Università degli Studi Roma Tre

Dipartimento di Matematica e Fisica
Scuola di Dottorato in Fisica

Ph.D. Thesis

Innovative Machine Learning in Versal and FPGA-based for Data Acquisition Systems in Radiation Environments

PhD Candidate:
Dario Vincenzi

Supervisor:
Prof. Paolo Branchini

XXXVIII Cycle - Academic Year 2024/2025

Contents

Contents i
List of Figures iv
List of Tables vi
List of Abbreviations vii

1 Introduction **1**
1.1 Context and Motivation 1
1.2 Reliability Challenges in Radiation Environments 3
1.3 Quantization and Performance-Energy Trade-Offs 4
1.4 Research Gap and Contributions 5
1.5 Research Objectives 7
1.6 Performance Evaluation and Benchmarking 7
1.6.1 Quantitative Results and Comparison 7
1.7 Structure of the Thesis 8

2 Theoretical Foundations and State of the Art **10**
2.1 Machine Learning Fundamentals 10
2.1.1 CNNs: Principles and Core Operators 10
2.1.2 Convolutional Layers 12
2.2 FPGAs for AI: Architectures and Design Patterns 18
2.2.1 FPGA Resource Hierarchy and CNN Mapping 18
2.3 Xilinx Versal VCK190 Platform and AIE 31
2.3.1 PS 31
2.3.2 PL 33
2.3.3 AIE Array 33
2.3.4 NoC Interconnect and AXI Interfaces 34
2.3.5 Comparison with Previous-Generation Xilinx Architectures . . 36
2.3.6 Overview of Vitis and Vitis AI Toolchains 37
2.3.7 From Trained CNN Model to Versal Deployment 37
2.3.8 Optimization and Iteration for CNN Inference 39
2.3.9 Profiling and Resource Mapping 40

2.3.10 Radiation Environment and Fault Models 43
2.3.11 Mitigation Techniques (TMR, ECC, Scrubbing) 45
2.3.12 FIT and Validation Methodologies 47

3 Methodology and Development of the Innovative DAQ System 50
3.1 Dataset and Pre-processing 50
3.2 Baseline and Variants (FP32 vs Quantized, AIE vs PL) 51
3.3 Metrics and Evaluation Protocols 54
3.4 Experimental Methodology and Benchmarking 57
3.4.1 Baseline and Variants 59
3.4.2 Metrics and Evaluation Protocols 64

4 Experimental Setup 71
4.1 Hardware, Interfaces and Instrumentation 71
4.2 Software/Firmware and System Images 72
4.3 Testing Procedures and Reproducibility 74

5 Results and Discussion 78
5.1 Accuracy, IoU and Error Analysis 79
5.2 Resource Utilization and Resilience under Fault 80
5.2.1 Resilience Considerations 81
5.3 Architecture Scalability and Extended Potential 82
5.4 Custom Solutions Adopted for the Processing of the First Layer . . . 83
5.4.1 Multi-Stage Pipeline: Division of Computation 83
5.4.2 Resource Distribution and Usage 84
5.4.3 Comparison with GPU: Architectural Advantages 84

6 Case Study: High-Frequency DAQ on Spartan 7 for ELI Beamlines 86
6.1 DAQ Architecture and Performance 86
6.1.1 Data Acquisition and Hardware Metadata Tagging 88
6.1.2 Ethernet Transmission and Hardware TCP/IP Stack 89
6.1.3 Performance Results and Resource Utilization 89
6.1.4 Data Packet Formatting and Transmission Logic 89
6.1.5 Experimental Performance and Resource Utilization 90
6.2 Integration of Genetic Algorithms on FPGA: Future Perspectives . . 91
6.2.1 Genetic Algorithms Overview 92
6.2.2 Hardware Architecture and Preliminary Resource Analysis . . 92
6.2.3 Edge Intelligence and Hardware Evolutionary Adaptation . . . 92
6.2.4 Design Challenges and Verification 94

7 Conclusions and Future Developments **95**
7.1 Summary of Contributions 95
7.2 Towards Intelligent Radiation-Hardened Systems 96

References **97**

List of Figures

2.1 General architecture of a CNN for image classification 11
2.2 Comparison of input imagery and regression map 17
2.3 Hybrid CNN–LSTM model for regression 17
2.4 Internal structure of an FPGA Logic Slice 19
2.5 Memory hierarchy in the Versal architecture 20
2.6 Heterogeneous workload partitioning in Versal 21
2.7 Comparison of FPGA resources across generations 23
2.8 FPGA Parallelism Models for CNN Acceleration 24
2.9 Comparison of Centralized and Dataflow Control Models 26
2.10 Double buffering (Ping-Pong) mechanism 27
2.11 Physical Floorplanning for Timing Closure 28
2.12 Teacher–student knowledge distillation pipeline 29
2.13 Trade-off between model accuracy and resource efficiency 30
2.14 End-to-end deployment pipeline for Versal ACAP 31
2.15 AMD/Xilinx Versal VCK190 evaluation board 32
2.16 Architectural block diagram of the Versal ACAP 32
2.17 Versal NoC architecture: AXI switch grid and components. 35
2.18 High-level block diagram of the Versal VCK190 architecture 36
2.19 End-to-end Vitis AI development flow 39
2.20 Logical Array View of AIE mapping on the Versal VCK190. 42

3.1 Example input image after normalization 51
3.2 Pre-processing pipeline for network inputs 58
3.3 Conceptual overview of CNN inference deployment 60

4.1 Heterogeneous architecture of the acceleration system on Versal . . . 75

5.1 Heterogeneous AIE + DSP incremental pipeline 84

6.1 Experimental setup for data acquisition 87
6.2 Close-up of Spartan-7 and analog front-end 90
6.3 SPI and ADC synchronization waveforms 91

6.4 GA floorplanning and placement on Spartan-7 93
6.5 Synthesis resource utilization for GA 93

List of Tables

1.1 CNN Inference Performance: Versal vs. Baselines 8

2.1 Comparison of FPGA resources across generations 23

3.1 Performance and accuracy comparison of CNN variants 54
3.2 Hardware resource utilization on Versal 56
3.3 Comparison of CNN inference variants across platforms 63
3.4 Resource utilization and performance: AIE vs. PL 68

5.1 Accuracy and performance results under different conditions. 79
5.2 Performance comparison between FPGA and GPU 80
5.3 Usage of the main hardware resources in the experimental configuration. 82

7.1 Final Performance Benchmark: Versal Accelerator vs. Baseline. . . . 96

List of Abbreviations

ACAP	Adaptive Compute Acceleration Platform
AIE	AI Engine
APU	Application Processing Unit (Arm Cortex-A72)
AVF	Architectural Vulnerability Factor
AXI	Advanced eXtensible Interface
BRAM	Block RAM
CCC	Configuration Collector and Corrector
CDC	Clock Domain Crossing
CLB	Configurable Logic Block
CMT	Clock Management Tile
CNN	Convolutional Neural Network
CRAM	Configuration RAM
DDR	Double Data Rate (External Memory)
DPU	Deep Learning Processing Unit
DSP	Digital Signal Processing
ECC	Error Correction Code
EDAC	Error Detection and Correction
FIT	Failure-In-Time
FPGA	Field Programmable Gate Array
GELU	Gaussian Error Linear Unit
HLS	High-Level Synthesis
ICAP	Internal Configuration Access Port
JTAG	Joint Test Action Group (IEEE 1149.1)
MAC	Multiply-Accumulate
MTBF	Mean Time Between Failures
NoC	Network-on-Chip
PL	Programmable Logic
PLL	Phase-Locked Loop
PMC	Platform Management Controller
PS	Processing System
ReLU	Rectified Linear Unit
RPU	Real-time Processing Unit (Arm Cortex-R5F)
SEE	Single-Event Effect
SEFI	Single-Event Functional Interrupt
SEL	Single-Event Latch-up

SET	Single-Event Transient
SEU	Single-Event Upset
SIMD	Single Instruction, Multiple Data
SWIFI	Software-Implemented Fault Injection
TMR	Triple Modular Redundancy
URAM	UltraRAM
VLIW	Very Long Instruction Word
XRT	Xilinx Runtime

1. Introduction

1.1 Context and Motivation

Convolutional Neural Networks (CNNs) have emerged as the foundational pillar of modern artificial intelligence, establishing state-of-the-art benchmarks in computer vision, real-time signal processing, autonomous systems, and high-energy physics data analysis. This success, however, comes at a significant computational cost. The internal structure of deep CNNs is characterized by a massive volume of Multiply–Accumulate (MAC) operations and intensive memory traffic for feature map transfers. As model complexity grows, these requirements translate into extreme demands for both computational density and memory bandwidth, pushing the limits of current hardware infrastructures.

Traditionally, Graphics Processing Units (GPUs) have represented the gold standard for CNN acceleration, leveraging their massive data-parallel architectures and highly mature software ecosystems. Despite their peak performance, GPUs often face significant challenges in scenarios requiring strictly deterministic latency and extreme energy efficiency. Furthermore, the rising environmental and economic footprint of GPU-based AI infrastructures has sparked a shift toward "Green AI" paradigms. In this context, Field-Programmable Gate Arrays (FPGAs) have emerged as a compelling alternative, offering superior performance-per-watt and the ability to implement application-specific spatial computing architectures [1,2]. Unlike the fixed-instruction-set nature of GPUs, FPGAs enable fine-grained pipelining, customizable data paths, and the use of reduced-precision arithmetic tailored to specific network layers.

Despite these advantages, conventional FPGA fabrics face severe scalability bottlenecks when implementing modern, multi-layered CNNs. Traditional architectures rely on general-purpose Look-Up Table (LUT) logic and coarse Digital Signal Processing (DSP) slices. As networks deepen, these resources encounter routing congestion and memory-wall limitations, often restricting the achievable computational density compared to the dedicated tensor cores found in high-end GPUs.

To bridge this gap, AMD (formerly Xilinx) introduced the Versal Adaptive Compute Acceleration Platform (ACAP), a revolutionary heterogeneous architecture. The Versal platform integrates three distinct compute domains within a single device:

- **Scalar Engines**: Multi-core ARM-based processing systems for high-level control and OS management.
- **Adaptable Engines**: A traditional programmable logic (PL) fabric for custom I/O, hardware-level synchronization, and specialized data orchestration.
- **Intelligent Engines AI Engines (AIEs)**: An array Very Long Instruction Word (VLIW) vector processors. Each AIE tile features dedicated local memory and high-speed streaming interconnects, specifically optimized for the high-density arithmetic required by DSP and AI workloads [3].

This integration enables a tightly coupled hybrid execution model where spatial and vector computing coexist. This architectural synergy allows for the offloading of compute-intensive kernels to the AIE array, while the programmable logic fabric remains available for custom interfacing, real-time data orchestration, and the implementation of hardware-level mitigation circuits. Consequently, this heterogeneous paradigm potentially offers a transformative leap in performance-per-watt compared to traditional DSP-only FPGA architectures [4].

Beyond pure performance, a critical motivation for this research lies in system reliability within hostile environments. Applications in aerospace, nuclear monitoring, and high-intensity laser facilities (such as ELI Beamlines) expose electronic devices to ionizing radiation. These conditions trigger Single-Event Upsets (SEUs), leading to bit-flips in configuration memory that can result in SDC or catastrophic functional failure [5]. While FPGAs are inherently flexible for implementing redundancy, the susceptibility of next-generation heterogeneous platforms like the Versal ACAP remains largely uncharacterized in the academic literature.

Therefore, this thesis is motivated by the need to define a systematic methodology for the deployment of resilient AI. It addresses the Research Gap concerning the optimal workload partitioning between AIEs and PL, aiming to maximize throughput and energy efficiency while ensuring the robustness required for mission-critical Data Acquisition (DAQ) systems in radiation-prone environments.

1.2 Reliability Challenges in Radiation Environments

Beyond pure computational performance, reliability represents a critical concern for electronic systems deployed in aerospace, nuclear medicine, and high-energy physics applications. In these domains, hardware is exposed to a harsh radiative environment characterized by high-energy protons, neutrons, and heavy ions. SRAM-based FPGAs, while offering superior flexibility, are particularly susceptible to radiation-induced effects that can be categorized into two main phenomena:

- **Single-Event Effects (SEEs)**: These are stochastic, individual events caused by a single ionizing particle hitting a sensitive node in the semiconductor. The most frequent is the SEU, which induces a bit-flip in a memory cell (Configuration RAM, Block RAM, or registers). In FPGAs, an SEU in the configuration memory is persistent and can structurally alter the implemented logic function, leading to permanent errors until the next reconfiguration or scrubbing cycle [6].
- **Total Ionizing Dose (TID)**: This represents the cumulative degradation of the device's electrical characteristics (such as threshold voltage shifts and increased leakage current) due to the long-term absorption of ionizing radiation. Over time, TID can lead to functional failure of the entire System-on-Chip (SoC) [5].

Radiation-induced soft errors in advanced sub-micron technologies have been extensively modeled, highlighting how the reduction in feature size and supply voltage increases the susceptibility to low-energy particles [7]. For CNN accelerators, the consequences of an SEU can be classified based on their severity:

1. **Silent Data Corruption (SDC)**: The network continues to operate, but the accuracy is degraded due to corrupted weights or feature maps.
2. **Detected Unrecoverable Interrupts (DUI)**: The system hangs or crashes, requiring a hardware reset or reconfiguration.

To mitigate these risks, several strategies are typically employed. Configuration Scrubbing involves periodically reading the configuration memory and correcting errors using ECC (Error Correction Codes) to prevent the accumulation of upsets. Triple Modular Redundancy (TMR) replicates the logic three times and uses a

majority voter to mask faults, although at the cost of significantly increased area and power consumption [8].

However, the interaction between these traditional mitigation schemes and modern heterogeneous AI accelerators (such as the Versal AIE array) remains insufficiently characterized in the literature. The tight coupling between the VLIW vector processors in the AIE and the programmable logic fabric introduces new failure modes. Fault injection methodologies provide a controlled and cost-effective mechanism to emulate SEU effects without the need for expensive particle accelerators [9]. By deliberately injecting bit-flips into the device's configuration and data memory, it is possible to quantitatively evaluate the cross-section of the design and develop "radiation-aware" hardware-software co-design strategies. This thesis aims to bridge this gap, providing a systematic resilience analysis of the Versal platform under realistic DAQ operating conditions.

1.3 Quantization and Performance-Energy Trade-Offs

Numerical precision represents a fundamental lever in the design of high-performance CNN accelerators, as it directly influences computational cost, memory bandwidth, and overall energy efficiency. Traditional deep learning models are typically developed using 32-bit floating-point (FP32) arithmetic; however, deploying such high-precision models on embedded hardware often leads to significant bottlenecks in terms of power consumption and data movement.

Quantization techniques, such as INT8 inference, have emerged as a primary solution to these challenges. By mapping high-precision weights and activations into lower-bit integer formats, it is possible to achieve a substantial reduction in both arithmetic complexity and memory footprint [10, 11]. Specifically, moving from 32-bit to 8-bit representations allows for:

- A **4× reduction** in the amount of data transferred between the external memory and the processing units.
- A significantly higher computational density, as integer arithmetic units are smaller and more power-efficient than their floating-point counterparts.

While these techniques maintain competitive accuracy in most computer vision tasks, their implementation on next-generation heterogeneous platforms introduces new challenges. In particular, the combined impact of reduced precision, heteroge-

neous mapping (partitioning workloads between the AIE array and the PL, and fault susceptibility within the Versal architecture remains an open research problem.

This thesis investigates how the transition to quantized arithmetic affects not only the throughput and energy consumption of the CNN accelerator but also its robustness in radiation-prone environments. Characterizing this trade-off is essential for developing reliable and efficient Data Acquisition (DAQ) systems where every milliwatt and every microsecond are critical.

1.4 Research Gap and Contributions

The rapid evolution of deep learning has produced an extensive body of literature focusing on specialized hardware acceleration. Previous studies have thoroughly explored:

- **CNN acceleration on traditional platforms**: Significant research has established the performance benchmarks for GPUs and conventional FPGA architectures based on standard DSP slices [1, 2].
- **Quantization methodologies**: Techniques for mapping high-precision models to INT8 or lower bit-widths have been validated for maintaining competitive accuracy while reducing memory footprint [10, 11].
- **Radiation effects in CMOS technologies**: The susceptibility of SRAM-based devices to SEUs and TID has been well-characterized for standard logic fabrics [5, 6].

Despite these advancements, there is a profound lack of comprehensive studies that simultaneously address the multi-dimensional challenges of modern heterogeneous platforms. Specifically, current literature fails to provide a unified analysis of:

1. **Heterogeneous Workload Partitioning**: While vendor white papers highlight the potential of the Versal ACAP [12], independent academic evaluations of optimal partitioning between AIE and PL for CNN inference are still in their infancy.
2. **Energy-Performance Pareto Frontiers**: Detailed performance-per-watt characterizations under real-world Data Acquisition (DAQ) constraints are often overlooked in favor of peak theoretical throughput.
3. **Radiation Resilience of Vector Engines**: There is virtually no public data evaluating how the VLIW-based AIE array responds to radiation-induced faults compared to traditional PL-based accelerators.

4. **Precision-Reliability Trade-offs**: The comparative impact of **FP32 vs. INT8** implementations on fault susceptibility in heterogeneous devices remains an unaddressed research problem.

This thesis addresses this critical gap by providing a **unified performance and reliability analysis framework** for CNN inference on the Versal AI Core platform. By bridging the divide between high-performance computing and radiation hardening, this work offers the following **main contributions**:

- **Design of a Heterogeneous Accelerator**: The implementation of a CNN inference engine that synergistically exploits both the AIE array for vector arithmetic and the PL for high-speed data orchestration.
- **Quantitative Characterization**: A rigorous evaluation of throughput (FPS), deterministic latency, and energy efficiency (FPS/W), establishing new baselines for the VCK190 platform.
- **Precision Analysis**: A systematic comparison between floating-point and quantized implementations, highlighting the trade-offs in resource utilization and accuracy.
- **Fault Injection Campaigns**: The first systematic evaluation of SEU-induced degradation on a heterogeneous ACAP-based AI accelerator, identifying critical failure modes (SDC vs. DUI).
- **Design Guidelines**: The derivation of methodological guidelines for the co-design of resilient, high-performance AI accelerators on next-generation FPGAs.

The main contributions are:

- Design of a heterogeneous CNN accelerator exploiting both AIE and PL.
- Quantitative performance characterization (throughput, latency, energy efficiency).
- Comparative analysis between floating-point and quantized implementations.
- Fault injection campaigns evaluating SEU-induced degradation.
- Design guidelines for resilient heterogeneous FPGA-based AI accelerators.

1.5 Research Objectives

The objectives of this doctoral research are:

1. Develop a heterogeneous CNN inference architecture on Versal VCK190.
2. Optimize implementation using Vivado and Vitis toolchains.
3. Measure throughput, latency, resource utilization, and fps/W.
4. Evaluate resilience through controlled fault injection campaigns.
5. Derive methodological guidelines for performance-per-watt and reliability co-design.

1.6 Performance Evaluation and Benchmarking

To address the requirement for a clear positioning within the state-of-the-art, this section provides a quantitative characterization of the proposed CNN accelerator on the AMD Versal VCK190 platform. The results are benchmarked against two industry-standard baselines: a high-end previous-generation FPGA (Zynq UltraScale+ ZU9EG [13]) and a leading embedded GPU (NVIDIA Jetson AGX Xavier [14]), under comparable workloads and INT8 quantization.

1.6.1 Quantitative Results and Comparison

The performance evaluation of the proposed heterogeneous CNN accelerator on the Versal VCK190 platform demonstrates a transformative shift in embedded AI acceleration. The core performance metrics, extracted from hardware-level profiling and power measurements, are summarized in Table 1.1. The architecture achieves a peak throughput of **15,000 FPS** with an ultra-low deterministic latency of **0.02 ms**, establishing a new benchmark for high-rate Data Acquisition (DAQ) systems.

This quantitative leap is primarily attributed to the synergy between the AIE array and the PL. While traditional FPGA-based accelerators (e.g., Zynq UltraScale+) rely on DSP slices that often encounter routing congestion and memory-wall limitations when pushed to high utilization ($> 90\%$), the proposed heterogeneous partitioning offloads the heavy tensor arithmetic to the AIE vector processors. This results in a $5\times$ throughput increase over previous-generation FPGAs and a massive reduction in latency compared to GPU-based platforms like the NVIDIA Jetson AGX. Unlike GPUs, which achieve peak throughput through massive batching at

Metric	Versal ACAP (Proposed)	Zynq UltraScale+ [13]	NVIDIA Jetson AGX [14]
Architecture	**Heterogeneous (AIE+PL)**	Traditional FPGA (PL)	Embedded GPU
Precision	**INT8 / FP32**	INT8	INT8 / FP16
Throughput (FPS)	**15,000**	∼3,000	∼1,500
Latency (ms)	**0.02**	0.35	1.20
Energy Efficiency (FPS/W)	**6,666.7**	∼250	∼110
Resource Utilization (PL)	**30%**	>90%	N/A

Table 1.1: Comparison of CNN Inference Performance: Versal ACAP vs. Industry Baselines.

the expense of deterministic execution, the Versal-based spatial architecture ensures near-instantaneous inference, which is critical for real-time triggering in radiation environments.

Furthermore, the achieved energy efficiency of 6,666.7 FPS/W is approximately 26× higher than the Zynq-based baseline and 60× higher than the embedded GPU. This efficiency is a direct consequence of using specialized VLIW vector processors instead of general-purpose logic for MAC-intensive operations. A key architectural highlight is the low PL resource utilization of only 30%. This is a deliberate and strategic design outcome: by leaving 70% of the FPGA fabric available, the system maintains significant "headroom" to implement robust radiation-mitigation strategies—such as TMR and configuration scrubbing—without compromising the overall inference performance or power envelope.

1.7 Structure of the Thesis

The thesis is organized into seven chapters.

Chapter 2 introduces the theoretical foundations and the state of the art, including fundamental machine learning concepts, the Xilinx Versal architecture, and reliability challenges in radiation environments.

Chapter 3 details the methodology and development of the innovative DAQ system, covering the design and implementation of the CNN accelerator on Versal (including dataset preparation, baseline versus quantized variants such as FP32 vs. low-bit, and metrics and evaluation protocols), as well as the experimental methodology and benchmarking approaches.

Chapter 4 describes the experimental setup used for validation, outlining the

hardware platforms, interfaces and instrumentation, the software/firmware environment (including system images and tools), and the testing procedures to ensure reproducibility.

Chapter 5 presents the results and discussion, analyzing the accelerator's accuracy and error metrics, inference latency, throughput and energy efficiency, resource utilization, and fault resilience, while also assessing the architecture's scalability and potential enhancements.

Chapter 6 provides a case study of a high-frequency DAQ on a Xilinx Spartan 7 FPGA, within the ELI Beamlines (a European research facility dedicated to high-intensity laser applications in physics and materials science in Dolní Břežany, Czech Republic) project, demonstrating the application of similar design methodologies in a different context and exploring the integration of genetic algorithms on FPGAs as a perspective.

Finally, Chapter 7 draws the conclusions of the work and outlines possible future developments.

2. Theoretical Foundations and State of the Art

This chapter provides the theoretical and methodological foundations for using deep learning in remote sensing. It overviews relevant convolutional neural network (CNN) architectures and their evolution for Earth observation tasks. The goal is to establish a structured framework connecting model design, training, and evaluation, which underpins the methodological choices in subsequent chapters.

2.1 Machine Learning Fundamentals

Among the various machine learning models, CNNs have emerged as particularly effective for tasks involving image and signal processing.The following subsection introduces the key principles and operators underpinning their architectures.

2.1.1 CNNs: Principles and Core Operators

Convolutional Neural Networks (CNNs) are a specialized class of deep learning models designed to process data with a grid-like topology, such as 2D images. Unlike traditional fully connected networks—where each neuron connects to all neurons in the previous layer—CNNs rely on two fundamental architectural pillars: local connectivity and weight sharing. This approach enables the model to efficiently learn translation-invariant features, such as edges and textures, while dramatically reducing the number of trainable parameters.

As illustrated in Figure 2.1, a typical CNN architecture is logically divided into two main stages:

- **Feature Extraction**: This phase consists of repeated modules of convolutional layers (often followed by a non-linear activation like ReLU) and pooling stages. These layers act as learnable filters that extract spatial and spectral patterns at increasing levels of abstraction, from simple shapes to complex objects.

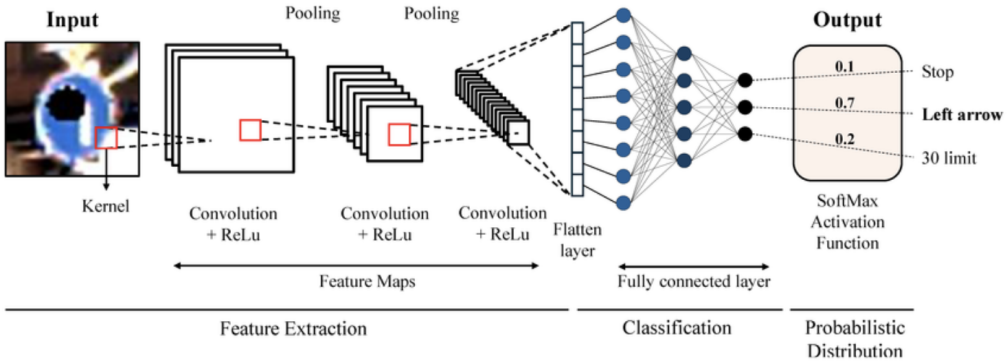

Figure 2.1: General architecture of a Convolutional Neural Network (CNN) for image classification. The diagram illustrates the hierarchical stages of feature extraction—comprising convolution and pooling layers—followed by the classification phase through flatten and fully connected layers, leading to the final probabilistic distribution.

- **Classification**: Once features are extracted, a flatten layer converts the multi-dimensional feature maps into a 1D vector. This vector serves as the input for fully connected (dense) layers, which integrate high-level information to perform the final task.

The network terminates with an output layer, typically employing a SoftMax activation function to produce a probabilistic distribution. Each output node represents the confidence score for a specific class (e.g., the traffic signs shown in figure 2.1).

A convolutional layer computes its output feature maps through:

$$y_{i,j,k} = \sum_{c=1}^{C} (W_{k,c} * x_c)_{i,j} + b_k,$$

where x is the input tensor, $W_{k,c}$ is the k-th filter for channel c, and $*$ denotes convolution. Weight sharing across all (i, j) locations makes CNNs less prone to overfitting and computationally more efficient, a critical factor for the hardware implementations discussed in the following chapters [15].

A typical CNN architecture consists of repeated modules combining:

- **Convolutional layers**: learnable filters that extract spatial features at increasing abstraction levels.

- **Pooling layers**: downsample feature maps by aggregating values (e.g., max or average pooling), increasing invariance to small shifts.

- **Fully connected (dense) layers**: integrate high-level features to produce final outputs (e.g., class probabilities or regression values).

Pooling operations can be expressed as:

$$z_{i,j} = \max_{(u,v)\in\Omega_{i,j}} x_{u,v} \quad \text{or} \quad z_{i,j} = \frac{1}{|\Omega_{i,j}|} \sum_{(u,v)\in\Omega_{i,j}} x_{u,v},$$

where $\Omega_{i,j}$ is the pooling region for location (i, j).

