

Leveraging Posit Arithmetic for Deep Learning on FPGAs: Comparative Insights with IEEE Floating and Fixed-Points Formats

Kawthar DELLEL^{1,2}, Emanuel TRABES^{1,4}, Hassene FAIEDH^{2,3}, and Carlos VALDERRAMA¹

¹ Service d'électronique et de Microélectronique, University of Mons, Belgium

² Laboratory of Electronics and Microelectronics, University of Monastir, Tunisia

³ Higher Institute of Applied Sciences Technology, University of Sousse, Sousse, Tunisia

⁴ Department of Electronics, Universidad Nacional de San Luis, Argentina

Abstract. The posit number system has gained attention as a compelling alternative to traditional floating- and fixed-point arithmetic for deep learning applications, offering higher accuracy per bit and improved numerical behavior in low-precision regimes. While prior studies have explored its benefits for inference and hardware acceleration, limited work has addressed the integration of posit arithmetic into the training process itself. In this paper, we propose a Quantization-Aware Training (QAT) methodology tailored specifically for the posit format. Our approach captures the non-uniform precision and rounding dynamics of posit arithmetic during both forward and backward passes, allowing the model to learn within the constraints of deployment-time precision. Unlike conventional quantization methods that apply posit conversion post-training, our framework aligns optimization with deployment constraints from the outset, leading to improved robustness and accuracy at low bit widths. To evaluate the practicality of our approach, we explore FPGA-based deployment, demonstrating that our posit-QAT model is well-suited for efficient, low-precision inference on resource-constrained platforms.

Keywords: QAT · Posit Arithmetic · FPGA · CNNs.

1 Introduction

Convolutional Neural Networks (CNNs) have transformed artificial-intelligence research, particularly in computer-vision tasks, by hierarchically extracting spatial features and capturing complex patterns. This progress, however, comes at the cost of substantial computational and energy demands—limitations that become acute when deploying models on embedded or edge devices. Low-precision arithmetic offers an attractive trade-off, shrinking memory footprint and power consumption while incurring only modest accuracy loss.

Crucial to successful low-bitwidth learning is the choice of numerical representation. Conventional fixed-point and IEEE floating-point formats impose rigid

compromises between dynamic range and precision. The recently proposed *posit* number system, by contrast, provides tapered precision and a logarithmic value distribution, delivering higher accuracy near zero and a wider dynamic range for the same word length. These properties align well with typical deep-learning workloads, where most activations and gradients cluster around small magnitudes.

Empirical studies confirm that posits achieve lower numerical error per bit than either fixed- or floating-point formats, sustaining competitive accuracy at 8-bit and even sub-8-bit precision [2, 6]. Algorithmic techniques such as dynamic range-aware quantization, layer-wise adaptive exponent scaling, and continual learning-aware encoding further exploit these advantages, reducing memory bandwidth and mitigating quantization noise with negligible accuracy loss [6, 7]. Hardware investigations echo these findings: custom posit multiply-accumulate (MAC) units deliver up to $2\times$ better performance-per-area and energy efficiency, while block- and logarithmic-posit variants simplify logic and deepen pipelines in FPGA implementations [5, 8]. Posit-based CNNs even exhibit heightened resilience to faults and noise, thanks to tapered precision’s ability to preserve small values without overflow across wide dynamic ranges [2, 6].

Yet a critical gap remains: most workflows still rely on 32-bit floating point during training, creating a mismatch between training and low-bit inference that undermines posit’s potential. Existing posit-aware training attempts often fall back on heuristic approximations or high-bit posits, leaving end-to-end low-precision learning an open challenge [2, 6].

Contributions We close this gap with a unified quantization-aware training (QAT) framework that embeds exact posit arithmetic into both forward and backward passes. Our approach:

- preserves posit’s numerical benefits throughout optimization and deployment;
- maintains accuracy at aggressive bit-widths without resorting to float32 fallbacks;
- maps naturally onto FPGA-based accelerators equipped with custom posit MACs, demonstrating practical deployability.

