From NAS to Hardware: Deploying Quantized Automatic Modulation Classification Models on FPGA for 6G Edge Intelligence

David Góez*, Paola Soto*, Nina Slamnik-Kriještorac*,
Natalia Gaviria Gomez†, Johann Marquez-Barja*, Miguel Camelo Botero*

*University of Antwerp - imec, IDLab, Belgium
† Universidad de Antioquia, Electronics Engineering Department, Colombia

***Abstract*—Future 6G edge-intelligent radios require neural networks that satisfy strict latency and energy constraints, yet the practical behaviour of quantization-aware Neural Architecture Search (NAS) models on real Field-Programmable Gate Array (FPGA) hardware remains insufficiently explored. This work evaluates a set of Pareto-optimal architectures generated by a quantization-aware NAS *MONAS-LQ*, together with state-of-the-art reference models, when deployed on an FPGA using the Brevitas-FINN-Vivado toolchain. The results show that all models found by *MONAS-LQ* maintain accuracy within 0.30% of server-side execution, with several exhibiting slight improvements, while energy consumption remains below 20 mJ/sample and latency is dominated by early convolutional layers rather than overall model size. Compared to existing quantized architectures, the *MONAS-LQ* models achieve more favourable accuracy–efficiency trade-offs. These findings highlight the relevance of hardware-aware NAS for deriving deployable and energy-efficient Deep Learning (DL) models tailored to the resource constraints of future edge-intelligent radio systems.**

***Index Terms*—FPGA, Deep Learning, Neural Networks, Model Optimization, Real-Time Inference.**

I. Introduction

Recent 3rd Generation Partnership Project (3GPP) efforts toward Fifth-Generation (5G)-Advanced mark a transition to Artificial Intelligence (AI)-Native Radio Access Network (RAN) architectures, paving the way for Sixth-Generation Mobile Networks (6G) systems in which Machine Learning (ML) models execute directly at the network edge [1], [2]. Such edge-centric intelligence must operate under strict latency and energy constraints while supporting autonomous decision making in heterogeneous and dynamic environments [3]. This shift toward a distributed *edge continuum* increases the need for efficient and dependable inference mechanisms deployable across diverse hardware platforms [4].

Automatic Modulation Classification (AMC) is a key enabler of resilient wireless systems [5], providing real-time spectrum awareness by classifying modulation formats from streaming In-Phase and Quadrature Components (I/Q) samples [6], [7]. As edge nodes from Unmanned Aerial Vehicles (UAVs) [8] to industrial Internet of Things (IoT) devices [9] become increasingly autonomous, local AMC is critical for interference mitigation, dynamic spectrum access, and robust connectivity in multi-node 6G scenarios [10].

Deep Learning (DL) has significantly advanced AMC, owing to its superior feature extraction capabilities and consistently higher classification accuracy across diverse signal representations and preprocessing strategies [11]. However, deploying such models at the edge remains challenging due to high sampling rates, real-time constraints, and limited computational resources. Field-Programmable Gate Array (FPGA)s offer an attractive solution by enabling deeply pipelined, low-power, and low-latency execution, but these benefits require Neural Network (NN) architectures that are explicitly designed for hardware efficiency [12]–[14].

Prior work has explored DL-based radio inference on reconfigurable hardware [12], [15], [16], and our earlier analysis showed the importance of quantization-aware design for efficient Convolutional Neural Network (CNN)-based AMC [7][1]. Nonetheless, most existing systems rely on fixed-size predefined architectures or uniform quantization strategies, limiting their suitability for heterogeneous and resource-constrained edge deployments. To address these limitations, our previous work introduced Multi-Objective Neural Architecture Search for Layer-wise Quantized (MONAS-LQ), a hardware-aware Neural Architecture Search (NAS) framework that jointly explores CNN topologies and layer-wise quantization strategies for efficient AMC [17]. While MONAS-LQ identifies Pareto-optimal *MLQ* models, the relationship between model complexity and performance on real edge hardware remains poorly understood.

To address this gap, this work evaluates Pareto-optimal architectures generated by quantization-aware MONAS-LQ alongside state-of-the-art quantized models [7], [15] using the Brevitas–FINN–Vivado toolchain on an FPGA. We measure inference latency, energy consumption, and classification accuracy under identical hardware constraints. This systematic comparison reveals that latency does not necessarily scale with model size, highlighting the importance of hardware-aware DL design for efficient edge-native intelligence.

