Integer Quantization of Graph Neural Networks for Real-Time FPGA Track Finding

Andrea Cardini, Pelayo Leguina, Elena Aller Gutierrez, and Santiago Folgueras
Universidad de Oviedo, Spain

***Abstract*—Real-time track finding for displaced-muon signatures in the CMS Level-1 trigger must operate under strict fixed-latency constraints of 12.5 μs while processing high-throughput detector data. Because muon hits map naturally onto sparse, irregular graphs, graph neural networks (GNNs) are attractive candidates; however, mapping message-passing models to field-programmable gate arrays (FPGAs) requires careful co-design of numerical precision, microarchitecture, and high-level synthesis (HLS) implementation. This work presents a reproducible, bit-exact workflow bridging GNN design and FPGA prototyping for fixed-latency inference. The methodology is demonstrated by implementing a two-layer GraphSAGE network onto an XCVU13P FPGA, using the Cora citation network as a fixed-size benchmark for firmware evaluation that decouples deployment feasibility from the physics task. Starting from a 32-bit floating-point reference that exceeds the available FPGA resource budget, we derive an integer-only datapath through post-training quantization (INT8 weights and activations, INT32 biases), a power-of-two scale approximation that replaces rescaling multipliers with arithmetic shifts, and data-driven bit-width narrowing. Every stage is validated bit-exactly against a Python integer emulator in Vitis HLS C-simulation. The optimized INT8 power-of-two design achieves an inference latency of 19 clock cycles (52.8 ns at the nominal 360 MHz clock) at 20% DSP, 6% FF, and 27% LUT utilization, with 75.0$\pm$1.1% accuracy compared to the 78.0$\pm$0.8% for FP32. Accuracy is reported as the average across multiple training seeds. The resulting workflow establishes a concrete, transferable path toward fixed-latency GNN-based track reconstruction in the CMS Level-1 trigger.**

***Index Terms*—FPGA, GNN, HLS, LHC, quantization, real-time trigger.**

I. Introduction

The High-Luminosity phase of the Large Hadron Collider (HL-LHC) at CERN [1] is planned to increase the instantaneous luminosity delivered to the experiments. This is achieved primarily through higher beam intensity and focusing, resulting in more proton-proton interactions per bunch crossing, referred to as pileup. At the Compact Muon Solenoid (CMS) experiment [2], the average number of proton-proton interactions per bunch crossing, i.e., every 25 ns, is expected to increase from about 60 to 140–200. To select potentially interesting events while operating in this high-pileup environment, the hardware-level trigger system, also known as the Level-1 (L1) trigger, is undergoing its Phase-2 upgrade [3]. The upgraded on-detector and back-end electronics based on field-programmable gate arrays (FPGA) will deliver higher-granularity detector primitives at greater bandwidth to trigger boards with substantially expanded on-chip compute resources, enabling the real-time execution of data-driven pattern-recognition algorithms within the fixed 12.5 μs L1 latency budget [3]. Such algorithms could include machine-learning inference for the identification of compatible hits from muons traversing the detector, namely a track-finding algorithm. Combining detector hits to reconstruct the trajectory of a muon naturally maps onto a graph, where detector hits and their geometrical compatibility are represented as nodes and edges, respectively. A graph neural network (GNN) was therefore chosen as the architecture for the displaced-muon reconstruction algorithm. For successful deployment, the GNN must operate within a 12.5 μs latency budget and fit within the available FPGA resource constraints. This manuscript presents a systematic evaluation of integer quantization, numerical representation, and high-level synthesis (HLS) implementation choices for a GNN-based inference within FPGA latency and resource constraints. The study is performed on a fixed-size firmware-proxy benchmark, decoupling deployment feasibility considerations from the physics application in the CMS L1 trigger.

This work is supported by ERC StG–101115353: INTREPID, granted to Prof. Santiago Folgueras. Funded by the European Union.

II. Firmware Requirements for the Phase-2 Level-1 Trigger

In the context of the HL-LHC, the L1 trigger system of the CMS experiment would have to process events at a rate of 40 MHz, producing an output rate of 750 kHz. Such events would then be sent to the software-level trigger for further processing and filtering. The maximum time available for the L1 decision is 12.5 μs, set by the on-detector buffer capacity before data are either discarded or transferred to the CPU farm.

This work focuses on the deployment of a track-finding algorithm in the muon system, and in particular in the overlap region between the barrel and endcap detectors. This region, extending approximately in the pseudorapidity range $0.8 < |\eta| < 1.24$, is characterized by the presence of three distinct detector technologies [4]: drift tube (DT) chambers, cathode strip chambers (CSCs), and resistive plate chambers (RPCs). Currently, muon tracks in this region are reconstructed by the Overlap Muon Track Finder (OMTF) algorithm [5], [6], which combines information provided by the different detectors, such as position and bending direction.