Field-Programmable Gate Arrays provide an ideal proving ground: they enable rapid prototyping of posit datatypes while retaining favorable energy and area characteristics. Rather than proposing new hardware, we show that networks trained with our posit-QAT strategy drop directly into existing low-precision pipelines, validating the method’s real-world applicability.

By aligning training with the constraints—and advantages—of the posit format, this work extends posit arithmetic from inference to optimization, providing a missing cornerstone in the broader pursuit of efficient, precision-aware deep learning.

2 Methodology

2.1 Posit Number System

The *posit* format, introduced by Gustafson [1], replaces the rigid field layout of IEEE-754 with a self-delimiting structure that trades bits between range and precision on demand. A posit is written `posit(n , es)`, where n is the total word length and es bounds the exponent field. Each encoded value contains

- a sign bit s ;
- a variable-length *regime* field r , unary-coded to produce a scale factor k ;
- up to es exponent bits e ;
- a fraction (mantissa) field f .

The decoded real value is

$$x = (-1)^s \textit{useed}^k 2^e (1 + f), \quad \textit{useed} = 2^{2^{es}}.$$

Regime coding. A run of 1’s terminated by 0 yields $k > 0$; a run of 0’s terminated by 1 yields $k < 0$. Examples: `1110` → $k = +2$, `0001` → $k = -2$, `10` → $k = 0$.

Tapered precision. Because regime length $\ell(k)$ varies with magnitude, the remaining bit budget $n - 1 - es - \ell(k)$ available to the fraction shrinks for very large or very small values, concentrating precision near $|x| \approx 1$. This *tapered precision* supplies high relative accuracy where neural activations and gradients cluster, yet still offers wide dynamic range.

Special values. Posits eliminate signed zeros, $\pm\infty$, and NaNs; a single pattern (all ones) encodes *Not-a-Real* (NaR) for invalid operations. Arithmetic is therefore totally ordered and free of many IEEE corner cases.

These properties make the posit format attractive for low-precision deep learning, where every bit of dynamic range and accuracy counts.

Compared to IEEE-754 floating-point formats, posit arithmetic exhibits several favorable properties:

- **Improved accuracy:** For a given bit width, posits typically achieve lower absolute and relative error, particularly in low-precision regimes.
- **Increased dynamic range:** The use of variable-length regime encoding permits wider representable ranges without increasing the exponent size.
- **Simplified exceptional cases:** The absence of NaNs, infinities, and signed zeros streamlines hardware and software implementations.
- **Numerical regularity:** Posits offer symmetric behavior around zero and more predictable rounding characteristics, contributing to improved numerical stability.

These advantages position the posit format as a compelling alternative to both floating-point and fixed-point formats, particularly in resource-constrained environments where reduced bit widths and precision-efficiency tradeoffs are critical. Arithmetic is therefore totally ordered and free of many IEEE corner cases.

These properties make the posit format attractive for low-precision deep learning, where every bit of dynamic range and accuracy counts.

2.2 Quantization-Aware Training with Fake Posit Quantization

To exploit posits during learning, we adopt *quantization-aware training* (QAT) with a custom fake-quantizer that emulates $\text{posit}(n, es)$ arithmetic in the forward pass while keeping gradients in full precision.

Forward-pass emulation Given a real tensor x :

1. **Log decomposition.** Write $|x| = 2^k m$ with $k = \lfloor \log_2 |x| \rfloor$ and $m \in [1, 2)$.
2. **Regime/exponent split.** Decompose k as $k = q \cdot 2^{es} + r$, $r \in [0, 2^{es} - 1]$, where q is the regime value.
3. **Bit-budget allocation.** The remaining fraction length is $b_f = n - 1 - es - \ell(q)$. Truncate m to b_f bits.
4. **Reconstruction.** Combine the quantised pieces to obtain $\hat{x} = \text{sign}(x) \text{useed}^q 2^r \tilde{m}$.

Back-propagation Gradients are propagated with a straight-through estimator (STE): $\partial \hat{x} / \partial x \approx 1$. This keeps the computation graph intact and lets the network adapt its weights to the posit constraints during optimisation.