The remainder of this paper is organized as follows. Section II presents the MONAS-LQ framework, Section III de-

[1]https://github.com/DGoezSanchez/A-Methodology-to-Design-Quantized-Deep-Neural-Networks-for-Automatic-Modulation-Recognition

ISBN 978-3-903176-82-9

scribes the FPGA deployment flow, Section IV reports the results, and Section V concludes the paper and outlines future work.

II. Overview of the MONAS-LQ Framework

The MONAS-LQ [17] framework is a hardware-aware NAS approach designed to automatically discover efficient layer-wise quantized CNN architectures for AMC. The method jointly explores NN depth and per-layer numerical precision under explicit accuracy–efficiency objectives, making it suitable for deployment on resource-constrained edge hardware such as FPGA platforms.

Unlike conventional NAS approaches that apply quantization only after architecture selection or single architectures with uniform precision across layers, MONAS-LQ incorporates quantization directly into the search space. This enables the joint optimization of network structure and bit-width assignments, allowing heterogeneous precision across layers.

The search problem is formulated as a multi-objective optimization that balances classification accuracy and computational efficiency. The computational efficiency is quantified using the Quantization-Aware Score (QA-Score), which combines Bit Operations per Second (BOPS) and weight bit-width (W-bits) into a normalized metric relative to a baseline architecture:

$$\text{QA-Score} = 0.5 \cdot \left(\frac{\text{BOPS}}{\text{BOPS}_{\text{baseline}}} \right) + 0.5 \cdot \left(\frac{\text{W-bits}}{\text{W-bits}_{\text{baseline}}} \right), \quad (1)$$

where the baseline architecture is used to normalize the computational metrics so that the QA-Score remains comparable across candidates. In MONAS-LQ, the chosen baseline is the 8-bit VGG10 model with 64 filters, which is commonly used in CNN-based AMC studies such as [15]. Therefore, **lower QA-Score is better**. This formulation ensures that candidate architectures are evaluated using metrics that correlate with hardware implementation cost.

To efficiently explore the design space, MONAS-LQ employs a multi-stage search pipeline:

- **Stage 1 — Hyperparameter Pre-Selection (HPO):** A lightweight hyperparameter search identifies a suitable number of convolutional filters for the backbone architecture. This step reduces the dimensionality of the subsequent NAS search space.
- **Stage 2 — Dataset Bootstrapping:** A set of randomly sampled architectures with layer-wise quantization is instantiated, fully trained, and evaluated. The resulting dataset contains accuracy, BOPS, and QA-Score measurements that serve as training data for surrogate modeling.
- **Stage 3 — Predictor-Guided Exploration:** A Multi-Objective Learning Predictor (MOLP) is trained to jointly estimate accuracy, BOPS, and the composite objective. MOLP consists of a shared feature extraction block followed by independent regression heads. This surrogate model enables rapid evaluation of large numbers of candidate architectures, significantly reducing the computational cost of the search.
- **Stage 4 — Hybrid PSO-Based NAS:** The architecture search is performed using a Particle Swarm Optimization (PSO) strategy. Most candidate architectures are evaluated using the predictor, while a subset is fully trained to refine the Pareto front and update the surrogate model. The search terminates using a patience-based early stopping rule once improvements stagnate.

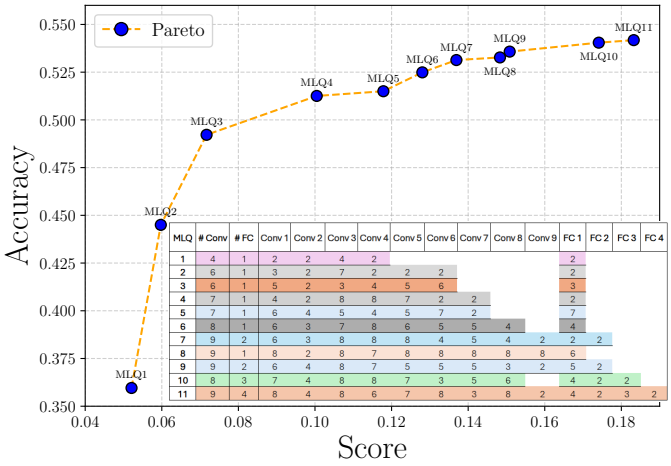

Figure 1: Pareto-optimal accuracy–efficiency trade-offs identified by the MONAS-LQ search, showing the set of Pareto-optimal architectures (denoted as *MLQ*) evaluated in this work.