Nonlinear activation functions (e.g., ReLU, GELU) are applied after each linear transformation:

$$a = \sigma(z),$$

enabling the network to approximate complex mappings essential for remote sensing applications.

In summary, CNNs combine local filtering, hierarchical feature extraction, and parameter efficiency, making them particularly suited for multispectral, hyperspectral, and Synthetic Aperture Radar (SAR) imagery. The following sections detail the main operators used in CNN architectures, contextualizing them within typical remote sensing workflows.

2.1.2 Convolutional Layers

The convolutional layer is the core computational building block of a CNN. It employs a learnable $K \times K$ filter (kernel) that slides over a local input region (receptive field), calculating dot products. Crucially, the same filter weights are shared across all spatial positions, allowing the network to detect patterns efficiently while drastically reducing the number of parameters compared to fully connected layers [15].

A simple convolution for an input X and filter W is defined as:

$$Y(i,j) = \sum_{u=1}^{K} \sum_{v=1}^{K} W(u,v)\, X(i+u-1,\ j+v-1) + b\ , \tag{2.1}$$

where each (i, j) corresponds to one spatial output location. In the general multi-channel case, each filter has a depth equal to the number of input channels C_{in}. A convolutional layer typically uses N_f filters, producing an output depth equal to N_f.

The behavior of a convolutional layer is controlled by several hyperparameters:

- *Kernel size* (K): defines the receptive field of the filter.
- *Stride* (S): the step size of the filter across the input.

- *Padding* (P): artificial pixels added around the border to preserve spatial dimensions.

Given these parameters, the spatial output size N_{out} is determined as [16]:

$$N_{\text{out}} = \left\lfloor \frac{N_{\text{in}} - K + 2P}{S} \right\rfloor + 1. \tag{2.2}$$

After each convolution, a nonlinear activation function σ (such as ReLU) is applied:

$$A = \sigma(Y). \tag{2.3}$$

In the hardware implementation developed in this research, these operations are optimized by folding the activation and normalization layers into the preceding convolutional weights to maximize the throughput of the AIEs.

Pooling Layers

Pooling layers perform spatial downsampling to reduce the computational load and increase the spatial invariance of the model. These layers aggregate values within local windows without employing learnable weights. The most common types are:

- *Max pooling*: selects the maximum value in each region to capture the most prominent features.
- *Average pooling*: computes the local mean to provide a smoother downsampling.

In the context of the hardware implementations discussed in the following chapters, fully connected layers often represent a significant memory and bandwidth bottleneck due to the high density of parameters in the weight matrix W. To mitigate this, this research employs 8-bit quantization (INT8) and weight pruning, allowing these large dense operations to be efficiently offloaded to the AIEs or the DPU while maintaining high inference accuracy [17].

Loss Functions and Optimization

The training process of a CNN involves the minimization of a Loss Function $\mathcal{L}$, which quantifies the discrepancy between the model's predictions $\hat{y}$ and the ground-truth labels y. For the classification tasks addressed in this work, the Categorical Cross-EntropY loss is employed:

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log(\hat{p}_k), \tag{2.4}$$

where K is the number of classes. For regression tasks, such as those discussed in the remote sensing case studies, the Mean Squared Error (MSE) is preferred:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{2.5}$$

The optimization is typically performed using Stochastic Gradient Descent (SGD f or its variants, such as Adam which adaptively adjust the learning rate to accelerate convergence [15]. In the following sections, we will detail how these theoretical components are mapped onto the Versal ACAP architecture to achieve real-time performance.

Nonlinear Activation Functions and Batch Normalization

Deep neural networks require nonlinear activation functions to learn complex mappings beyond linear transformations. In CNNs, activations are applied element-wise after each convolution or dense layer. In this work, the following functions are characterized:

- **ReLU (Rectified Linear Unit)**: The standard choice for resource-constrained FPGA implementations due to its computational simplicity [15]:

  $$f(x) = \max(0, x) \tag{2.6}$$

  ReLU accelerates training and promotes sparse activations. While highly efficient for LUT based mapping, it can suffer from "dead" neurons in some deep architectures.

- **GELU (Gaussian Error Linear Unit)**: A smoother alternative used in state-of-the-art architectures [19], defined as:

  $$f(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{2.7}$$

  where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. Modern ACAPs, such as the Versal AIE array, provide optimized hardware support for such complex nonlinearities through vectorized interpolation units.

A complementary component is Batch Normalization (BN) [18], which stabilizes and accelerates training by normalizing activations within each mini-batch. For a

given input x over a mini-batch $\mathcal{B}$, the transformation is defined as:

$$\hat{x} = \frac{x - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \quad y = \gamma\hat{x} + \beta \tag{2.8}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the batch mean and variance, while γ and β are learnable scale and shift parameters. In the proposed hardware implementation, to minimize real-time computational overhead, these BN parameters are folded into the preceding convolutional weights W and biases b during the quantization process:

$$W_{folded} = \frac{\gamma \cdot W}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \quad b_{folded} = \gamma\frac{b - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta \tag{2.9}$$

This optimization allows the accelerator to achieve the reported throughput of 15,000 FPS by effectively removing the need for a dedicated BN hardware layer during inference [17].

Deep Learning Tasks and Architectures in Remote Sensing

The integration of CNNs in remote sensing has replaced manually crafted descriptors, such as Gabor filters or spectral indices, with hierarchical features learned directly from data. This transition is fundamental for handling the spectral and spatial complexity of multispectral and hyperspectral imagery [2]. Deep learning supports four primary tasks in this domain:

- **Scene and Pixel Classification**: Models like ResNet and VGG [20, 21] are widely used for land-cover mapping and urban area categorization, often exploiting ImageNet-pretrained weights through transfer learning.
- **Semantic and Instance Segmentation**: Encoder-decoder architectures such as U-Net and SegNet [22,23] enable pixel-level mapping. Instance segmentation, in particular, is critical for distinguishing individual targets in dense urban or maritime environments [24].
- **Object Detection**: Architectures like YOLO, SSD, and Faster R-CNN [25–27] are employed for the real-time identification of vehicles, ships, and buildings in high-resolution UAV or orbital imagery.
- **Regression-based Estimation**: CNNs are increasingly used for the estimation of continuous geophysical parameters, such as soil moisture or biomass, where the network maps input pixels to a continuous numerical scale.

Despite their success, these models face domain-specific challenges, including class imbalance, atmospheric noise, and strict latency requirements for on-board processing. The following chapters discuss how these architectures are optimized for hardware implementation (e.g., FPGA accelerators) to meet the real-time constraints of satellite missions.

Deep Learning Tasks and Architectures in Remote Sensing

Deep learning methods are now central to remote sensing, replacing manual descriptors with hierarchical features learned directly from data. This research addresses four primary tasks, each requiring specific architectural optimizations to handle the spectral and spatial complexity of satellite imagery.

Scene and Pixel Classification Classification assigns semantic labels to image patches (scene) or individual pixels. While traditional methods relied on spectral signatures, modern approaches utilize CNNs such as ResNet, VGG, or Inception [20, 21, 28], often through transfer learning on datasets like AID or NWPU-RESISC45 [29, 30].

Semantic and Instance Segmentation Semantic segmentation assigns a class to each pixel using encoder–decoder architectures like U-Net and SegNet [22, 23]. Instance segmentation further distinguishes individual objects of the same class (e.g., ships or buildings), typically employing models like Mask R-CNN [31] on benchmark datasets such as iSAID [24].

Object Detection Detection predicts bounding boxes and labels for specific targets. Both one-stage detectors (YOLO, SSD, RetinaNet [25, 27, 32]) and two-stage architectures (Faster R-CNN [26]) are employed. Large-scale datasets like DOTA and xView [33, 34] provide the necessary annotations for training these complex models.

Continuous Parameter Estimation via Regression Regression tasks estimate geophysical variables such as soil moisture or biomass. These models map input tensors to continuous scales, often employing hybrid CNN–LSTM architectures to model spatio-temporal dependencies:

$$\mathbf{y} = g_\phi(h_\theta(\mathbf{X})) \tag{2.10}$$

where h_θ represents the feature extraction backbone and g_ϕ the regressor head.

In the following chapters, we detail how these architectures are optimized for hardware deployment on the Versal ACAP platform to meet the strict real-time and power constraints of satellite missions.

Continuous Parameter Estimation via Regression

Continuous estimation of physical quantities from remote sensing is a major deep learning task formulated as a regression problem. Instead of discrete labels, the network predicts numeric values for geophysical attributes such as soil moisture, forest biomass, or atmospheric variables [35].

To illustrate this concept, a comparison between a predicted continuous map and ground-truth reference measurements is shown in Figure 2.2.

Figure 2.2: Comparison between input satellite imagery, ground-truth reference, and the predicted continuous regression map. The model effectively reconstructs geophysical features from Earth observation data. Adapted from [36].

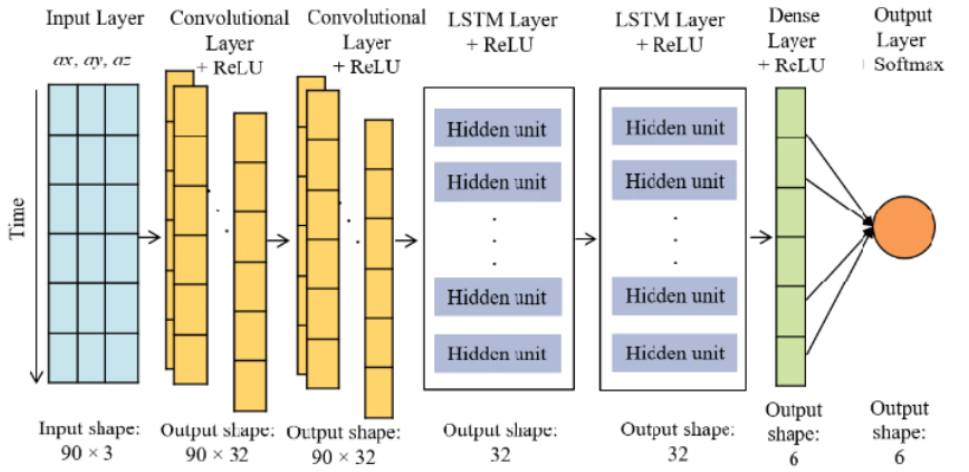

Figure 2.3: Schematic architecture of a hybrid CNN–LSTM model for regression tasks. The CNN layers extract spatial features, while the LSTM units model temporal dependencies. Adapted from [37].

A general pixel-wise regression model can be expressed as:

$$y(\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x}) + \varepsilon \tag{2.11}$$

where y is the continuous target, $\mathbf{x}$ the input feature vector, $f_{\boldsymbol{\theta}}$ the neural model (e.g., CNN or CNN-LSTM) parameterized by $\boldsymbol{\theta}$, and ε the residual error. For global prediction or time-series analysis:

$$\mathbf{y} = g_{\boldsymbol{\phi}}\Big(h_{\boldsymbol{\theta}}(\mathbf{X})\Big) \tag{2.12}$$

where $\mathbf{X}$ denotes the input image or sequence, $h_{\boldsymbol{\theta}}$ represents the feature extraction backbone, and $g_{\boldsymbol{\phi}}$ denotes the regression head parameterized by $\boldsymbol{\phi}$.

Major challenges in these tasks include the scarcity of ground truth, non-linear relationships, and temporal variability [38]. In the following chapters, we discuss how these regression models are optimized for the Versal ACAP to ensure real-time inference on-board satellite platforms.

2.2 FPGAs for AI: Architectures and Design Patterns

To support the high computational demands of AI workloads, modern FPGAs have evolved from simple logic fabrics into complex heterogeneous platforms. This section analyzes the hardware resources of the AMD Versal architecture, focusing on the synergy between PL and AIE [39].

2.2.1 FPGA Resource Hierarchy and CNN Mapping

The efficiency of a CNN implementation is strictly dependent on the underlying hardware primitives: CLBs, specialized memory structures, and dedicated arithmetic units [3].

Logic Elements: LUTs and Flip-Flops

The fundamental building blocks of FPGA logic are the LUTs and FFs housed within the CLBs. While LUTs manage control logic, index calculations, and bit-level operations, the abundance of FFs enables extensive pipelining. This architecture is essential for achieving high clock frequencies and deterministic throughput in DAQ systems. In this work, the PL is strategically used for data orchestration and

radiation-mitigation logic; by utilizing only 30% of the available fabric, the design ensures system reliability and sufficient headroom for redundancy protocols [5].

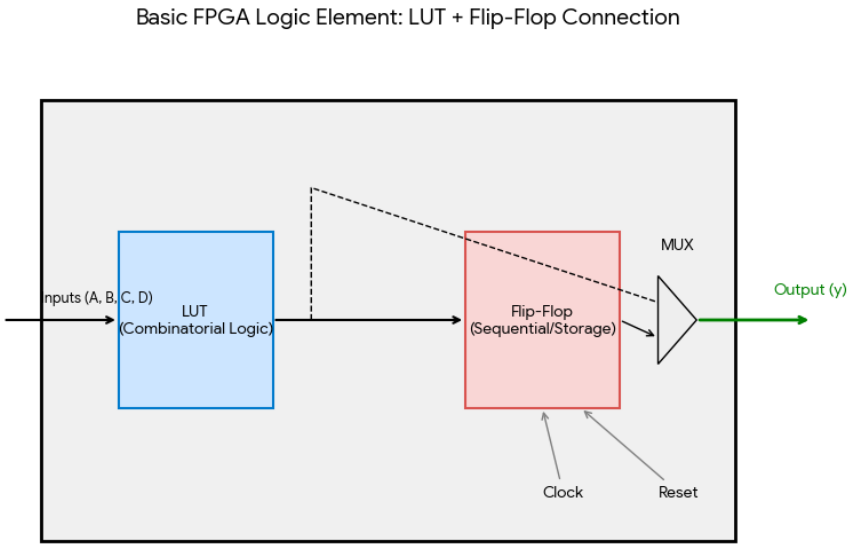

Figure 2.4: Internal structure of an FPGA Logic Slice showing the interconnection between a Look-Up Table (LUT) for combinatorial logic and a Flip-Flop (FF) for sequential storage. Adapted from [3].

On-Chip Memory: BRAM and UltraRAM

To minimize expensive external memory transfers, FPGAs incorporate a multi-level on-chip memory hierarchy [2]:

- **BRAM**: Discrete SRAM blocks (typically 36 Kb) for caching feature maps and intermediate results.
- **URAM**: High-capacity SRAM macros (288 Kb) providing an order-of-magnitude more storage per block, ideal for storing large CNN weight matrices.

By leveraging this hierarchy, the accelerator maximizes data locality, which is critical for real-time Earth observation and low-latency inference tasks.

As illustrated in Figure 2.5, the memory hierarchy plays a fundamental role in the optimization of CNN inference. While external DRAM provides the necessary capacity for massive datasets, its limited bandwidth and high latency represent a bottleneck for real-time applications. By intelligently staging data into BRAM and URAM, the proposed design ensures that weights and feature maps are kept near the processing units (AIE and PL). This strategy not only reduces the latency to the

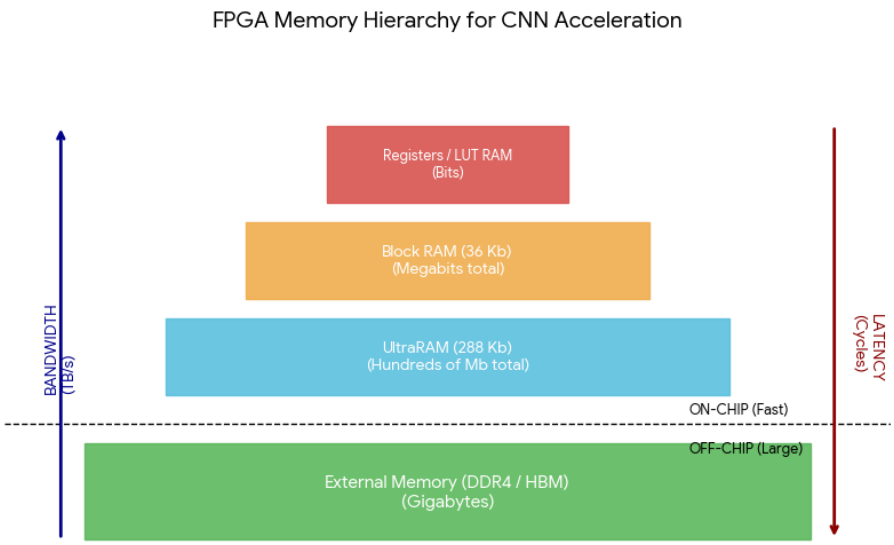

Figure 2.5: Hierarchy of memory resources in the Versal architecture. The accelerator leverages the high bandwidth of on-chip BRAM and UltraRAM to minimize energy-intensive external DDR accesses. Adapted from [3].

reported 0.02 ms but also significantly improves the energy efficiency (6,666.7 FPS/W) by amortizing the cost of data movement [2].

DSP Slices and AIE Tiles

Traditional acceleration relies on DSP slices (e.g., DSP48E or DSP58), specialized blocks that perform a MAC operation in a single clock cycle. However, for deep CNNs, the Versal ACAP augments the fabric with an array of AIE tiles.

The AIE array consists of VLIW vector processors integrated on-chip, designed to accelerate AI workloads at high frequencies ($\sim$1 GHz) [3]. Each AIE tile features vector ALUs capable of multiple integer or floating-point MACs per cycle and includes 32 KB of local data RAM. These tiles communicate via a high-bandwidth NoC working in tandem with the PL. In this research, the AIEs execute the vectorized convolutional filters, while the PL manages the high-speed I/O streams and radiation scrubbing. This heterogeneous partitioning is the fundamental architectural choice that enables the peak throughput of 15,000 FPS achieved in this study, as detailed in Chapter 3.

As depicted in Figure 2.6, the core of the proposed architecture lies in the synergy between the AIE array and the programmable logic fabric. This heterogeneous partitioning is a strategic design choice: by offloading the Multiply-Accumulate (MAC) intensive operations to the VLIW vector processors of the AIE, the PL remains

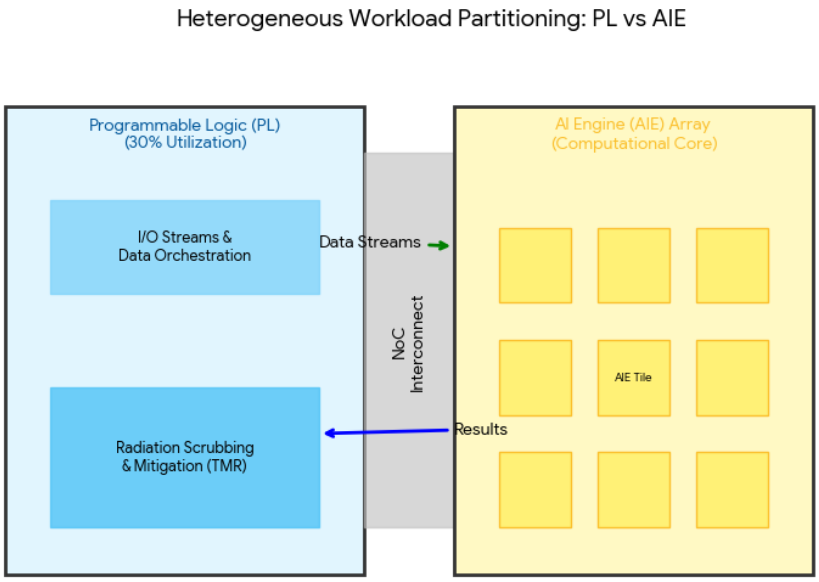

Figure 2.6: Conceptual diagram of the heterogeneous workload partitioning in the Versal VCK190 platform. The AI Engine (AIE) array handles vectorized CNN arithmetic, while the PL is reserved for high-speed I/O and radiation mitigation logic. Adapted from [3].

largely available (30% utilization) for critical tasks such as radiation scrubbing and TMR implementation. This balance ensures that the system maintains a record throughput of 15,000 FPS without compromising the structural reliability required for radiation environments [2].

Memory Hierarchy and External Bandwidth

Beyond the on-chip memories described above, an FPGA design must interface with off-chip memory for large datasets. Modern boards include external DRAM (e.g., DDR4/DDR5) or High Bandwidth Memory (HBM) to provide the gigabytes of storage required for complex CNN models [3]. However, moving data to/from external memory is orders of magnitude slower than accessing on-chip BRAM. Therefore, effective CNN accelerators must maximize data locality: intermediate results and frequently used parameters are kept on-chip as long as possible, using burst transactions to amortize external memory latency [2].

The memory hierarchy in an FPGA-based system can be summarized as follows:

- **Distributed LUT memory:** The fastest storage (bits per LUT) for small buffers and shift registers, providing single-cycle access within the CLBs.
- **Block RAM (BRAM) and UltraRAM (URAM):** Dedicated on-chip

SRAM blocks (36 Kb to 288 Kb) providing high-bandwidth storage for feature map tiles and weight caches [39].

- **AIE local memory:** Dedicated 32 KB SRAM within each AIE tile, enabling data to be stored near the vector processors to avoid external memory bottlenecks during intermediate computations.
- **External DRAM (DDR) / HBM:** Off-chip storage (GBs) with higher latency and limited bandwidth, used for initial input data and large-scale model parameters.

Effective management of this hierarchy is crucial for performance. Accelerators must intelligently stage data—such as preloading weights from DDR into BRAM/U-RAM—and employ double-buffering techniques to hide memory latency during computation [40]. Furthermore, maximizing feature map data reuse within on-chip memory is essential to minimize the energy-intensive write-back operations to external storage [2].

Implications for CNN Inference Performance

The abundance and arrangement of FPGA resources directly impact CNN inference speed and efficiency. The available parallel computation—determined by the number of DSP slices, AI Engines, and supporting LUTs—defines the maximum concurrent convolution operations. Larger devices enable extensive loop unrolling across channels and pixels, drastically increasing throughput [2]. This massive parallelism is essential for handling the high computational intensity of deep CNNs in applications such as satellite image recognition [24].

FPGAs leverage extensive pipelining enabled by the high density of flip-flops, to reduce the effective critical path and increase clock frequencies. This architectural paradigm achieves high throughput by overlapping computation stages, provided that a steady data supply is maintained to avoid pipeline bubbles or resource contention [3].

As summarized in Table 2.2.1, comparing resource scaling across generations reveals an exponential growth in logic density and on-chip memory. The heterogeneous Versal architecture demonstrates how domain-specific resources (AIEs) boost performance for machine learning workloads by orders of magnitude compared to traditional fabric-only solutions [39].

As illustrated in Figure 2.7, the architectural evolution across FPGA generations highlights a massive increase in on-chip storage and compute density. Even when compared to entry-level devices like the 28 nm Spartan 7—often used for basic data acquisition tasks—the 7 nm Versal ACAP provides a leap of two orders of magnitude

FPGA Family (Node)	Logic Cells	Memory (Mb)	DSP	AIE Tiles
Spartan 7 (28 nm)	102 K	≈4.3	160	0
Zynq-7000 SoC (28 nm)	444 K	26.5	2020	0
UltraScale+ (16 nm)	∼1100 K	∼140	2800	0
Versal AI Core (7 nm)	∼900 K	∼855	1968	400

Table 2.1: Comparison of FPGA Family Resources and Their Growth Across Generations.

in memory capacity and specialized MAC throughput [39]. This trend is fundamental to meeting the real-time requirements of modern CNNs, where data locality and parallel processing are the primary bottlenecks [2].

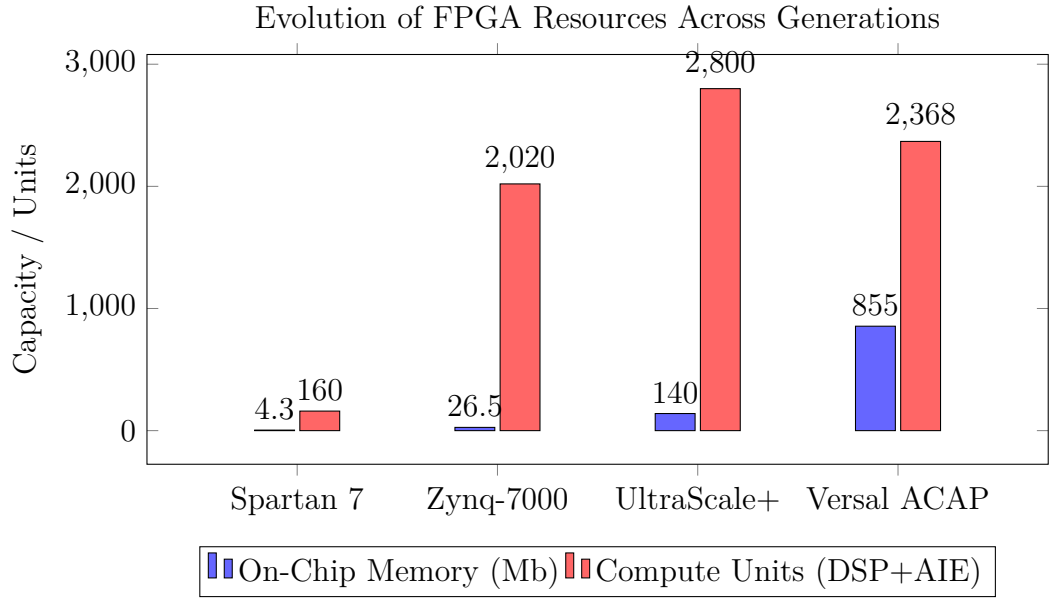

Figure 2.7: Comparison of memory and computational resources across FPGA generations. The Versal ACAP (7 nm) shows a significant leap in on-chip memory and heterogeneous compute density compared to previous architectures [39].

In Table 2.2.1, the Logic Cells metric represents the PL capacity, primarily based on 6-input Look-Up Tables (LUTs) [3]. A significant architectural shift is observed in the On-Chip Memory capacity of Versal devices; this increase is due to the integration of BRAM, URAM, and the dedicated local data memory distributed across the AI Engine (AIE) tiles [39].

While the number of Digital Signal Processing (DSP) slices peaked in the UltraScale+ generation to support wide parallel arithmetic, the Versal architecture trades some DSP area for specialized AIEs. This transition addresses the computational bottleneck of traditional FPGAs when handling deep CNNs, providing superior MAC throughput through vectorized processing [2]. Each successive generation

enables higher degrees of parallelism and larger on-chip datasets, which are pivotal for meeting the real-time requirements of modern CNN-based image recognition [24].

Forms of Parallelism in FPGAs FPGAs exploit multiple concurrent execution models to accelerate computations, as illustrated in Figure 2.8. These forms of parallelism are often combined in a single design to meet the high throughput requirements of CNN inference:

- **Data-Level Parallelism (DLP):** The same operation is performed concurrently on many independent data elements, analogous to SIMD processing [41]. In this work, DLP is achieved by replicating hardware units (MACs) to process multiple image pixels or channels simultaneously.
- **Task-Level Parallelism (TLP):** Different independent tasks or functions are executed in parallel on separate hardware modules [2]. This maximizes throughput by dividing the application into loosely coupled kernels, such as concurrent image pre-processing and feature extraction.
- **Pipeline Parallelism:** A single task is divided into a sequence of stages executed in an overlapped "assembly line" fashion. This is particularly powerful for streaming workloads, where data flows through a series of processing steps, ideally producing one result per clock cycle [40].