For the Phase-2 upgrade, the OMTF will run on a custom ATCA (Advanced Telecommunications Computing Architecture) board housing Xilinx Virtex UltraScale+ FPGA devices with high-speed optical transceivers. The study presented in

this manuscript targets an XCVU13P FPGA from the Xilinx Virtex UltraScale+ family, which provides 12,288 digital signal processing (DSP) slices, 1,728,000 look-up tables (LUTs), 3,456,000 flip-flops (FFs), 360 Mb of on-chip UltraRAM memory, and 128 GTY transceivers capable of operating at up to 32.75 Gb/s. This device is a larger member of the same FPGA family than the XCVU9P device quoted in Ref. [3] as the design choice for the OMTF, which provides 6,840 DSP slices, 1,182,000 LUTs, and 2,364,000 FFs.

III. Graph Network Architecture

Initial studies [7], [8] suggested that architectures based on graph sample-and-aggregate operations (GraphSAGE) [9] or more general message passing neural networks (MPNNs) could provide good performance for the target use case, consistent with prior demonstrations of GNN inference within sub-microsecond FPGA latency budgets for real-time particle reconstruction [10]. The GNN deployed in this study is a deliberately simplified two-layer GraphSAGE, chosen to establish a firmware deployment methodology, rather than to optimize physics performance.

Fig. 1 shows a schematic representation of the network constructed for firmware deployment studies: a two-layer GraphSAGE model [9] with mean neighbour aggregation, a linear input-feature projection, and statically bounded loops that permit full unrolling and deterministic pipelining in HLS [10], [11].

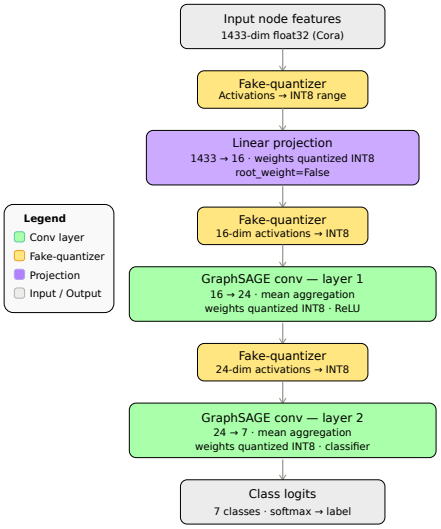

Fig. 1. Schematic representation of the tested GraphSAGE-based network.

Relative to the original GraphSAGE formulation [9], the root-node self-term is omitted: each node embedding is updated solely from its aggregated neighbourhood, discarding the concatenation with the root representation. This choice eliminates a separate root weight matrix from the HLS datapath and reduces the number of multiply-accumulate (MAC) operations per node per layer, simplifying the fixed-size loop structure required for complete array unrolling [11]. The self-contribution can be reinstated as an additional integer accumulation at the output of the aggregation stage, making it a straightforward extension when deploying on the physics dataset.

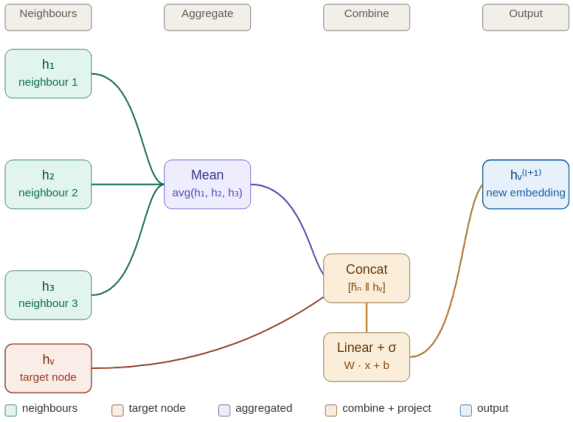

Fig. 2. Schematic representation of one GraphSAGE layer.

Fig. 2 shows a schematic representation of a GraphSAGE layer, where each node embedding is updated by aggregating information from its local neighbors.

The input data for the target GNN are represented as graphs. Detector stubs for each detection layer (DT, CSC, RPC) [2] are assigned to nodes using their coordinates, including radius r, pseudorapidity η, and azimuthal angle φ, together with the bending angle of the stub when it is recorded by a DT chamber. A maximum number of 8 detector layers (nodes) is considered per graph. Edges are constructed between stubs in detector regions that are geometrically compatible, meaning that a particle trajectory could plausibly cross both detector elements. A graphical representation of geometrically compatible hits forming a graph is presented in Fig. 3.

