





Machine Learning Model Compression for Radiation/Particle Classification on SoC-FGPA

Location: **ICTP MLab, Trieste Italy**

PROJECT SUPERVISOR's **DR. Romina Molina, DR. Maynor Ballina**

Trainees:
Mohammad H.Alzahrani | Electrical Engineer
Moath A.Algahtani | Software Engineer

Aug 2025

TABLE OF CONTENTS

ABSTRACT **3**

1. INTRODUCTION **3**

2. PROJECT OBJECTIVES AND GOALS **4**

3. METHODOLOGY AND APPROACH **4**

3.1. Dataset and Preprocessing 4
3.2. Neural Network Archtectures 5
3.3. Compression Techniques 5
3.4. Hardware-Software Co-Design Workflow 5

4. RESULTS **7**

4.1. Quantization-Aware Training without Knowledge Distillation 7
4.2. Effect of Knowledge Distillation 8
4.3. Interpretation 8

5. CHALLENGES AND OBSTACLES **8**

6. NEXT STEPS AND FUTURE WORK **9**

7. RESOURCES AND TOOLS USED **9**

REFERENCES **10**

APPENDIX **11**

LIST OF FIGURES

Figure 1. Sample gamma and neutron radiation signals from the dataset. 4
Figure 2. Structure of the classification neural network used for gamma–neutron discrimination. 5
Figure 3. ComBlock interface architecture for PS–PL communication. 6
Figure 4. Physical view of the FPGA cluster at ICTP MLab. 7
Figure 5. Validation accuracy of models under quantization-aware training (QAT) without knowledge distillation. 7
Figure 6. Validation accuracy of models under QAT with knowledge distillation. 8

ABSTRACT

This paper reports the progress of deploying compressed neural networks for radiation particle classification on system-on-chip field-programmable gate arrays (SoC-FPGAs). The project addresses the challenge of meeting strict memory and latency requirements while maintaining classification accuracy between gamma and neutron signals. Neural networks with dense and convolutional layers were developed and subjected to pruning, quantization, and knowledge distillation. Experiments demonstrated that quantization-aware training preserved near-perfect accuracy at 16-bit precision, while aggressive 4-bit quantization led to performance collapse. Knowledge distillation enhanced the stability of compressed student models, achieving approximately 99% accuracy when combined with 16-bit quantization. These findings align with state-of-the-art approaches such as deep compression pipelines [1] and distillation strategies [2], while being adapted to FPGA workflows for resource-constrained deployment [3].

1. INTRODUCTION

The classification of radiation events, specifically the discrimination between gamma and neutron particles, is a central task in nuclear safety, high-energy physics, and medical imaging. Traditional methods relying on handcrafted features struggle under real-time constraints and overlapping signals. Deep learning has emerged as a promising alternative, offering automated feature extraction and robust classification capabilities.

However, modern neural networks are resource-intensive, posing challenges for embedded deployment. Han et al. showed that pruning and quantization can compress large models by nearly 50× without accuracy loss [1]. Hinton et al. introduced knowledge distillation, where a smaller student model inherits generalization ability from a larger teacher [2]. Recent reviews confirm that hybrid compression techniques are essential for deploying efficient AI at the edge [4].

For FPGA deployment, Molina et al. demonstrated end-to-end workflows combining pruning, quantization, and KD on SoC platforms, maintaining accuracy while drastically reducing resource usage [3]. Xiang et al. further showed that activation-aware quantization can accelerate large models on edge devices [5]. These insights provide the foundation for

applying compression to radiation classification, where real-time performance and hardware efficiency are critical.

This project extends these advances by designing, compressing, and deploying gamma–neutron classifiers on Zynq-based SoC-FPGAs, integrating pruning, quantization, quantization-aware training, and knowledge distillation within a hardware–software co-design framework.

2. Project Objectives and Goals

The project's objectives are fourfold. First, to design convolutional–dense neural networks capable of distinguishing gamma and neutron radiation events with high accuracy. Second, to systematically apply compression techniques—pruning, quantization, and KD—to create compact yet accurate models suitable for FPGA hardware. Third, to implement a hardware–software co-design workflow on Zynq SoCs, partitioning tasks between processing system (PS) and programmable logic (PL) while using ComBlock IP for communication. Fourth, to deploy and validate compressed models on FPGA boards and clusters, measuring latency, throughput, and resource usage.

