End-to-end codesign of Hessian-aware quantized neural networks for FPGAs

JAVIER CAMPOS, Fermilab, Batavia, United States
JOVAN MITREVSKI, Fermilab, Batavia, United States
NHAN TRAN, Fermilab, Batavia, United States
ZHEN DONG, University of California Berkeley, Berkeley, United States
AMIR GHOLAMINEJAD, University of California Berkeley, Berkeley, United States
MICHAEL W. MAHONEY, University of California Berkeley, Berkeley, United States
JAVIER DUARTE, University of California San Diego, La Jolla, United States

We develop an end-to-end workflow for the training and implementation of co-designed neural networks (NNs) for efficient field-programmable gate array (FPGA) hardware. Our approach leverages Hessian-aware quantization of NNs, the Quantized Open Neural Network Exchange intermediate representation, and the hls4ml tool flow for transpiling NNs into FPGA firmware. This makes efficient NN implementations in hardware accessible to nonexperts in a single open sourced workflow that can be deployed for real-time machine-learning applications in a wide range of scientific and industrial settings. We demonstrate the workflow in a particle physics application involving trigger decisions that must operate at the 40-MHz collision rate of the CERN Large Hadron Collider (LHC). Given the high collision rate, all data processing must be implemented on FPGA hardware within the strict area and latency requirements. Based on these constraints, we implement an optimized mixed-precision NN classifier for high-momentum particle jets in simulated LHC proton-proton collisions.

CCS Concepts: • **Networks → Network algorithms**; **Data path algorithms**; • **Computing methodologies → Neural networks**; • **Hardware → Hardware accelerators;**

Amir Gholaminejad also with International Computer Science Institute.
Michael W. Mahoney also with International Computer Science Institute and Lawrence Berkeley National Laboratory.
J. Campos, N. Tran, A. Gholaminejad, M. W. Mahoney, and J. Duarte are supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under the "Real-time Data Reduction Codesign at the Extreme Edge for Science" Project (DE-FOA-0002501). J.M. is supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the DOE, Office of Science, Office of High Energy Physics. J.D. is also supported by the DOE, Office of Science, Office of High Energy Physics Early Career Research program under Grant No. DE-SC0021187, and the U.S. National Science Foundation (NSF) Harnessing the Data Revolution (HDR) Institute for Accelerating AI Algorithms for Data Driven Discovery (A3D3) under Cooperative Agreement No. OAC-2117997. N.T. is also supported by the DOE Early Career Research program under Award No. DE-0000247070.
Authors' Contact Information: Javier Campos, Fermilab, Batavia, Illinois, United States; e-mail: jcampos@fnal.gov; Jovan Mitrevski, Fermilab, Batavia, Illinois, United States; e-mail: jmitrevs@fnal.gov; Nhan Tran, Fermilab, Batavia, Illinois, United States; e-mail: ntran@fnal.gov; Zhen Dong, University of California Berkeley, Berkeley, California, United States; e-mail: zhendong@berkeley.edu; Amir Gholaminejad, University of California Berkeley, Berkeley, California, United States; e-mail: amirgh@berkeley.edu; Michael W. Mahoney, University of California Berkeley, Berkeley, California, United States; e-mail: mmahoney@stat.berkeley.edu; Javier Duarte, University of California San Diego, La Jolla, California, United States; e-mail: jduarte@ucsd.edu.

ACM 1936-7406/2024/06-ART36
https://doi.org/10.1145/3662000

Additional Key Words and Phrases: Neural networks, field programmable gate arrays, firmware, high-level synthesis

ACM Reference Format:
Javier Campos, Jovan Mitrevski, Nhan Tran, Zhen Dong, Amir Gholaminejad, Michael W. Mahoney, and Javier Duarte. 2024. End-to-end codesign of Hessian-aware quantized neural networks for FPGAs. *ACM Trans. Reconfig. Technol. Syst.* 17, 3, Article 36 (June 2024), 22 pages. https://doi.org/10.1145/3662000

1 INTRODUCTION

Machine learning (ML) is pervasive in big data processing, and it is becoming increasingly important as data rates continue to rise. In particular, ML taking place as close to the data source as possible, or *edge ML*, is increasingly important for both scientific and industrial applications, including applications such as data compression, data volume reduction, and feature extraction for real-time decision-making [18]. Integrating ML at the edge, however, is challenging because of area, power, and latency constraints. This is especially the case for **deep learning (DL)** and **neural network (NN)** models. Deployment of NNs for edge applications requires carefully optimized protocols for training as well as finely tuned implementations for inference. This typically requires efficient computational platforms such as **field-programmable gate arrays (FPGAs)** and **application-specific integrated circuits (ASICs)**. Developing a NN algorithm and implementing it in hardware within system and task constraints is a multistep *codesign* process with a large decision space. Among other things, this space includes options related to *quantization*, or using reduced precision operations. In this article, we present a completely open source, end-to-end workflow accessible to nonexperts for NN quantization and deployment in FPGAs.

