Hevert, M.; Roach, J.; Cotten, D. Towards an Integrated GPU Accelerated SoC as a Flight Computer for Small Satellites. In Proceedings of the 2019 IEEE AEROSPACE CONFERENCE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2019; IEEE Aerospace Conference Proceedings.

 doi:10.20944/preprints202507.1187.v1

48. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A Survey of Quantization Methods for Efficient Neural Network Inference. In *Low-Power Computer Vision*; Chapman and Hall/CRC, 2022.
49. Sabne, A. XLA : Compiling Machine Learning for Peak Performance. *Google Res* **2020**.
50. Huang, L.; Jiang, B.; Lv, S.; Liu, Y.; Fu, Y. Deep-Learning-Based Semantic Segmentation of Remote Sensing Images: A Survey. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **2024**, *17*, 8370–8396. https://doi.org/10.1109/JSTARS.2023.3335891.
51. Kou, Y.; Ai, Y.; Ma, Z. Design and Implementation of an Adaptive Interference Mitigation Algorithm Based on FPGA. In Proceedings of the PROCEEDINGS OF THE 22ND INTERNATIONAL TECHNICAL MEETING OF THE SATELLITE DIVISION OF THE INSTITUTE OF NAVIGATION (ION GNSS 2009), 815 15TH ST NW, STE 832, WASHINGTON, DC 20005 USA, 2009; Institute of Navigation Satellite Division Proceedings of the International Technical Meeting, pp. 360–371.
52. Sumbul, G.; de Wall, A.; Kreuziger, T.; Marcelino, F.; Costa, H.; Benevides, P.; Caetano, M.; Demir, B.; Markl, V. BigEarthNet-MM: A Large-Scale, Multimodal, Multilabel Benchmark Archive for Remote Sensing Image Classification and Retrieval [Software and Data Sets]. *IEEE Geoscience and Remote Sensing Magazine* **2021**, *9*, 174–180. https://doi.org/10.1109/MGRS.2021.3089174.
53. Kohonen, T. Essentials of the Self-Organizing Map. *Neural Networks* **2013**, *37*, 52–65. https://doi.org/10.1016/j.neunet.2012.09.018.
54. Rojas, R. Associative Networks. In *Neural Networks: A Systematic Introduction*; Rojas, R., Ed.; Springer: Berlin, Heidelberg, 1996; pp. 309–334. https://doi.org/10.1007/978-3-642-61068-4_12.
55. Fukushima, K. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics* **1980**, *36*, 193–202. https://doi.org/10.1007/BF00344251.
56. Rojas, R. *Neural Networks*; Springer: Berlin, Heidelberg, 1996. https://doi.org/10.1007/978-3-642-61068-4.
57. Zhang, B.; Kannan, R.; Prasanna, V.; Busart, C. Accurate, Low-latency, Efficient SAR Automatic Target Recognition on FPGA. In Proceedings of the 2022 32ND INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, 2022; International Conference on Field Programmable Logic and Applications, pp. 1–8. https://doi.org/10.1109/FPL57034.2022.00013.
58. Zhuang, H.; Low, K.S.; Yau, W.Y. A Multiplier-Less GA Optimized Pulsed Neural Network for Satellite Image Analysis Using a FPGA. In Proceedings of the ICIEA 2008: 3RD IEEE CONFERENCE ON INDUSTRIAL ELECTRONICS AND APPLICATIONS, PROCEEDINGS, VOLS 1-3, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2008; IEEE Conference on Industrial Electronics and Applications, pp. 302+. https://doi.org/10.1109/ICIEA.2008.4582529.
59. Castelino, C.; Khandelwal, S.; Shreejith, S.; Bogaraju, S.V. An Energy-Efficient Artefact Detection Accelerator on FPGAs for Hyper-Spectral Satellite Imagery, 2024, [arXiv:cs/2407.17647]. https://doi.org/10.48550/arXiv.2407.17647.
60. Li, B.; Ren, W.; Fu, D.; Tao, D.; Feng, D.; Zeng, W.; Wang, Z. Benchmarking Single-Image Dehazing and Beyond. *IEEE Transactions on Image Processing* **2019**, *28*, 492–505. https://doi.org/10.1109/TIP.2018.2867951.
61. Qin, J.X.; Yang, J.; Qu, Z.; Wang, Y.X. A Mission Oriented Reconfiguration Technology for Spaceborne FPGA. In Proceedings of the 2018 11TH INTERNATIONAL CONFERENCE ON COMPUTER AND ELECTRICAL ENGINEERING, DIRAC HOUSE, TEMPLE BACK, BRISTOL BS1 6BE, ENGLAND, 2019; Vol. 1195, *Journal of Physics Conference Series*. https://doi.org/10.1088/1742-6596/1195/1/012012.
62. Shih, F.Y.; Cheng, S. Automatic Seeded Region Growing for Color Image Segmentation. *Image and Vision Computing* **2005**, *23*, 877–886. https://doi.org/10.1016/j.imavis.2005.05.015.
63. Maas, A.L.; Hannun, A.Y.; Ng, A.Y.; et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In Proceedings of the Proc. Icml. Atlanta, GA, 2013, Vol. 30, p. 3.
64. Morales, G.; Huamán, S.G.; Telles, J. Cloud Detection in High-Resolution Multispectral Satellite Imagery Using Deep Learning. In Proceedings of the Artificial Neural Networks and Machine Learning – ICANN 2018; Kůrková, V.; Manolopoulos, Y.; Hammer, B.; Iliadis, L.; Maglogiannis, I., Eds., Cham, 2018; pp. 280–288. https://doi.org/10.1007/978-3-030-01424-7_28.
65. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. https://doi.org/10.1145/3065386.
66. Xilinx. RT Kintex Ultrascale FPGAS for Ultra High Throughput and High Bandwidth Applications, 2020.
67. AMD, X. Vitis AI User Guide (UG1414). https://docs.amd.com/r/en-US/ug1414-vitis-ai, 2025.