2.3 Quantised CNN Architecture

We evaluate the fake-posit QAT on a compact convolutional network fully quantised in both weights and activations:

- **Stages.** Three convolutional stages (3×3 kernels, channels 32–64–128) each comprise two quantised convolutions + ReLU, followed by 2×2 max-pooling.
- **Head.** Flattened features feed two fully connected layers (256, 128 units) and a final classifier.
- **Quantisation.** Every conv/linear op applies the fake-posit operator (default $\text{posit}(16, 1)$). Batch-norm layers remain in float32 for stability.

During training the forward path strictly observes posit precision, so reported accuracy reflects real deployment behaviour.

2.4 FPGA Implementation

To gauge hardware cost, each network layer is synthesised with Xilinx Vitis HLS for a Zynq-7000 device. Convolution modules use line-buffer shift registers for streaming 3×3 windows; max pooling employs pipelined comparators; fully connected layers implement streamed matrix-vector products. All blocks expose AXI-Stream interfaces and share a templated datatype, permitting compilation with either float, fixed-point, or posit. This modular flow yields per-layer resource, latency, and energy figures that quantify the savings obtained when models are trained with our posit-QAT pipeline.

3 Experimental Results

We benchmark the impact of numeric format on both *model accuracy* and *hardware cost*. All networks were trained on CIFAR-10 with the architecture described in Section ??, using identical hyper-parameters and **quantization-aware training (QAT)** for every format.

3.1 Classification Accuracy

Table 1. Test accuracy on CIFAR-10 after QAT.

Format	Float32	Posit16	Fixed16	Posit8	Fixed8	Posit6	Posit4
Accuracy (%)	87.16	87.61	87.37	87.08	86.97	85.90	10.00

Key observations.

- **Posit16** achieves the highest accuracy (87.61%), edging out full-precision Float32 despite using half as many bits.
- At 8bits, **Posit8** slightly outperforms Fixed8, confirming that tapered precision and wider dynamic range help preserve information under aggressive quantization.
- Extremely tight budgets (Posit4) collapse to chance-level accuracy, indicating that 4bits cannot support both range and gradient fidelity. Future work may explore mixed-precision or layer-adaptive bit-widths to circumvent this limit.

3.2 Hardware Cost on a Zynq-7000 FPGA

To quantify resource savings, each layer was synthesized with Xilinx Vitis HLS for a xc7z020 device. Results are reported *post place-and-route*.

Convolution. Posit8 cuts LUT and FF usage by more than 50% compared with Float32 and slashes DSP demand by 98%, while sustaining the same clock

Table 2. 3×3 Convolution layer (padding = 1, ReLU).

Format	LUTs	FFs	DSPs	BRAMs	SRLs	F _{max} (MHz)
Float32	11 854	14 185	156	12	424	103.2
Posit8	5 676	4 821	3	33	12	102.9
Fixed16	1 430	1 697	0	12	66	142.2

Table 3. Fully-connected (1×32 output, ReLU).

Format	LUTs	FFs	DSPs	BRAMs	SRLs	F _{max} (MHz)
Float32	498	730	5	1	7	118.0
Posit8	1 511	1 228	1	0	0	120.3
Fixed16	341	520	0	0	0	135.4

speed. Fixed16 is still the lightest, but at the cost of the accuracy drop seen in Table 1.

Dense layer. The variable-length regime in posits complicates wide matrix–vector products: Posit8 needs $\sim 4\times$ more LUTs and FFs than Fixed16 and still relies on a DSP. Optimized encode/decode pipelines or hybrid formats are promising routes to narrow this gap.

Table 4. 2×2 Max-pooling (stride = 2).

Format	LUTs	FFs	DSPs	BRAMs	SRLs	F _{max} (MHz)
Float32	376	443	2	0	25	132.2
Posit8	358	414	0	0	13	173.7
Fixed16	310	422	0	0	32	126.4

Pooling. Here Posit8 matches Fixed16 resources but attains the highest clock rate (173.7MHz), benefiting from simple compare–select logic and the small 8-bit datapath.