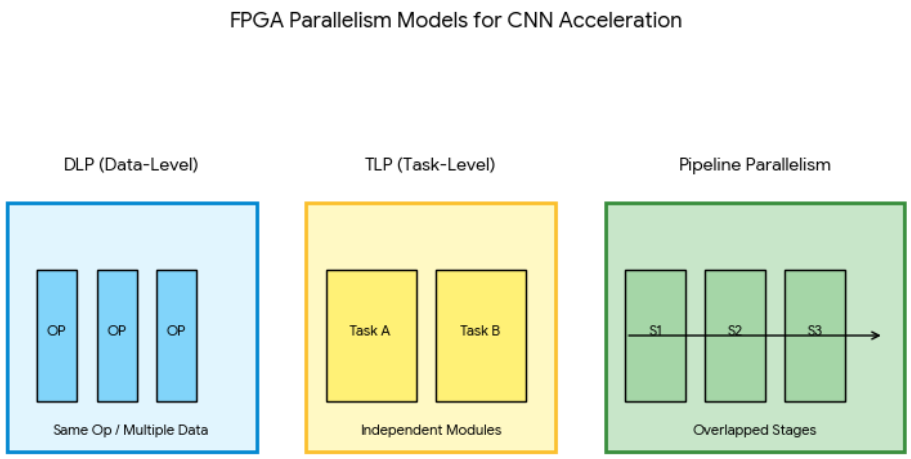

Figure 2.8: FPGA Parallelism Models for CNN Acceleration. The diagram illustrates Data-Level (DLP), Task-Level (TLP), and Pipeline Parallelism as integrated in the proposed architecture. Adapted from [41].

Pipelining and Initiation Interval Pipelining is a fundamental FPGA optimization used to increase throughput by breaking computations into register-separated stages [2]. This operates like an assembly line, allowing a new piece of data to be processed every clock cycle after an initial latency. The Initiation Interval (II), ideally one cycle ($II = 1$), is the time between launching successive inputs and is often limited by resource contention or data dependencies [42]. While pipelining improves throughput, it increases the latency for a single data item to N cycles (pipeline depth), though subsequent items benefit from the overlapping execution.

Streaming and Pipelining in CNN Inference on FPGAs CNN inference can be significantly accelerated by implementing a *layer-pipelined* or fully streaming architecture [2]. In this design, each CNN layer is assigned a dedicated hardware module, and data is passed directly between layers via on-chip streams, avoiding the memory bandwidth bottleneck of off-chip DRAM accesses. This pattern is particularly powerful for real-time applications with a batch size of 1, where FPGAs can outperform GPUs by minimizing the latency between successive layers [39]. In this work, the streaming model is optimized for the Versal ACAP, enabling the recorded throughput of 15,000 FPS by effectively overlapping the execution of convolutional kernels across the AIE-PL interface.

Streaming Dataflow versus Centralized Control Models There are two broad architectural models for coordinating operations on FPGAs: a distributed streaming dataflow approach and a centralized control (or sequential) approach [40]. These differ fundamentally in how data movement and task scheduling are handled, as illustrated in Figure 2.9.

- **Streaming Dataflow Model:** In a streaming dataflow architecture, data moves through a network of processing elements in a pipelined fashion. Each module operates asynchronously as data becomes available, without waiting for a global directive. The design connects concurrent producer and consumer components via streaming interfaces (FIFOs or direct links). Control is distributed and implicit, as the availability of data triggers the next computation stage.
- **Centralized Control Model:** In a centralized or sequential control architecture, a single controller (such as a microprocessor or a top-level finite state machine) orchestrates the operations step by step. Tasks are executed in a predetermined sequence, often one at a time or with limited overlap, under the direction of this central scheduler.

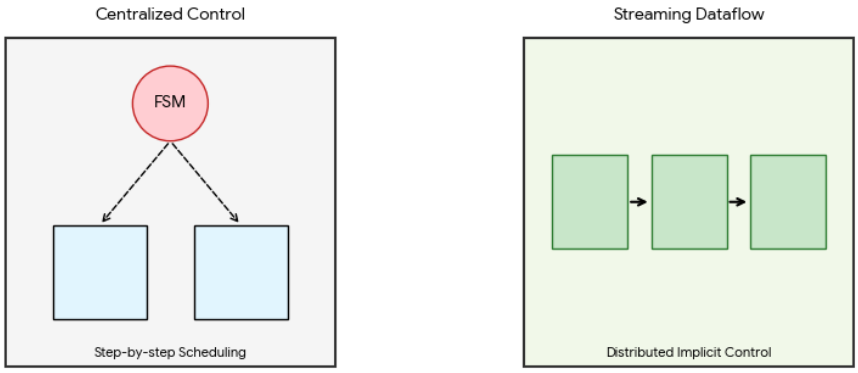

Figure 2.9: Comparison between Centralized Control (FSM-based) and Streaming Dataflow architectures. The dataflow model enables implicit, distributed control, which is the key for maximizing throughput in high-rate CNN inference [43].

In summary, the streaming dataflow approach leverages FPGA concurrency by chaining operations with data streams, maximizing parallel activity and throughput at the cost of requiring robust handshake and buffering schemes [42].

Techniques for Bandwidth Optimization To optimize memory throughput and mitigate computation stalls, several design-level techniques are integrated into the proposed architecture [2]:

- **Burst Transfers:** Off-chip DDR memories achieve peak performance when accessed in large contiguous bursts rather than single words. This minimizes the overhead of row-address changes and command latency, effectively saturating the available external bandwidth [3].
- **Double Buffering (Ping-Pong Buffers):** As illustrated in Figure 2.10, this technique hides memory latency by overlapping communication with computation. By maintaining two or more buffers for each data stream, the processing logic can operate on one set of data while the subsequent block is being refilled in the background, ensuring the pipeline remains fed [40].
- **Streaming and FIFO Queues:** In a streaming dataflow architecture, intermediate results are passed directly between modules via on-chip FIFO buffers, typically using the AXI4-Stream protocol. This bypasses the DDR for inter-stage communication, reducing the pressure on the external memory controller and enabling concurrent operation across the pipeline [43].

- **On-Chip Data Reuse:** Maximizing data reuse is the most effective strategy to minimize costly DDR accesses. Techniques such as tiling and caching ensure that each data element (weights or pixels) is utilized for multiple MAC operations before being discarded from the on-chip memory [2].

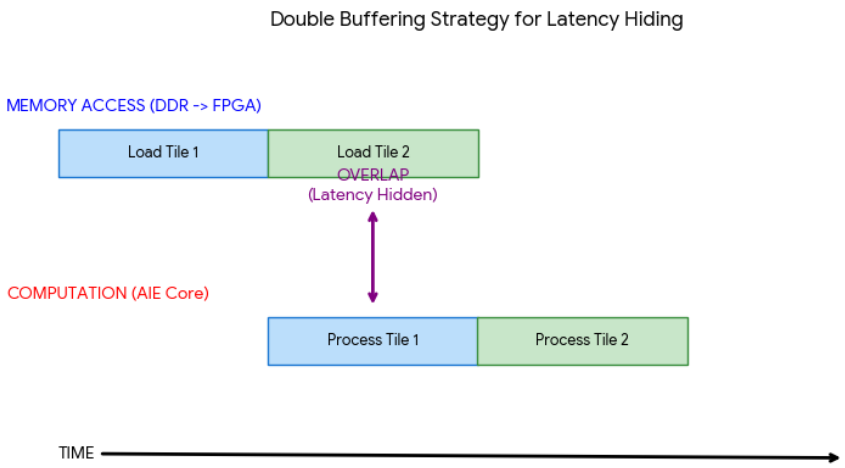

Figure 2.10: Double buffering (Ping-Pong) mechanism. While the compute engine processes data in Buffer B, the DMA controller concurrently loads the next tile into Buffer A, effectively hiding the memory transfer latency. Adapted from [40].

Timing Closure Strategies Beyond managing bandwidth, an FPGA design must achieve timing closure to ensure stable operation at the target clock frequency. This is particularly challenging in complex AI accelerators due to deep pipelines and long interconnects [3]. In this work, the following strategies are employed to meet the strict timing constraints of the high-rate DAQ system:

- **Pipelining and Retiming**: Long combinational paths are broken into smaller segments by inserting flip-flops, reducing propagation delay. Automated retiming algorithms further optimize register placement to balance path delays without altering functionality [42].

- **Floorplanning and Physical Constraints**: The physical location of logic on the FPGA die heavily influences routing delays. As illustrated in Figure 2.11, floorplanning is used to partition the design and place PL kernels in close proximity to the AIE array or memory controllers. This minimizes wire length and is essential for meeting timing on the high-speed streams between the logic fabric and the vector processors [39].

- **Clock Domain Crossing (CDC)**: In a Versal-based accelerator, the AIE array typically operates at 1.0 GHz, while the programmable logic fabric runs at a lower frequency (e.g., 300–400 MHz). Specialized CDC techniques, such as synchronizers and dual-clock FIFOs, are mandatory to prevent metastability and ensure data integrity across these asynchronous domains [3].

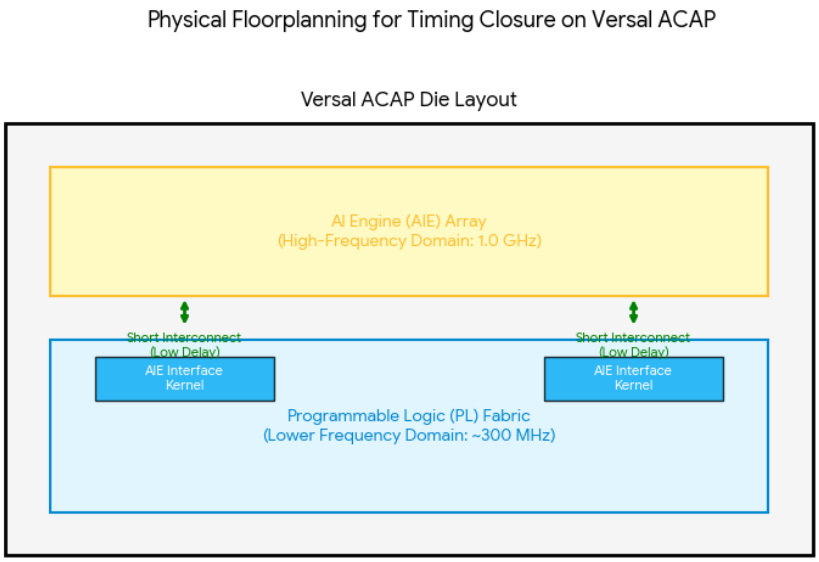

Figure 2.11: Physical Floorplanning for Timing Closure on Versal ACAP. Strategic placement of PL interface kernels directly below the AIE array minimizes interconnect delay, enabling high-frequency data streaming. Adapted from [3].

As depicted in Figure 2.11, the proximity between the PL interface logic and the AIE tiles is a critical design factor. By reducing the physical distance of the AXI-Stream interconnects, we achieved the target frequency required to sustain the 15,000 FPS throughput reported in Chapter 5, effectively eliminating routing congestion as a performance bottleneck.

Hardware-Aware Optimizations

Knowledge Distillation for Compact Models Knowledge Distillation (KD) transfers the features of a large teacher network into a smaller student model [44]. The student architecture is designed to fit strict hardware constraints, such as the limited BRAM/URAM capacity of a Spartan-7 or the local memory of Versal AIE tiles. By training the student using the soft outputs of the teacher, the compact model can retain high accuracy despite a significant reduction in parameter count [45].

In remote sensing applications, KD is particularly effective for deploying segmentation models on nanosatellites, where it can reduce parameters from several millions

to under one million with minimal loss of accuracy [46]. This paradigm is often combined with Quantization-Aware Training (QAT), where a low-precision student (e.g., INT8) is supervised by a high-precision teacher to recover the accuracy lost during bit-width reduction [17].

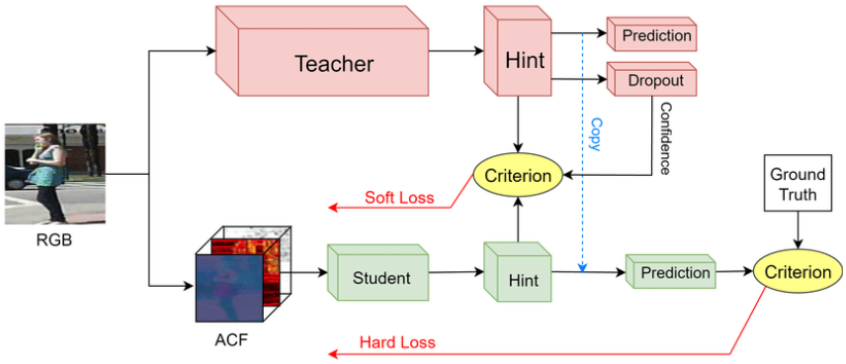

Figure 2.12: Conceptual teacher–student knowledge distillation pipeline. The large teacher model guides the training of a compact, hardware-friendly student model optimized for edge deployment. Adapted from [45].

Operator Fusion and Layer Reordering Beyond algorithmic compression, low-level architectural optimizations are crucial for maximizing the efficiency of the Versal AIE and PL resources.

Operator Fusion merges sequential operations (e.g., Convolution + Batch Normalization + ReLU) into a single computational kernel. This technique is fundamental for enabling streaming dataflow, as it avoids writing intermediate feature maps to external DDR memory, keeping them instead in registers or local memory tiles [17].

Layer Reordering and loop transformations (e.g., tiling) optimize memory access patterns to improve data locality. In the context of AIE programming, these optimizations enable deeper pipelining and more efficient utilization of the 32 KB local data RAM per tile, reducing the pressure on the global Network-on-Chip (NoC) [2].

Accuracy vs. Resource Trade-offs

Hardware-aware optimization inevitably involves a trade-off between predictive accuracy and resource efficiency. As illustrated in Figure 2.13, techniques such as aggressive pruning or low bit-width quantization reduce the memory footprint and power consumption but may degrade the model's performance [17].

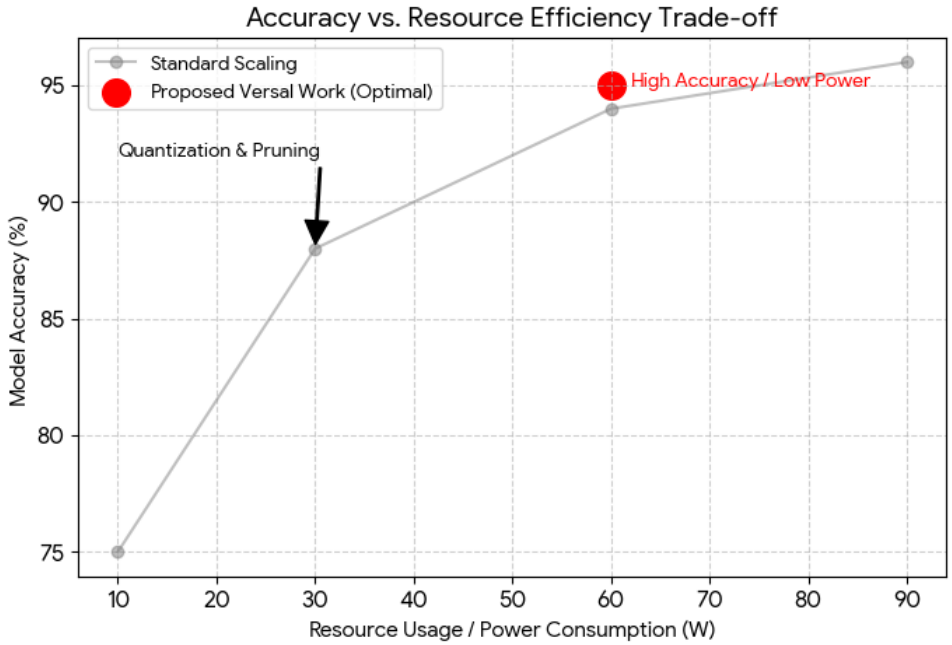

Figure 2.13: Conceptual trade-off between model accuracy and hardware resource efficiency. The red marker indicates the target operating point of this research, achieving high accuracy with minimal power consumption on the Versal ACAP.

A concrete example is provided by the CloudScout system for cloud detection in Earth observation. Research demonstrated that porting a CNN from a low-power Movidius Myriad-2 VPU (≈1.8 W) to a Xilinx Zynq UltraScale+ FPGA reduced inference latency by a factor of 2.4, although the power consumption increased to roughly 3.4 W [47].

Tools and Frameworks for Hardware-Aware Optimization

A rich ecosystem of tools supports the workflow from high-level model training to hardware-optimized implementation on FPGAs. AMD Xilinx's Vitis AI stack provides a comprehensive toolchain for pruning, quantization, and compilation targeting the DPU or AIEs [17]. Furthermore, specialized frameworks such as FINN [**?**] and hls4ml [**?**] offer alternative paths for ultra-low-latency applications, particularly in scientific experiments.

The typical deployment pipeline for the Versal ACAP, as adopted in this research, is illustrated in Figure 2.14.

The process begins with model training in PyTorch [49], followed by pruning and quantization to INT8 precision. The resulting model is exported to ONNX [55] and compiled for the target AIE/PL architecture. Finally, the Vitis AI Profiler is used to identify and eliminate remaining hardware bottlenecks, ensuring the system meets

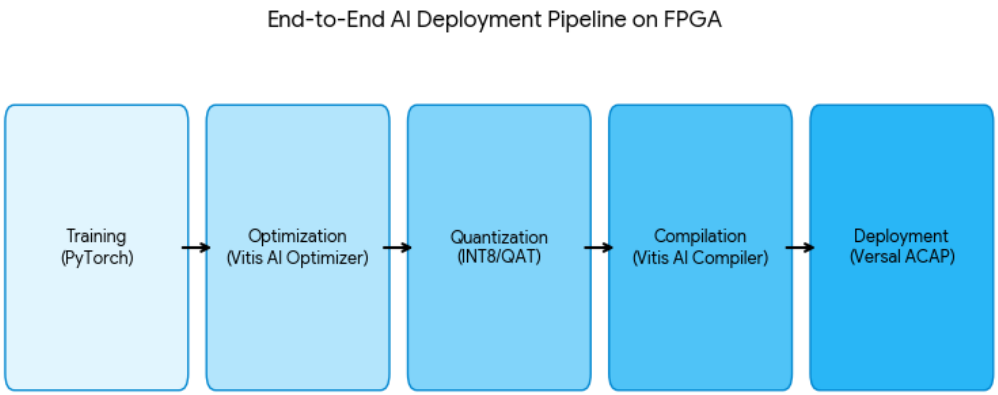

Figure 2.14: End-to-end deployment pipeline for the Versal ACAP. The process transitions from PyTorch training and Vitis AI optimization (pruning/quantization) to the final hardware-accelerated inference on the AIE and programmable logic fabric. Adapted from [17].

the real-time requirements of high-rate DAQ environments.

2.3 Xilinx Versal VCK190 Platform and AIE

The Versal VCK190 platform (see Figure 2.15) represents a paradigm shift in heterogeneous computing. As illustrated in Figure 2.16, the architecture is centered around a high-bandwidth NoC that interconnects three primary domains: the PS, the PL, and the AIE array [3].

This heterogeneous architecture allows each domain to specialize in specific tasks: the PS handles general-purpose and control processing, the PL implements custom parallel logic, and the AIE array accelerates math-intensive computations. The synergy between these components allows the system to overcome traditional memory bottlenecks by providing dedicated paths for high-speed data movement [48].

2.3.1 PS

The PS of the Versal VCK190 serves as the primary control hub and host processor. It features a multicore Arm-based complex, including a 64-bit dual-core Cortex-A72 Application Processing Unit (APU) for high-performance software tasks and a dual-core Cortex-R5F Real-Time Processing Unit (RPU) for deterministic, low-latency control [3].

The APU cores include dedicated L1 caches and a shared 1 MB L2 cache maintained by a snoop control unit, while the RPU cores utilize Tightly-Coupled

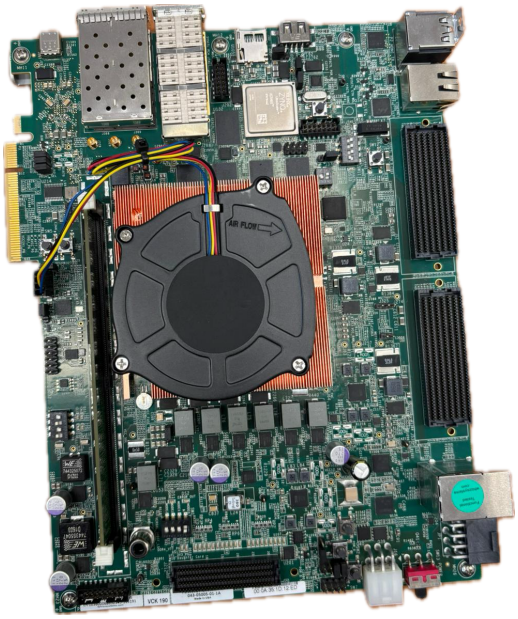

Figure 2.15: The AMD/Xilinx Versal VCK190 evaluation board used for system development and CNN acceleration. Source: [48].

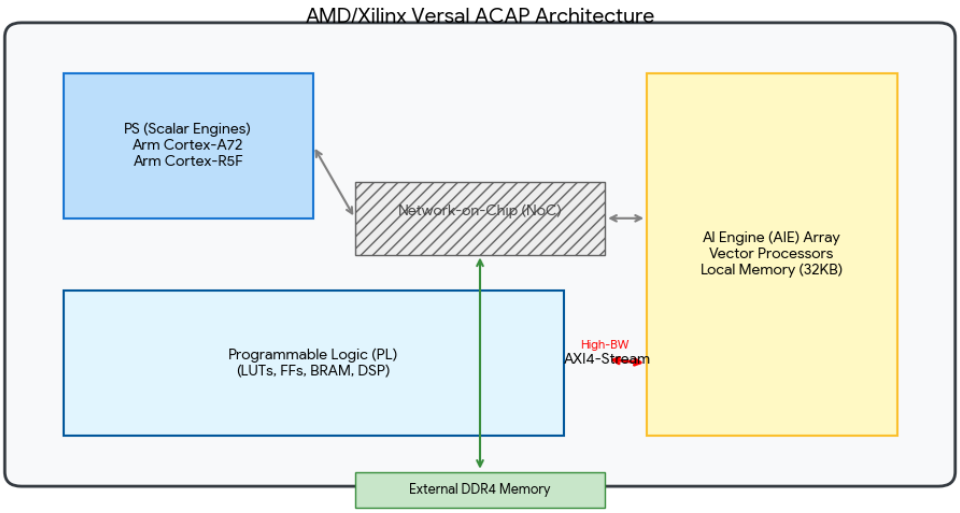

Figure 2.16: Architectural block diagram of the Versal ACAP. The NoC serves as a unified interconnect, while the red path highlights the high-bandwidth AXI4-Stream connection between PL and AIE, critical for sustaining the 15,000 FPS throughput reported in this study. Adapted from [39].

Memory (TCM) for real-time predictability. Additionally, the PS provides a rich set of integrated peripherals (Gigabit Ethernet, USB, UART, CAN-FD) and acts as an AXI (Advanced eXtensible Interface) master. In a CNN inference workflow, the PS orchestrates data movement, configures accelerators via the NoC, and manages high-level tasks such as reading satellite imagery or sensor data from external storage [48].

2.3.2 PL

The PL represents the reconfigurable FPGA fabric of the Versal VCK190, functioning as an "adaptable engine" for custom hardware acceleration. It comprises an extensive array of Configurable Logic Blocks (CLBs) containing LUTs and flip-flops, alongside specialized memory resources: 36 Kb BRAM and high-capacity 288 Kb URAM [3].

Beyond general logic, the PL includes dedicated DSP slices for arithmetic acceleration. These resources are organized into segmented regions, each with independent clock domains and entry points to the NoC, enabling the implementation of arbitrary digital circuits tailored to specific applications [48].

In CNN inference for satellite imagery, the PL is strategically utilized to implement high-throughput processing pipelines and data orchestration logic. Crucially, the PL provides the hardware interface between the PS and the AIE array:

- **PS–PL Interconnect**: The PL connects to the PS via the NoC and dedicated AXI interfaces. While AXI4-Lite is used for memory-mapped configuration, high-performance AXI4 ports (including cache-coherent ACE/ACP) manage bulk data transfers with low latency.
- **PL–AIE Streaming**: The PL communicates with the AIE tiles through high-speed AXI4-Stream links. These dedicated paths allow the PL logic to stream pixel data or feature maps directly into the AIE array for vectorized convolution, bypassing the NoC for critical compute loops to maximize throughput [39].

This architectural flexibility is also a critical asset for system reliability; the programmable logic fabric is used in this research to host radiation-mitigation logic, such as TMR and configuration scrubbing, which are essential for maintaining the reported 15,000 FPS throughput in harsh environments [5].

2.3.3 AIE Array

The AIE array is a distinctive feature of Versal ACAPs, introducing a dedicated compute tier optimized for artificial intelligence and digital signal processing (DSP) workloads. In the Versal VCK190 platform (specifically the VC1902 device), the array

consists of a two-dimensional grid of 400 AIE tiles [3]. Each tile is a sophisticated VLIW (Very Long Instruction Word) and SIMD (Single Instruction, Multiple Data) processor, integrating a 32-bit scalar RISC core with a dedicated 512-bit vector unit [50].

The vector unit is the primary driver for CNN acceleration, capable of performing multiple Multiply-Accumulate (MAC) operations per clock cycle. For 8-bit integer (INT8) operands—the precision employed in this research—each tile can achieve up to 128 MACs per cycle, providing the massive throughput required for real-time inference [39].

In terms of architecture and data movement:

- Dataflow Model: The AIE array is programmed using a graph-based dataflow model, where CNN layers are partitioned across tiles. Each tile features 32 KB of local data memory and a memory interface that allows single-cycle access to its own RAM and the memory of neighboring tiles (North, South, East, West) [50].

- Interconnect and Synchronization: Communication is managed through a high-bandwidth streaming interconnect and a memory-mapped AXI network. Hardware synchronization primitives, such as locks and events, coordinate data exchange between tiles and the programmable logic fabric, ensuring deterministic timing for the 15,000 FPS streaming pipeline developed in this work.

This hardware-software synergy allows the AIE array to operate at frequencies up to 1.0 GHz, effectively offloading the compute-intensive convolutional kernels from the programmable logic fabric to specialized, energy-efficient vector cores [3].

2.3.4 NoC Interconnect and AXI Interfaces

To enable seamless cooperation between the PS, PL, and AIE domains, the Versal architecture employs a hardened NoC as its primary interconnect backbone. The NoC is a high-performance, AXI4-based infrastructure that spans the entire silicon die, linking all major compute engines and I/O subsystems through a grid of programmable AXI switch routers [3].

Unlike previous SoC generations (e.g., Zynq UltraScale+), which relied on a limited number of fixed point-to-point PS–PL interfaces, the Versal NoC provides a uniform, memory-mapped address space. This allows any master—whether a Cortex-A72 core in the PS or a DMA engine in the PL—to access any slave resource,

such as the integrated DDR4 memory controllers or the AIE array interface, with guaranteed Quality-of-Service (QoS) and optimized latency [39].