Quantization-aware training (QAT) has been shown to be very successful in scaling down model sizes for FPGAs [10, 11, 16, 21, 36, 44]. With QAT, large NNs can be quantized to 8 bits and below, with comparable accuracy to the baseline. **Quantized NNs (QNNs)** generally have considerably reduced model sizes and latencies. **Hessian-aware quantization (HAWQ)** [65] is a mixed-precision integer-only quantization framework for PyTorch [53] with promising applications. HAWQ is able to quantize the model to very small bit widths by using mixed-precision guided by second-order (Hessian) information. In this approach, sensitive layers (determined by Hessian information) are kept at higher precision and insensitive layers are kept at lower precision. FPGAs are a natural use case for this: They can benefit from this approach, since mixed-precision computations are much better supported by FPGAs than other hardware such as GPUs.

While these features make HAWQ an interesting choice for QAT with FPGAs, there does not currently exist a streamlined process to deploy it onto FPGAs directly. To address this, we introduce additional functionality to HAWQ to export QNNs as **Quantized Open Neural Network Exchange (QONNX)** [52] intermediate representations. Then the QONNX representation can be ingested by hls4ml [24], an open source Python library for NN translation and deployment in FPGA and ASIC hardware. The hls4ml package is designed to be accessible for both hardware experts and nonexperts, and it is flexible enough to deploy QNNs with a broad range of quantization bit widths on different FPGA and ASIC platforms. It is a popular tool for both scientific and industry edge ML applications [1, 28, 51].

To demonstrate the performance of our end-to-end workflow, we develop a NN for real-time decision-making in particle physics. The CERN **Large Hadron Collider (LHC)** is the world's largest and most powerful particle accelerator. Particles collide in detectors every 25 ns, producing tens of terabytes of data. Because of storage capacity and processing limitations, not every collision event can be recorded. In these experiments, the online trigger system filters data and

stores only the most "interesting" events for offline analysis. Typically, the trigger system uses simple signatures of interesting physics, e.g., events with large amounts of deposited energy or unusual combinations of particles, to decide which events in a detector to keep. There are multiple stages of the trigger system, and the first stage, referred to as the **level-1 trigger (L1T)** [2, 60], processes data at 40 MHz with custom ASICs or FPGAs. Over the past years, the LHC has increased its center of mass collision energy and instantaneous luminosity to allow experiments to hunt for increasingly rare signals. With the extreme uptake of accumulated data, ML methods are being explored for various tasks at the L1T [13, 14]. One such task is *jet tagging*: identifying and classifying collimated showers of particles from the decay and hadronization of quarks and gluons using *jet substructure* information [41, 42]. ML methods show great promise over traditional algorithms in increasing our capability to identify the origins of different jets and discover new physical interactions [8, 59]. Although we focus on a single application in this article, the potential scientific use cases for this workflow are numerous and span multiple domains [18]. Even for LHC trigger applications, there are dozens of potential tasks, ranging from charged particle tracking [19, 27], anomaly detection [31, 32, 46], calorimeter clustering [37, 55], long-lived particle identification [6, 15], and Higgs boson identification [43, 49]. In this article, we lay out an optimal workflow for these and more, depicted in Figure 1.

Within the context of developing a NN for real-time decision making for particle physics applications, the original contributions of this article are the following:

- We take advantage of the QONNX format to represent QNNs with arbitrary precision and mixed-precision quantization to extend HAWQ for QONNX intermediate representation support.
- We perform Hessian-aware quantization on a **multilayer perceptron (MLP)** model used in jet tagging benchmarks, and we study in detail the effects of quantization on each layer for model performance and efficiency.
- We use hls4ml to present optimized resources and latency for FPGA hardware implementations of NNs trained in HAWQ.