Posit arithmetic delivers the best trade-off when computation is *convolution-dominated*: it halves resources versus Float32 with no loss—and even a slight gain—in accuracy. Dense layers remain an open optimisation target; nonetheless, the overall results reinforce the suitability of posits for edge-scale CNNs where memory, power, and dynamic-range demands must be balanced carefully.

3.3 Hardware Implementation Results

In Table 2 the hardware metrics demonstrate that the Posit8 format offers a compelling trade-off between precision and resource efficiency. Compared to Float32, Posit8 achieves a drastic reduction in LUTs (over 50%) and flip-flops while nearly eliminating the need for DSP blocks—requiring only 3 compared to 156. While Fixed16 achieves the lowest resource usage, it lacks dynamic range and numerical fidelity, which can limit model accuracy.

Table 5. Post-Place-and-Route Hardware Metrics for 3×3 Convolution Layer (Padding=1, ReLU)

Format	LUTs	FFs	DSPs	BRAMs	SRLs	Freq (MHz)
Float32	11854	14185	156	12	424	103.2
Posit8	5676	4821	3	33	12	102.9
Fixed16	1430	1697	0	12	66	142.2

Table 6. Post-Place-and-Route Hardware Metrics for Linear Layer (1×32 Output, ReLU)

Format	LUTs	FFs	DSPs	BRAMs	SRLs	Freq (MHz)	Timing Met
Float32	498	730	5	1	7	118.0	
Posit8	1511	1228	1	0	0	120.3	
Fixed16	341	520	0	0	0	135.4	

In Table 3, while Posit8 shows clear hardware savings in convolution and pooling layers, its overhead in fully connected layers is significant. Our results show it requires over 4× more LUTs and FFs than Fixed16, and still depends on a DSP block. This stems from the variable-length regime and dynamic scaling logic that complicate efficient implementation of matrix-vector multiplications. Addressing this will require tailored hardware optimizations or hybrid encoding schemes to retain Posit’s precision benefits without incurring high area and power costs in dense layers.

Table 7. Post-Place-and-Route Hardware Metrics for MaxPooling2D Layer (2×2 Kernel, Stride 2)

Format	LUTs	FFs	DSPs	BRAMs	SRLs	Freq (MHz)
Float32	376	443	2	0	25	132.2
Posit8	358	414	0	0	13	173.7
Fixed16	310	422	0	0	32	126.4

In Table 4 The Posit8 format demonstrates a strong balance between efficiency and performance. It eliminates DSP usage like Fixed16, but achieves a significantly higher operating frequency (173.7MHz vs. 126.4MHz), enabling faster computation. Additionally, Posit8 uses only 8 bits, offering reduced memory footprint and potential power savings compared to 16-bit formats.

4 Discussion

Our experiments confirm that *posit* arithmetic can deliver competitive—and in some cases superior—accuracy at substantially lower bit-widths than IEEEFloat32 or conventional fixed-point formats. Two findings stand out:

1. **Accuracy under QAT.** Posit16 tops the leaderboard on CIFAR-10 (Table 1), edging out 32-bit float with only half the storage. At 8bits, Posit8 matches Float32 and exceeds Fixed8, illustrating how tapered precision and a wider dynamic range preserve information that uniform fixed-point quantizers lose.
2. **Hardware savings in convolution-dominated layers.** On the Zynq-7000 FPGA, Posit8 cuts LUT and DSP usage by more than 50% in 3×3 convolutions (Table 2) and attains the highest clock rate in max-pooling (Table 7). These wins make the format attractive for latency-constrained, edge-scale CNNs.

Limitations. The same advantages do not yet extend to dense layers: the variable-length regime encoder/decoder inflates area in fully connected blocks, where simple, regular fixed-point arithmetic excels (Table 6). In addition, batch-normalisation layers remain in full precision, partially masking the pure low-bit behaviour. Finally, 4-bit posits collapse to chance-level accuracy, indicating that an absolute floor on precision exists without mixed-precision or layer-adaptive bit-widths.