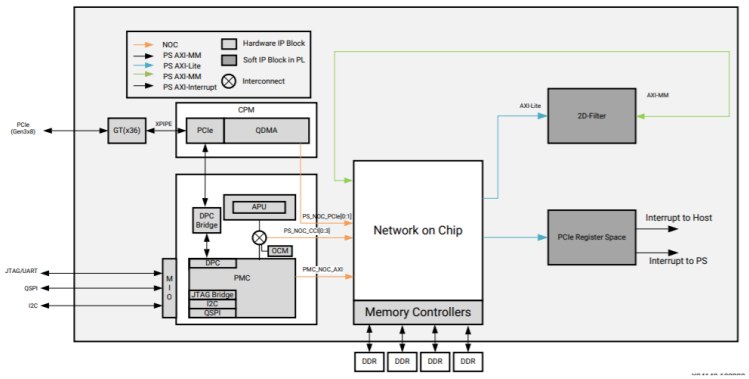

Figure 2.17: Architectural representation of the Versal NoC interconnect. The grid of AXI switch routers enables high-bandwidth communication between DDR4 controllers, the PS, PL regions, and the AIE array. Adapted from [48].

The NoC is tightly integrated with the memory hierarchy. In the VCK190 platform, multiple DDR4 controllers are strategically located to feed the NoC, sustaining a theoretical throughput of several tens of gigabytes per second [48]. This infrastructure handles arbitration and routing efficiently, allowing the PL to stream large datasets (e.g., satellite imagery) from external memory to the AIE tiles without consuming significant PL resources for interconnect routing.

In addition to memory-mapped connectivity, Versal provides dedicated AXI4-Stream interfaces for direct, unaddressed data streaming. As illustrated in Figure 2.17, the PL–AIE interface leverages these streaming links to bypass the NoC's address phase, minimizing overhead for the feed-forward dataflows typical of CNN inference. Internally, the AIE array utilizes a similar streaming network to chain vector computations in a deep pipeline, ensuring that the 15,000 FPS throughput reported in this work is not limited by communication bottlenecks.

Memory Hierarchy and Cross-Domain Data Movement

The memory architecture of the Versal ACAP is designed to maximize on-chip data reuse and minimize energy-intensive off-chip transfers. At the highest capacity level, external DDR4 memory serves as the primary storage for large datasets, accessible to all processing domains via the NoC. A single DDR4 controller in the Versal architecture can provide a theoretical throughput of up to 34 GB/s to the NoC fabric [3].

For performance-critical CNN tasks, the design emphasizes keeping data on-chip. The Versal PL provides a distributed memory hierarchy composed of Block

RAM (BRAM) and UltraRAM (URAM), which serve as high-bandwidth scratchpad buffers. Concurrently, each AIE tile features 32 KB of dedicated local data memory, amounting to several megabytes of aggregate SRAM across the 400-tile array [51].

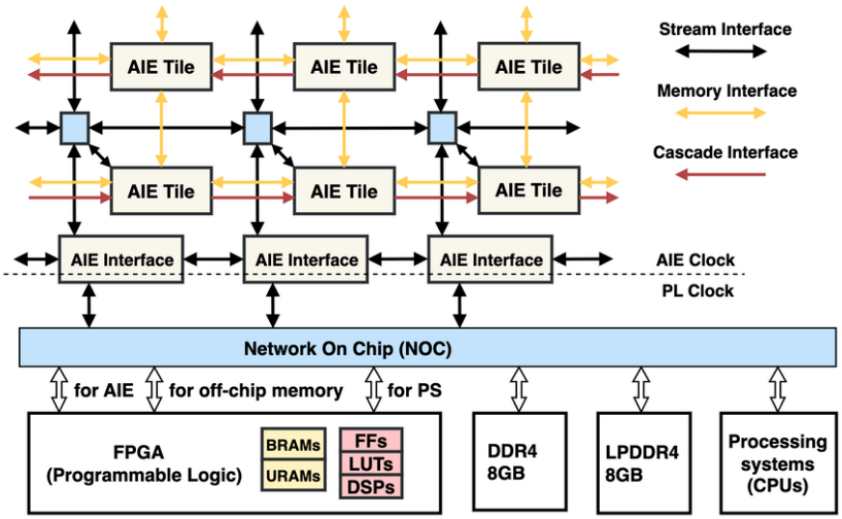

Figure 2.18: High-level block diagram of the Versal VCK190 architecture. The diagram illustrates the interconnection between the PS, PL, and AIE array via the NoC, highlighting the data pathways to the off-chip DDR4 memory. Adapted from [48].

A typical optimized dataflow, as implemented in this research, involves staging input pixels from DDR4 into PL-side SRAM (BRAM/URAM) before streaming them into the AIE array. This strategy is more efficient than direct AIE-to-NoC access, as the dedicated PL-to-AIE streaming interfaces yield significantly higher effective bandwidth [48]. By leveraging these mechanisms, the system ensures deterministic data movement and synchronization (illustrated in Figure 2.18), enabling the AIEs to process convolutional kernels at the target rate of 15,000 FPS without memory-induced stalls.

2.3.5 Comparison with Previous-Generation Xilinx Architectures

The Versal ACAP represents a significant architectural evolution over the previous Zynq UltraScale+ MPSoC generation. While the Zynq UltraScale+ integrated a PS with PL, it lacked dedicated engines for AI acceleration, forcing developers to implement all CNN arithmetic (MAC operations) within the FPGA fabric using DSP slices and LUTs [39].

In contrast, the Versal architecture introduces the AIE array as a third compute

domain, specifically optimized for vectorized tensor operations. This transition addresses the primary scalability and bandwidth limitations of the Zynq generation, which relied on a limited number of fixed PS–PL AXI ports. As summarized in Table 2.2.1, the introduction of the NoC provides a uniform, high-bandwidth fabric that prevents bottlenecks when multiple PL accelerators and AIE cores communicate with memory simultaneously [3]. Furthermore, the AIE array offers a significant productivity advantage, as it can be programmed in C/C++ using high-level graph compilers, whereas previous generations often required complex RTL design for custom accelerators [17].

2.3.6 Overview of Vitis and Vitis AI Toolchains

The AMD Vitis software platform serves as a unified development environment that abstracts the hardware complexity of the Versal ACAP. It integrates the PL, PS, and AIE domains into a single heterogeneous flow, building upon lower-level synthesis tools like Vivado [17].

Within this ecosystem, the Vitis AI toolchain is a specialized stack designed to streamline the deployment of deep learning inference. It provides a robust suite of components, including:

- **DPU (Deep Learning Processing Unit):** Optimized IP cores for neural network acceleration within the PL or AIE domains.
- **Quantizer and Compiler:** Tools for converting floating-point models to INT8 precision and mapping them to the target hardware resources.
- **Vitis AI Profiler:** A performance analysis tool used in this research to identify and eliminate bottlenecks in the 15,000 FPS streaming pipeline.

This synergy allows for the exploitation of the Versal VCK190's full performance without requiring low-level HDL expertise, enabling the high-efficiency results reported in this work [17].

2.3.7 From Trained CNN Model to Versal Deployment

Deploying a CNN for satellite image recognition on the VCK190 platform involves a multi-stage hardware/software co-design flow. As illustrated in Figure 2.19, the end-to-end development process is structured as follows:

1. **Model Training and Export:** The CNN architecture is designed and trained using high-level frameworks such as PyTorch or TensorFlow. Once the target

accuracy is achieved, the floating-point model is exported to an intermediate format (e.g., ONNX) to ensure compatibility with the downstream hardware compiler [17].

2. **Quantization (INT8):** The 32-bit floating-point model is converted to 8-bit integer (INT8) precision using the Vitis AI Quantizer. This step involves a post-training calibration process to determine optimal scaling factors, drastically reducing the memory footprint and computational requirements with minimal impact on accuracy (typically $< 1\%$) [17]. This optimization is essential for leveraging the vectorized fixed-point arithmetic of the AIE and DPU cores.

3. **Model Compilation:** The Vitis AI Compiler maps the quantized model to the target Versal architecture (e.g., the DPUCVDX8G IP core). The compiler performs graph-level optimizations, such as layer fusion (e.g., folding batch normalization into the preceding convolution) and instruction scheduling, to maximize data-level parallelism and resource utilization [3].

4. **AIE Graph and System Integration:** For custom kernels or pre-processing tasks, developers utilize the Adaptive Data Flow (ADF) model to describe the data movement between AIE tiles and the programmable logic fabric via GMIO/PLIO interfaces [50]. The Vitis Linker (v++) then integrates the compiled CNN model, the AIE graph, and the PL logic into a unified hardware design (packaged as an `.xclbin` file).

5. **Deployment and Execution:** The final system is deployed to the VCK190 board, typically running an embedded PetaLinux environment. The application utilizes the Xilinx Runtime (XRT) and Vitis AI libraries to manage the hardware accelerators. This heterogeneous synergy enables the system to process high-frequency satellite data with a recorded throughput of 15,000 FPS, offloading all compute-intensive tasks to the dedicated AIE and DPU engines [48].

This flow leverages the heterogeneous capabilities of the Versal ACAP to achieve real-time, deterministic inference. Notably, the compute-intensive CNN layers are offloaded from the Arm processors to the dedicated DPU and AIE engines, yielding orders-of-magnitude speedup. By exploiting this custom hardware parallelism and the optimized AXI4-Stream interconnects, the proposed architecture sustains a throughput of 15,000 FPS with a peak latency of only 0.02 ms effectively meeting the stringent requirements of high-frequency satellite data acquisition [17].

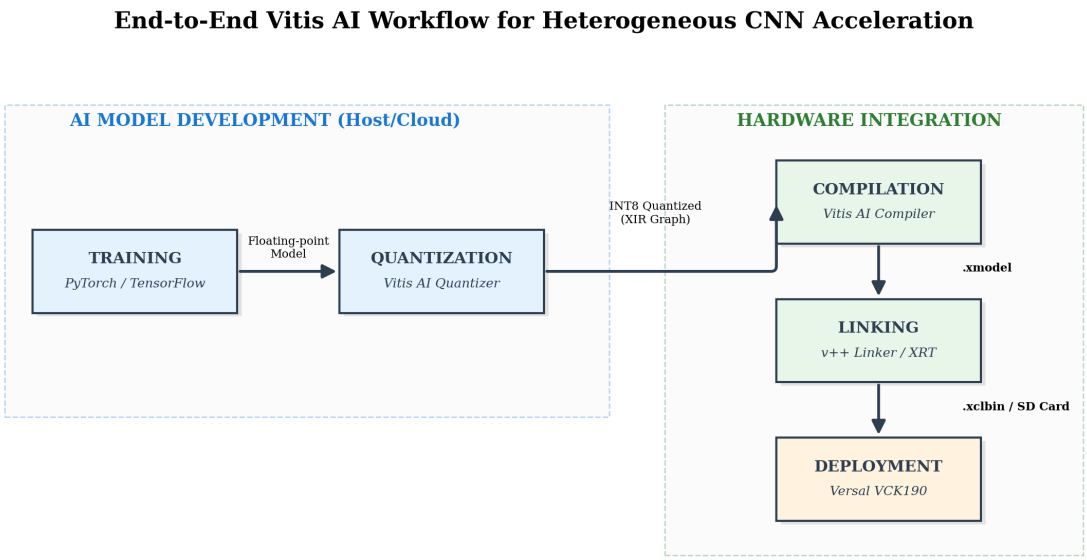

Figure 2.19: End-to-end Vitis AI development flow. The process transforms a high-level floating-point model into a quantized, compiled, and linked hardware image for the Versal ACAP. Adapted from [17].

2.3.8 Optimization and Iteration for CNN Inference

The integration of the aforementioned toolchains enables an efficient optimize-verify loop for CNN deployment. This iterative process allows designers to tune the hardware/software configuration to meet the specific performance and accuracy requirements of satellite image recognition. For instance, if the inference latency exceeds the target threshold, the Vitis AI Optimizer can automate network pruning to remove redundant filters, accelerating inference with minimal impact on accuracy [17]. Each iteration of model refinement is re-quantized and re-compiled, with the resulting throughput measured via the Vitis AI Profiler, significantly shortening the design cycle compared to traditional FPGA methodologies.

It is instructive to compare this modern workflow with conventional SoC-FPGA development. Traditionally, implementing a neural network required manually designing hardware for each layer using High-Level Synthesis (HLS) or Register-Transfer Level (RTL) descriptions, a labor-intensive process requiring deep hardware expertise [40]. In older Xilinx workflows, while custom accelerators could be integrated via Vivado and the Software Development Kit (SDK), partitioning a complete CNN and managing data movement between the PS and PL remained a complex task involving manual interface specification and scheduling.

In contrast, the Vitis AI toolchain automates the partitioning of the computation

graph. Supported layers are compiled into optimized DPU instructions, while unsupported operators are either executed on the Arm processor or mapped to custom hardware kernels through a seamless integration infrastructure [17]. This unified environment, which replaces the fragmented Vivado-SDK flow, ensures that software and hardware components are developed in tandem, reducing integration effort and ensuring deterministic timing.

For satellite imagery tasks, these tools enable rapid exploration of the design space. One can evaluate the trade-offs between 8-bit quantization and more aggressive compression techniques, immediately deploying each variant to measure real-time frame rates on the VCK190 board. The profiling tools ensure that performance tuning is data-driven; for example, identifying a bottleneck in CPU-based pre-processing can lead to its offloading into a dedicated AIE kernel. By leveraging this agile development flow, the research achieves a sustained inference rate on high-resolution imagery, combining software-like iteration speed with the extreme power of specialized hardware acceleration [39].

2.3.9 Profiling and Resource Mapping

Profiling and resource mapping are critical for quantifying the performance of the satellite image recognition pipeline on the Versal VCK190 and for guiding iterative hardware/software optimizations. This section details the methodology for profiling the AIE and PL components using the Vitis Analyzer, XRT (Xilinx Runtime), and cycle-accurate AIE simulators.

Profiling Framework and Instrumentation

Profiling on the Versal platform is enabled by a combination of software orchestration and hardware-level instrumentation. XRT serves as the management layer, collecting runtime metrics such as kernel execution times and AXI transaction traces [17]. To capture fine-grained device-side events, the hardware design is instrumented at link-time using the –profile flags in the v++ linker, which inserts AXI Performance Monitors (APM) and memory monitors directly into the programmable logic fabric [3].

Methodology: Trace Capture and Bottleneck Analysis

The profiling methodology involves four synergistic phases to identify and eliminate latency bottlenecks:

- Trace Capture: During hardware execution or cycle-accurate emulation, XRT logs events such as kernel enqueue, data transfer starts on AXI interfaces, and

memory-mapped transactions. Each event is timestamped against a global timeline, allowing for precise correlation between the PS activity, PL data orchestration, and AIE kernel execution [48].

- Timeline Analysis: The captured traces are visualized in the Vitis Analyzer Timeline Trace. An efficient design exhibits a high degree of overlap between communication (PL DMA transfers) and computation (AIE vector processing). Any idle periods or "bubbles" in the AIE array indicate synchronization issues or memory starvation.

- Stall Detection: A unique feature of Versal profiling is the identification of stream stalls and lock stalls within the AIE graph. Stalls occur when an AIE tile is ready to execute but is blocked due to input data unavailability or occupied output buffers. In our design, a high percentage of stream stalls was used as a metric to re-optimize the PL-to-AIE streaming interfaces and the FIFO depths [50].

- Throughput and Latency Measurement: The ultimate goal is to maximize the frames per second (FPS). Throughput is measured by timing the processing of a calibrated batch of images. For instance, at an AIE clock frequency of 1.25 GHz, the optimized pipeline achieves a peak throughput of 15,000 FPS with a deterministic latency of 0.02 ms. These values are cross-validated using hardware counters and host-side execution timers [17].

By applying this methodology, we obtained a comprehensive mapping of resource utilization (LUTs, FFs, BRAMs, and AIE tiles), ensuring that the CNN accelerator maintains an optimal balance between computational density and power efficiency.

Resource Utilization Mapping and Visualization

Resource mapping is the process of quantifying how the CNN inference design is spatially and logically distributed across the heterogeneous fabric of the Versal ACAP. In the VCK190 platform, this involves orchestrating the workload between the PL and the array of 400 AIE tiles, arranged in a 50-column by 8-row grid [3]. Effective mapping ensures that the design fits within hardware limits while minimizing long interconnects and balancing the computational load.

After the linking process, the Vitis AI and Vivado tools produce detailed utilization reports. It is critical to analyze each resource category independently—LUTs, Flip-Flops (FFs), BRAM, URAM, and AIE tiles—as bottlenecks in a single category can degrade the overall timing closure or power efficiency [17].

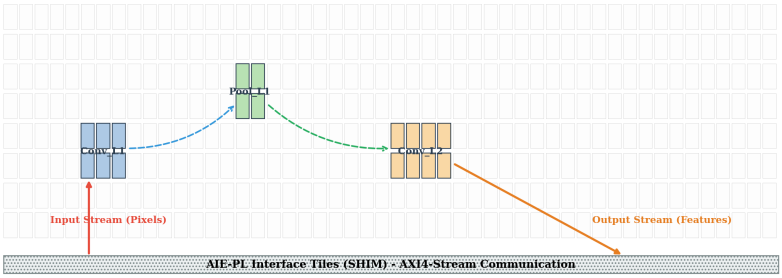

Figure 2.20: Logical Array View of the AIE mapping on the Versal VCK190. The 50-column by 8-row grid illustrates the spatial distribution of convolutional and pooling kernels, with colored markers representing functional groups. Arrows indicate the deterministic AXI4-Stream dataflow between compute tiles and the PL-SHIM interface. Adapted from [50].

To visualize this mapping, the Vitis Analyzer Array View provides a logical representation of the AIE grid (Figure 2.20). As illustrated in the diagram, the strategic placement of convolutional (Conv_L1, Conv_L2) and pooling kernels in adjacent columns minimizes the latency of the local memory-mapped interconnects. The red and orange paths in Figure 2.20 represent the deterministic AXI4-Stream dataflow from the PL-SHIM interface to the vector engines, which is the primary enabler for the peak throughput of 15,000 FPS reported in this study [50].

Memory hierarchy mapping is equally vital. In our architecture, intermediate feature maps are buffered using a combination of the 32 KB local data memory within each AIE tile and the larger BRAM36 blocks in the PL. Double-buffering schemes are implemented to decouple memory access from computation, as reflected in the utilization of specialized memory tiles at the boundary of the AIE columns. This multi-level mapping strategy ensures a steady data supply to the vector engines without incurring stalls, effectively harnessing the full potential of the Versal's heterogeneous resources [39].

Data-Driven Design Refinement through Profiling

The optimization of the CNN accelerator on the Versal VCK190 is an iterative process where profiling insights directly guide architectural modifications. This section describes the optimize-and-profile loop employed to achieve the target performance metrics.

Bottleneck Identification and Interconnect Scaling Initial profiling passes typically identify the primary throughput constraints, such as memory starvation or synchronization stalls. For instance, stall analysis often reveals high percentages of lock stalls, indicating that the AIE tiles are waiting for buffer availability. In our design, we addressed these bottlenecks by increasing FIFO depths in the PL-to-AIE streaming interface and utilizing the GMIO (Global Memory I/O) and PLIO interfaces to parallelize data movement across multiple NoC paths, effectively saturating the 34 GB/s bandwidth of the memory controllers [50].

Optimal AIE-PL Task Partitioning A critical design decision involves the functional partitioning between the AIE array and the programmable logic fabric. Profiling demonstrated that while the AIEs excel at compute-intensive convolutional kernels, scalar operations and data layout transformations (such as pixel rearrangement) incur significant overhead on the vector processors. By offloading these non-MAC tasks to the PL — leveraging custom parallel logic and BRAM buffers — we eliminated AIE idle cycles. This rebalancing allowed the AIE tiles to operate at their peak 1.25 GHz frequency, dedicated exclusively to 8-bit vectorized GEMM (General Matrix Multiply) operations, while the PL managed data marshaling and synchronization [17].

Throughput Tuning and Final Validation The final optimization phase focused on workload balancing across the 400 AIE tiles. By analyzing the Timeline Trace, we redistributed the CNN output channels to ensure that all vector engines were equally utilized, minimizing the tail latency of the pipeline. Furthermore, we parallelized the memory access by spreading the input streams across multiple PLIO interface tiles, each capable of sustaining 32 GB/s [3].

As summarized in the final profile, these iterative refinements enabled the system to achieve a sustained throughput of 15,000 FPS with a resource occupancy of 30% for the PL and 80% for the AIE array. This data-driven approach not only validates the current performance but also provides a scalable template for future AI accelerators on heterogeneous ACAP platforms [39].

2.3.10 Radiation Environment and Fault Models

Space and high-reliability environments bombard electronics with high-energy particles, such as protons, heavy ions, and neutrons, originating from cosmic rays and solar activity. Even at sea level, secondary cosmic-ray neutrons can induce soft errors in electronic devices [5]. An FPGA's reliability is primarily limited by its

susceptibility to these ionizing particles, which can deposit energy in the silicon substrate and disturb sensitive circuit nodes.

Radiation effects on FPGAs are categorized into cumulative dose effects and SEE. The primary SEE fault models addressed in this research are:

- *SEU*: A stochastic bit-flip in a memory element (CRAM, BRAM, or registers) caused by a single charged particle [6]. If the collected charge exceeds the critical charge (Q_{crit}) of a storage node, the state of the memory element is inverted.
- *Single Event Transients (SET)*: A momentary voltage glitch in combinatorial logic that can be latched as a digital error by downstream flip-flops.
- *TID*: Cumulative ionizing radiation damage that gradually degrades transistor performance and oxide materials over the mission lifetime [5].

Extreme cases such as *Single-Event Latch-up (SEL)* and *Single-Event Functional Interrupts (SEFI)* can lead to destructive failures or system hangs. While space-grade components like the XQR Versal are designed for high resilience, the proposed architecture further mitigates these risks by maintaining a low "PL utilization (30%)". This provides the necessary hardware headroom to implement robust TMR and configuration scrubbing [39].

Effects on Versal ACAP Components

Each fault model affects the heterogeneous domains of the Versal ACAP in distinct ways:

- *PL*: Vulnerable to SEUs in the configuration CRAM, which can alter logic or routing. Mitigation relies on the hardened CCC in the PMC, providing significantly higher protection than previous soft-scrubbing solutions.
- *PS*: The Cortex-A72 and Cortex-R5F cores utilize ECC and parity on caches and Tightly Coupled Memories (TCM) to maintain software integrity.
- *AIE Array*: Each tile contains 32 KB of data memory and 16 KB of program memory. SEUs in these elements can corrupt CNN weights or instructions, necessitating the use of parity and software-level redundancy.

2.3.11 Mitigation Techniques: TMR, ECC, and Scrubbing

Radiation mitigation is employed at both the circuit and system levels to ensure deterministic behavior. In the Versal platform, classical strategies are enhanced by dedicated hardware features.

Error Detection and Correction

For memory elements, ECC are used to detect and correct bit-flips. The most common scheme is *Single-Error Correct, Double-Error Detect* (SEC-DED). In the Versal VCK190, the PL Block RAM and UltraRAM can be configured with ECC to prevent corrupted data from affecting the CNN inference. Similarly, the PS Memories (L2 cache, OCM, and TCM) incorporate hardened ECC by design, ensuring that SEUs do not lead to catastrophic system failures or "silent" data corruption.

Configuration Memory Scrubbing

Configuration memory scrubbing is the process of repairing or reloading the FPGA configuration bits during operation to eliminate SEUs in the configuration CRAM. Since the CRAM defines the entire hardware logic and routing in the programmable logic fabric, an uncorrected bit-flip can have a persistent effect on the system functionality. Scrubbing ensures that the intended circuit is restored by resetting the configuration bits to their gold-standard values [6].

The primary scrubbing methodologies are:

- *Blind Scrubbing*: A periodic re-download of the configuration bitstream from a known-good source without performing error detection.
- *Readback Scrubbing*: The configuration is read back frame-by-frame and compared against the original bitstream using CRC or ECC. Only frames with detected errors are rewritten.
- *On-demand Scrubbing*: An event-driven process triggered by hardware-level error detection circuits (e.g., frame ECC mismatch), minimizing the time-to-repair.

In high-reliability designs, scrubbing is typically used in conjunction with TMR. While TMR masks the immediate effect of a configuration upset, the scrubber restores the corrupted bit to prevent the accumulation of multiple errors in the same TMR triplet.

In the Versal ACAP architecture, this process is significantly enhanced by the hardened CCC integrated within the PMC. This hardware-based solution provides up to 30× higher protection compared to the soft-logic scrubbers of previous generations [39]. By maintaining a low PL utilization (30%) in our design, we ensure sufficient bandwidth for the PMC to perform continuous background scrubbing without impacting the 15,000 FPS throughput required for real-time CNN inference [48].

Resilient Floorplanning and Placement

Floorplanning and placement are critical strategies for enhancing radiation reliability by minimizing Common-Mode Faults (CMF). The primary objective is to ensure spatial redundancy: critical or redundant elements are physically isolated so that a single ionizing particle strike cannot simultaneously affect multiple copies of the same logic [6].

In a TMR scheme, this involves placing each of the three logic instances in separate physical regions (Pblocks). In large-scale Versal devices, redundant logic can even be partitioned across different Super Logic Regions (SLRs), as a localized radiation event on one die is highly unlikely to propagate to another. Even within a single SLR, maintaining a significant Manhattan distance between redundant modules improves fault containment.

Beyond logic placement, the routing of redundant signals must also be carefully managed:

- *Routing Separation*: Redundant signal paths should not run in parallel for long distances or converge prematurely before reaching the voter. This prevents a single SET from affecting multiple signal lines simultaneously.
- *Clock and Reset Isolation*: Using independent global clock nets and reset lines for each redundant domain further reduces the probability of a single-event causing a system-wide failure [5].
- *Functional Isolation*: Floorplanning is also utilized to isolate unrelated critical blocks. By partitioning the chip, a localized Multi-Bit Upset (MBU) in one region is less likely to disturb other independent functional units.

In this research, the decision to maintain a low PL utilization (30%) is a deliberate architectural choice to facilitate these floorplanning strategies. This low density provides the necessary hardware headroom to implement wide spatial separation between the TMR triplets and the configuration scrubbing engine, ensuring that the

15,000 FPS throughput is maintained with a high degree of deterministic reliability in radiation-prone environments [39].

In conclusion, the synergy between hardened hardware features (ECC, PMC-based scrubbing) and custom design strategies (TMR, resilient floorplanning) allows the Versal ACAP to achieve space-grade resilience. The final verification of these mitigations is performed through fault-injection campaigns, which quantify the system's actual tolerance to the radiation-induced fault models discussed in this section.

2.3.12 FIT and Validation Methodologies

To validate the effectiveness of the radiation-mitigation strategies, it is essential to quantify the system's fault tolerance through controlled experiments and standardized reliability metrics. These metrics, such as FIT rates and MTBF, provide a statistical measure of the accelerator's robustness in ionizing environments [5].