3. Methodology and Approach

3.1. Dataset and Preprocessing

The dataset consisted of 10,913 gamma and 27,696 neutron signals. Gamma signals exhibited sharper peaks and noise variability, while neutron signals showed smoother decays. A 70–30 split ensured generalizable training and validation.

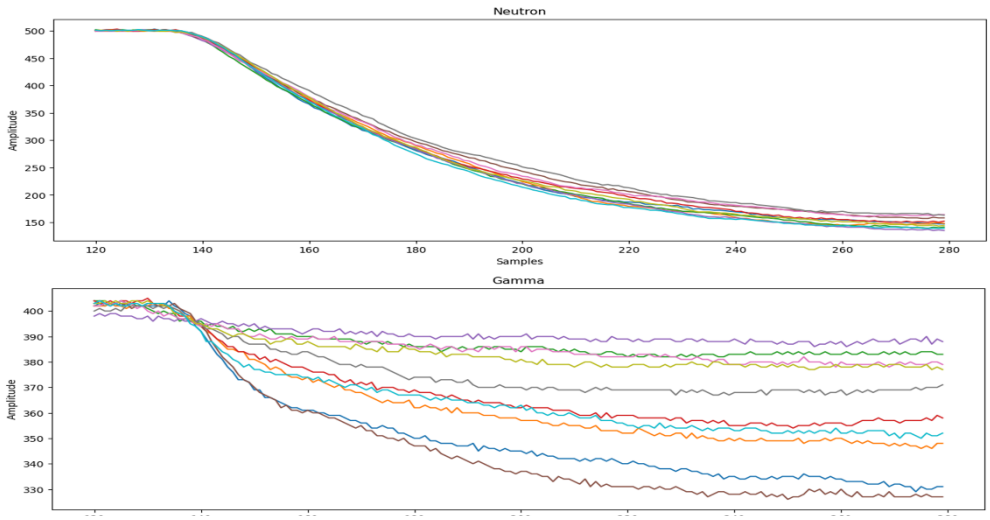


Figure 1. Sample gamma and neutron radiation signals from the dataset.

3.2. NEURAL NETWORK ARCHTECTURES

The classification model combined one-dimensional convolutional layers for temporal feature extraction from a sample signal with dense layers for classification. ReLU activations were used in hidden layers, sigmoid in the output, and binary cross-entropy as the loss function.

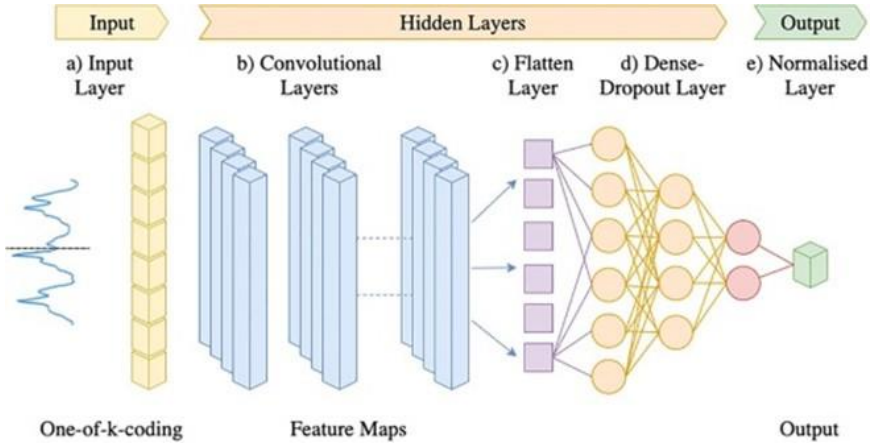


Figure 2. Structure of the classification neural network used for gamma–neutron discrimination.