The rest of this article is structured as follows. In Section 2, we introduce the key steps that comprise the end-to-end codesign workflow for QNNs to be deployed on FPGAs, including an overview of quantization and HAWQ. We present the task and discuss how NNs are evaluated and trained in Section 3. Preliminary QAT results with homogeneous quantization and Hessian-based quantization are presented in Section 4, and our extension to HAWQ is presented in Section 5. We then cover the firmware implementation of NNs, specifically the resource usage and estimated latency, in Section 6. In Section 7 we illustrate the complete workflow using a second physics model intended for the **Compact Muon Solenoid (CMS)** high-granularity calorimeter. This highlights the versatility and applicability of the methodology, emphasizing its relevance beyond specific references. Finally, a summary is presented in Section 8.

2 BACKGROUND AND RELATED WORK

In this section, we provide an overview of quantization and HAWQ (in Section 2.1), and then we cover automatic bit-width selection (in Section 2.2) and firmware generation tools (in Section 2.3).

2.1 Quantization

Quantization in NNs refers to reducing the numerical precision used for inputs, weights, and activations. In *uniform affine quantization*, values are quantized to lower precision integers using a mapping function defined as

$$q = \text{quantize}(r) = \text{Clip}(\text{Round}((r/S) - Z), \alpha, \beta), \tag{1}$$

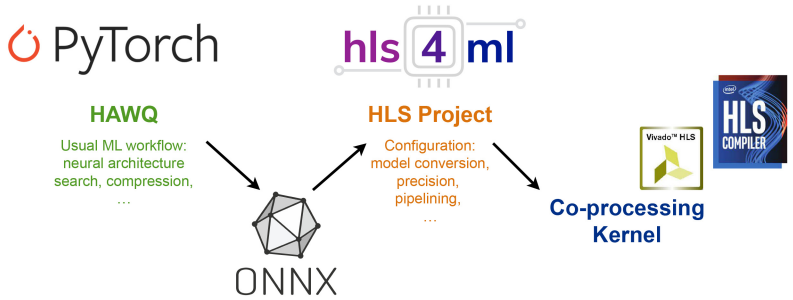

Fig. 1. An overview of the entire workflow, integrating HAWQ, QONNX, and hls4ml, is designed to facilitate a seamless transition from HAWQ to firmware implementations. Each stage is highly configurable; for instance, HAWQ supports mixed-precision quantization. Similarly, QONNX provides graph transformations and optimization support, while hls4ml allows control over precision and parallelism.

where r is the floating-point input, S is the *scale factor*, and Z is the *zero point* [29]. The Round function is the round-to-nearest operation clipped/clamped at α and β. Because all quantization bins are uniformly spaced, this mapping function in Equation (1) is referred to as uniform quantization. Nonuniform quantization methods whose bin sizes are variable are more difficult to implement in hardware [45]. Real values can be recovered from the quantized values through *dequantization*:

$$\tilde{r} = \text{dequantize}(q) = S(q + Z), \tag{2}$$

where $\tilde{r} - r$ is known as the quantization error. The scale factor divides a given range of real values into 2^b bins, with

$$S = \frac{\beta - \alpha}{2^b - 1}, \tag{3}$$

where $[\alpha, \beta]$ is the clipping range and b is the bit width. Choosing the clipping range is referred to as *calibration*. A simple approach is to use the minimum and maximum of the values, i.e., $\alpha = r_{\min}$ and $\beta = r_{\max}$. This is an asymmetric quantization scheme, because the clipping range is not necessarily symmetric with respect to the input, i.e., it could be that $-\alpha \neq \beta$. A symmetric quantization approach uses a symmetric clipping range of $-\alpha = \beta$, such as $-\alpha = \beta = \max(|r_{\max}|, |r_{\min}|)$, and replaces the *zero point* with $Z = 0$.

The latest publication of the Hessian-aware quantization, HAWQv3 [65], introduces a completely new computational graph with an automatic bit-width selection policy based on it is previous works [22, 23]. In HAWQv3, which for simplicity we refer to here simply as HAWQ, quantization follows Equation (1) with additional hardware-inspired restrictions. HAWQ executes its entire computational graph using only integer multiplication, addition, and bit shifting, without any floating-point or integer division operations. The clipping range is symmetric for weights $\beta = 2^b - 1 = -\alpha$, while activations can be either symmetric or asymmetric. The real-valued scale factors are pre-calculated by analyzing the range of outputs for different batches and fixed at inference time, a process called *static quantization*. HAWQ avoids floating-point operations and integer divisions by restricting all scale factors to be dyadic numbers (rational numbers of the form $b/2^c$, where b and c are integers). To illustrate a typical computation, consider a layer with input h and weight tensor W. In HAWQ, h and W are quantized to $S_h q_h$ and $S_W q_W$, respectively, where S_h and S_W are the real-valued scale factors, and q_h and q_W are the corresponding quantized integer values. The output result, denoted by a, can be computed as