Each candidate architecture is encoded as a fixed-length vector describing (i) the number of convolutional and dense layers and (ii) the quantization level of each layer. This representation supports heterogeneous bit-width assignments (1–8 bits) while remaining compatible with PSO update rules.

The output of MONAS-LQ consists of both the globally optimal architecture according to the composite objective and a set of Pareto-optimal solutions representing different accuracy-efficiency trade-offs. The Pareto models obtained during the search, illustrated in Figure 1, will be referred to hereafter as *MLQ*.

III. Deployment Flow for FPGA-Based Acceleration

The architectures selected through MONAS-LQ are subsequently translated into hardware accelerators and deployed on the Zynq-7000 ZC702 FPGA. Figure 2 shows the end-to-end workflow, which follows the standard Brevitas-FINN-Vivado toolchain for low-precision CNN deployment.

The deployment process consists of five stages:

1) **Quantization-Aware Training:** Each selected model is instantiated with its assigned layer-wise bit-widths and trained offline using Brevitas. All training is performed on the host machine; the FPGA is only used for inference during evaluation.
2) **Model Export:** The trained network is exported to ONNX and processed by FINN, which lowers the graph into hardware-compatible streaming operators while preserving the quantization metadata.

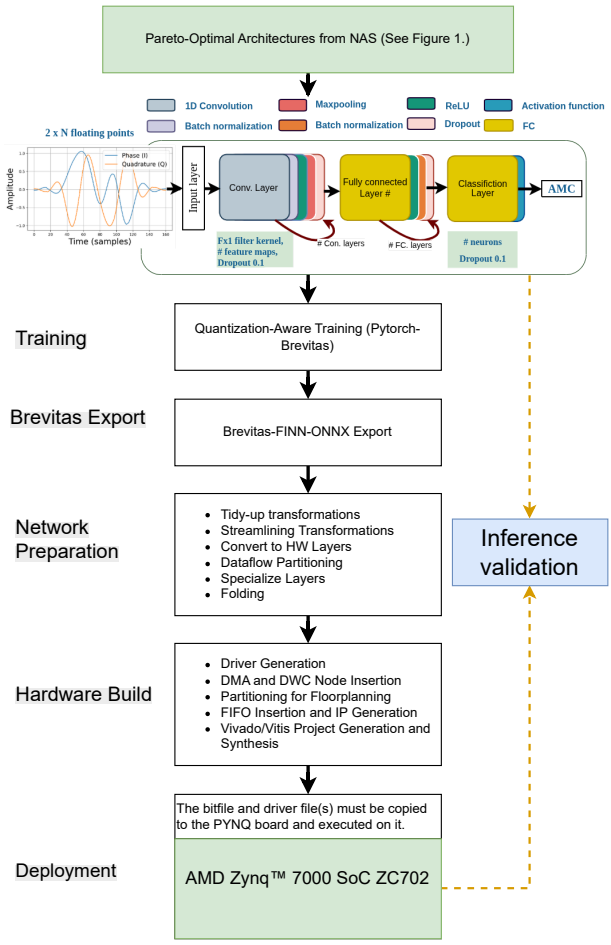

Figure 2: End-to-end design flow from quantized model selection to FPGA deployment.

3) **Network Preparation:** FINN applies graph transformations—including streamlining, layer conversion, dataflow partitioning, specialization, and operator folding—to enable efficient execution in a fully pipelined architecture.
4) **Hardware Build:** The transformed graph is synthesized into a dataflow accelerator. FINN generates the required IP blocks and AXI interfaces, and Vivado produces the final bitstream for the ZC702 device.
5) **Deployment and Validation:** The generated bitstream is loaded onto the board, and the accelerator is evaluated in inference-only mode to measure latency, energy consumption, and functional correctness under identical hardware conditions.

A. Target FPGA Platform

The implementation targets the Zynq-7000 ZC702 development board, whose programmable logic resources impose practical limits on accelerator size and parallelism. Table I summarizes key features, including Look-Up-Tables (LUT)s, registers, Block Random Access Memory (BRAM), and Digital Signal Processing slice (DSP) slices, to aid interpretation of the results in Section IV-A.

Table I: Technical Specifications of the Zynq-7000 FPGA Development Board.