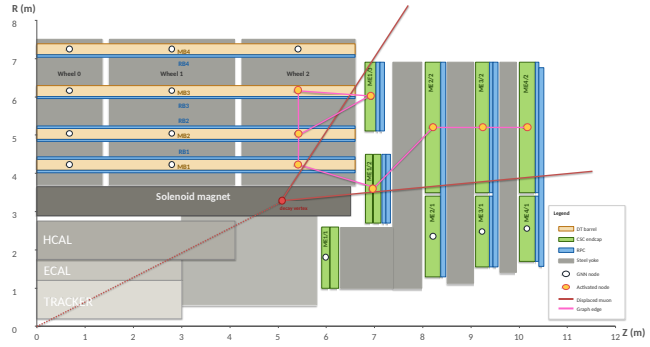

Fig. 3. Schematic representation of the CMS muon system being traversed by a long-lived particle decaying into a pair of muons. The red lines represent the displaced muons, while the yellow circles represent activated detector hits that form a graph for the track-finding GNN.

The objective of the present work is not to evaluate the physics performance but to establish a methodology for firmware deployment. Therefore, the resource and latency requirements for the deployment of this graph network are evaluated on a benchmark dataset, as described in the next Section.

IV. Dataset and Data Preparation

The Cora citation network [12] is used in this work as a firmware-proxy benchmark to develop and validate the quantization and FPGA deployment methodology. Cora consists of 2,708 scientific publications represented as nodes and 10,556 citation edges, with an average node degree of approximately 3.9. Each node is represented by a 1,433-dimensional binary bag-of-words feature vector and is labelled with one of seven research-topic classes. The standard Planetoid data splits [13] are adopted: 20 nodes per class, corresponding to 140 nodes in total, for training; 500 nodes for validation; and 1,000 nodes for testing. Before training, node features are row-normalized so that each feature vector sums to unity.

The full 2,708-node graph is not suitable for the fixed-size, fully unrolled HLS architecture considered in this deployment study. A compact fixed-size subgraph is therefore extracted for hardware-level validation: a 2-hop neighborhood around a representative training node is computed and truncated to N_{dep} nodes. The synthesised design uses $N_{\mathrm{dep}} = 8$, matching the fixed eight-node graphs of the target muon application (Section III). The adjacency matrix of the extracted subgraph is row-normalized without self-loops,

$$\tilde{A}_{ij} = \frac{1}{\deg(i)}, \quad (i,j) \in \mathcal{E}, \tag{1}$$

where $\deg(i) = \sum_{j:(i,j)\in\mathcal{E}} 1$ is the number of incoming neighbours of node i, isolated nodes ($\deg(i) = 0$) take $\tilde{A}_{ij} = 0$, and $\mathcal{E}$ denotes the set of directed edges in the extracted subgraph, consistent with the mean-aggregation operation used in the deployed GraphSAGE layer.

The 1,433-dimensional Cora features would require approximately $1433 \times F_h$ MAC operations per node per layer, with F_h being the number of hidden features per node per layer. This is prohibitive for the fixed-latency inference architecture targeted here. A trainable linear projection layer reduces the input dimension to F_{proj} before the first graph convolution. This is precomputed off-chip and omitted from the model under study; its latency is therefore not included in the reported figures and must be absorbed upstream within the same fixed-latency pipeline. The FPGA-targeted model therefore operates with architecture 16→24→7, corresponding to projected input features ($F_{\mathrm{proj}} = 16$), hidden features, and output classes, respectively. The resulting model contains approximately 1.1 k weights, making it suitable for complete array unrolling in HLS.

V. Quantization Methods

Deploying floating-point GNNs on FPGAs is resource-prohibitive for the fully unrolled architecture considered in this study. As shown in Table I, a 32-bit floating-point (FP32) implementation of the 16→24→7 model reaches 283% of the available XCVU13P DSP budget, 123% of the available LUT budget, and 128% of the available FF budget. A sequence of progressively optimized quantization schemes is therefore developed, each trading a bounded accuracy loss for substantial reductions in latency and resource usage.

A. Float32 and Fixed-Point Baselines

A floating-point reference is obtained by converting the trained 64-bit PyTorch model to 32-bit floating point and synthesizing it with Vitis HLS using native `float` types. The resulting implementation is not deployable on a single XCVU13P device because the required DSP, LUT, and FF resources exceed the available budget.

A first quantized variant is obtained with the Xilinx `ap_fixed<8,8>` arithmetic for activations and weights while retaining floating-point scale factors during dequantization. This is denoted with *Fixed-PTQ* for "fixed point - post-training quantization" (PTQ).