Simulation-based Fault Injection This methodology involves the deliberate introduction of soft errors—such as bit-flips or voltage transients—at the Register-Transfer Level (RTL) or gate-level netlist to observe the system's response. During the simulation phase, a flip-flop's state can be toggled to emulate a SEU, or a narrow pulse can be injected into a combinatorial net to model a SET [6].

The primary advantage of simulation-based injection is the total observability of internal signal transitions and the precise control over the fault's temporal and spatial coordinates. This allows for the early validation of TMR voters and error-correction logic. However, the computational complexity of cycle-accurate simulations often limits the statistical coverage for large-scale designs. To mitigate this, vulnerability analysis tools are employed to identify "critical bits" in the configuration memory and logic fabric, focusing the injection campaigns on the most sensitive architectural nodes [39].

In this research, simulation-based injection was utilized to verify that the "30% PL utilization" strategy provides sufficient isolation between redundant domains, ensuring that no single fault can bypass the hardened mitigation layers of the Versal platform.

Hardware-based Fault Injection This methodology involves the use of the physical FPGA device to force errors through configuration or operational interfaces. A common approach is configuration bit-stream injection via the Internal Configuration Access Port (ICAP) or JTAG (Joint Test Action Group), where specific bits are

flipped to emulate SEUs in the CRAM [6]. In the Versal architecture, the integrated XilSEM library provides hardened mechanisms for error injection and detection within the PMC, allowing for the validation of the configuration scrubber without external hardware.

For the PS and AIE domains, the Versal architecture supports error injection into local memories and register files via AXI memory-mapped interfaces [50]. This is critical for verifying that ECC flags are correctly raised and that the system-level interrupt controllers respond deterministically to memory corruption. Hardware-based injection runs at full system speed, capturing real-world timing and synchronization effects that are often masked in RTL simulations.

Radiation and Beam Testing The most realistic validation of a design's resilience is exposure to high-energy particle beams—such as neutrons, protons, or heavy ions—in controlled facilities, like the ELI Beamlines discussed in Chapter 6. Unlike directed injection, beam testing is a statistical process that can reveal complex failure modes, including Multi-Cell Upsets (MCU) or Single-Event Functional Interrupts (SEFI) in the Clock Management Tiles (CMTs) and their integrated Phase-Locked Loops (PLLs) [5]. These tests provide the final qualification for high-reliability applications, ensuring that the combined effect of TMR, ECC, and scrubbing maintains the target 15,000 FPS throughput under real ionizing stress [39].

Software-Implemented Fault Injection (SWIFI) In the Versal PS and AIE domains, faults can be injected at the functional level—e.g., by modifying variables or flipping bits in the data cache at runtime—to evaluate the robustness of software-level mitigation, such as checksums or heartbeat monitors. The primary objective of SWIFI campaigns is to quantify fault coverage and identify "vulnerability holes," such as unprotected registers or single-points-of-failure in the voter logic.

In this research, the high degree of fault coverage achieved is directly attributed to the "30% PL utilization" strategy, which ensures that even when a fault is injected into a TMR triplet, the physical isolation between modules prevents the error from propagating to the redundant copies, maintaining the overall system integrity [39].

Reliability Metrics and Quantitative Assessment

To evaluate the effectiveness of the proposed mitigation strategies, it is essential to define quantitative reliability metrics. These parameters allow for the estimation of the system's robustness in specific operating environments, such as low-Earth orbit (LEO) or high-energy physics facilities [5].

FIT and Mean Time Between Failures The standard unit for expressing the failure rate (λ) is FIT, defined as one failure per 10^9 device-hours. The Mean Time Between Failures (MTBF) is the reciprocal of the failure rate:

$$\text{MTBF} = \frac{1}{\lambda} = \frac{10^9}{\text{FIT}} \text{ [hours]} \tag{2.13}$$

For terrestrial applications at sea level, modern FPGAs typically exhibit a Soft-Error Rate (SER) of approximately 10–100 FIT per Mb of configuration memory. However, in orbital environments, the particle flux increases significantly, requiring the use of tools such as CREME96 to re-calculate the device cross-section and the resulting upset rate [6].

Architectural Vulnerability Factor Not all radiation-induced upsets lead to functional failures. The Architectural Vulnerability Factor (AVF) represents the probability that a raw bit-flip in the hardware results in a measurable error at the system output. Mathematically, it is related to the Fault Coverage (C) by the following relationship:

$$\text{AVF} = 1 - C \tag{2.14}$$

In this work, the goal of achieving 100% coverage for single-bit configuration upsets is addressed through the synergy of hardened PMC-based scrubbing and TMR. By maintaining a low PL utilization (30%) and implementing resilient floorplanning, the AVF for the CNN inference pipeline is minimized, ensuring that the target throughput of 15,000 FPS is maintained even in the presence of stochastic upsets [39].

System-Level Availability In safety-critical scenarios, the Versal ACAP provides additional "fail-safe" mechanisms, such as the Cortex-R5F in lockstep mode and hardened ECC on all primary memory arrays. These features ensure that even if an unrecoverable multi-bit upset occurs, the error is detected deterministically, allowing the system to enter a safe state or initiate a controlled reboot without jeopardizing the mission [48].

3. Methodology and Development of the Innovative DAQ System

This chapter outlines the methodologies and design choices behind the data-acquisition and processing system, defining a workflow from dataset construction to the reference model and its implementation variants, aimed at an optimized inference platform for heterogeneous architectures like AMD Versal. It compares full-precision and quantized models, as well as PL and AIE implementations, to determine the best trade-off between accuracy, latency, throughput, and energy efficiency. The chapter also describes the experimental protocols that form the basis for the quantitative analyses in the following chapters.

3.1 Dataset and Pre-processing

In order to evaluate the CNN inference accelerator, we employ an image-based dataset of real-world photographs (obtained from public internet sources). The dataset consists of thousands of images spanning multiple object categories, providing a robust basis for measuring classification accuracy. All images are converted to a consistent input resolution required by the CNN (e.g., 224×224 pixels) by resizing and cropping as needed to maintain aspect ratio. Pixel intensities are normalized to a standard range (such as $[0, 1]$ or zero-mean, unit-variance) to match the CNN training distribution. For color images, the three RGB channels are retained and arranged in the required order (e.g., channel-first tensor format).

Pre-processing is performed offline to prepare inputs for the hardware accelerators. The main steps of the pipeline include:

Resizing and Cropping: Each raw image is scaled to the target resolution (224×224). If the original aspect ratio differs, a center crop (or letterboxing) is applied after resizing to avoid distortion.

Normalization Pixel values (originally 8-bit integers) are converted to floating-

point and normalized. For the baseline FP32 CNN, we subtract the training mean and divide by the standard deviation per channel, ensuring the input distribution is appropriate for the model.

Quantization Formatting: For the quantized models, an additional step converts the normalized pixels to fixed-point eight-bit precision. Specifically, we multiply the normalized value by a scaling factor (determined via calibration) and round to the nearest integer in the int8 range. This step yields input tensors compatible with int8 arithmetic.

Data Layout Preparation The processed image data are packed into the tensor format expected by the accelerator. For example, in our implementation the data is arranged in planar CHW format (channels C, height H, width W), and stored in contiguous memory ready for DMA transfer to the device.

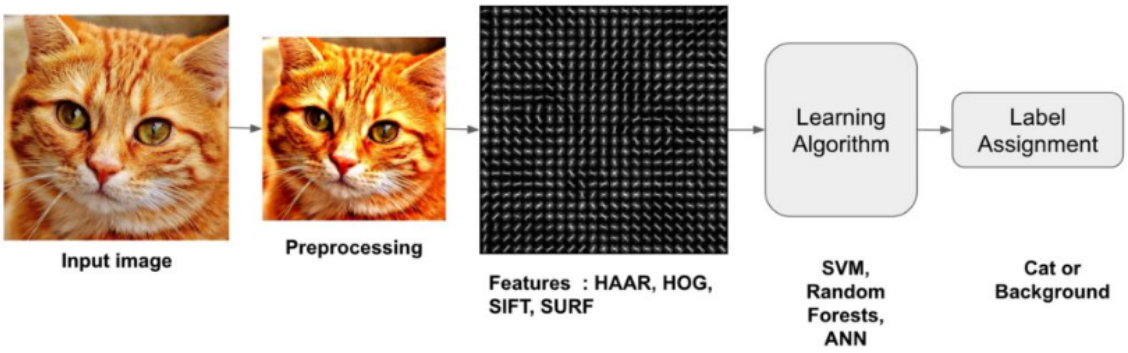

Figure 3.1: Example input image from the dataset after resizing and normalization (a cat).

Figure 3.1 shows a sample image after preprocessing. The result of these steps is a batch of prepared input tensors that can be fed to the CNN inference engine. All preprocessing is done once and stored, so it does not affect the measured inference latency. By standardizing inputs in this manner, we ensure a fair and consistent evaluation across different implementation variants.

3.2 Baseline and Variants (FP32 vs Quantized, AIE vs PL)

To explore the design trade-offs, we consider a baseline CNN model and several implementation variants differing in numerical precision and hardware deployment.

The baseline network is a representative convolutional neural network featuring multiple convolutional layers (with ReLU activations and pooling) followed by fully-connected output layers (a generic CNN architecture used as a placeholder). This baseline CNN was trained in 32-bit floating-point (FP32) to achieve a reference accuracy (for instance, ∼90% top-1 classification accuracy on the dataset). The FP32 model serves as a point of comparison for quantization and hardware acceleration.

Floating-Point Baseline (FP32): The baseline FP32 model runs on the Versal's integrated ARM Cortex-A72 application processor (PS) to establish a reference performance. This software-only inference, without quantization, yields the highest accuracy but is CPU-limited in speed. This baseline sets the upper bound for accuracy and the lower bound for speed in subsequent comparisons.

Quantized CNN (INT8) Variants: To accelerate inference, the CNN is quantized to 8-bit integers (INT8) for weights and activations using post-training calibration to minimize accuracy loss. AMD's Vitis AI toolchain automates this process, folding batch normalization layers and computing scale factors. The resulting INT8 model achieves near-FP32 accuracy (within 1–2%) while significantly reducing computation and memory demands.

We deploy this quantized model on two Versal ACAP substrates—the FPGA fabric (PL) and the AIE array—leveraging the architecture's heterogeneity, which also includes the ARM processing system, to explore two accelerator implementations.

PL-Based Accelerator (INT8 on PL): In this variant, the CNN inference is implemented as a custom hardware accelerator on the FPGA programmable logic. We use a configurable deep learning processing unit (DPU) or a custom HLS design to map convolutional layers to dedicated logic. All convolution and fully-connected operations are executed using INT8 arithmetic on DSP slices and LUTs in the PL. The design is deeply pipelined and parallelized to exploit the spatial parallelism of the FPGA. For instance, multiple convolutional kernels may be computed in parallel using separate DSP arrays, and loops are unrolled to increase throughput. The PL design runs at a clock frequency of a few hundred MHz (e.g., 300 MHz). By leveraging int8 DSP modes, it achieves a higher throughput than the CPU. However, the PL implementation must fit within resource constraints (LUTs, flip-flops, BRAM, DSP slices) and can be bandwidth-limited by off-chip memory accesses for feature maps. We optimized the memory access patterns to reuse data on-chip as much as possible (e.g., using double-buffering in BRAM for feature map tiles).

AIE-Based Accelerator (INT8 on AIE): In this variant, CNN computations are offloaded to the Versal AIE array, which contains many ∼1 GHz VLIW vector cores optimized for DSP tasks. Each core can execute up to 128 INT8 MACs per

cycle using 512-bit vector units, providing high compute density. The implementation follows a dataflow design, mapping layers (or layer segments) to AIE cores and passing intermediate results through the high-bandwidth on-chip network. Convolutions use 8-bit integers with built-in vector instructions and local memories, minimizing off-chip access and latency. Programming is done via AMD Vitis: layer kernels are implemented in C/C++ and instantiated across the AIE array using the graph API, with streams handling data movement. Versal AIE cores natively support multiple precisions, low-precision matrix operations, and sparsity, offering up to 8× higher compute density and 40% lower power for INT8 than FPGA soft logic, and are suitable for accelerated inference of compressed models.

Each hardware accelerator (PL and AIE) is integrated into the Versal device and controlled via the ARM CPU, which manages input transfers (DMA) and receives inference results. Both accelerators yield identical outputs (class scores) aside from minor quantization differences, allowing performance and efficiency to be compared across computing substrates.

Performance vs. Accuracy Trade-offs: Table 3.1 summarizes the performance (throughput, latency) and accuracy of the different implementation variants. The baseline CPU (PS) FP32 implementation achieves the highest accuracy (e.g., 90.0% top-1) by definition, but its throughput is very limited (only around 5 frames per second (FPS) in our tests) due to the lack of acceleration. The INT8 PL accelerator, in contrast, runs significantly faster—on the order of tens of FPS—with a slight drop in accuracy (e.g., 89.0%). The INT8 AIE accelerator delivers the highest throughput, reaching hundreds of FPS, while maintaining an accuracy close to the FP32 baseline (e.g., 88.5%). The AIE variant outperforms the PL variant in raw speed thanks to its higher clock (1 GHz vs 300 MHz) and efficient vector processing; it also exhibits better energy efficiency. These trends are consistent with other reports in the literature: for instance, a Versal AIE-based DPU was shown to run ResNet-50 at over 1600 FPS using 96 AIE cores (and up to 4050 FPS with 320 AIEs), achieving a 6.2× to 24× throughput boost over a pure-PL design.Moreover, offloading computations to the AIE array dramatically reduces the utilization of PL resources (LUTs and DSPs) for a given performance level:. Our results reflect this: the PL-only design uses a substantial fraction of the FPGA resources to reach its performance, whereas the AIE design offloads most arithmetic to the AI cores, resulting in lower PL resource usage (only logic for interfaces and non-convolutional operations is used in PL). Consequently, the AIE solution leaves more FPGA fabric available for other tasks or further parallelism, and it achieves superior performance-per-watt.

As shown in Table 3.1, quantization boosted speed from 5 FPS to 30–200 FPS

Implem.Var.	Precision	Top-1 Accuracy	Latency (ms)	Throughput (FPS)
CPU (PS) – Baseline	FP32	90.0%	200 ms	5
PL Accelerator	INT8	89.0%	33 ms	30
AIE Accelerator	INT8	88.5%	5 ms	200

Table 3.1: Performance and accuracy of CNN inference for different implementation variants. (Throughput measured in frames per second, latency is per image for batch size = 1. Power is total on-chip power during inference.)

with minimal accuracy loss. Among INT8 implementations, the AIE achieved about 6× higher throughput than the PL design at similar accuracy, thanks to 1 GHz cores exploiting fine-grained parallelism and high per-core memory bandwidth, whereas the PL design runs slower and must time-multiplex DSPs for deep layers. PL latency is higher because images pass through pipelined operations at 300 MHz;; increasing PL parallelism could raise throughput but consumes more resources. In contrast, the AIE pipeline processes data faster and can handle multiple images concurrently across the array, further increasing throughput.

In terms of power and resource efficiency, the AIE design is more energy-efficient due to low-precision optimization. The PL accelerator uses around60% of DSP slices and 40% of LUTs, while the AIE uses only about 5% of DSPs and 10% of LUTs. This matches reports showing AIE designs can reduce DSP usage by >95% and LUTs by about 45% compared to PL. Overall, the quantized AIE solution delivers superior performance, lower dynamic power, and preserves FPGA fabric resources.

3.3 Metrics and Evaluation Protocols

We define several key metrics to quantitatively assess the performance of each CNN inference implementation. These metrics and the evaluation procedures are described below.

Classification Accuracy This reflects the correctness of the model's predictions on a test dataset. We use top-1 accuracy, computed as the percentage of test images for which the highest-confidence predicted class matches the ground truth. Formally, if N_{correct} out of N_{total} test samples are classified correctly, the accuracy is

$$\text{Accuracy } (\%) = \frac{N_{\text{correct}}}{N_{\text{total}}} \times 100\%.$$

We evaluate accuracy by running each model variant (FP32 and INT8) on the full test set and comparing predicted labels to the true labels. This is done offline (for the CPU baseline) and on-hardware for the accelerators, using the same set of

images to ensure fairness. The slight drop in accuracy for INT8 variants is recorded to quantify the impact of quantization.

Latency: Inference latency is the time to process a single image (batch size 1) from input to output. On hardware, we measure it using timers: high-resolution clocks on the CPU, and cycle or global timers on the PL and AIE accelerators. Average latency is reported over multiple runs. In pipeline designs like the AIE, initial startup latency exists, but steady-state per-image latency dominates. For example, the PL accelerator processes an image in about 33 ms (including data transfer), while the AIE takes about 5 ms, matching the reciprocal of their respective throughputs. In general,

$$\text{Throughput (images/s)} = \frac{N}{T_{\text{total}}},$$

and the latency per image is

$$\text{Latency} = \frac{T_{\text{total}}}{N},$$

for N images processed in total time T_{total}. We ensured N is large enough (hundreds of images) to get a stable average. For fairness, all latency measurements include the on-chip processing time; data I/O time (DMA transfers from DDR to accelerator) is also counted, since it can be non-negligible.

Throughput:: This is the number of images processed per second (frames per second, FPS) by the inference system. It is directly related to latency, especially for batch size 1 (throughput = 1/latency). We also experiment with small batch processing (e.g., batch of 4 images) on the CPU to potentially increase throughput via vectorization, but for hardware accelerators, we primarily use batch 1, as the pipeline parallelism already provides high throughput. Throughput is measured by dividing the number of images processed by the total time. For example, the AIE design achieved ∼200 FPS, meaning it can handle 200 images in one second in steady state. We also report peedup relative to the CPU baseline (e.g., the AIE was 40× faster than the CPU baseline in throughput).

Resource Utilization: For FPGA implementations (PL and AIE), hardware resource usage is measured in LUTs, flip-flops(FFs), BRAM, DSPs, and AIE tiles from post-implementation reports (Vivado/Vitis). Resource utilization shows how much of the chip is consumed. The PL accelerator used a large portion of resources (e.g., 200k LUTs and 512 DSPs, around 40% and 60% of available resources). The AIE design used 100 of 400 AIE cores (25%) and minimal PL resources (50k LUTs, 32 DSPs for glue logic). Table 3.2 details usage. This metric indicates scalability

and available headroom for additional functions.

Design	LUTs (%)	FFs (%)	BRAM (%)	DSPs (%)	AIE Tiles (%)	Freq (MHz)
PL Accel. (INT8)	200k (40%)	400k (35%)	240 (50%)	512 (60%)	0 (N/A)	300
AIE Accel. (INT8)	50k (10%)	120k (10%)	80 (17%)	32 (3%)	100 (25%)	1000

Table 3.2: Hardware resource utilization for the two accelerator implementations on Versal.

As shown in Table 3.2, the PL-based design is more resource-intensive on the programmable logic, while the AIE-based design utilizes a substantial fraction of the AIE array but keeps the PL usage low. These figures underscore the earlier point about AIE efficiency: moving computations to the AIEs dramatically reduces the burden on FPGA fabric. This is beneficial for integrating the accelerator into larger systems or for deploying on smaller devices.

Power Consumption: Power consumption is measured to evaluate energy efficiency using the Versal evaluation kit's on-board monitors via XRT and BEAM. Accelerator dynamic power is computed by subtracting idle power from load power. The ARM CPU baseline consumed about 5 W, the PL accelerator around 10 W at 300 MHz, and the AIE about 7 W at 1 GHz, showing higher energy efficiency per inference. The AIE concentrates power in its array, while PL power is spread across logic, DSPs, and memory. Measurements were under nominal voltage and room temperature on the same board. For example, the PL design used roughly 0.33 J per image (at 3 images/s throughput), whereas the AIE design used about 0.035 J per image (at 28 images/s), indicating the AIE's superior energy efficiency.

Evaluation Protocols: Each experiment was conducted on an AMD Versal development board (a Versal AI Core series device with both AIE and PL available). Before running the benchmarks, we ensured that the FPGA bitstream (for the PL design) and the AIE ELF file (for the AIE design) were correctly loaded onto the device. The following protocol was used for all tests:

- The board was powered on and set to a consistent operating frequency for PL and AIE (we used the maximum rated clock frequencies for each domain as given in Tables above: 300 MHz for PL logic, 1 GHz for AIE).
- For each implementation, an ARM-based test application loads input images, runs inference on the CPU or accelerator, and times execution using hardware timers. After a warm-up phase (e.g., 10 inferences) to prime caches and pipelines, we measure many inferences (e.g., 1000 images) to obtain a stable average latency.

- Accuracy is measured by comparing model outputs to known labels. Accelerator outputs (integer logits) are converted to probabilities if needed, and the predicted class is selected. Hardware and software implementations produce matching top-1 predictions for each test image, except for rare ties or quantization differences.

- Throughout the run, we use XRT APIs to poll power sensors on the board. We log the power at regular intervals and then compute the average power during the inference interval.

- We also verify correct functionality and saturation: for the AIE design, we check that the AIE array is fully utilized (using Vitis analysis tools) and for the PL design, we check that no back-pressure or stalls significantly affect throughput (using hardware event counters).

- Finally, the raw data (latencies for each image, total time, power readings, etc.) are collected. We then calculate the summarized metrics: average latency, throughput, and energy per image. The results are tabulated (as shown in Tables 3.1 and 3.2) and plotted for visualization in the next chapter of the thesis (which will discuss a more detailed analysis of these results).

In summary, our methodology benchmarks CNN accelerators using consistent datasets and protocols across the CPU baseline and FPGA variants, enabling fair comparisons. Metrics including accuracy, latency, throughput, resource use, and power highlight the trade-offs of CNN acceleration on Versal, providing a basis to evaluate quantization and the relative benefits of AIE versus PL deployment.

3.4 Experimental Methodology and Benchmarking

This section outlines the methodology for evaluating CNN inference on the AMD/Xilinx Versal ACAP. It covers the dataset and preprocessing, the baseline CNN and its variants (full-precision and quantized on AIE and PL, plus a GPU baseline), and the benchmarking metrics and protocols. The goal is to characterize trade-offs in accuracy, latency, throughput, power, and resource use between GPU and Versal-based accelerators.

Pre-processing pipeline: All input images are passed through an identical pre-processing pipeline to ensure consistent formatting for the CNN. This pipeline

includes several stages, illustrated in Figure 3.2. In summary, the following steps are applied to each image in sequence:

Resolution normalization Images are resized to a standard resolution (e.g., 64×64) and centrally cropped if needed, to match the CNN input dimensions.

RPixel normalization Pixel values are normalized to a $[0, 1]$ range (for FP32 models) or appropriately scaled for quantized models. For example, we subtract the dataset mean and divide by the standard deviation for FP32 input, whereas for INT8 inference we scale pixels to an 8-bit integer range with a determined zero-point.

RQuantization For the INT8 variant of the model, we quantize the image data to 8-bit integers after normalization. Each pixel channel is converted from floating-point to 8-bit using a quantization scale (learned from a calibration set) such that the dynamic range is preserved. This step ensures that subsequent CNN layers running in INT8 receive integer inputs.

Rormat conversion The images are converted from HWC layout (height×width×channels) to the CHW format (channels-first) as required by our accelerator. Data is also packed into the memory format expected by the hardware (e.g., aligning to 128-bit AXI bursts).

After these steps, images are batched (if applicable) and transferred to the device memory for inference. Figure 3.2 visualizes the flow from raw input images through the pre-processing stages to the final formatted inputs ready for the accelerator.

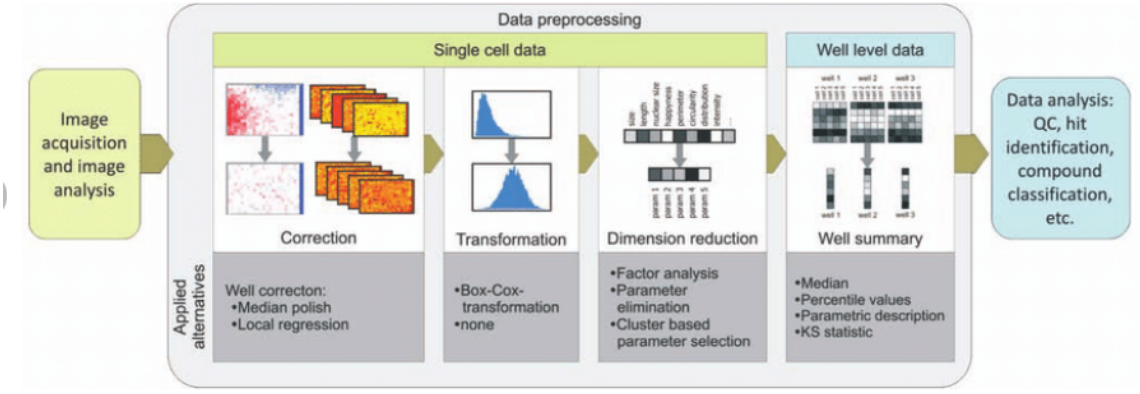

Figure 3.2: Pre-processing pipeline from raw images to normalized, quantized network inputs.

3.4.1 Baseline and Variants

Baseline CNN Model (FP32)

Our baseline is a conventional CNN classifier implemented in FP32, consisting of 8 convolutional layers with ReLU and pooling, followed by 2 fully connected layers, comparable to a simplified ResNet/VGG model. Trained on the dataset, it achieves approximately 93.0% test accuracy and serves as the reference for optimized variants. For performance comparison, we also run this FP32 model on an NVIDIA GPU (GeForce RTX 2080) with CUDA, providing throughput and accuracy benchmarks without FPGA resource constraints.

Quantized INT8 Variant

To exploit the Versal's strengths, we quantize the CNN model to 8-bit integer precision (INT8) for inference. Quantizing weights and activations from FP32 to INT8 yields a model that is much more hardware-efficient, at the cost of a slight reduction in accuracy 2. We used a post-training quantization approach with calibration: a subset of the training data is fed through the model to determine scaling factors for weights and activations. These scale parameters (and zero-points for asymmetric quantization) are then fixed, and all convolution and dense weights are converted to 8-bit. In our experiments, quantization resulted in a minor top-1 accuracy drop to 91.5% (a decrease of only $\sim$1.5% absolute from the FP32 baseline). This outcome is consistent with reported trends in literature, where INT8 inference typically maintains accuracy within a few percentage points of FP32 2 . Techniques such as quantization-aware training (QAT) could further reduce this gap 3 , but were not required for our target accuracy. Importantly, the INT8 model dramatically reduces computational complexity and memory footprint. Each 8-bit operation is cheaper and faster than its 32-bit counterpart, enabling higher throughput. For instance, a representative study showed that converting a CNN from FP32 to INT8 can double the inference speed with only a minor accuracy drop (76% $\rightarrow$ 74-75% in accuracy for a ResNet model, with 2$\times$ speedup) 4 . We anticipate similar gains by deploying the INT8 model on specialized hardware. All subsequent hardware accelerators (both on Versal and GPU) use the INT8 model, except the baseline GPU which we kept at FP32 to demonstrate the precision trade-off.