Programmable Logic (FPGA Fabric)	
FPGA device	Xilinx Zynq-7000 XC7Z020-1CLG484C
Logic cells	≈85 k
Slice LUTs	53 200
Slice registers	106 400
Slices	13 300
Block RAM	140 × 36 Kb (4.9 Mb total)
DSP slices	220
User I/O (PL)	up to 200 I/O pins
PL I/O banks	4 high-range I/O banks (1.2–3.3 V)
Clock management	up to 4 clock management tiles (CMTs)

The number of available LUTs and registers determines how many compute and control operators can run in parallel. DSP slices constrain the maximum number of simultaneous multiply-accumulate operations, which is critical for convolution-heavy designs. BRAM availability limits the amount of locally stored weights and feature maps; insufficient BRAM forces more external memory access, increasing latency and power consumption.

B. Streaming CNN Accelerator Architecture

Within FINN, each NN layer is mapped to its own hardware stage in a streaming pipeline. Pipeline registers are inserted automatically to satisfy timing constraints. Once the pipeline is filled, the design achieves an initiation interval of $\mathrm{II} = 1$, accepting a new input every clock cycle. The inference latency L is therefore set by the pipeline depth D and the clock period $T_{\mathrm{clk}} = 1/F_{\max}$.

Convolutional and fully connected layers are implemented using the Matrix–Vector Units (MVAU), which performs matrix-vector multiplication followed by activation. Parallelism is controlled by the number of Processing Elements (PE) and Single Instruction, Multiple Data (SIMD) lanes. PE defines how many output channels are computed in parallel, while SIMD defines how many input channels participate simultaneously in each dot product.

For a convolutional layer with input size $H \times W$, kernel size $K \times K$, C_{in} input channels, and C_{out} output channels, the approximate cycle count of the MVAU is

$$\mathrm{CYC}_{\mathrm{MVAU}} = \frac{H \cdot W \cdot C_{\mathrm{out}} \cdot K^2 \cdot C_{\mathrm{in}}}{\mathrm{PE} \cdot \mathrm{SIMD}}. \tag{2}$$

The total latency of layer L_i also includes padding, convolution input generation, activation thresholding, and optional pooling:

$$\begin{aligned}\mathrm{Latency}(L_i) = \mathrm{CYC}_{\mathrm{Pad}_i} + \mathrm{CYC}_{\mathrm{CIG}_i} + \mathrm{CYC}_{\mathrm{MVAU}_i} \\ + \mathrm{CYC}_{\mathrm{Thresh}_i} + \mathrm{CYC}_{\mathrm{Pool}_i}.\end{aligned} \tag{3}$$

In practice, the MVAU is the dominant contributor to inference time.

C. FPGA System Integration

The final accelerator is packaged as a Vivado IP core using Xilinx Vivado Design Suite (v2022.1-64-bit) and integrated via AXI-Stream interfaces.

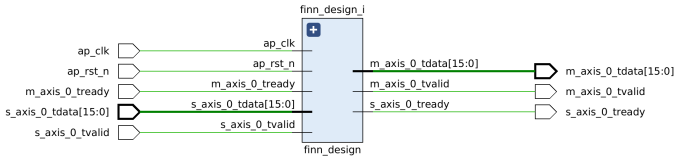

Figure 3: AXI-Stream interface of the MLQ-based accelerator, showing clocking, reset, and streaming data signals used for end-to-end inference on the FPGA

Figure 3 shows the main interface signals: system clock (`ap_clk`), reset (`ap_rst_n`), input stream (`s_axis_0_tdata`), and output predictions (`m_axis_0_tdata`). The `tvalid`/`tready` handshake ensures synchronized streaming between modules. Together, these interfaces enable continuous dataflow through the accelerator, allowing end-to-end inference to run entirely in the programmable logic without processor intervention.

IV. Experimental Results

Following our previous work [17], the eleven *MLQ* models (i.e., the Pareto-optimal architectures produced by MONAS-LQ and shown in Figure 1) were evaluated together with three state-of-the-art quantized baselines: uniform quantized FINN B [15], non-uniform quantized Q-CNN with 32 (QCNN-32F) and 64 filters (QCNN-64F) [7], and uniform 8-bit VGG10 model with 32 filters. The results are organized into three categories: FPGA computational resource utilization, inference latency, and energy consumption.

In terms of dataset, all experiments were conducted using the RadioML 2018.01A dataset [18][2]. A homogeneous 10% subset of the dataset was employed, following the same data partitioning methodology adopted in our previous works [7], [17].