B. Integer-Only Post-Training Quantization

A fully integer arithmetic pipeline is obtained by replacing all floating-point operations with integer MACs and fixed-point rescaling [14]. The trained floating-point model is calibrated on a representative subgraph to derive per-tensor symmetric quantization scales:

$$s = \frac{\max|x|}{127}, \quad q = \mathrm{clip}\Big(\Big\lfloor \frac{x}{s} \Big\rceil, -128, 127\Big), \tag{2}$$

where $\lfloor\cdot\rceil$ denotes rounding to the nearest integer, x represents the floating-point tensor (weights or activations) being quantized, and q the resulting 8-bit integer (INT8) weight matrices and INT8 activations.

Calibration protocol: The quantization scales are derived from a single forward pass over a fixed representative subgraph comprising the full 2-hop neighbourhood (a calibration subgraph of $N_{\mathrm{cal}} = 32$ nodes) extracted around a training node. The scales are computed using symmetric per-tensor min/max observers, with the quantization grid aligned to zero (zero-point $= 0$). The calibration procedure evaluates all nodes in the extracted subgraph without resampling; as the subgraph is deterministic, derived scales are reproducible across runs. Quantization is applied symmetrically per tensor (not per channel), meaning a single scale applies to all elements of a weight matrix or activation map.

The row-normalized adjacency matrix is scaled to fixed-point integers as

$$A_{ij}^{K} = \Big\lfloor \tilde{A}_{ij} \cdot K \Big\rceil, \quad K = 2^{K_b}, \quad K_b = 12, \tag{3}$$

enabling full integer aggregation. Biases (b_o) are converted to the accumulator domain:

$$b_o^{\mathrm{int}} = \left\lfloor \frac{b_o}{s_{\mathrm{hid}} \cdot s_w} \right\rceil, \tag{4}$$

where s_{hid} is the aggregation-output (hidden) activation scale consumed by the linear layer and s_w is the weight scale, so that b_o^{int} lies in the same domain as the integer products $\hat{h}\, w$ accumulated in (7).

The complete integer forward pass for each layer $l \in \{1, 2\}$ of the deployed GraphSAGE model is:

$$T_{i,f}^{(l)} = \sum_j A_{ij}^K \cdot h_{j,f}^{(l-1)}, \tag{5}$$

$$\hat{h}_{i,f}^{(l)} = \mathrm{sat}_8\left(\left\lfloor \frac{T_{i,f}^{(l)} \cdot \beta_{\mathrm{fp}}^{(l)} + 2^{M-1}}{2^M} \right\rfloor\right), \tag{6}$$

$$a_{n,o}^{(l)} = b_o^{(l)} + \sum_f \hat{h}_{n,f}^{(l)} \cdot w_{o,f}^{(l)}, \tag{7}$$

$$h_{n,o}^{(l)} = \mathrm{sat}_8\left(\rho_l\left[\left\lfloor \frac{a_{n,o}^{(l)} \cdot \gamma_{\mathrm{fp}}^{(l)} + 2^{M-1}}{2^M} \right\rfloor\right]\right), \tag{8}$$

The integer aggregation accumulator $T_{i,f}^{(l)}$ (node i, feature channel f, layer l) accumulates the INT8 features $h_{j,f}^{(l-1)}$ of all its graph-neighbours j, weighted by the fixed-point adjacency entries A_{ij}^K. Because A_{ij}^K are integers scaled by $K = 2^{12}$, the partial sums $T_{i,f}^{(l)}$ span a range up to $d_{\max} \cdot K \cdot 127$ and must be held in a wider accumulator before rescaling.

Equation (6) requantises T to the INT8 range, yielding the aggregated feature $\hat{h}_{i,f}^{(l)}$. The product $T \cdot \beta_{\mathrm{fp}}^{(l)}$ is a fixed-point multiply: it simultaneously undoes the K-scaling of the adjacency and converts between the input quantization scale $s_{\mathrm{in}}^{(l)}$ and the aggregation-output scale $s_{\mathrm{hid}}^{(l)}$. Adding 2^{M-1} before the right-shift by M implements round-to-nearest, while the outer sat_8 clamps the result to the signed INT8 range $[-128, 127]$. With $M = 24$ fractional bits, the largest scale multiplier (the layer-1 linear multiplier $\gamma_{\mathrm{fp}}^{(1)} \approx 1.8 \times 10^5$) occupies 18 bits, fitting the 18-bit operand port of a DSP48E2 slice (a Xilinx DSP slice optimized for multiply-accumulate operations); the aggregation multipliers $\beta_{\mathrm{fp}}^{(l)}$ are narrower (≤ 13 bits).