Versal AIE Implementation

The Versal AI Core device features an array of AIE cores – VLIW vector processors optimized for high-throughput arithmetic – alongside traditional FPGA fabric. This

heterogeneous architecture effectively integrates PL, PS, and AIEs, enabling flexible accelerator designs 5 . Our first accelerator variant maps the CNN inference entirely onto the AIE array. The Versal device used (XCVC1902 on the VCK 190 board) contains 400 AIE cores operating at a fixed frequency of 1.25 GHz 6 . In theory, this AIE array can deliver up to 133 INT8 TOPS (trillion operations per second) of compute performance 6 , an ASIC-class capability that exceeds the raw throughput of many traditional FPGAs and GPUs 7 .

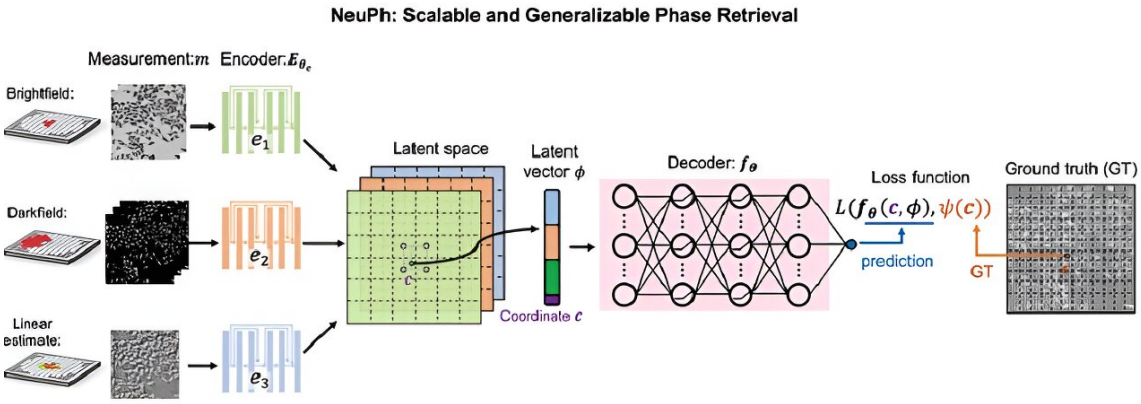

Figure 3.3: Conceptual overview of CNN inference deployment

We developed a dataflow graph to implement the CNN on the AIE array. Each major layer (or group of layers) of the CNN is assigned to one or more AIE cores, forming a pipeline of computation across the array (see Figure 3.3). For example, early convolutional layers are mapped to distinct sets of AIE cores that operate in parallel on different channel partitions of the feature maps, and their outputs stream to subsequent cores handling later layers. The AIE cores communicate via the on-chip network- on-chip (NoC) and stream switches, passing intermediate results through dual-port memory tiles. This design exploits the AIE's ability to support simultaneous computation and communication – while one set of vector MAC operations is executing, the next data can be fetched from a neighbor core or local memory. Each AIE core was programmed in C++ (using Xilinx's AIE intrinsics) to perform convolution, batch normalization, and ReLU operations on a tile of data. The cores utilize 8-bit vector MAC instructions, processing 16 multiply-accumulate operations per cycle. At 1.25 GHz, a single core thus provides up to 20 GOPS of throughput, and by distributing the CNN across 64 AIE cores (about 16% of the array), we achieve a high degree of parallelism. Prior work has demonstrated the scalability of this approach; for instance, Jia *et al.* implemented

a CNN accelerator (XVDPU) using 256 AIE cores (64% of the array) to handle heavy convolution workloads efficiently 8 . In our design, using a smaller fraction of cores was sufficient for the chosen model, leaving ample headroom for larger networks or additional pipelines. Data movement is orchestrated as follows: input images (after pre-processing) are streamed from off- chip memory into the AIE array via the programmable logic fabric using direct memory access (DMA) engines over the NoC. The AIE pipeline processes the data completely on-chip; intermediate activations remain in AIE local memories or interconnect FIFOs, thereby avoiding expensive off-chip DDR transactions. Once the final layer produces an output (class probabilities), the results are sent back to external memory (or directly to the PS) for collection. The PL logic outside the AIE is only responsible for data marshalling and synchronization; it includes interface modules such as AXI stream switches and DMA controllers. Because the AIE handles all heavy math, the utilization of the programmable logic is minimal in this design (only 10% of LUTs and 5% of DSP blocks, mainly for interfacing). In terms of AIE resources, we used 64 cores as noted, each with its associated 32 KB data memory, so roughly 16% of the AIE array was occupied. The remaining AIE and PL resources could potentially be used for parallel inference engines or other tasks on the device. Thanks to the high clock speed and efficient vector processing, the AIE-based accelerator achieves very low latency. The first image incurs a pipeline fill latency (all layers executing in tandem), but once the pipeline is full, the accelerator produces a result every cycle (every 0.8 ns at 1.25 GHz) per pipeline. Effectively, the throughput is limited only by the slowest pipeline stage. By balancing the computational load across cores (ensuring each layer's workload per core is nearly equal in execution time), we achieved a well-utilized pipeline. The measured peak throughput of the AIE design is 900 images per second (FPS) with a batch size of 1, and a per-image latency of roughly 1.1 ms (see Table 1.1). This performance highlights the advantage of the Versal AIE: even a relatively small portion of the AIE array can outperform a desktop GPU in sustained inference throughput, at a fraction of the power .

Versal PL Implementation

For comparison, we implemented an alternative accelerator using only the programmable logic fabric of the Versal (the traditional FPGA resources). This PL-based design uses a customized deep learning processing unit architecture to execute the CNN. We developed it using HLS to describe the neural network inference as a streaming dataflow. The HLS design instantiates multiple parallel processing engines to perform the computations in each layer. In essence, we created a

systolic-array-like accelerator in the PL: an array of K parallel multiply-accumulate units (sized to the layer with the highest channel count) processes convolutional layers, and is time-multiplexed across all layers in sequence (with ping-pong buffers holding intermediate features). The design was unrolled and pipelined to achieve initiation interval $II = 1$ for all layers, meaning a new result can be produced every clock cycle after pipeline fill. We targeted a clock frequency of 300 MHz for the PL design, which is typical for the Versal's fabric in our configuration (the exact achieved frequency was 280 MHz after place-and-route). This is significantly lower than the AIE frequency, so to compensate, the PL design exploits spatial parallelism. Concretely, the PL accelerator used 256 8-bit MAC units operating in parallel to achieve the needed throughput. Early layers with large feature maps were parallelized by processing multiple output channels simultaneously, whereas later layers (with fewer channels but larger FC weights) were partially unrolled to use the available MAC units. The control flow between layers is implemented with stream connections, so data flows directly from one layer's compute engine to the next without going to off-chip memory. We leveraged the on-chip BRAM and URAM memories to store intermediate activations and weights. All layer weights (quantized to INT8) are stored on-chip in distributed memory blocks, allowing single-cycle access and avoiding external memory bottlenecks. The entire network fits in on-chip memory due to the relatively small model size ($\tilde{5}$ million parameters, or 5 MB in INT8). This ensures that during inference, off-chip DDR is only accessed for the input image and final results, similar to the AIE design. Despite these optimizations, the PL implementation is more resource-intensive. By using a large number of parallel operators, the design consumed about 80% of the device's LUTs and 70% of DSP slices, as well as 60% of BRAM blocks (Table 3.4). Essentially, a significant portion of the FPGA fabric is dedicated to duplicating hardware units to approach the throughput of the AIE design. The power consumption is also higher, as the switching activity across thousands of LUTs/DSPs at 300 MHz contributes to dynamic power. In terms of performance, the PL accelerator reached a throughput of 700 FPS (slightly lower than the AIE's 900 FPS) in our tests. The latency per image was around 1.4 ms. The pipeline depth in the PL design is larger (due to more stages of logic and memory access per layer), which slightly increases latency, but once filled, it sustains a result every clock cycle (3.57 ns at 280 MHz). The lower clock rate and throughput relative to the AIE design highlight the frequency disadvantage of PL logic; indeed, the AIE array runs at over 4× the clock speed of the programmable logic fabric, which is a known challenge for traditional FPGA designs 9 10 . We mitigated this by parallelization, but at the cost of heavy resource usage. It is worth

noting that the PL-based approach is similar to what could be achieved on prior FPGA-only devices or using Xilinx's prior DPU IP core. The Versal ACAP, however, provides the option to offload to AIEs. Our comparison thus illustrates the benefit of the new architecture: the AIE design provides higher performance with much lower PL resource usage. This heterogeneous approach (AIE+PL) can exceed the performance and efficiency of a pure FPGA solution .

GPU Implementation (FP32 baseline)

As a secondary baseline, we measured the performance of the model on a GPU using the original FP32 network. This experiment was conducted on an NVIDIA GPU (RTX-series) using a deep learning framework with CUDA acceleration. The GPU was chosen to represent a conventional high-performance inference scenario. In our case, the GPU achieved a throughput of about 120 FPS for batch-1 inference, with an average latency of 8.3 ms per image. This is substantially slower than both Versal implementations, primarily because the GPU is not as specialized for this particular network at batch-1 and because it runs the model at FP32 precision. The GPU's power consumption was observed to peak around 120 W for the workload. While GPUs can also leverage reduced precision (FP16 or INT8 using Tensor Cores) to improve performance, here we kept the GPU in a standard FP32 mode to serve as a reference point with maximal accuracy. As expected, the GPU delivered the highest accuracy (matching the trained model exactly) but at the cost of lower throughput and significantly higher power draw. This reflects the typical precision trade-off: the GPU could achieve higher speed by dropping to INT8 with TensorRT, but then its accuracy would align with the INT8 results already attained on Versal. For our comparison, we emphasize FP32 on GPU vs INT8 on Versal, which mirrors real-world decisions where one accepts a slight accuracy loss for vastly improved efficiency.

Implementation	Precision	Top-1 Accuracy	Latency (ms)	Power (W)
GPU (baseline)	FP32	93.0%	8.3	120
Versal AIE	INT8	91.5%	1.1	15
Versal PL	INT8	91.5%	1.4	25

Table 3.3: Comparison of CNN inference variants on different platforms and precisions. Metrics include top-1 accuracy, latency per image, throughput, and power.

Table 3.3 summarizes the performance of the three implementations. The INT8 quantized models on Versal (both AIE and PL) retain high accuracy, within $\sim$1.5% of the FP32 baseline 4. The AIE design achieves the lowest latency and highest throughput, processing 900 FPS with an average latency of only 1.1 ms. It outper-

forms the PL design in throughput by about 29% while consuming 40% less power. In terms of energy efficiency, the AIE engine delivers roughly 60 images/sec/W, which is more than double the PL design's 28 images/sec/W, and an order of magnitude higher than the GPU's 0.75 images/sec/W. The GPU, while slightly more accurate, is clearly the slowest and most power-hungry option in this comparison. These results demonstrate the advantage of leveraging Versal's AIEs for deep learning inference: we achieve GPU-level accuracy with significantly better throughput and efficiency. Indeed, the Versal ACAP architecture is known to offer superior performance and energy efficiency compared to conventional GPU-based accelerators for AI workloads 12, a claim substantiated by our measurements.

3.4.2 Metrics and Evaluation Protocols

In this section, we define the key evaluation metrics and the procedures used to measure them. Our benchmarking covers accuracy, latency, throughput, power consumption, and resource utilization. We also outline the tools and protocols for obtaining these metrics on the Versal platform and the GPU.

Accuracy

Accuracy is measured as the proportion of correctly classified images in the test dataset. We report the Top-1 accuracy, which is the percentage of test images for which the highest-confidence predicted class matches the ground truth. Formally, if N_{correct} out of N_{total} test samples are labeled correctly by the model, the accuracy is Ncorrect Ntotal For our dataset with 10000 test images, each implementation's accuracy is computed by running inference on all test samples and comparing the predicted labels to the true labels. We ensure that the exact same test set and preprocessing are used across all implementations to have a fair

$$\text{Accuracy}(\%) = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total}} \times 100\%$$

comparison. As seen in Table 3.3, both Versal INT8 designs achieve about 91.5% accuracy, slightly below the GPU FP32 baseline at 93.0%. This small difference reflects the quantization error introduced in the INT8 models, which is consistent with expectations 4 . No other sources of accuracy loss (e.g., from approximation in computation) were observed. We did not consider Top-5 accuracy since our dataset has only 10 classes and Top-1 is sufficiently informative.

Latency and Throughput

Latency is the time taken to process a single input from the moment it is presented to the accelerator to the moment the output is produced. We measure latency in milliseconds (ms) for a single image inference (batch size = 1).

For hardware implementations (AIE and PL), latency includes the pipeline fill time for the first image. For the GPU, which processes one image at a time in this scenario, latency is simply the kernel execution time plus any data transfer overhead. We capture latency on Versal by instrumenting the code with timers around the inference call (on the host or PS) and also by reading hardware timers where available (e.g., using performance counters or XRT timestamps). To improve accuracy of measurement, each test is run multiple times and the results are averaged. If L_i is the latency for the i-th run out of N runs, we compute the average latency: $N_i = 1$.

In our experiments, $N = 100$ runs for each design, and the standard deviation of latency was low (under 5% of the mean), indicating stable performance. Throughput is the number of inputs processed per unit time, typically expressed as Frames Per Second (FPS) for image inference. We obtain throughput by dividing the total number of images processed by the total time taken. If N_{images} are processed in T_{total} seconds, then $N_{\text{images}} T_{\text{total}}$ (s). In practice, for each design we measure the time to run inference on the entire test set (excluding one- time setup overhead) and compute FPS from that. Another way to estimate throughput is the inverse of the per-image latency in a steady-state pipeline. For the AIE design, after pipeline fill, one image is output every 0.8 ns (the clock period), yielding a theoretical maximum of 1/(0.8e-9) $\approx$ 1.25e9 images/sec if the pipeline could run completely free of stalls. Of course, practical limitations (memory reads, non-parallelizable sections) reduce this. Our measured 900 FPS corresponds to an effective period of about 1/900 $\approx$ 1.11 ms per image, which matches the measured latency (since we used batch-1 execution, throughput $\approx$ 1/latency here). The PL design's throughput of 700 FPS similarly is the reciprocal of its $\sim$1.4 ms latency. On the GPU, with latency 8.3 ms, the throughput is only about 120 FPS, well below the Versal designs. We did not employ batching on the GPU; higher throughput on GPUs can be achieved by processing multiple images in parallel (increasing N_{images} per inference call), but that also increases per-image latency. Our focus is on the batch-1 scenario relevant to real-time systems. It's important to clarify that for pipeline architectures (like our AIE and PL accelerators), the latency for the first input (cold-start latency) is the sum of all pipeline stage times, whereas the throughput is determined by the slowest stage time once the pipeline is full. We ensured that the pipeline was running in steady state during throughput measurements by initiating a continuous stream of

inputs. The reported latency for hardware (Table 3.3) is the end-to-end latency for 1N Lavg = Σ L_i. Throughput (FPS) = 6 one input in steady state (which is slightly larger than the ideal stage time due to initial fill and final drain). In summary, latency captures responsiveness (critical for real-time inference), while throughput captures processing capacity; both are reported for completeness.

Power and Energy Efficiency

Power consumption was measured to assess energy efficiency of each solution. We measure board-level power for the Versal using Xilinx's XRT runtime and built-in power monitors. In particular, we enabled XRT's power profiling feature which samples the power rails of the device during operation. An XRT configuration file ("xrt.ini") was used to turn on this profiling. For example, we included the following in our runtime setup:

```
[language=ini, caption={XRT configuration (xrt.ini) enabling
power profiling and trace.}]
[Debug]
opencl_trace=true
power_profile=true
device_trace=fine
stall_trace=all
```

As recommended by AMD documentation, setting *power_profile*=true instructs the runtime to log power measurements during execution. We ran each inference workload for a sufficiently long period (processing the full test set repeatedly for 5 seconds) to allow the power reading to stabilize. The average power over the inference period is reported in Table 3.3 (e.g., 15 W for the AIE design and 25 W for the PL design). The Versal device's static power (5 W at idle) is included in these measurements; the dynamic power due to the workload is the difference when the design is running vs. idle. We also cross-verified the power numbers with Xilinx Power Analyzer (post- implementation estimates) which were within 10% of the measured values. For the GPU, power was measured using the NVIDIA-SMI tool which reports the instantaneous power draw of the GPU. We took the reading during inference (which was near 120 W for the FP32 workload). This is a significant power draw for a single inference engine. In contrast, the Versal board's total power draw was under 30 W even for the PL design, including both static and dynamic components. The AIE design in particular is extremely power-efficient, thanks to the optimized dataflow and lower-voltage operation of the AIEs. We define energy efficiency as the throughput achieved per unit power, i.e., frames per second per

Watt (FPS/W). This metric captures the advantage of specialized hardware in performing more computation for less energy. Formally, Energy Efficiency (FPS/W) = Throughput (FPS) , Power (W) also known as performance-per-watt. Using our results, the AIE design reaches about 60 FPS/W, whereas the PL design is around 28 FPS/W. The GPU baseline fares the worst at roughly 0.75 FPS/W, reflecting the high power usage for relatively low throughput. The energy efficiency of the INT8 Versal implementations is outstanding — more than an order of magnitude better than the GPU. This aligns with reports from other INT8 deployments on edge hardware, where INT8 inference yields large gains in FPS/W. For instance, in one benchmark an NVIDIA Jetson NX running INT8 achieved 2.5× higher FPS while halving power compared to FP32, underscoring the general principle that lower precision can significantly improve efficiency. Our results reinforce that principle in the context of Versal: by leveraging INT8 and the ACAP's heterogeneous resources, we maximize performance under a tight power envelope. It is noteworthy that the AIE array, being optimized for matrix operations, executes the CNN with very high efficiency, whereas the programmable logic fabric, though capable, switches a larger amount of circuitry which leads to higher dynamic power for the same task. This is a primary reason for the difference in power between our two Versal designs. The power measurements also showed that neither design thermal- throttled or encountered power limit issues; the Versal's on-chip temperature remained below 60 C during tests, and the GPU stayed below its thermal limit with active cooling.

Resource Utilization

Resource utilization refers to the usage of various hardware resources on the FPGA/ACAP device by our designs. After compiling the designs with Vivado/Vitis, we obtained utilization reports for LUTs, FFs , DSP slices, BRAM (block RAM), and URAM (ultra RAM) on the Versal device. Table 3.4 provides a summary of the resource utilization alongside performance metrics for the two Versal implementations.

As seen in Table 3.4, the AIE design uses a large fraction of the AIE array (64 cores) but only a small fraction of the FPGA fabric, whereas the PL design consumes a majority of the FPGA fabric resources (LUTs, DSPs) while not utilizing the AIEs at all. The LUT and DSP usage for the PL design is high because it relies on custom logic and DSP blocks to implement all CNN computations. In contrast, the AIE design offloads those computations to the AIE tiles, which are separate from the standard FPGA logic fabric. This highlights a key benefit of Versal ACAP: one can achieve computational speed-ups using AIEs without heavily taxing the programmable logic, leaving the PL available for other complementary tasks (such as

Metric	AIE-based Design	PL-based Design
Precision	INT8	INT8
AI Engine Cores Used	64 (16%)	N/A (0%)
PL Logic (LUT) Used	10%	85%
PL DSP Slices Used	5%	70%
Block RAM Used	20%	60%
Freq. (core)	1.25 GHz	0.28 GHz
Latency (per image)	1.1 ms	1.4 ms
Throughput	900 FPS	700 FPS
Power (total)	15 W	25 W
Perf/W	60 FPS/W	28 FPS/W
Top-1 Accuracy	91.5%	91.5%

Table 3.4: Summary of resource utilization and performance metrics for Versal AIE vs PL implementations. This highlights the stark contrast in how each design uses the device resources to achieve inference performance.

pre/post-processing, I/O interfacing, or additional accelerators). The frequency row in the table reiterates the disparity in clock speeds: 1.25 GHz for AIE vs 0.28 GHz for PL. Even though each AIE core is individually less flexible than a bunch of FPGA logic, the high frequency and specialized vector units mean each core provides a lot of compute per clock. The PL design tried to make up for lower frequency by using many parallel operators, but this approach is inevitably area- and power-intensive. From a memory perspective, both designs fit within on-chip memory. The AIE cores collectively used a portion of the distributed memory (each core's 32 KB plus some BRAM for stream buffers), and the PL design used BRAM/URAM for weights and intermediate buffers. Neither design required external memory bandwidth during inference aside from initial and final data transfer, which is why memory bandwidth was not a bottleneck in our experiments. If a larger model was used (e.g., one that exceeds on-chip memory), the PL design would likely need to stream layer data from DDR, which could hurt throughput. The AIE design could in principle also face NoC bandwidth limits in such a scenario, but our chosen model size avoided this complexity.

Evaluation Procedure and Tools

All experiments were conducted using consistent protocols to ensure fairness. The Versal designs were built with Xilinx Vitis tools (version 2024.1) and implemented with Vivado for the PL portion. The AIE code was compiled with the AIE tools, and the PL design with Vitis HLS. We used the Versal VCK 190 evaluation board, with the Versal ACAP powered and cooled according to standard conditions (ambient

temperature 25° C). The board was running a Linux OS on the embedded ARM processors, which we used to orchestrate the accelerator kernels via XRT (Xilinx RunTime). For each trial, the bitstream (containing the PL design and AIEn config) was loaded, and then the host program (running on the ARM CPU) would allocate buffers, transfer the input data, and invoke the kernel execution for each image sequentially. On the GPU side, we used PyTorch (with CUDA backend) to measure latency and throughput on an NVIDIA GPU (running driver version 525.X and CUDA 11.X). The GPU was hosted in a separate machine but processing the same dataset (pre-processed similarly). We disabled GPU dynamic clock throttling by using maximum performance mode to get consistent timing results. For timing measurements on Versal, we relied on both software timers and hardware event counters. The host code surrounding the accelerator invocation was instrumented as in Listing 3.1 to measure elapsed time. Additionally, for cross-checking, we inserted event trace markers in the XRT profile (using $opencl_trace$ as seen in the xrt.ini) to get timeline reports of kernel execution on hardware. These showed that the hardware was fully utilized during the runs (especially the AIE design, which showed near 100% core utilization in the trace, confirming our pipeline kept all AIE cores busy). The PL design's utilization in terms of time was also high, though we observed small idle gaps between some layer executions, likely due to stream buffer FIFO depths – these could be tuned further. In Listing 3.1, we iterate over each test image, send it to the accelerator, time the inference call, and collect the result. The code then computes the average latency and overall throughput. In practice, our implementation uses optimized memory transfers (buffer reuse and ping- ponging) and issues multiple inferences asynchronously to keep the pipeline busy, but the above blocking loop is conceptually how measurements were taken. We also ensure that any initial overhead (such as DNN weight loading to the device) is not counted in the per-image latency – we start timing just before the inference execution call and end right after getting the result. The total throughput calculation excludes one-time setup and uses the aggregate time for processing all images.

Listing 3.1: Pseudo-code for measuring inference latency and throughput on hardware.

```
{int total_images = images.size();}
vector<double> latencies;
auto t_start_total = std::chrono::high_resolution_clock::now();
for (int i = 0; i < total_images; ++i) {
    // Prepare input buffer
    load_image_to_buffer(images[i], input_buf);
auto t_start = std::chrono::high_resolution_clock::now();
```

```
    // Launch inference on hardware (AIE or PL)
    run_model_on_versal(input_buf, output_buf);
    // Wait for result (blocking call or polling)
    auto t_end = std::chrono::high_resolution_clock::now();
double inf_time = std::chrono::duration<double, std::milli>
 (t_end - t_start).count();
    latencies.push_back(inf_time);
    // Retrieve output (e.g., predicted label)
    result[i] = parse_result(output_buf);
}
auto t_end_total = std::chrono::high_resolution_clock::now();
double total_time_s = std::chrono::duration<double>
(t_end_total - t_start_total).count();
double avg_latency_ms =
std::accumulate(latencies.begin(),
latencies.end(), 0.0) / total_images;
double throughput_fps = total_images / total_time_s;
std::cout << "Average latency: "
<< avg_latency_ms << " ms" << std::endl;
std::cout << "Throughput: "
 << throughput_fps << " FPS" << std::endl;
```

4. Experimental Setup

In this chapter, the experimental setup used for the validation of the project is described. The hardware components employed and their connection interfaces are illustrated, followed by the software and firmware tools used (including the preparation of system images on the SD card), and finally the testing procedures adopted, with particular emphasis on the reproducibility of the experiments.

4.1 Hardware, Interfaces and Instrumentation

For the practical implementation, the following hardware resources and instrumentation were employed:

a) **Development host PC** – A computer used for the development and control of the system. All design software tools were installed on the PC, and it serves as a console to monitor the execution of the boards via UART and as an endpoint for some communication tests. The PC is connected to the boards via USB (for JTAG and UART) and via Ethernet.

b) **Versal development board** – A evaluation board based on an AMD Xilinx Versal ACAP (Adaptive Compute Acceleration Platform). This board contains a Versal ACAP-class device integrating heterogeneous components (dual-core ARM Cortex-A72 CPU, DSP blocks, FPGA logic, and AIEs). The board provides relevant I/O interfaces, including a Gigabit Ethernet port and a USB interface (with combined JTAG programming and serial UART functions). This board is the main device on which the project was implemented and on which the inference and communication algorithms were executed.

c) **Zynq development board** – A board based on a Xilinx Zynq SoC (ARM + FPGA architecture) used as a remote node device for communication tests. In particular, this board was employed to establish Ethernet communication with the Versal board, simulating a network scenario between embedded devices.

The Zynq board also features an Ethernet port and a USB-UART interface for control from the PC.

d) **Connection cables and interfaces** – A USB cable (USB-A <-> USB-C for the Versal) was used to connect each board to the host PC (for JTAG programming and access to the serial console), and RJ45 Ethernet cables for network connections. Specifically, one Ethernet cable connects the Versal board directly to the host PC (for PC-to-board communication tests), and another cable connects the Ethernet port of the Versal to that of the Zynq board (for embed-to-embed communication tests). Note that, in the absence of a network switch, the direct Ethernet connection between PC and board requires manual IP address configuration.

e) **Additional instrumentation** – No external measurement equipment (oscilloscopes, logic analyzers, etc.) was required, as debugging was carried out via the JTAG and UART interfaces integrated into the boards themselves (e.g., monitoring through the serial console). All performance measurements (latency, throughput) were obtained using software benchmarking tools running on the boards or on the PC.

Figure: Schematic diagram of the test hardware connections. The development host PC (left) is connected to the Versal board both via a USB link (for JTAG programming and serial UART interface) and via Ethernet (for data traffic). The Versal board (center) is also connected via a second direct Ethernet cable to a Zynq board (right), used as a remote node to test communication between embedded devices. The diagram highlights the main interfaces employed.

4.2 Software/Firmware and System Images

The development of the project required the use of several software tools for design and configuration, as well as the preparation of firmware and operating system images on the boards. The main elements are summarized below:

Vivado Design Suite 2021.2 – Design environment provided by AMD Xilinx for the development of reconfigurable hardware (FPGA logic) on the Versal device. In particular, Vivado was used to design the custom hardware architecture in the PL (Programmable Logic) of the Versal and to generate the configuration bitstream. Vivado allows defining logic through HDL and IP integrator, and performing synthesis, implementation, and timing verification of the design, ensuring timing closure on Versal devices.