A. FPGA computational resource utilization

FPGA computational resource utilization is evaluated using standard hardware metrics, including Multiplication, Addition and Multiply-accumulate (MAC), LUTs, registers, BRAM, and DSP slices. These parameters are fundamental for assessing the efficiency, scalability, and practical feasibility of deploying each model on the target hardware platform. Table II reports the FPGA resource utilization on the ZC702.

Resource utilization on the ZC702 is analyzed with respect to (i) the total device capacity from Table I and (ii) the computational load of the QCNN(64F) model, which defines the reference baseline with 12,864,512 MAC operations (100%). Under this criterion, the eleven *MLQ* models (Pareto 1–11) require between 2,998,272 and 3,353,600 MAC, corresponding to 23.3–26.1% of the QCNN(64F) workload, i.e., a 3.8–4.3× reduction in computational complexity.

In terms of logic resources, the *MLQ* configurations utilize 14,211–18,041 LUTs (26.7–33.9%) and 24,441–35,577 registers (23.0–33.4%), remaining consistently below the QCNN(64) utilization levels. BRAM usage ranges from 20 to 64 BRAM36 blocks (14.3–45.7%), well below the QCNN(64) requirement of 107 BRAM36 blocks, while DSP consumption is limited to 6–11 units (2.7–5.0%). Although QCNN(64) exhibits the highest BOPS and weight precision, its overall Score increases only marginally, indicating diminishing efficiency gains as model complexity grows.

By contrast, the *MLQ* models achieve a more favorable trade-off, progressively improving their Score through controlled increases in BOPS and weight bit-width. Nevertheless, such aggregate metrics alone are insufficient to characterize how computation is distributed across the network. As a result, FPGA resource utilization depends not only on quantization precision and network depth, but also on architectural balance, operator folding[3], and dataflow efficiency.

B. Performance Evaluation

Performance evaluation is conducted on the FPGA reference designs, considering inference latency, throughput, energy consumption, and classification accuracy. The results, summarized in Table III, present a fair comparison under identical parallelization configurations, highlighting the trade-offs between computational efficiency and predictive performance across the evaluated architectures.

1) Inference latency: Table III reports the inference latency for all evaluated architectures. The *MLQ* models achieve consistent and competitive performance, with *MLQ* 6, 8, and 11 showing the lowest expected latencies (34.92–35.46 ms per sample). These values outperform reference designs such as VGG10 (8-bit) and FINN-B, and remain well below the QCNN-(64F) model, whose larger depth results in a substantially higher latency of 113.22 ms.

Measured latencies on the ZC702 are systematically higher than Vivado predictions, with an average deviation of 20.6% across the *MLQ* models. This difference is mainly due to data-transfer overheads between the processing system and programmable logic, which are not fully captured by synthesis-level timing estimates.

To understand how latency accumulates, Figure 4 shows the per-layer latency breakdown for each architecture. Convolutional layers dominate the runtime, and the first convolutional layer (*Conv1*) contributes the largest share across all models. Because *Conv1* operates on the highest-resolution feature maps and appears early in the pipeline, it requires significantly more cycles than subsequent layers and therefore dictates the overall latency.

[2]https://www.kaggle.com/datasets/aleksandrdubrovin/deepsigio-radioml-201801a-new/data

[3]In FINN-based dataflow architectures, operator folding refers to the time-multiplexing of hardware operators to reduce resource usage. It is controlled by the degree of parallelism, typically defined by the number of PE and SIMD lanes in MVAU, trading off throughput for lower FPGA resource utilization.

Table II: FPGA resource utilization (MACs, LUTs, registers, BRAM, DSP) for MLQ architectures and reference models on the ZC702 platform.