3.3. COMPRESSION TECHNIQUES

Pruning eliminated low-magnitude weights, with structured pruning preferred for hardware efficiency. Quantization reduced precision from 32-bit to 8- or 16-bit integers, aided by quantization-aware training (QAT). Knowledge distillation (KD) trained small student models on teacher outputs, improving performance under compression [2]. Quantization-aware pruning (QAP) combined pruning and quantization in training, aligning with state-of-the-art workflows [4].

3.4. HARDWARE-SOFTWARE CO-DESIGN WORKFLOW

Deployment targeted Zynq-7000 SoCs, which integrate an ARM Cortex-A9 processing system (PS) with FPGA programmable logic (PL). This heterogeneous architecture allowed for the partitioning of tasks: high-level control and data movement were managed by the PS, while low-level computation such as convolution and dense matrix multiplications were accelerated in the PL. Vivado was employed for hardware block design, providing a graphical IP integrator for creating block diagrams, while Vitis supported embedded software development, enabling the PS to configure and monitor custom

accelerators in the PL. HLS4ML served as the bridge between machine learning models and hardware implementation, automatically converting trained TensorFlow/Keras networks into C++ and synthesizable RTL. Optimization pragmas such as loop pipelining, loop unrolling, and array partitioning were applied to maximize concurrency and minimize latency.

Communication between the PS and PL relied on the ComBlock IP, which abstracted the complexities of AXI interconnects and provided three modes of interaction: registers for direct monitoring, FIFO buffers for streaming data, and dual-port RAM for shared memory. This modular architecture simplified the exchange of commands and data during inference and was crucial for integrating compressed neural networks into the system. The design of the ComBlock IP is shown in **Figure 3**, which illustrates its memory-mapped registers, FIFOs, and shared memory interface.

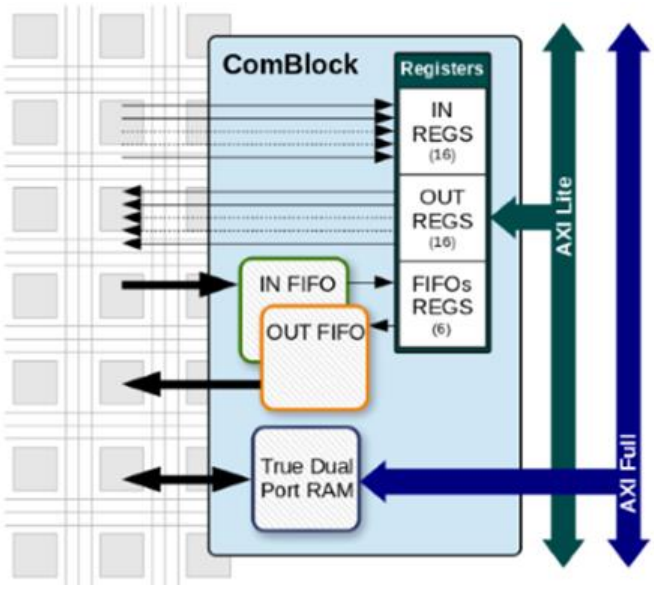


Figure 3. ComBlock interface architecture for PS–PL communication.

To validate large-scale experiments, the deployment extended beyond a single board to the FPGA cluster at ICTP. As shown in **Figure 4**, this cluster consisted of multiple interconnected ZedBoards configured to perform distributed computation, allowing compressed models to be tested in parallel and benchmarked under cluster conditions. This setup enabled real-time testing of scalable inference scenarios, which are critical for radiation detection systems where throughput and redundancy are essential.



Figure 4. Physical view of the FPGA cluster at ICTP MLab.

4. RESULTS

4.1. QUANTIZATION-AWARE TRAINING WITHOUT KNOWLEDGE DISTILLATION

At 32-bit and 16-bit precision, models achieved ~100% accuracy. At 8-bit, accuracy peaked near 80% but fluctuated, and at 4-bit, accuracy collapsed to ~50%, equivalent to random guessing. These results confirm that overly aggressive quantization destabilizes training [5].