Equation (7) is the standard matrix–vector step: the pre-scaled bias integer $b_o^{(l)}$ is added to the dot product of the rescaled aggregation output $\hat{h}^{(l)}$ and the quantized weight row $w_{o,\cdot}^{(l)}$, whose entry $w_{o,f}^{(l)}$ is the INT8 weight connecting input channel f to output channel o. Both operands are INT8, so the accumulator $a_{n,o}^{(l)}$, where n denotes the node index (equivalent to i in the aggregation step), is INT32, large enough to hold up to $F = 24$ signed products without overflow,

Equation (8) brings the INT32 accumulator back to INT8 by the same round-then-shift mechanism, now using $\gamma_{\mathrm{fp}}^{(l)}$. The layer activation function ρ_l is a rectified linear unit (ReLU) for hidden layers and the identity for the output layer; because ReLU only zeroes negative values, it can be applied entirely in the integer domain before saturation, with no floating-point step.

$$\beta_{\mathrm{fp}}^{(l)} = \left\lfloor \frac{s_{\mathrm{in}}^{(l)}}{K \cdot s_{\mathrm{hid}}^{(l)}} \cdot 2^M \right\rfloor, \tag{9}$$

$$\gamma_{\mathrm{fp}}^{(l)} = \left\lfloor \frac{s_{\mathrm{hid}}^{(l)} \cdot s_w^{(l)}}{s_{\mathrm{out}}^{(l)}} \cdot 2^M \right\rfloor. \tag{10}$$

Here $s_{\mathrm{in}}^{(l)}$ denotes the scale of the activations entering the aggregation of layer l (the projected-input scale for $l = 1$ and the hidden scale s_{hid} for $l = 2$), while $s_{\mathrm{hid}}^{(l)}$ and $s_{\mathrm{out}}^{(l)}$ are the aggregation-output and linear-output scales, respectively. Equations (6) and (8) each require one wide-integer multiplication by a fixed-point scale, which maps to DSP48 slices in the synthesized design.

C. INT8 with Power-of-Two Scaling

The four scale multiplications in (6)–(8) consume a significant fraction of the available DSP budget. Constraining the scales to powers of two (PO2) eliminates these multiplications, replacing them with compile-time arithmetic right-shifts. In this work, each scale is approximated by its nearest power of two:

$$\beta \;\approx\; \tilde{\beta} = 2^{-S_\beta}, \quad S_\beta = \lfloor -\log_2 \beta \rceil, \tag{11}$$

Choosing the nearest power of two bounds the resulting multiplicative scaling error by a factor of $\sqrt{2}$. Under this constraint, (6) and (8) reduce to shift-and-round operations:

$$\hat{h}_{i,f}^{(l)} = \mathrm{sat}_8\left(R_{S_\beta^{(l)}}\left(T_{i,f}^{(l)}\right)\right), \tag{12}$$

$$h_{n,o}^{(l)} = \mathrm{sat}_8\left(\rho_l\left[R_{S_\gamma^{(l)}}\left(a_{n,o}^{(l)}\right)\right]\right), \tag{13}$$

where $R_S(x)$ denotes round-to-nearest division by 2^S, implemented on the signed accumulator as a single rounding addition followed by an arithmetic right-shift:

$$R_S(x) = \left\lfloor \frac{x + 2^{S-1}}{2^S} \right\rfloor = \left(x + 2^{S-1}\right) \gg S, \tag{14}$$

where $\gg$ is an arithmetic right-shift. This is the same rounding rule used in (6) and (8), and rounds half-integers toward $+\infty$. No multipliers are required for rescaling in this configuration.

For the two-layer 16→24→7 model, the four shift amounts are derived analytically from the PTQ calibration scales:

$$S_{\beta^{(1)}} = \left\lfloor -\log_2\left(\tfrac{s_{\mathrm{in}}}{K \cdot s_{\mathrm{hid}}}\right) \right\rceil, \tag{15}$$

$$S_{\beta^{(2)}} = \left\lfloor -\log_2\left(\tfrac{1}{K}\right) \right\rceil = K_b = 12 \quad \text{(exact)}, \tag{16}$$

$$S_{\gamma^{(1)}} = \lfloor -\log_2 s_{w_1} \rceil, \tag{17}$$

$$S_{\gamma^{(2)}} = \left\lfloor -\log_2\left(\tfrac{s_{\mathrm{hid}} \cdot s_{w_2}}{s_{\mathrm{out}}}\right) \right\rceil. \tag{18}$$

Notably, $S_{\beta^{(2)}} = K_b = 12$ is exact because $\beta^{(2)} = s_{\mathrm{hid}}/(K \cdot s_{\mathrm{hid}}) = 1/K = 2^{-12}$ is already a power of two. The layer-1 linear scale reduces to $\gamma_1 = s_{w_1}$, as in (17), because the layer-1 aggregation output and the layer-1 linear output share the hidden scale ($s_{\mathrm{hid}}^{(1)} = s_{\mathrm{out}}^{(1)}$). For the reference model, the four shift constants are $S_{\beta^{(1)}} = 17$, $S_{\beta^{(2)}} = 12$, $S_{\gamma^{(1)}} = 7$, and $S_{\gamma^{(2)}} = 8$.