Opportunities for optimisation.

- *Hardware.* Streamlined regime encoders, approximate MACs, or hybrid posit/fixed pipelines could bring the cost of dense layers in line with fixed-point while retaining posit accuracy.
- *Software.* Quantised batch normalisation and normalisation-free architectures would expose the raw effect of the numeric format and may reduce reliance on float32 fall-backs.
- *System-level.* Layer-wise or channel-wise bit-width scheduling, determined during training, could push overall precision below the 8-bit envelope without the failure seen in Posit4.

These observations underscore the importance of *hardware–software co-design*: training must be aware of the numeric format, and hardware must be tailored to its idiosyncrasies.

5 Conclusion

We have presented a unified posit quantization-aware training pipeline and evaluated it on both accuracy and real FPGA cost. The study shows that:

- **Posit16** surpasses 32-bit float accuracy while halving the word length.
- **Posit8** preserves accuracy comparable to Float32 but reduces convolutional resource use by more than 50% and virtually eliminates DSP dependence.

These results position the posit number system as a viable path toward low-power, high-throughput inference on resource-constrained hardware. Remaining challenges lie chiefly in fully connected layers and end-to-end low-bit

pipelines. Future work will therefore (i) co-optimize posit arithmetic units for dense workloads, (ii) integrate quantised normalisation and mixed-precision scheduling, and (iii) extend evaluation to larger models and datasets. Through such co-design, we expect posits to become a key enabler of next-generation efficient AI hardware.

References

1. Gustafson and Yonemoto, “Beating Floating Point at its Own Game: Posit Arithmetic,” *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, pp. 71–86, juin 2017, doi: 10.14529/jsfi170206.
2. J. Lu et al., “Training Deep Neural Networks Using Posit Number System,” in 2019 32nd IEEE International System-on-Chip Conference (SOCC), Sep. 2019, pp. 62–67. doi: 10.1109/SOCC46988.2019.1570558530.
3. S. D. Ciocirlan, D. Loghin, L. Ramapantulu, N. Țăpuș, and Y. M. Teo, “The Accuracy and Efficiency of Posit Arithmetic,” in 2021 IEEE 39th International Conference on Computer Design (ICCD), Oct. 2021, pp. 83–87. doi: 10.1109/ICCD53106.2021.00024.
4. O. Desrentes, D. Resmerita, and B. Dupont de Dinechin, “A Posit8 Decompression Operator for Deep Neural Network Inference,” in *Next Generation Arithmetic*, J. Gustafson and V. Dimitrov, Eds., Cham: Springer International Publishing, 2022, pp. 14–30. doi: 10.1007/978-3-031-09779-9-2.
5. S.-F. Hsiao, S.-C. Lin, G.-L. Chen, S.-H. Yang, Y.-C. Yuan, and K.-C. Chen, “Design of an Efficient Deep Neural Network Accelerator Based on Block Posit Number Representation,” in 2024 International VLSI Symposium on Technology, Systems and Applications (VLSI TSA), Apr. 2024, pp. 1–4. doi: 10.1109/VL-SITSA60681.2024.10546390.
6. H. F. Langroudi et al., “Alps: Adaptive Quantization of Deep Neural Networks with Generalized PositS,” in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Jun. 2021, pp. 3094–3103. doi: 10.1109/CVPRW53098.2021.00346.
7. V. Karia, A. Ziyarah, and D. Kudithipudi, “PositCL: Compact Continual Learning with Posit Aware Quantization,” in *Proceedings of the Great Lakes Symposium on VLSI 2024*, in GLSVLSI ’24. New York, NY, USA: Association for Computing Machinery, juin 2024, pp. 645–650. doi: 10.1145/3649476.3660371.
8. A. Ramachandran, Z. Wan, G. Jeong, J. Gustafson, and T. Krishna, “Algorithm-Hardware Co-Design of Distribution-Aware Logarithmic-Posit Encodings for Efficient DNN Inference,” in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, in DAC ’24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6. doi: 10.1145/3649329.3656544.