MONAS-LQ / Device	BOPs	W_bits	QA-Score	Total MACs	MAC load [% of QCNN]	LUTs	LUTs [%]	Regs	Regs [%]	BRAM36	BRAM36 [%]	DSP	DSP [%]
ZC702 capacity	–	–	–	–	–	53 200	100.0	106 400	100.0	140	100.0	220	100.0
1	40581136	66490	0.051	2 998 272	23.3	18 041	33.9	35 577	33.4	20	14.3	6	2.7
2	66164096	46627	0.059	3 256 320	25.3	16 713	31.4	29 020	27.3	27	19.3	7	3.2
3	66317792	46664	0.059	3 256 320	25.3	15 550	29.2	27 333	25.7	28	20.0	7	3.2
4	59837976	86692	0.071	3 299 328	25.6	16 173	30.4	28 208	26.5	33	23.6	8	3.6
5	105712976	86626	0.100	3 299 328	25.6	14 211	26.7	24 441	23.0	32	22.9	8	3.6
6	111721136	120745	0.117	3 320 832	25.8	15 141	28.5	26 622	25.0	37	26.4	9	4.1
7	125730656	125294	0.128	3 341 312	26.0	17 268	32.5	30 602	28.8	58	41.4	11	5.0
8	134009616	134129	0.136	3 331 584	25.9	16 123	30.3	28 887	27.1	45	32.1	10	4.5
9	121027296	181744	0.147	3 341 312	26.0	16 729	31.4	30 019	28.2	64	45.7	11	5.0
10	141987312	156091	0.150	3 331 584	25.9	16 123	30.3	28 887	27.1	45	32.1	10	4.5
11	149238480	203563	0.174	3 353 600	26.1	16 173	30.4	28 208	26.5	33	23.6	8	3.6
QCNN-32F	65900672	171152	0.109	3 345 408	26.0	16 021	30.1	28 488	26.8	72	51.4	10	4.5
VGG10-32F/8b	213100928	564424	0.358	3 345 408	26.0	14 168	26.6	25 256	23.7	94	67.1	10	4.5
FINN B	99202048	262724	0.166	3 345 408	26.0	14 041	26.4	24 719	23.2	70	50.0	10	4.5
QCNN-64F	243566768	400588	0.311	12 864 512	100.0	20 349	38.2	39 940	37.5	107	76.4	10	4.5

Table III: End-to-end latency, energy per sample, and classification accuracy for all architectures under the same parallelization configuration (PE = 1, SIMD = 1).

Project		Latency ms/sample		Energy	FPGA	Server	$\lvert\Delta$Acc$\rvert$
		Expected	FPGA	mJ/sample	acc. %	acc. %	
MLQ	1	31.34	42.25	19.45	43.05	42.96	0.09
	2	34.25	42.29	19.55	47.89	47.90	0.01
	3	34.25	42.30	19.53	52.09	51.89	0.20
	4	34.70	42.25	19.52	53.29	53.26	0.03
	5	34.70	42.35	19.58	53.44	53.31	0.13
	6	34.92	40.96	18.95	54.58	54.40	0.18
	7	35.13	42.24	19.54	54.14	53.90	0.24
	8	35.03	40.98	18.96	54.34	54.33	0.01
	9	35.13	41.02	18.97	54.39	54.36	0.03
	10	35.25	41.15	19.03	53.81	53.76	0.05
	11	35.46	40.12	19.03	53.98	53.95	0.03
QCNN-32F		35.16	42.23	19.62	52.84	52.65	0.19
VGG10 32F/8-bit		35.16	41.88	19.60	54.24	54.19	0.05
FINN-B		35.16	41.99	19.77	52.43	52.34	0.09
QCNN-64F		131.98	113.22	52.72	54.40	54.15	0.25

These results confirm that end-to-end latency is driven by a small subset of cycle-intensive layers rather than by overall model size. Effective hardware optimization should therefore focus on parallelizing early convolutional stages, particularly *Conv1*, while avoiding unnecessary optimization of later layers with negligible latency impact. This behaviour is consistent with our previous sensitivity analysis in [7], which also identified *Conv1* as the dominant contributor to inference cost in quantized CNN architectures.

2) Energy Consumption and Accuracy Preservation: Energy measurements were obtained using the TI UCD9248 regulators on the ZC702 board, accessed via PMBus (I^2C). The total energy per inference was computed as $E = \sum_t P(t)\,\Delta t$, where $P(t)$ is the instantaneous power derived from the VCCINT, VCCAUX, and VCCBRAM rails, and Δt denotes the sampling period of the monitoring system.