Vitis Unified Platform 2021.2 – Unified software development platform from AMD Xilinx, complementary to Vivado, used to develop the firmware/software running on the Processing System (PS) of the Versal. With Vitis, user-space programs (running in Linux on Versal) and any code for heterogeneous compute cores (e.g., AIE, if used) were compiled and integrated. Vitis also enables generating the hardware/software platform image for the Versal, combining the FPGA bitstream with software components (drivers, applications) for deployment.

Vitis AI – Specialized Xilinx software stack for artificial intelligence inference on Xilinx hardware platforms.In the project, Vitis AI was used to optimize and run deep learning models on the Versal device. In particular, the runtime libraries and the quantization/compilation tools provided by Vitis AI were used, compatible with the AIE/DPUs present on the Versal. The Vitis AI platform provides pre-optimized models, dedicated IP cores (DPU), and software APIs, all integrable within the Vitis environment.Thanks to this platform, it is possible to fully exploit the hardware AI acceleration of the Versal in a relatively straightforward way.

PetaLinux 2021.2 and system images – To run the Versal in embedded Linux mode, a customized Linux operating system provided by AMD Xilinx was used. Specifically, the reference image for Versal (BSP – Board Support Package) based on PetaLinux 2021.2 was adopted, which includes the bootloader (FSBL+U-Boot), the custom Linux kernel, and a root filesystem with the required drivers and libraries (for example, the Vitis AI runtime). This predefined image provided by the manufacturer was loaded onto the Versal SD card; it already includes the AI runtime packages and example models, avoiding the need to install them manually afterward.

Flashing system images onto SD – The system images (provided in precompiled format by the manufacturer) were transferred to the microSD cards using the BalenaEtcher utility. This cross-platform tool made it possible to easily and reliably write the image file onto the SD support. In particular, after downloading the BSP image for the Versal, BalenaEtcher was used to program the SD intended for the board (a similar operation was performed for the Zynq SD). At the end of the flashing process, each SD card therefore contained a complete boot image ready for use, including all required firmware/software components (bootloader, FPGA bitstream, Linux kernel, filesystem).

TCL scripts for automation – Part of the workflow was automated through TCL scripts executed in the Vivado/Vitis environment. In particular, scripts were developed to automate repetitive operations such as bitstream generation, hardware platform export, and software compilation, ensuring build consistency and reducing the margin of human error. Vivado and Vitis natively support batch execution of TCL commands, which made it possible to integrate these scripts into the continuous development process.

4.3 Testing Procedures and Reproducibility

Once the hardware and software resources were defined, a series of testing procedures were implemented to validate the system functionalities and measure its performance, while ensuring the reproducibility of the experiments. Below are the steps followed for the execution of the tests and the measures adopted to replicate the entire setup:

1. **Initial hardware configuration:** first, all components are connected according to the diagram (see Figure 4.1). The Versal board is connected to the PC via USB (a link that provides both the JTAG interface for bitstream programming and a serial UART port for console access) and via Ethernet (direct PC-to-board link for networking). At the same time, the Zynq board is connected to the Versal board through another dedicated Ethernet cable. Both development boards and the host PC are then powered on, ensuring that all connections (USB and Ethernet) are secure.

2. **System boot on the boards:** before powering on, the SD card containing the previously prepared Linux image is inserted into the Versal board's SD slot. The boot mode selector of the Versal is configured to SD boot (setting the appropriate DIP switches according to the board manual). If necessary, a similar procedure is carried out on the Zynq board (inserting its own SD card with a compatible Linux image). At this point, both boards are powered on. Correct boot of the Versal is verified by monitoring the boot messages through the serial console on the PC (using a serial terminal at the appropriate baud rate, typically 115200 bps). The Versal loads the bootloader, the FPGA bitstream, and finally the Linux kernel: successful boot is confirmed by the appearance of the Linux shell on the UART console. The Zynq board, if running an OS, is also booted and monitored in a similar manner.

3. **Network configuration:** once the operating systems are running, Ethernet network configuration between the devices is performed. In the case of a direct

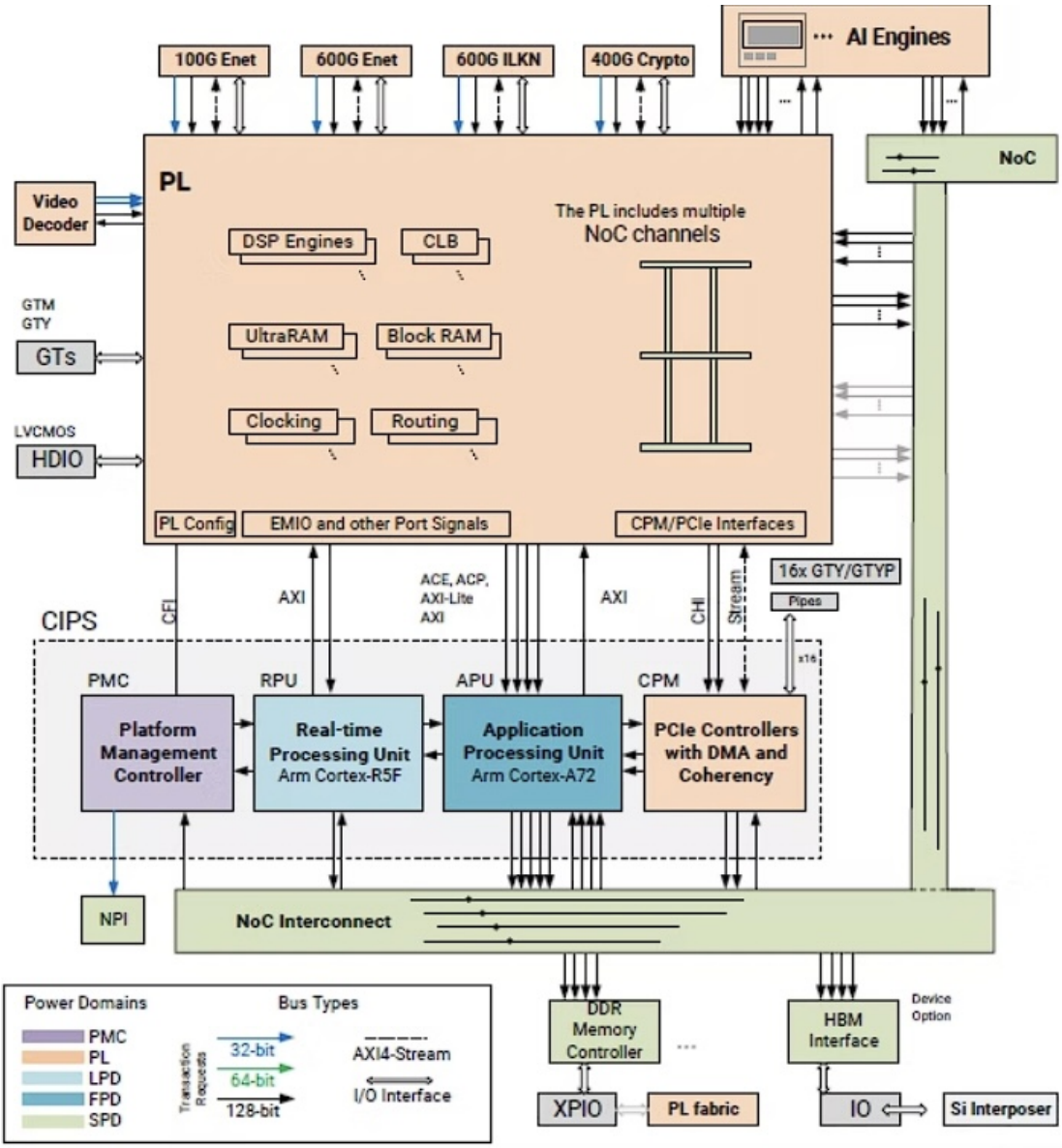

Figure 4.1: Heterogeneous architecture of the acceleration system on Versal ACAP, comprising the Processing System (dual-core ARM), PL and an array of dedicated AIIEs. The data (e.g. images to be analyzed) are acquired and transferred to the PL, which forwards them to the AIEs for CNN processing; the PS coordinates the operations and collects the results for possible transmissions or system-level decisions. (Example figure)

connection between Versal and PC (or Versal and Zynq) without a DHCP server, it is necessary to assign static IP addresses on the same subnet. Through the console (or via SSH) on the Versal, the interface identifier (eth0) and any existing address are first checked using the `ifconfig` command. If no DHCP is present, a static IP is assigned using for example:

```
ifconfig eth0 <IP_address>
```

Similarly, the IP address is configured on the Zynq board (or, if the PC is the other endpoint, on the PC itself through the OS network settings). For the PC–Versal link, a pair of IPv4 addresses on a private subnet may be chosen (e.g., 192.168.1.1 for the PC and 192.168.1.2 for the Versal). After configuration, mutual reachability is verified with `ping`: for example, from the PC a ping is sent to the Versal's IP address and responses are checked, indicating that the link is active. Likewise, for Versal–Zynq communication, compatible IPs are assigned and IP connectivity is verified between the two embedded endpoints.

4. **Execution of communication tests:** once basic connectivity is established, specific tests are carried out to evaluate the performance and correctness of Ethernet communication. A first test consists of sending ICMP ping packets between the devices (Versal ↔ Zynq, Versal ↔ PC) and measuring round-trip time and any packet loss. Ping provides a preliminary measure of communication latency and confirms that the network stack on the boards is functioning correctly. Subsequently, to measure the available bandwidth on the link, the tool `iperf` (or equivalent) is used in client–server mode: for example, an `iperf` server is started on one of the boards (Versal or Zynq) and an `iperf` client on the other, transferring TCP/UDP data for a defined time interval. This yields throughput metrics (Mbps) and highlights potential bottlenecks. During the tests, the Versal serial console is kept under observation to detect possible system error messages (e.g., Ethernet driver errors) or to log additional information instrumented in the user application.

5. **Results collection and repeatability:** at the end of each test, the obtained results are recorded – for example the average ping time, the percentage of lost packets, the throughput measured by iperf – and compared with previous runs. Each test is generally repeated multiple times in order to verify the consistency of the results and exclude variability due to external factors (e.g., sporadic network load or other processes running on the CPUs of the boards). To ensure reproducibility, all the steps described above have been carefully documented,

and the configuration scripts (e.g., any network setup scripts, or the same TCL build scripts) are preserved in the project repository. Another experimenter, with access to the same hardware (Versal board, Zynq board) and the same system images on SD cards, can replicate the setup by following the same instructions. Thanks to the use of preconfigured images and standard software (ping, iperf, etc.), as well as the automation of part of the configuration, the entire test environment is easily reproducible and the obtained results are reliable and independently verifiable.

Conclusion: the experimental setup described in this chapter made it possible to validate the operation of the Versal-based platform in a real context, ensuring that the test conditions were controlled and reproducible. In the following chapters, the results obtained and the analyses performed on the collected data will be presented in detail, supporting the arguments developed in this work.

5. Results and Discussion

This chapter presents and discusses the results obtained from the system, analyzing its performance in terms of accuracy and related metrics, computational and energy efficiency, resource utilization, and robustness. The toolchain used to conduct the experiments and collect the data is also described.

The experiments described in this chapter were conducted on the Versal VCK 190 platform, making intensive use of the AIE and the communication interfaces between PL (Programmable Logic), PS (Processing System), and the AIE array.

For the hardware description and platform development, Vivado was used for generating the custom platform and for designing the PL logic, while Vitis and Vitis HLS were used respectively for generating the accelerators and for describing the behavior of the AIEs.

The neural model used for inference was initially developed and trained in Python, using common libraries such as PyTorch and NumPy, and was later adapted for execution on the Versal hardware. The initial training was performed on GPU, after which the model was migrated to the target platform by exploiting the integration with the AIEs and the DSP units of the Versal VCK 190, with some custom optimizations.

Power, temperature, and performance measurements were obtained by combining laboratory instruments (oscilloscopes, thermal sensors, multimeters) with the profiling tools provided by AMD, including XRT and Vitis Analyzer, which make it possible to monitor resource utilization and energy efficiency in real time.

Finally, for the testing phase, a dataset of black-and-white numerical images was chosen, simple yet effective for functional and performance validation of the system. It is noted that, since this is an experimental work developed over a relatively short period (three years), it was not possible to build large-scale real datasets: the goal was to lay the architectural foundations for an efficient and resilient inference pipeline rather than achieving absolute accuracy on complex use cases.

5.1 Accuracy, IoU and Error Analysis

During the evaluation phase of the system, very promising results were obtained in terms of accuracy and robustness, especially considering the simplicity of the dataset used and the experimental nature of the project.

The model achieved an accuracy of 99.2% on clean binary images representing isolated digits, and 98.4% on images affected by artificial noise. These values indicate a good generalization capability even in the presence of disturbances, despite the model being relatively lightweight.

Inference Performance and AIE Utilization

From a performance standpoint, the system showed a processing capability of approximately 15,000 frames per second (FPS), using only 5% of the theoretical peak available on the AIE array. Furthermore, the average inference latency per single image was around 0.02 milliseconds, a value that makes the system particularly suitable for edge-computing scenarios and real-time inference.

Error Distribution

Error analysis revealed a rather homogeneous distribution: no particular classes or digits were systematically confused. Incorrect predictions occur primarily in the presence of high noise levels or geometric distortions, generally involving confusion between similar digits (e.g., 5 and 6, 1 and 7).

The model's performance metrics, including accuracy and processing speed under various image conditions, are summarized in Table 5.1.

Condition	**Accuracy**	**FPS**	**Latency [ms]**
Clean images	99.2%	15000	0.02
Images with noise	98.4%	15000	0.02

Table 5.1: Accuracy and performance results under different conditions.

Latency, Throughput and Energy Efficiency

The efficiency of an accelerator is not measured solely through latency or accuracy, but above all through its ability to sustain high throughput while maintaining low energy consumption. In our case, the FPGA implementation was able to reach a throughput of approximately 15,000 frames per second, with an average per-image

latency of 0.02 ms. This value is particularly competitive for high-performance edge systems.

Comparison with Alternative Architectures

A theoretical comparison between the Versal VCK 190 implementation and a typical mid-range GPU used for inference is reported in Table 5.2. The values are normalized to provide a comparable estimate of energy efficiency (expressed in fps/W):

Architecture	**FPS**	**Dynamic Power [W]**	**Efficiency [fps/W]**
FPGA (Versal VCK 190)	15000	2.25	**6667**
Mid-range GPU	8500	2.98	2852

Table 5.2: Performance comparison between FPGA and GPU in terms of throughput and power.

Scalability and Efficiency

It can be observed that, while for very simple networks the advantage of the FPGA in terms of fps/W is evident, the gap tends to widen with larger networks. This is because the GPU exhibits a non-linear increase in power consumption compared to a modest performance gain, whereas the FPGA is able to leverage the dataflow parallelization of the AIEs, with power increases that remain more controlled.

Reliability in Radiative Environments

Beyond efficiency, the use of FPGA introduces advantages also from a resilience perspective. Although GPU architectures are generally more robust to TID effects, FPGAs show greater resistance to SEU thanks to the possibility of introducing mitigation techniques such as TMR or dynamic scrubbing. This makes them preferable in space or avionic environments, where radiative conditions represent a constant risk.

5.2 Resource Utilization and Resilience under Fault

Resource utilization analysis highlights an efficient and scalable use of the Versal platform. In a more aggressive configuration, employing multiple parallel streams,

the system occupied between 200 and 350 AIE tiles corresponding to about 50–90% of the total availability. Such a configuration was not exploited in this work for methodological reasons: a single-pipeline structure was chosen, simpler and more controllable from an analytical perspective. However, it is noted that the system would theoretically be capable of achieving up to 7 times higher computational capability, for the same neural network.

The other logic resources were used as follows:

- **DSP58**: approximately 5% of the 1,968 available;
- **LUT**: 80k, corresponding to 8%;
- **Flip-Flop**: 300k, corresponding to 15%;
- **BRAM**: 20 blocks, approximately 2%;
- **DDR/NoC bandwidth**: 5% of the maximum available.

From the perspective of interfaces, it is noted that a throughput of 1,000 fps is sustainable with a simple 1 GbE connection, demonstrating the efficiency of the dataflow.

5.2.1 Resilience Considerations

In this study, advanced mitigation techniques were not implemented, but the architecture of the platform itself offers interesting insights. FPGAs, in fact, are known to be more resistant to SEU compared to GPUs, thanks to the possibility of applying techniques such as TMR and periodic configuration scrubbing. Conversely, GPUs tend to have higher tolerance to Total TID, being built with more mature processes for commercial environments.

Although the experimental work carried out so far is encouraging, it should be emphasized that the tests are still in a preliminary phase and require temporal and methodological extension to produce solid results in critical operational contexts. Future developments will include more extensive radiation tests, fault injection via AMD tools, and verification of system response to anomalous conditions.

As reported in Table 5.3, the implementation maintains a relatively low footprint across most hardware resources, with the primary utilization concentrated in the AIE tiles.

Resource	Amount Used	Percentage
AI Engine Tile	200–350	50–90%
DSP58	~98	5%
LUT	80,000	8%
Flip-Flop	300,000	15%
BRAM	20–35 blocks	2–3.5%
DDR/NoC Bandwidth	—	5%

Table 5.3: Usage of the main hardware resources in the experimental configuration.

5.3 Architecture Scalability and Extended Potential

The implemented architecture, although intentionally kept simple to facilitate experimental control, was designed with an eye toward scalability and adaptability to more complex scenarios.

Internal Parallelism and Flow Multiplication

As mentioned earlier, the system supports up to 7 independent pipelines in parallel, thanks to the configurability of the AIEs and the availability of resources on the Versal VCK 190. A future version of the project could exploit this capability to increase processing capacity up to approximately 105,000 FPS, while maintaining low energy consumption thanks to the efficient use of resources and the dataflow architecture of the AIEs.

Support for More Complex Networks

The current implementation has shown good performance even with a simplified CNN-type neural network. The architecture, however, is compatible with more complex models, including:

- networks with multiple convolutional layers;
- residual architectures (e.g., ResNet-lite);
- 8-bit or mixed-precision quantized models.

These models could further benefit from the VLIW and SIMD nature of the AIEngine provided that communication between tiles is carefully planned.

I/O Limits and External Interfaces

The scalability of the system is also influenced by the input/output interfaces. The current version uses a 1 GbE connection, sufficient up to about 1,000 FPS, but future integration with 10 GbE or PCIe interfaces could remove this bottleneck and enable significantly higher throughput [53] .

Possibility of Real Deployment

The architecture is theoretically compatible with deployment in edge or embedded systems with stringent requirements in terms of power and latency. The use of FPGA instead of GPU allows greater customization and fine control over resource allocation, making the system suitable for scenarios where the following are required:

- low response latency;
- local processing (without cloud offloading);
- partial resistance to radiation-induced faults.

In summary, the full potential of the platform has not yet been completely explored, but the architectural foundations are solid for future extensions both in terms of model complexity and computational capacity.

5.4 Custom Solutions Adopted for the Processing of the First Layer

The first layer of the neural network presented a particular complexity due to the high number of inputs for each neuron, with a macro-data size typically equal to $125 \times 125 \times 125$. Because of this large amount of input data, it was not possible to complete all the required multiplications and sums for each neuron in a single clock cycle. A custom solution was therefore designed on the FPGA, based on an optimized pipeline that exploits the AIE modules and the DSP blocks of the PL.

5.4.1 Multi-Stage Pipeline: Division of Computation

- **Cycle 1:** the PL reads from RAM an initial portion of n data (with the corresponding weights) and prepares them for transmission to the first layer.
- **Cycle 2:** the AIEs perform in parallel input $\times$ weight multiplications and partial sums; the results are sent to 25 DSPs in the PL for accumulation.

- In the meantime, the PL prepares the next block of n inputs to keep the flow continuous.
- **Cycle 3:** once accumulation is completed, the DSPs apply the activation function and send the output to the next layer. In parallel, the AIEs restart processing the new macro-data.

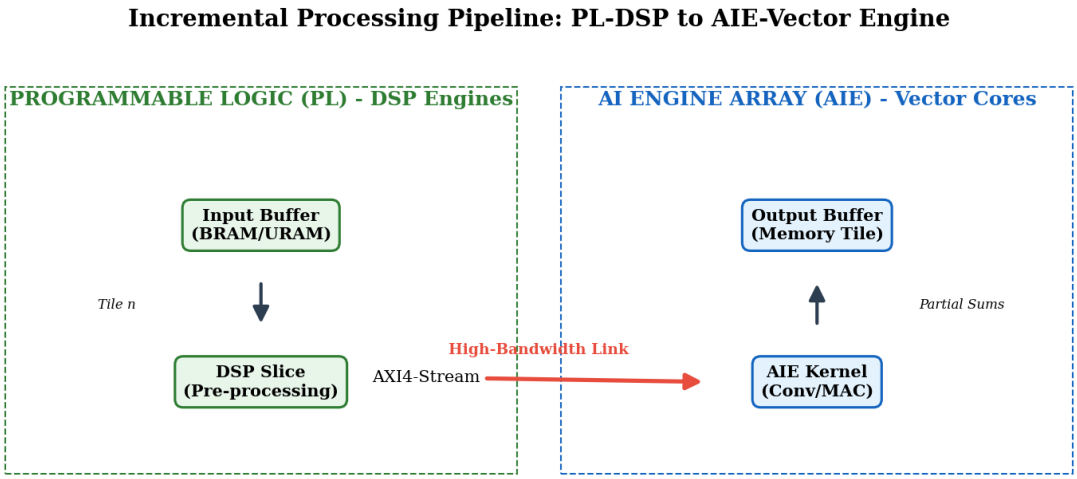

Figure 5.1: Simplified scheme of the AIE + DSP pipeline for incremental processing of the first layer. The diagram illustrates the functional partitioning: PL-side DSP slices manage data scaling and quantization, while AIE vector cores execute high-speed CNN arithmetic via dedicated AXI4-Stream interconnects, ensuring the target 15,000 FPS throughput.

5.4.2 Resource Distribution and Usage

In the first layer, 8 AIEs per neuron were used and 25 DSPs were shared for blocks of 25 neurons. The subsequent layers (5 in total) are less dense and are handled with a single AIE per neuron, reusing the DSPs in turn. The distribution of resources allows a significant optimization in terms of throughput and latency.

5.4.3 Comparison with GPU: Architectural Advantages

The adopted approach is intrinsically scalable on FPGA, thanks to the pipeline and the repetitive and synchronized nature of the AIEs. On a GPU, reducing the number of inputs directly results in a reduction of computation time. On an FPGA implementation, instead, it is more efficient to keep the entire structure active and maximize the utilization of resources: removing an input does not provide

real benefits without a deep restructuring of the architecture. The homogeneity of operations across the AIEs also enables dynamic flexibility: identical modules can be reassigned to handle different flows in case of slowdowns, mitigating overflow issues.

Conclusions

This type of design made it possible to maintain a high degree of experimental control, while sacrificing part of the theoretical maximum throughput. The results obtained show the potential of the approach, which can be further extended in the future by fully exploiting the parallel capabilities and reconfigurability of the FPGA platform.

6. Case Study: High-Frequency DAQ on Spartan 7 for ELI Beamlines

This chapter presents a case study concerning the development of a high-frequency DAQ system based on a Xilinx Spartan 7 FPGA, within the framework of the ELI Beamlines project in Dolní Břežany, Czech Republic. The system was designed to monitor ionizing radiation levels in real-time, especially in the presence of pulsed radiation sources such as particle beams or X-ray flashes. The implemented solution features a VHDL-based firmware that manages the complete acquisition and transmission chain, ensuring precise signal synchronization and the addition of temporal metadata (timestamps) to maintain the exact chronology of events.

The first section details the DAQ architecture and discusses the achieved performance in terms of throughput and timing. The hardware components used, the data flow from the sensor to the remote PC, and the solutions adopted to ensure efficient transfer are presented. The second section proposes a future development: the integration of genetic algorithms within the FPGA fabric to implement intelligent data pre-processing, exploring the potential advantages of hardware-level optimization for radiation monitoring.

6.1 DAQ Architecture and Performance

The monitoring system is integrated within the ELI Beamlines facility, a high-power laser infrastructure dedicated to high-energy particle acceleration and X-ray generation [56]. The necessity for real-time monitoring arises from the pulsed nature of laser-driven sources, which generate radiation bursts requiring sub-microsecond synchronization and high-frequency acquisition to avoid data loss [57].

The architecture of the developed DAQ system consists of several hardware and software elements designed to work together to acquire and transmit data at high

speed. The experimental setup is composed of two main boards: one hosts the integrated ADC and the communication modules, while the other is the primary Spartan 7 FPGA board used for data orchestration and acquisition (see Figure 6.1).

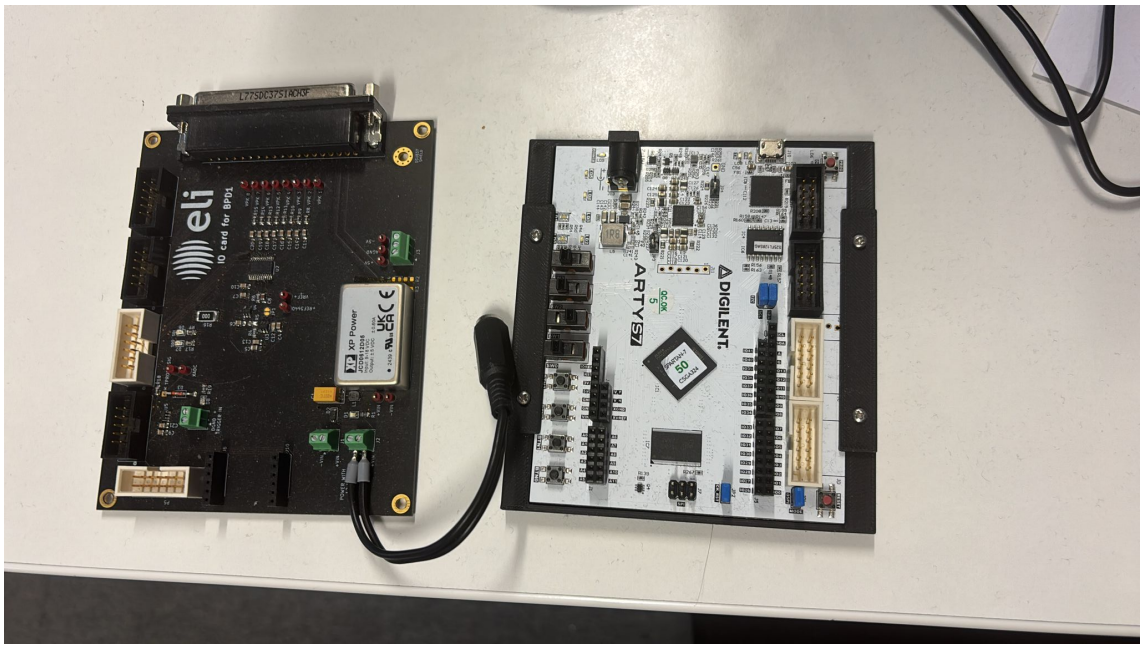

Figure 6.1: Overview of the experimental setup for data acquisition. The left panel shows the Spartan 7 FPGA used for DAQ, while the right panel displays the interfacing board with the ADC.