$$a = (S_W S_h)(q_W * q_h), \tag{4}$$

where $*$ denotes a low-precision integer matrix multiplication (or convolution). The result is then quantized to $S_a q_a$ for the following layer as

$$q_a = \text{Int}\left(\frac{a}{S_a}\right) = \text{Int}\left(\frac{S_W S_h}{S_a}(q_W * q_h)\right), \tag{5}$$

where S_a is a precalculated scale factor for the output activation. This avoids floating point operations and integer divisions by implementing Equation (5) with integer multiplication and bit shifting.

2.2 Automatic Bit Width Selection

Many methods have been proposed to measure the sensitivity to quantization or developed automatic schemas for bit settings. For example, HAQ [63] proposed a **reinforcement learning (RL)** method to determine the quantization policy automatically. The method involves an RL agent receiving direct latency and energy feedback from hardware simulators. Reference [64] formulated a **neural architecture search (NAS)** problem with a differentiable NAS to explore the search space efficiently. Reference [50] proposed periodic functions as regularizers, where regularization pushes the weights into discrete points that can be encoded as integers. One disadvantage of these exploration-based methods is that they are often sensitive to hyperparameters or initialization. More recently, AutoQkeras [16] was proposed as a method to optimize both model area (measured by the number of logical elements in the FPGA design) and accuracy, given a set of resource constraints and accuracy metrics, e.g., energy consumption or bit-size. Different from these previous methods, HAWQ [23] introduced an automatic way to find the mixed-precision settings based on a second-order sensitivity metric. In particular, the Hessian (specifically the top Hessian eigenvalue) can be used to measure the sensitivity. This approach was extended in Reference [22], wherein the sensitivity metric is computed using the average of all the Hessian eigenvalues. The Hessian provides valuable information to guide quantization, but it is not restricted to a particular quantization scheme. In theory, other methods could be applied, such as stochastic binary, power-of-two, look-up table, small floating-point, or clustering-based quantization.

2.3 Firmware Generation Tools

Although ML methods have shown promising results on edge devices, fitting these algorithms onto FPGAs is challenging, often very time-consuming, and it requires the expertise of domain experts and engineers. Several directions aim to solve this issue. One direction, field-programmable deep neural networks [33] is a framework that takes TensorFlow-described **deep neural networks (DNNs)** as input and automatically generates hardware implementations with **register transfer level (RTL)** and **high-level synthesis (HLS)** hybrid templates. Another direction, fpgaConvNet [62], specifically targets **convolutional NNs (CNNs)** and is an end-to-end framework for the optimized mapping of CNNs on FPGAs. Interestingly, fpgaConvNet proposes a multi-objective optimization problem to account for the CNN workload, target device, and metrics of interest.

These and other tools indicate a growing desire to deploy more efficient and larger ML models on edge devices in a faster and more streamlined process. This desire arises in many scientific and industrial use cases [4, 18]. Particle physics applications are a particularly strong stress test of such tools. This is due to the extreme requirements in computational latency and data bandwidth, as well as environmental constraints such as low-power and high-radiation and cryogenic environments. Furthermore, particle physics practitioners are not necessarily ML experts or hardware experts, and their applications and systems require open source tools (to the extent possible) and flexible deployment across different FPGA and ASIC platforms. The hls4ml tool originated from these use cases and supports multiple architectures and frameworks, such as Keras [12], QKeras [16, 30],

and PyTorch [53]. Previous work has successfully implemented DL models for FPGAs and ASICs for particle physics [20]. Currently, it is steadily increasing its scope of supported architectures, frameworks, hardware optimizations, and target devices, with the support of a growing scientific community. Another tool, FINN [7, 61] from AMD/Xilinx, aims to solve the problem of bringing NNs (more specifically, QNNs) to FPGAs by using generated HLS code. Both tools create a streamlined process to deploy DL models as efficiently as possible, without requiring large development effort and time. The two tools are similar in their goals, hls4ml and FINN, although there are differences in their flows, layer support, and targeted optimizations. Both of them support QONNX, an open source exchange format, which represents QNNs with arbitrary precision, so that there can be interoperability between the flows. This is ideal for HAWQ, as it can target multiple hardware-generating tools. In this work, however, we focus only on hls4ml, which has implementations for FPGAs and ASICs and optimizations for a wider range of bit widths. Additionally, the emphasis of this work is on FPGAs. Nevertheless, we note that previous research has used hls4ml to develop ASIC accelerators for NNs [47, 48]. Consequently, the workflow we present here has the potential to be applied for ASIC development as well.