A schematic representation of the INT8-PO2 pipeline is shown in Fig. 4. In the synthesized Vitis HLS kernel,

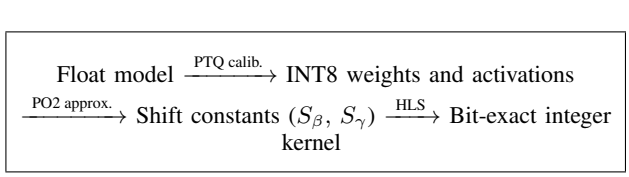

Fig. 4. INT8-PO2 quantization and deployment pipeline. PTQ calibration on a representative subgraph yields per-tensor scales, which are approximated by powers of two. The resulting shift constants are embedded as compile-time macros in the Vitis HLS kernel, replacing the rescaling multiplications with arithmetic right shifts. Bit-exact agreement between the Python integer emulator and the HLS C-simulation is verified before synthesis.

shift amounts are encoded as preprocessor macros, such as `BETA1_SHIFT`, `BETA2_SHIFT`, `EFF_SCALE1_SHIFT`, and `EFF_SCALE2_SHIFT`, allowing the tool to resolve them into wiring and shift logic during elaboration.

D. Data-Driven Bit-Width Optimization

A 32-bit accumulator is sufficient to prevent overflow in any INT8 MAC chain considered here, but it wastes routing and flip-flop resources. A data-driven profiling pass runs an integer emulation of the full forward pass and records the maximum absolute value m of each intermediate signal: linear accumulator m_a, aggregation temporary m_T, and adjacency entry m_A. The minimal sufficient bit-width for a signed integer type is then:

$$B = \lceil \log_2(m+1) \rceil + 1 + \delta, \tag{19}$$

where $\delta = 2$ is a safety margin and the extra $+1$ accounts for the sign bit.

The theoretical worst-case bounds,

$$m_T^{\text{th}} = d_{\max} \cdot A_{\max}^K \cdot 127, \quad m_a^{\text{th}} = m_b + F \cdot 127 \cdot w_{\max}, \tag{20}$$

where $d_{\max}$ is the maximum node degree in the subgraph, $A_{\max}^K \le K = 2^{12}$ is the largest fixed-point adjacency entry, $w_{\max}$ and m_b are the largest absolute INT8 weight and integer bias $|b_o^{\text{int}}|$ respectively, 127 is the maximum INT8 magnitude, and F is the layer fan-in ($F^{(1)} = 16$, $F^{(2)} = 24$; Eq. (20) uses the worst case $F = 24$). The adjacency magnitude requires no statistical bound: $A_{\max}^K \le K = 2^{12}$ follows directly from (3), so m_A is fixed deterministically by applying (19) with $m_A = A_{\max}^K$ rather than by profiling.

Applied together with PO2 shift optimization, data-driven bit-width narrowing yields the *INT8-PO2 Opt.* configuration, represented in Fig. 5.

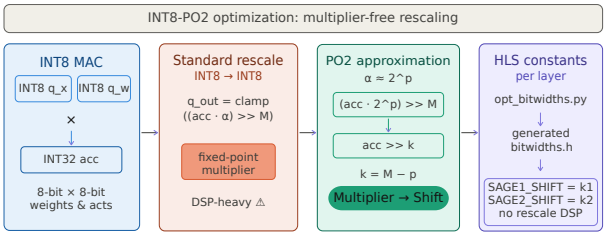

Fig. 5. Schematic representation of the optimized integer quantization with power of two scaling: *INT8-PO2 Opt.*

E. Quantization-Aware Training

Quantization-aware training (QAT) inserts differentiable fake-quantization operators around weight and activation tensors during the forward pass, allowing the model to adapt its parameters to the quantization grid under gradient descent. The Brevitas library [15] provides symmetric per-tensor INT8 fake quantizers with straight-through estimators and moving-average min/max observers. The QAT model is initialized from the pre-trained floating-point weights and fine-tuned for 200 epochs with quantization applied at INT8 precision to both weights and activations.

VI. Quantization and HLS Synthesis Results

The GraphSAGE model was tested using a custom graph-network implementation derived from the hls4ml package [11]. Tests were performed on the Cora dataset described in Section IV, using the quantization methods defined in Section V.