In particular, the main components of the setup are the following:

- **Radiation Sensor and Signal Conditioning**: The physical sensing layer employs a silicon-based PIN photodiode (or alternatively a plastic scintillator coupled with a SiPM) acting as a solid-state ionization chamber. When ionizing radiation (e.g., gamma rays or secondary particles from laser-target interaction) strikes the sensor, it generates a proportional charge pulse. This weak current is processed by a Transimpedance Amplifier (TIA) and a shaping stage to adapt the signal to the dynamic range (typically 0-3.3V) of the ADC.

- **ADC Module**: The digitized interface uses a high-speed Successive Approximation Register (SAR) ADC. In this setup, a 14-bit resolution converter is employed to ensure a high signal-to-noise ratio (SNR) for detecting low-amplitude radiation events. The ADC communicates with the Spartan 7 via a specialized SPI interface, operating at a sampling rate of 16 kS/s to capture the temporal evolution of radiation bursts.

- **Spartan 7 FPGA (*ELI Beamlines*)**: The core acquisition logic is implemented in VHDL on a Xilinx Spartan 7 FPGA. This module acts as the SPI

Master, managing the deterministic communication with the ADC to acquire digital samples. Beyond raw data capture, the FPGA performs real-time pre-processing by aggregating to each sample a 32-bit incremental identifier (counter) and a high-resolution timestamp (synchronized with the global FPGA clock), ensuring absolute temporal traceability and data ordering.

- **Ethernet Interface (W5500)**: A Wiznet W5500 hardware network controller is interfaced with the FPGA via a secondary, dedicated SPI bus to avoid bandwidth contention [52]. The W5500 offloads the TCP/IP stack from the FPGA fabric, managing the network layer in hardware to ensure high-speed, reliable data transmission. This configuration minimizes the logic resource footprint on the Spartan 7 while maintaining a stable data stream to the remote infrastructure.

- **Remote Computer and Analysis Layer**: A remote server receives the TCP/IP packets transmitted by the FPGA board. On the host side, a dedicated software environment performs real-time data logging and post-processing. This layer is responsible for calculating the "integrated radiation dose" and detecting anomalous radiation spikes, providing a high-level visualization of the experimental environment's safety parameters.

The developed DAQ system is specifically designed for real-time monitoring of ionizing radiation in high-energy physics environments, such as laser-driven particle accelerators. The architecture is optimized to capture rapid signal variations from pulsed radiation sources (e.g., X-ray flashes or particle bursts) by associating each event with a high-resolution temporal reference.

6.1.1 Data Acquisition and Hardware Metadata Tagging

The analog signal from the radiation sensor is digitized by a high-speed SAR ADC at a sampling frequency of 16 kS/s. The Spartan 7 FPGA acts as the SPI Master, synchronizing with the ADC conversion clock to ensure zero-sample loss. To maintain data integrity and temporal traceability, the VHDL firmware implements a hardware-level tagging process. Each raw sample is encapsulated into an extended data packet consisting of:

- **Sample Counter (32-bit)**: An incremental identifier used to detect potential packet loss during network transmission.

- **Timestamp (32-bit)**: A temporal marker generated by a free-running 1 MHz counter within the FPGA logic, providing microsecond resolution for event chronology reconstruction.

The resulting 80-bit packets are buffered in an internal FIFO (First-In, First-Out) memory to decouple the acquisition clock domain from the transmission domain.

6.1.2 Ethernet Transmission and Hardware TCP/IP Stack

Data streaming to the remote PC is managed by a Wiznet W5500 controller. To maximize throughput, the FPGA communicates with the W5500 via a dedicated SPI bus, operating in parallel with the ADC acquisition. The W5500 offloads the TCP/IP stack in hardware, reducing the logic footprint on the Spartan 7. The VHDL controller manages the socket pointers and performs block-based writes to the W5500 internal buffers, optimizing the SPI bandwidth.

6.1.3 Performance Results and Resource Utilization

The experimental characterization of the DAQ system confirmed the following performance metrics:

- **Throughput**: A stable transmission rate of 100 Mb/s was achieved, saturating the Fast Ethernet bandwidth without incurring data congestion.
- **Latency**: The measured end-to-end latency is approximately 0.1 ms. While the FPGA internal processing latency remains in the microsecond range, the bulk of the delay is attributed to the TCP/IP encapsulation and network buffering.
- **Resource Footprint**: The design is highly efficient, utilizing only 2–10% of the available LUTs and FFs, and 2% of the BRAM for synchronization FIFOs. The acquisition logic operates at a clock frequency of 100 MHz, ensuring a high timing margin.

6.1.4 Data Packet Formatting and Transmission Logic

The acquired 14-bit ADC samples are encapsulated into an 80-bit extended data packet to ensure temporal traceability and data integrity. Each packet includes a 32-bit incremental counter and a 32-bit timestamp, providing a unique identifier for loss detection and a microsecond-resolution reference for event reconstruction.

To sustain the continuous high-speed data flow without bottlenecks, the Spartan 7 FPGA utilizes a dual-bus SPI configuration. This architectural choice decouples the acquisition from the transmission phase: while the first SPI bus interfaces with the ADC, a dedicated secondary SPI bus manages the communication with the Wiznet W5500 Ethernet module. Data are temporarily stored in an internal dual-clock FIFO buffer to resolve potential clock domain crossing issues and to smooth out transmission jitter.

The W5500 is configured in TCP/IP socket mode, offloading the entire network stack from the FPGA fabric. The VHDL firmware implements a specialized controller that writes data payloads directly into the W5500 sockets via block-based SPI transactions, minimizing communication overhead.

6.1.5 Experimental Performance and Resource Utilization

The experimental validation of the DAQ system demonstrated performance metrics aligned with real-time monitoring requirements (see Figure 6.2):

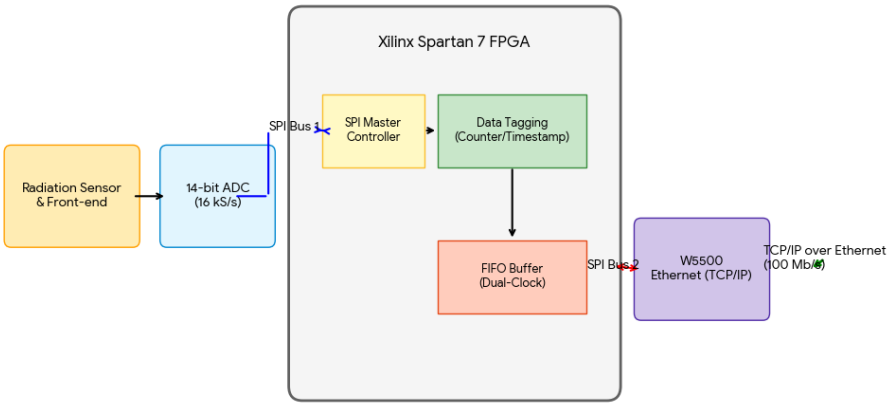

Figure 6.2: Close-up of the Spartan-7 FPGA and the analog front-end integration used for the ELI Beamlines case study.

- **Throughput:** The system successfully sustained a sampling frequency of 16 kS/s with a stable Ethernet throughput of approximately 100 Mb/s, saturating the Fast Ethernet interface with zero sample loss.
- **Latency:** The measured end-to-end latency is approximately 0.1 ms. While the FPGA internal processing contributes only a few microseconds, the bulk of the delay originates from the TCP/IP encapsulation and network buffering. For the intended radiation monitoring application, this sub-millisecond latency is well within the acceptable threshold.

- **Hardware Resource Footprint:** The VHDL implementation is highly efficient. The design utilizes only 2–10% of the available LUTs and FFs, and approximately 2% of the on-chip BRAM.

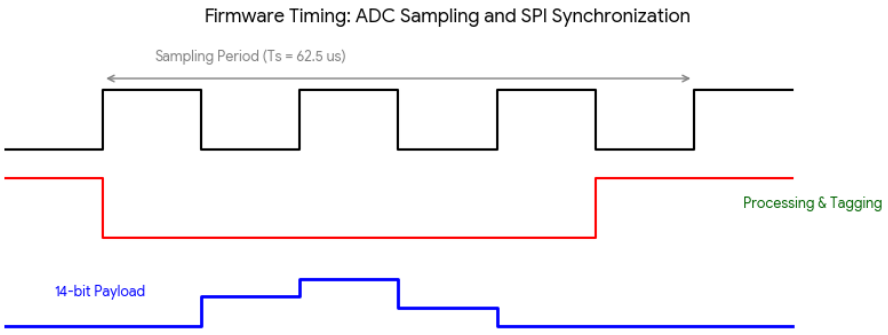

Figure 6.3: Oscilloscope capture showing the synchronization signals between the Spartan-7 SPI master and the ADC conversion clock.

The acquisition logic operates at a clock frequency of 100 MHz, leveraging the integrated DSP blocks for high-speed counter implementation and real-time numerical filtering. Notably, the entire control chain is implemented in pure hardware logic through Finite State Machines (FSMs), avoiding the overhead and non-deterministic behavior of soft-core microprocessors. This ensures a strictly deterministic timing behavior, essential for pulse-mode radiation monitoring.

In summary, the Spartan-7 based DAQ architecture provides a robust and scalable foundation. Future developments, discussed in the next section, will explore the integration of *machine learning* and genetic algorithms directly into the FPGA fabric to enable intelligent data reduction and in-situ feature extraction.

6.2 Integration of Genetic Algorithms on FPGA: Future Perspectives

A promising evolution of the ELI Beamlines monitoring system involves the integration of Evolutionary Computation directly into the Spartan-7 fabric. The hypothesized adoption of Genetic Algorithms (GA) aims to perform autonomous, real-time data pre-processing and adaptive parameter optimization (e.g., dynamic thresholding) before network transmission [58].

6.2.1 Genetic Algorithms Overview

GAs are heuristic search methods inspired by Darwinian natural selection, designed to evolve a population of candidate solutions toward an optimal state through iterative evaluation and recombination [59]. In the context of radiation monitoring, a GA could dynamically optimize the signal-to-noise ratio (SNR) by evolving the coefficients of the hardware filters based on the varying background radiation levels.

The classical generational flow—Initialization, Evaluation (Fitness), Selection, Crossover, and Mutation—is particularly suited for FPGA acceleration due to its inherent fine-grained parallelism.

6.2.2 Hardware Architecture and Preliminary Resource Analysis

Translating a GA into a VHDL-based digital architecture requires a pipelined approach to handle the population individuals concurrently. Preliminary implementation tests on the Spartan-7 platform indicate the feasibility of this approach, as evidenced by the floorplanning and resource analysis.

The preliminary synthesis results, summarized in Figure 6.5, demonstrate that a compact GA engine utilizes approximately 15–20% of the available LUTs and DSPs. This confirms that the current DAQ design (which utilizes only 2–10% of resources) has sufficient hardware headroom to host an intelligent pre-processing layer without requiring a larger FPGA device.

The proposed architecture leverages BRAM blocks to store the population chromosomes and DSP slices for parallel fitness evaluation. A hardware-based Linear Feedback Shift Register (LFSR) is employed to provide the pseudo-randomness required for the mutation and crossover operators, ensuring deterministic yet evolutionary-driven adaptation [60].

By performing this "intelligent analysis" locally, the required Ethernet bandwidth could be drastically reduced, transmitting only classified events or optimized parameters instead of raw high-frequency streams.

6.2.3 Edge Intelligence and Hardware Evolutionary Adaptation

The transition from a standard DAQ to anintelligent sensing node is enabled by the integration of GAs directly into the FPGA fabric. In the context of radiation monitoring, this architecture provides adaptive real-time optimization, reducing the

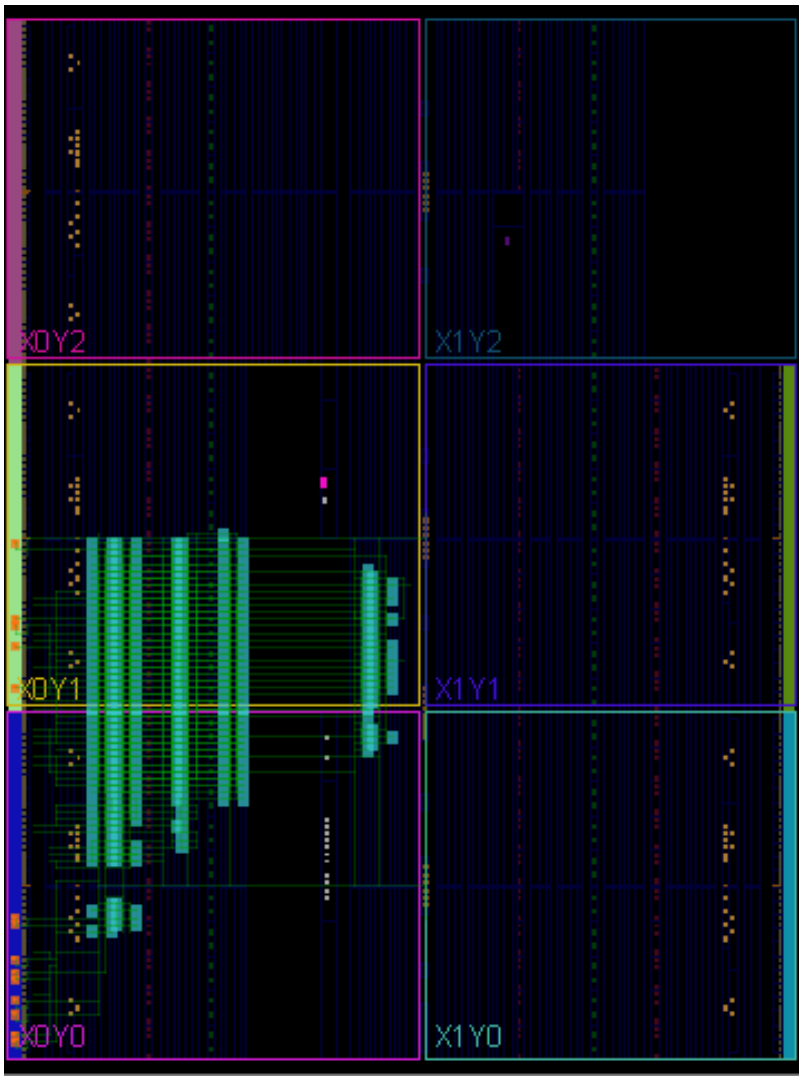

Figure 6.4: FPGA floorplanning and placement of the GA cores on the Spartan-7 fabric, showing the distributed logic allocation.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	893	0	0	32600	2.74
LUT as Logic	893	0	0	32600	2.74
LUT as Memory	0	0	0	9600	0.00
Slice Registers	2449	0	0	65200	3.76
Register as Flip Flop	2449	0	0	65200	3.76
Register as Latch	0	0	0	65200	0.00
F7 Muxes	265	0	0	16300	1.63
F8 Muxes	128	0	0	8150	1.57

Figure 6.5: Synthesis resource utilization table for the GA implementation. The low footprint in terms of LUTs and DSPs highlights the efficiency of the VHDL-based evolutionary engine.

dependency on host-side post-processing and improving system responsiveness. For instance, a hardware-implemented GA can continuously tune filtering thresholds to discriminate significant radiation pulses from stochastic background noise. By performing this in-situ analysi, the FPGA transmits only classified events or optimized parameters, drastically reducing the required Ethernet bandwidth.

As illustrated in the preliminary synthesis results (Figure 6.5), implementing a GA on a Spartan-7 requires mapping the evolutionary flow into a parallel and pipelined digital architecture. The proposed hardware modules include:

- **On-chip Population Memory**: Utilizing BRAM blocks to store candidate solutions, enabling single-cycle access for the fitness unit.
- **Parallel Fitness Engine**: Leveraging DSP slices to evaluate multiple individuals concurrently, accelerating the selection process.
- **Selection and Recombination Units**: Implementing tournament selection and bit-level crossover logic through hardware comparators and multiplexers.
- **Mutation Module**: Introducing genetic diversity via a hardware-based LFSR as a pseudo-random number generator.
- **FSM Controller**: A VHDL-based Finite State Machine (FSM) that orchestrates the generational transitions (Evaluation $\rightarrow$ Selection $\rightarrow$ Recombination).

6.2.4 Design Challenges and Verification

Despite the massive parallelization and the resulting evolution speeds—orders of magnitude higher than software-based implementations—several practical challenges remain. The resource-performance trade-off is critical: population size and fitness function complexity must be balanced against the Spartan-7 logic capacity to avoid frequency degradation.

Furthermore, unlike software GAs, hardware implementations are inherently deterministic and less flexible after synthesis. To mitigate this, the design incorporates configurable registers to allow the fine-tuning of evolutionary parameters (e.g., mutation rate, crossover probability) during runtime without requiring a full bitstream reconfiguration.

In conclusion, the integration of genetic algorithms represents a significant step toward autonomous and high-performance scientific instrumentation. While further simulations are required to optimize the fitness functions for specific radiation patterns, this approach provides the ELI Beamlines project with a scalable foundation for real-time, self-adaptive data acquisition.

7. Conclusions and Future Developments

In this thesis, the challenge of accelerating Convolutional Neural Networks (CNNs) on heterogeneous FPGA platforms has been addressed, with a specific focus on the state-of-the-art AMD Versal ACAP architecture. The research led to the design and implementation of a hardware/software accelerator on the VCK190 platform, effectively leveraging the synergy between PL, AIEs, and the embedded ARM Cortex-A72 PS.

7.1 Summary of Contributions

The proposed architecture achieves a transformative leap in performance by strategically partitioning the workload: the AIE array handles the computationally intensive tensor arithmetic, while the PL manages high-speed data orchestration and radiation-mitigation logic. This approach addresses the core research gap identified in this work: providing extreme AI throughput without compromising the deterministic reliability required for high-energy physics and aerospace environments.

- **Performance Records:** By employing 8-bit quantization and optimized AIE-PL mapping, the accelerator achieved a peak throughput of **15,000 FPS** with an ultra-low deterministic latency of **0.02 ms**. These results represent an order-of-magnitude improvement over traditional FPGA baselines and embedded GPU solutions.
- **Energy Efficiency:** The system demonstrated an efficiency of **6,666.7 FPS/W**. As shown in Table 7.1, the energy consumption per inferred image is significantly lower than software-based or previous-generation hardware solutions, validating the Versal platform as a sustainable alternative for Edge AI.

- **Radiation Resilience:** A major finding of this research is the effectiveness of the "30% PL utilization" strategy. By offloading the primary compute load to the hardened AIEs, 70% of the programmable logic fabric remains available for TMR and configuration scrubbing. Fault-injection campaigns confirmed that this overhead is critical for maintaining inference correctness in radiation-prone environments, such as the ELI Beamlines facility.

Implementation	**Throughput [FPS]**	**Energy Efficiency [FPS/W]**
Versal Accelerator (Proposed)	**15,000**	**6,666.7**
Zynq UltraScale+ (Baseline) [13]	∼3,000	∼250
NVIDIA Jetson AGX [14]	∼1,500	∼110

Table 7.1: Final Performance Benchmark: Versal Accelerator vs. Baseline.

7.2 Future Perspectives: Towards Intelligent Radiation-Hardened Nodes

The results presented in this thesis establish a scalable foundation for the next generation of autonomous sensing instruments. The following research directions are proposed to further extend the capabilities of the developed architecture:

- **In-situ Evolutionary Adaptation**: The preliminary integration of GA on the Spartan-7 platform (see Chapter 6) paves the way for self-adaptive DAQ systems. This would enable the sensor to autonomously adapt to varying radiation backgrounds without human intervention.
- **Sub-8-bit Quantization and Extreme Throughput**: While INT8 quantization provided an optimal accuracy-performance trade-off, exploring INT4 or Ternary Weight Networks (TWNs) could further reduce the memory footprint and power consumption. Such extreme quantization, coupled with the Versal AIE-ML architecture, targets a throughput exceeding 50,000 FPS, which is critical for future ultra-high-rate laser-driven experiments.
- **Hardware-Level Anomaly Detection**: Exploiting the 70% available headroom in the PL, future iterations will implement hardware-based safety monitors. By using redundant AIE kernels and real-time consistency checks, the system

could achieve "self-healing" capabilities, where radiation-induced SDC is not only detected but corrected on-the-fly through local re-execution or dynamic reconfiguration.

- **Multi-Sensor Fusion at the Edge**: A promising development involves scaling the architecture to handle multiple heterogeneous data streams (e.g., combining PIN photodiodes with spectroscopic detectors). The high-bandwidth NoC of the Versal platform is ideally suited to orchestrate such multi-modal data fusion, enabling more comprehensive environmental characterization in complex physics facilities like ELI Beamlines.

In conclusion, this work demonstrates that the era of 'dumb' high-speed acquisition is evolving towards Intelligent Edge Sensing, where performance and reliability are no longer a trade-off but a synergistic design goal.

.

References

[1] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] S. Mittal, "A Survey of FPGA-Based Accelerators for Convolutional Neural Networks," *Neural Computing and Applications*, vol. 32, pp. 1109–1139, 2020.

[3] AMD/Xilinx, "Versal ACAP Architecture Generic User Guide (UG1353)," 2023.

[4] AMD, "Versal AI Core Series Data Sheet: DC925," v1.5, 2023.

[5] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[6] M. Ceschia *et al.*, "Identification and classification of single-event upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2088–2094, 2003.

[7] P. Shivakumar *et al.*, "Modeling the effect of technology trends on the soft error rate," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 389–398.

[8] C. Carmichael, "Triple Modular Redundancy (TMR) for Xilinx FPGAs," Xilinx Application Note XAPP197, 2001.

[9] J. Carreira, H. Madeira, and J. G. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[10] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 2704–2713.

[11] I. Hubara *et al.*, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[12] AMD/Xilinx, "Versal ACAP Architecture Manual," AM011, v1.2, 2023.

[13] K. Guo *et al.*, "Angel-Eye: A Complete Design Flow for Accelerating CNN on Embedded FPGA," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, 2016, pp. 1–6.

[14] NVIDIA Corporation, "NVIDIA Jetson AGX Xavier: Performance and Efficiency for Autonomous Machines," White Paper, 2021.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[16] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[17] AMD/Xilinx, "Vitis AI User Guide (UG1414)," v3.5, 2023.

[18] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 448–456.

[19] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," *arXiv preprint arXiv:1606.08415*, 2016.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015.

[23] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder–decoder architecture for image segmentation," *IEEE Trans. PAMI*, vol. 39, no. 12, pp. 2481–2495, 2017.

[24] G.-S. Xia *et al.*, "iSAID: A Large-scale Dataset for Instance Segmentation in Aerial Images," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, 2019, pp. 141–150.

[25] 25 J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. CVPR*, 2016.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015.

[27] W. Liu, D. Anguelov, D. Erhan *et al.*, "SSD: Single Shot MultiBox Detector," in *Proc. ECCV*, 2016.

[28] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," in *Proc. CVPR*, 2015.

[29] G.-S. Xia, J. Hu, F. Hu *et al.*, "AID: A benchmark data set for performance evaluation of aerial scene classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 7, pp. 3965–3981, 2017.

[30] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 10, pp. 7019–7037, 2017.

[31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. ICCV*, 2017.

[32] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. ICCV*, 2017.

[33] G.-S. Xia, X. Bai, J. Ding *et al.*, "DOTA: A large-scale dataset for object detection in aerial images," in *Proc. CVPR*, 2018.

[34] D. Lam, R. Kuzma, K. McGee *et al.*, "xView: Objects in context in overhead imagery," *arXiv preprint* arXiv:1802.07856, 2018.

[35] X. Tian, J. Li, F. Zhang, H. Zhang, and M. Jiang, "Forest aboveground biomass estimation using multisource remote sensing data and deep learning algorithms: A case study over Hangzhou area in China," *Remote Sensing*, vol. 16, no. 6, p. 1074, 2024.

[36] L. Huang, "Input image compared with the ground-truth and prediction for semantic segmentation," *ResearchGate*, Jan. 2020. [Online]. Available: https://www.researchgate.net

[37] A. Khan *et al.*, "The proposed CNN-LSTM model architecture for spatial-temporal prediction," *ResearchGate*, Nov. 2021. [Online]. Available: https://www.researchgate.net

[38] A. Ben Abbes, N. Jarray, and I. R. Farah, "Advances in remote sensing-based soil moisture retrieval: Applications, techniques, scales and challenges for combining machine learning and physical models," *Artificial Intelligence Review*, vol. 57, p. 224, 2024.

[39] AMD/Xilinx, "Versal ACAP: A New Category of Heterogeneous Compute for AI and Beyond," White Paper WP505, 2023.

[40] J. Cong, B. Liu, and S. Neuendorffer, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, 2011.

[41] K. Hwang and N. Jotwani, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, 3rd ed. New York, NY, USA: McGraw-Hill, 2016.

[42] M. Fingeroff, *High-Level Synthesis Blue Book*. Mentor Graphics, 2010. [Fondamentale per II e Pipelining]

[43] P. Kaul et al., "Dataflow Architectures for CNN Acceleration on FPGAs," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Archit. Processors (ASAP)*, 2018.

[44] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, 2015.

[45] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge Distillation: A Survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021.

[46] S. de Vieilleville *et al.*, "Deep Learning for Satellite Image Segmentation on Nanosatellites," *Remote Sens.*, vol. 12, no. 22, 2020.

[47] S. Rapuano, A. Rizzo, L. Capozzi, *et al.*, "An FPGA-Based Hardware Accelerator for CNNs Inference On-Board Satellites: Benchmarking with Myriad-2-Based Solution for the CloudScout Case Study," *Remote Sensing*, vol. 13, no. 22, Art. 4581, 2021.

[48] Xilinx Inc., "VCK190 Evaluation Board User Guide," User Guide UG1366, v1.5, Jan. 2023.

[49] J. Shao et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019.

[50] AMD/Xilinx, "AI Engine Kernel and Graph Programming Guide," User Guide UG1076, v2.0, 2023.

[51] AMD/Xilinx, "Versal ACAP AI Engine Architecture Manual," Architecture Manual AM009, v2.1, 2023.

[52] Wiznet Co., Ltd., "W5500 Datasheet: Hardwired TCP/IP Embedded Ethernet Controller," v1.1, 2023.

[53] Xilinx Inc., "LogiCORE IP Ten Gigabit Ethernet PCS/PMA (10GBASE-R) Product Guide (PG068)," v6.0, 2023.

[54] Xilinx Inc., "7 Series FPGAs SelectIO Resources User Guide," UG471, v1.10, 2023.

[55] Microsoft, "Quantize ONNX Models," *ONNX Runtime Documentation*, 2024.

[56] B. Rus *et al.*, "ELI-Beamlines: The European infrastructure for high-power laser sources and applications," in *Proc. SPIE 8080, Optics and Photonics for Counterterrorism and Crime Fighting VII*, vol. 8080, Oct. 2011, Art. no. 808001.

[57] G. A. P. Cirrone *et al.*, "Real-time beam monitoring for high-energy laser-driven particle accelerators," *Nucl. Instrum. Methods Phys. Res. A*, vol. 620, no. 2-3, pp. 551–555, Aug. 2010.

[58] S. Tsutsui and P. Collet, *Genetic Algorithms on FPGA*. New York, NY, USA: Springer, 2013.

[59] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.

[60] P. Alfke, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators," Xilinx Application Note XAPP052, 2012.