The results show that the NAS-derived architectures are consistently more energy-efficient than the manually designed baselines. *MLQ* 6, 8, and 9 achieve the lowest energy consumption, with values of 18.95–18.97 mJ/sample, outperforming both VGG10 (19.60 mJ/sample) and FINN-B (19.77 mJ/sample). QCNN-(32F) exhibits a similar footprint (19.62 mJ/sample), while QCNN-(64F) incurs a significantly higher cost of 52.72 mJ/sample, reflecting the energy penalty

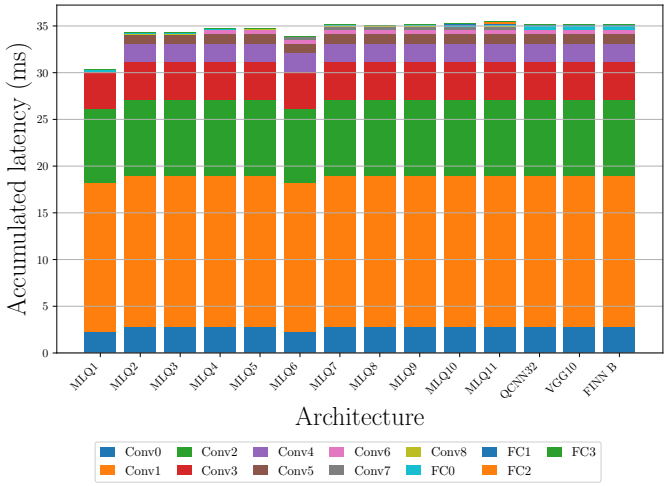

Figure 4: Per-layer latency breakdown for all architectures. Early convolutional layers, particularly Conv1, dominate overall latency and therefore represent the primary target for hardware-level parallelization.

associated with its larger computational load.

Across all architectures (Figure 5), the accuracy difference between server and FPGA execution remains below $\pm 0.30\%$, confirming that the low-precision hardware deployment does not compromise predictive performance. Several *MLQ* models even display marginal accuracy gains on the FPGA, including *MLQ* 7 (+0.18%), *MLQ* 8 (+0.24%), and *MLQ* 9 (+0.19%), while others such as *MLQ* 10 show nearly identical results across platforms.

C. Trade-Off Between Energy and Theoretical Complexity

Figure 5 compares the energy consumption, accuracy, and QA-Score of the *MLQ* architectures (01–11) against widely used reference models. Across all *MLQ* configurations, the energy per inference remains below 20 mJ/sample, showing only minor variation despite the increase in model capacity. This contrasts sharply with QCNN-(64F), which exceeds 50 mJ/sample, illustrating the strong dependence of energy consumption on excessive architectural complexity.

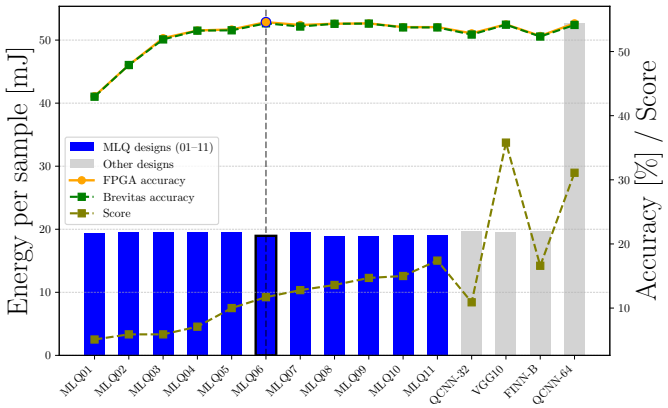

Figure 5: Energy consumption, FPGA and server accuracy, and QA-Score values for the MLQ architectures and reference models.

Accuracy remains stable across the *MLQ* designs, with both server-side and FPGA-based inference exhibiting differences within $\pm 0.30\%$. Several architectures even show small accuracy gains on the FPGA, which suggests that quantization-aware design and FINN's controlled parallelism can be effectively leveraged on reconfigurable hardware.

The QA-Score curve shows a steady improvement as model capacity grows, with *MLQ* 10 and *MLQ* 11 achieving the highest scores. Importantly, this increase is not accompanied by a proportional rise in energy consumption, indicating that additional representational power is achieved through targeted refinement of cycle-dominant layers rather than uniform scaling of the network. In contrast, VGG10 and FINN-B deliver lower or comparable QA-Score values at higher energy costs, reinforcing the efficiency gains enabled by the *MLQ* design space.

V. Conclusions and Future Work

This work evaluated a set of quantized architectures generated by the hardware-aware NAS framework MONAS-LQ and deployed on a Xilinx Zynq-7000 ZC720 FPGA. The study provided an empirical assessment of the accuracy–efficiency trade-offs offered by the Pareto-optimal architectures identified by MONAS-LQ (denoted as *MLQ*) under realistic edge hardware constraints.