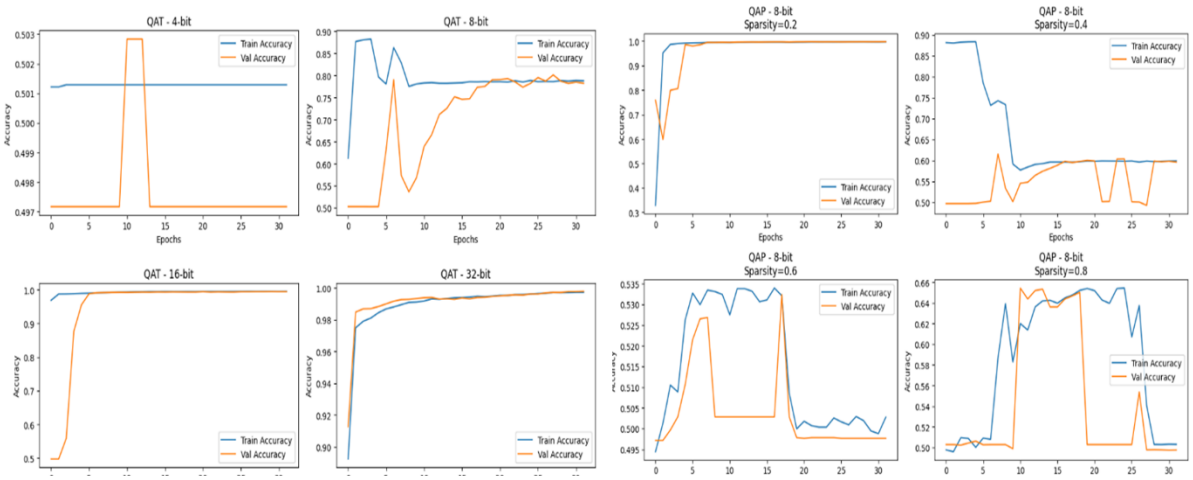


Figure 5. Validation accuracy of models under quantization-aware training (QAT) without knowledge distillation.

4.2. Effect of Knowledge Distillation

KD improved compressed models substantially. Student models with only 438 parameters achieved 75–80% accuracy at 8-bit QAP, though instability appeared beyond 0.2 sparsity. The strongest results came from KD combined with 16-bit QAT, preserving ~99% accuracy while reducing model size. These findings validate KD as a stabilizing factor in compressed FPGA-friendly networks [2–4].

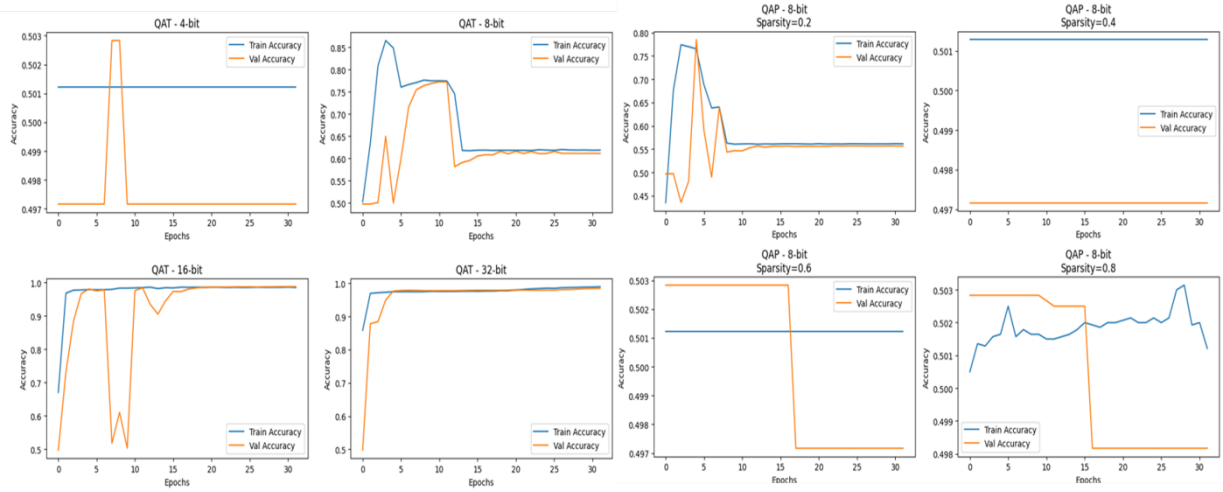


Figure 6. Validation accuracy of models under QAT with knowledge distillation.