The model was evaluated under different quantization configurations in order to compare their impact on firmware-oriented performance. The metrics compared are:

- latency, reported both in clock cycles and in nanoseconds at the nominal 360 MHz target clock frequency;
- FPGA resources required to deploy the model, namely DSP slices, FFs, and LUTs;
- estimated maximum clock frequency (Fmax) reported by HLS synthesis for an initiation interval (II) of 1;
- accuracy, estimated as the fraction of correctly classified nodes in the 1,000-node Cora test set.

All accuracy measurements are reported as the mean $\pm$ one standard deviation (std) over 5 random training seeds, with seed values from 42 to 46. Each seed uses the same training/validation/test split from the Planetoid dataset (140 training nodes, 500 validation nodes, 1000 test nodes). The standard deviation reflects variance in the stochastic training process, including weight initialization and optimizer dynamics. All quantization schemes use calibration on the same fixed representative subgraph ($N_{\text{cal}} = 32$), so calibration variance is negligible. The standard deviation value of approximately 1.0 percentage points (pp) observed across configurations reflects the inherent stochasticity of training on the relatively small Cora training set (140 nodes).

Table I lists the results of the quantization techniques tested. Each column corresponds to a different numerical implementation. The first one, labelled *Float32*, was the initial attempt at deploying the two-layer GraphSAGE model onto the FPGA proxy after converting the 64-bit weights to 32-bit floating point. As shown by the DSP, LUT, and FF utilization,

TABLE I
Integer Quantization and HLS Synthesis Study

Metric	Float32	Fixed-PTQ	INT8-PO2	QAT	INT8-PO2 Opt.*
Precision	FP32	Fixed(8|8)	INT8	INT8	INT8
Accuracy (%)	78.0±0.8	75.7±1.0	75.0±1.2	76.8±0.9	75.0±1.1
Latency (cycles)	603	77	28	55	19
Latency (ns) at 360 MHz	1675.0	213.9	77.8	152.8	52.8
Fmax (MHz)	—	397	450	493	502
DSP slices	34,880 (283%)†	6,976 (56%)	5,320 (43%)	6,760 (55%)	2,560 (20%)
FFs	4.45M (128%)†	262k (7%)	254k (7%)	561k (16%)	232k (6%)
LUTs	2.14M (123%)†	110k (6%)	188k (10%)	282k (16%)	477k (27%)

Latencies in ns are computed using the nominal target clock frequency of 360 MHz. Percentages for resources used are calculated with respect to the available resources of an XCVU13P FPGA.
†Resources exceeding 100% indicate that the Float32 design does not fit on a single XCVU13P device; no Fmax is reported for the Float32 design.
*Best overall design; selected for deployment.

all exceeding the available resources of the target XCVU13P FPGA, the model cannot be deployed in this form.

The *Fixed-PTQ* configuration reduces the latency to 77 cycles and brings the resource usage within device bounds, with 56% DSP utilization. While the resource utilization is below the device budget, a large number of DSP slices are still used because scale factors are not fully eliminated from the datapath.

The *INT8-PO2* configuration removes the rescaling multipliers by approximating the scale factors as powers of two, reducing the latency to 28 cycles and the DSP usage to 43% of the device, at the cost of 0.7 pp in accuracy relative to the floating-point-rescaled *Fixed-PTQ* baseline (comparable to the per-seed standard deviation). Finally, the *INT8-PO2 Opt.* configuration combines PO2 scaling with data-driven bit-width optimization, reaching 19 cycles of latency and 52.8 ns at the nominal 360 MHz clock frequency, accompanied by a resource usage of 20% DSP slices, 6% FFs, and 27% LUTs. This configuration is therefore selected as the design for deployment, as it provides the most favorable trade-off among the evaluated designs. The reduction in DSP utilisation is obtained by directing the synthesis tool to implement the narrowed MAC operations in LUT fabric rather than DSP48E2 slices; this accounts for the increase in LUT utilisation from 10% to 27% between the *INT8-PO2* and *INT8-PO2 Opt.* configurations.

The *QAT* entry in Table I recovers 1.8 pp of accuracy relative to the PTQ baseline, reaching an accuracy of 76.8% compared with the 75.0% accuracy for INT8-PO2 (a gap comparable to the per-seed standard deviation, so to be confirmed with a paired significance test). In the current HLS realization, this comes at the cost of higher latency and resource usage because the learned quantization scales are implemented through fixed-point rescaling operations rather than the fully multiplier-free PO2 approximation.