The results show that several *MLQ* models achieve competitive accuracy while reducing energy consumption to below 20 mJ/sample, outperforming baselines such as VGG10 (8-bit), FINN-B, and QCNN-(32F). Latency measurements further reveal that end-to-end performance is dominated by a small number of cycle-intensive layers-most notably the first convolutional layer-rather than by overall model size. This finding challenges the common assumption that smaller networks are inherently faster and highlights the importance of layer-level hardware behavior when deploying DL models at the edge.

All *MLQ* architectures, when compiled into FINN dataflow accelerators, were synthesized using the minimal parallelization configuration (PE = 1, SIMD = 1), resulting in fully folded implementations. This setup ensured a fair comparison across models but also exposed how the execution time of early layers ultimately shapes the overall latency profile.

Future work will incorporate per-layer folding and parallelization decisions directly into the NAS process, enabling joint exploration of network architecture and hardware mapping. We also plan to extend the search space toward multitask and spectrum-sensing models to support more capable and energy-efficient edge-intelligent radio systems.

Acknowledgment

This research is funded by the imec.icon project RAPIDNESS, which is co-financed by imec and Flanders Innovation and Entrepreneurship under project nr HBC.2024.0772

References

[1] X. Lin, "Artificial Intelligence in 3GPP 5G-Advanced: A Survey," *arXiv preprint arXiv:2305.05092*, 2023.

[2] H. Zhang *et al.*, "Revolution of Wireless Signal Recognition for 6G: Recent Advances, Challenges and Future Directions," *IEEE Communications Surveys & Tutorials*, 2025.

[3] H. F. Shahid *et al.*, "Iot service orchestration in edge-cloud continuum with 6g: A review," *IEEE Internet of Things Journal*, pp. 1–1, 2026.

[4] Y. Wei *et al.*, "Convolutional neural networks on the edge: A comparison between fpga and gpu," in *2023 CSTIC*, 2023, pp. 1–3.

[5] A. Q. M. S. Sayyed *et al.*, "Resilient wireless communications with selective deep neural network classification," in *2025 IEEE MILCOM*, 2025, pp. 782–787.

[6] T. J. O'Shea *et al.*, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[7] D. Goez *et al.*, "A methodology to design quantized deep neural networks for automatic modulation recognition," *Algorithms*, vol. 15, no. 12, p. 441, 2022.

[8] M. Wang *et al.*, "An ultra lightweight neural network for automatic modulation classification in drone communications," *Scientific Reports*, vol. 14, no. 1, p. 21490, 2024.

[9] T. T. An *et al.*, "Efficient automatic modulation classification for next-generation wireless networks," *IEEE Transactions on Green Communications and Networking*, vol. 10, pp. 249–259, 2026.

[10] Y. Wang *et al.*, "Distributed learning for automatic modulation classification in edge devices," *IEEE Wireless Communications Letters*, vol. 9, no. 12, pp. 2177–2181, 2020.

[11] S. Peng *et al.*, "A survey of modulation classification using deep learning: Signal representation and data preprocessing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7020–7038, 2022.

[12] A. Maclellan *et al.*, "RFSoC modulation classification with streaming CNN: data set generation & quantized-aware training," *IEEE Open Journal of Circuits and Systems*, 2024.

[13] S. M. Sali *et al.*, "Real Time FPGA Based CNNs for Detection, Classification, and Tracking in Autonomous Systems: State of the Art Designs and Optimizations," *arXiv preprint arXiv:2509.04153*, 2025.

[14] P. Antunes and A. Podobas, "FPGA-Based Neural Network Accelerators for Space Applications: A Survey," *arXiv preprint arXiv:2504.16173*, 2025.

[15] F. Jentzsch *et al.*, "RadioML meets FINN: Enabling future RF applications with FPGA streaming architectures," *IEEE Micro*, vol. 42, no. 6, pp. 125–133, 2022.

[16] J. A. Rothe and H. Shajaiah, "Resource and Performance Improvements of Optimized Convolutional Neural Networks for FPGA Implementations of Automatic Modulation Recognition," in *2025 CISS*. IEEE, 2025, pp. 1–6.

[17] D. Góez *et al.*, "Monas-lq: A fast nas approach for layer-wise quantized cnns optimized for amc," —, 2026, submitted for publication.

[18] T. J. O'Shea *et al.*, "Over-the-Air Deep Learning Based Radio Signal Classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.