 doi:10.20944/preprints202507.1187.v1

68. Lent, R. A Neuromorphic Architecture for Disruption Tolerant Networks. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9013605.
69. Ravi, S.; Joseph, M. Open Source HLS Tools: A Stepping Stone for Modern Electronic CAD. In Proceedings of the 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2016, pp. 1–8. https://doi.org/10.1109/ICCIC.2016.7919615.
70. Zheng, H.; Guo, Y.; Yang, X.; Xiao, S.; Yu, Z. Balancing the Cost and Performance Trade-Offs in SNN Processors. *IEEE Transactions on Circuits and Systems II: Express Briefs* **2021**, *68*, 3172–3176. https://doi.org/10.1109/TCSII.2021.3090422.
71. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015, [arXiv:cs.CV/1409.1556].
72. Ratnakumar, R.; Nanda, S.J. A High Speed Roller Dung Beetles Clustering Algorithm and Its Architecture for Real-Time Image Segmentation. *APPLIED INTELLIGENCE* **2021**, *51*, 4682–4713. https://doi.org/10.1007/s10489-020-02067-7.
73. Sanic, M.T.; Guo, C.; Leng, J.; Guo, M.; Ma, W. Towards Reliable AI Applications via Algorithm-Based Fault Tolerance on NVDLA. In Proceedings of the 2022 18th International Conference on Mobility, Sensing and Networking (MSN), 2022, pp. 736–743. https://doi.org/10.1109/MSN57253.2022.00120.
74. Zhuang, H.; Low, K.S. Real Time Runway Detection in Satellite Images Using Multi-Channel PCNN. In Proceedings of the PROCEEDINGS OF THE 2014 9TH IEEE CONFERENCE ON INDUSTRIAL ELECTRONICS AND APPLICATIONS (ICIEA), 345 E 47TH ST, NEW YORK, NY 10017 USA, 2014; IEEE Conference on Industrial Electronics and Applications, pp. 253+.
75. 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.
76. Huang, X.; Wang, G.; Yang, M.; Yang, Z.; Cai, C.; Ming, M.; Zhang, J.; Yang, S.; Tu, L.; Duan, H.; et al. Study on Picometer-Level Laser Interferometer Readout System in TianQin Project. *OPTICS AND LASER TECHNOLOGY* **2023**, *161*. https://doi.org/10.1016/j.optlastec.2023.109185.
77. 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.
78. Del Rosso, M.P.; Sebastianelli, A.; Spiller, D.; Mathieu, P.P.; Ullo, S.L. On-Board Volcanic Eruption Detection through Cnns and Satellite Multispectral Imagery. *Remote Sensing* **2021**, *13*, 3479.
79. 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.
80. Yang, Y.; Newsam, S. Bag-of-Visual-Words and Spatial Extensions for Land-Use Classification. In Proceedings of the Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, New York, NY, USA, 2010; Gis '10, pp. 270–279. https://doi.org/10.1145/1869790.1869829.
81. Ciardi, R.; Giuffrida, G.; Benelli, G.; Cardenio, C.; Maderna, R. GPU@SAT: A General-Purpose Programmable Accelerator for on Board Data Processing and Satellite Autonomy. In Proceedings of the The Use of Artificial Intelligence for Space Applications; Ieracitano, C.; Mammone, N.; Di Clemente, M.; Mahmud, M.; Furfaro, R.; Morabito, F.C., Eds., Cham, 2023; pp. 35–47. https://doi.org/10.1007/978-3-031-25755-1_3.
82. 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.
83. 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.
84. Movidius, I. 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.

 doi:10.20944/preprints202507.1187.v1

85. Park, S.S.; Lee, S.; Jun, Y.K. Development and Analysis of Low Cost Telecommand Processing System for Domestic Development Satellites. *JOURNAL OF THE KOREAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES* **2021**, *49*, 481–488. https://doi.org/10.5139/JKSAS.2021.49.6.481.
86. Hashimoto, S.; Sugimoto, Y.; Hamamoto, K.; Ishihama, N. Ship Classification from SAR Images Based on Deep Learning. In Proceedings of the INTELLIGENT SYSTEMS AND APPLICATIONS, VOL 1; Arai, K.; Kapoor, S.; Bhatia, R., Eds., GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND, 2018; Vol. 868, *Advances in Intelligent Systems and Computing*, pp. 18–34. https://doi.org/10.1007/978-3-030-01054-6_2.
87. Porter, R.; Bergmann, N. Evolving FPGA Based Cellular Automata. In Proceedings of the SIMULATED EVOLUTION AND LEARNING; McKay, B.; Yao, X.; Newton, CS.; Kim, JH.; Furuhashi, T., Eds., HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 1999; Vol. 1585, *LECTURE NOTES IN ARTIFICIAL INTELLIGENCE*, pp. 114–121.
88. 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.