VII. Conclusions

This work presents an end-to-end reproducible methodology for deploying GraphSAGE-based GNN inference on FPGAs under the strict latency and resource constraints of the CMS Phase-2 L1 trigger. Starting from a floating-point PyTorch model, the workflow systematically develops fixed-size graph extraction, integer-only post-training quantization, power-of-two scale approximation, data-driven bit-width optimization, and bit-exact validation against a Python integer reference. All components—from seed control and calibration subgraph selection to HLS parameter generation—are fully automated and version-controlled, enabling reproduction on alternative datasets and network architectures.

While the floating-point implementation is not deployable on the target XCVU13P FPGA, requiring 283% of the available DSP budget and exceeding by 20% both LUT and FF capacity, the optimized INT8-PO2 design reduces the latency to 19 cycles, corresponding to 52.8 ns at a nominal 360 MHz clock frequency, while using 20% of DSP slices, 6% of FFs, and 27% of LUTs. This demonstrates that multiplier-free integer rescaling and careful bit-width selection are essential for fitting message-passing GNNs into real-time FPGA trigger systems and sets the foundation for the deployment of more complex network architectures.

The present study uses Cora as a firmware-proxy benchmark to validate the methodology and quantify implementation trade-offs. Accuracy measurements include uncertainty quantification (mean $\pm$ std over multiple random seeds) to reflect the variability inherent in stochastic training. The same workflow can be applied to a physics-driven dataset, using detector-specific graph sizes and features, and integrating quantization-aware training to recover accuracy while preserving the low-latency integer datapath. The full training, quantization, and HLS-generation pipeline is publicly available at https://github.com/INTREPID-hep/graphsage-cora.

References

[1] G. Apollinari, O. Brüning, T. Nakamoto, and L. Rossi, "High Luminosity Large Hadron Collider HL-LHC," *CERN Yellow Rep.*, no. 5, pp. 1–19, 2015.

[2] CMS Collaboration, "The CMS Experiment at the CERN LHC," *JINST*, vol. 3, p. S08004, 2008.

[3] CMS Collaboration, "The Phase-2 Upgrade of the CMS Level-1 Trigger," CERN, Geneva, Technical Design Report CERN-LHCC-2020-004, CMS-TDR-021, 2020. [Online]. Available: https://cds.cern.ch/record/2714892

[4] CMS Collaboration, *The CMS muon project*, ser. Technical design report. CMS. Geneva: CERN, 1997. [Online]. Available: https://cds.cern.ch/record/343814

[5] K. Bunkowski, "The algorithm of the CMS Level-1 Overlap Muon Track Finder trigger," *Nucl. Instrum. Meth. A*, vol. 936, pp. 368–369, 2019, on behalf of the CMS Collaboration.

[6] W. M. Zabolotny and A. Byszuk, "Algorithm and implementation of muon trigger and data transmission system for barrel-endcap overlap region of the CMS detector," *JINST*, vol. 11, no. 03, p. C03004, 2016.

[7] P. Leguina, S. Folgueras, A. Cardini, and E. Aller, "Using AI on FPGAs for the CMS Overlap Muon Track Finder for the HL-LHC," in *EuCAIFCon 2025*, 9 2025, on behalf of the CMS Collaboration.

[8] A. Cardini, S. Folgueras, P. Leguina, E. Aller Gutierrez, P. Vischia, and C. Ramón Álvarez, "Enhancing the OMTF Trigger for Phase-2 with a GNN," Presented at the 7th Inter-Exp. LHC ML Workshop, 2025, on behalf of the CMS Collaboration.

[9] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[10] Y. Iiyama, G. Cerminara, A. Gupta, J. Kieseler, V. Loncar, M. Pierini, S. R. Qasim, M. Rieger, S. Summers, G. Van Onsem, K. Wozniak, J. Ngadiuba, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, K. Pedro, N. Tran, E. Kreinar, and Z. Wu, "Distance-Weighted Graph Neural Networks on FPGAs for Real-Time Particle Reconstruction in High Energy Physics," *Front. Big Data*, vol. 3, p. 598927, 2020.

[11] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and Z. Wu, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[12] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective Classification in Network Data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[13] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting Semi-Supervised Learning with Graph Embeddings," in *Proc. 33rd Int. Conf. Mach. Learn.*, ser. Proceedings of Machine Learning Research, vol. 48, 2016, pp. 40–48.

[14] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.

[15] A. Pappalardo, G. Franco, I. Colbert, F. Grob, T. Costigan, O. Savolainen, A. Stoian, A. Gerdelan, Y. Umuroglu, T. Paine, K. Witham, S. Khan, P. Monteagudo Lago, O. Peracha, L. Montero, J. Duarte, and F. Jentzsch, "Xilinx/Brevitas," 2021. [Online]. Available: https://github.com/Xilinx/brevitas