3 EXPERIMENTAL SETUP

In this section, we describe the benchmark ML task we explore for particle physics applications. As discussed above, although there are a much broader set of scientific and industrial applications, particle physics applications are a particularly good stress test of our end-to-end workflow. The concept behind the development of particle physics benchmarks is detailed more in Reference [25], and our jet tagging benchmark is one of the three described there.

3.1 Dataset and Task

We consider a jet classification benchmark of high-p_{T} jets to evaluate performance. Particle jets are radiation patterns of quarks and gluons produced in high-energy proton-proton collisions at the LHC. As these jets propagate through detectors like the A Toroidal LHC ApparatuS or CMS, they leave signals through the various subdetectors, such as the silicon tracker, electromagnetic or hadron calorimeters, or muon detectors. These signals are then combined using jet reconstruction algorithms. In the CMS trigger upgrade, jet classifiers must process inputs every 150 ns and achieve a latency less than 1 μs [13]. These are the real-time constraints for ML algorithms on the CMS trigger system. We use the benchmark presented in Reference [24] consisting of 54 features from simulated particle jets produced in proton-proton collisions. Of the 54 high-level features, 16 were chosen based on Table 1 of Reference [24]. The features are a combination of both mass ("dimensionful") and shape ("dimensionless") observables. The dataset [54] is a collection of 870,000 jets and is divided into two sets: a training set of 630,000 jets and a test set of 240,000 jets. The dataset underwent preprocessing: All features are standardized by removing the mean and scaling to obtain unit variance. The task is to discriminate jets as originating from one of five particles: W bosons, Z bosons, light quarks (q), top quarks (t), or gluons (g). Descriptions of each observable and particle jet can be found in Reference [17]. Additionally, we measure the accuracy given by the number of correctly classified jets divided by the total number of classified jets.

3.2 Model and Loss Definition

We implement all models with the architecture presented in Reference [24], an MLP with three hidden layers of 64, 32, and 32 nodes, respectively. The baseline model is the floating-point implementation of this MLP, i.e., with no quantization. All hidden layers use ReLU activations, and the output is a probability vector of the five classes filtered through the softmax activation function.

We aim to minimize the empirical loss function,

$$\mathcal{L}_c(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), \mathbf{y}_i) = \frac{1}{N}\sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i), \tag{6}$$

where ℓ is the categorical cross-entropy loss function and N is the number of training samples. The model, denoted by f_θ, maps each input $\mathbf{x}_i \in \mathbb{R}^{16}$ to a prediction $\hat{\mathbf{y}}_i \in [0,1]^5$, using parameters θ. Predictions are then compared with ground truth $\mathbf{y}_i$ to minimize the empirical loss. We train the NNs with L_1 regularization by including an additional penalty term to the loss,

$$\mathcal{L}(\theta) = \mathcal{L}_c(\theta) + \lambda \sum_{j=1}^{L} \|\mathbf{W}_j\|_1, \tag{7}$$

where the added penalty term is the elementwise norms of weight matrices, $\mathbf{W}_i$ is the "vectorized" form of weight matrix for the jth layer, and L is the number of layers in the model. The L_1 regularization term is scaled by a tunable hyperparameter λ. Typically, L_1 regularization is used to prevent overfitting, enabling statistical models to generalize better outside the training data. It is also known to promote sparsity, which is desirable to reduce the number of computations. Section 4 discusses the implications of L_1 regularization in QNNs concerning performance and other metrics discussed below.