4.3. Interpretation

Results show that pruning alone allowed up to 80% reduction with limited impact, but pruning combined with aggressive quantization caused collapse. KD mitigated these effects, allowing compressed models to retain generalization. The findings align with prior compression studies [1–4] and extend them to radiation signal classification.

5. Challenges and Obstacles

The major challenges were encountered during hardware synthesis and deployment. Vivado frequently crashed during bitstream generation, particularly with large projects, forcing repeated restarts. HLS4ML C++ generation and IP synthesis required long runtimes, sometimes hours, even for small modifications. Debugging was slowed by the lack of incremental synthesis, as each change required full re-synthesis and bitstream generation. These delays significantly extended the development cycle and limited the speed of experimentation, consistent with challenges noted in other FPGA workflows [3].

6. Next Steps and Future Work

Future work will focus on enhancing the efficiency and scalability of the deployment workflow. At present, each compressed model requires the entire pipeline—translation with HLS4ML, C++ synthesis, IP core generation, system integration in Vivado, and bitstream generation—to be repeated from scratch. This repetition is time-consuming and prone to errors, particularly given the instability of Vivado during bitstream generation.

The next stage of development will therefore seek to design an optimized workflow capable of deploying multiple models in a single integrated process, rather than repeating the complete toolchain for each model. Such an enhancement would allow comparative evaluation of different compression strategies under identical hardware conditions and would significantly reduce development time.

Additionally, remaining models that have not yet been fully deployed will be implemented on FPGA hardware to provide a complete performance benchmark. Scaling to multi-model deployment across FPGA clusters will be explored, enabling the simultaneous execution of multiple networks in distributed radiation detection scenarios. This direction will transform the workflow from a sequential, model-by-model approach into a flexible, parallelizable pipeline suitable for large-scale applications.

7. Resources and Tools Used

The hardware resources included Xilinx Zynq-7000 SoC boards (ZedBoard) and the FPGA cluster at ICTP. Vivado and Vitis were used for hardware design and embedded development, while Vivado/Vitis HLS supported IP core generation. HLS4ML bridged trained TensorFlow/Keras models to FPGA synthesis.

Training and compression were performed in Python using TensorFlow and the TensorFlow Model Optimization Toolkit (TFMOT). Knowledge distillation followed the teacher–student framework [2]. Preprocessing relied on NumPy and SciPy, and accuracy plots were generated with Matplotlib.

Communication between PS and PL was facilitated by ComBlock IP [3], which enabled register-based control, streaming through FIFOs, and shared memory access.

REFERENCES

[1] Han S, Mao H, Dally WJ. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. Proc. Int. Conf. Learn. Represent. (ICLR); 2016.

[2] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531; 2015.

[3] Molina RS, Morales IR, Crespo ML, Gil-Costa V, Carrato S, Ramponi G. An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. IEEE Embedded Systems Letters. 2021;14(8):1–4.

[4] Dantas PV, da Silva WS Jr, Cordeiro LC, Carvalho CB. A comprehensive review of model compression techniques in machine learning. Applied Intelligence. 2024;54:11804–44. https://doi.org/10.1007/s10489-024-05747-w.

[5] Xiang M, Fernando R, Wang B. On-device Qwen2.5: Efficient LLM inference with model compression and hardware acceleration. arXiv preprint arXiv:2504.17376; 2025.

[6] Coelho A, Molina RS, Crespo ML, Ramponi G, Carrato S. Automatic heterogeneous quantization of neural networks for deployment on FPGAs. IEEE Int. Conf. Reconfigurable Computing and FPGAs (ReConFig). 2021; p. 1–6.

[7] Coelho A, Molina RS, Crespo ML, Ramponi G. Automatic quantization for efficient neural network inference on SoC-FPGAs. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 2020;39(11):3769–82.

APPENDIX

https://drive.google.com/drive/u/0/mobile/folders/1u-zzeiYl2c-b40NMlP35H1ZEIgYokA1A?usp=sharing