3.3 Metrics: Bit Operations and Sparsity

Similarly to floating-point operations, **bit operations (BOPs)** [5] in QNNs are computed to estimate model complexity and the number of operations per inference. BOPs have been shown to accurately predict the area of hardware accelerators and, in turn, the power usage in processing elements [40]. This makes BOPs an easy-to-compute metric that is a useful approximation of the total area of a QNN. The bit operations of a fully connected layer with b_a bit input activations and b_W bit weights are computed as

$$\text{BOPs} = mn((1 - f_p)b_a b_W + b_a + b_W + \log_2(n)), \tag{8}$$

where n and m are the number of input and output features, respectively, and f_p is the fraction of weights pruned (i.e., equal to zero). This definition of BOPs only accounts for zero weight values within a layer. A more precise definition could compute the sum of all binary 1s in each weight individually. We do not consider this, as taking advantage of sparsity at this level of granularity requires task-specific architecture considerations that are not completely generalizable. From Equation (8), the number of BOPs is inversely related to the sparsity f_p. Sparse models are desired, as zero-weight multiplications are optimized out of the firmware implementation by HLS. This is a highly attractive feature of HLS, and it makes BOPs a noteworthy metric to observe. We measure the total BOPs of each quantization scheme, as well as its relation to accuracy (see Section 4) and hardware usage (see Section 6).

4 QUANTIZATION-AWARE TRAINING

In this section, we discuss the training procedure for homogeneous and mixed-precision quantization. We start in Section 4.1 with a discussion of single bitwidth quantization, which is also referred to as homogeneous quantization. Then, in Section 4.2, we discuss mixed-precision quantization, including how it can greatly improve classification performance, as well as its downsides. In particular, in Section 4.2.2, we cover a method to select automatically the bit width of each layer in a NN using second-order Hessian information, as well as a method obtained by imposing hardware constraints in the bit-width selection process.

Table 1. Classification Performance with Homogeneous Quantization

Precision		Baseline (%)	L_1 (%)	BN (%)	L_1+BN (%)
Weights	Inputs				
INT12	INT12	76.9	75.2	77.2	76.5
INT8	INT8	76.6	76.4	76.9	76.9
INT6	INT6	73.6	73.7	74.5	74.4
INT4	INT4	62.5	63.2	63.5	63.4
FP-32	FP-32	76.5	76.8	76.9	76.8

All weights, activations, and inputs are quantized with the same precision. Models are trained with and without L_1 regularization and BN. At INT8 and above, the accuracy is restored to baseline; but at INT6 and below, the accuracy is worse than baseline.

4.1 Homogeneous Quantization

Quantizing all layers with the same bit width is simple, but it can cause a significant loss in performance. In Table 1, we present the accuracy for different bit settings from INT12 to INT4 with homogeneous quantization using HAWQ. As expected, we see a significant performance degradation as we quantize below INT8 (and especially below INT6). To combat this, we employed two regularization techniques during training: L_1 regularization and **batch normalization (BN)** [38]. BN provides a more stable distribution of activations throughout training by normalizing the activations and producing a smoother loss landscape [57]. Although using BN raises performance on all quantization schemes, it fails to recover baseline accuracy for INT6 and INT4 quantization. Likewise, while L_1 regularization leads to some improvement, it falls short of restoring to its original baseline. Consequently, homogeneously quantizing a model with one bitwidth setting is insufficient for quantization below 8-bit precision.

In addition to employing regularization techniques, we can increase the input quantization bit width. In HAWQ, inputs are quantized before proceeding to the first layer, ensuring all operations are integer only. A possible failure point is quantization error introduced in the inputs for low bitwidths where key features needed to classify jets may be lost. We decouple the precision of the inputs from that of the weights and activations and increase it to INT16. Figure 2 shows results for 8-bit weights and below, with different bit widths for the activations. We find the following: (1) increasing the activation bit width significantly improves the classification performance of INT4 and INT6 weights; (2) similar improvements are obtained for INT16 quantized inputs, although this comes at the cost of increased hardware resource usage; and (3) L_1 and BN (applied alone or together) are insufficient for recovering the accuracy to baseline levels. For this study, BN is less desirable, as the batch statistics parameters are implemented with floating-point values, thereby increasing the latency and memory footprint. One option is to quantize these values or (even more promisingly) apply BN folding. The idea is to remove BN by using its parameters to update the fully connected (or convolution) layer's weights and biases for inference efficiency. However, after implementing BN folding using the procedure outlined in Reference [65], we found little to no effect on model performance. As mentioned above, L_1 serves as a regularization term to promote model generalization and mitigate overfitting without introducing additional parameters and latency overhead. As an additional advantage, it produces sparse weight matrices, which decreases the number of bit operations in hardware implementations. Therefore, in later sections, we continue to use L_1 during training for mixed-precision quantization and baseline. Figure 2 suggests model performance can greatly benefit from more fine-grained quantization settings.

However, manually adjusting all these quantization settings can be time-consuming and suboptimal. An optimized bit-setting scheme is needed to simultaneously minimize the loss and hardware

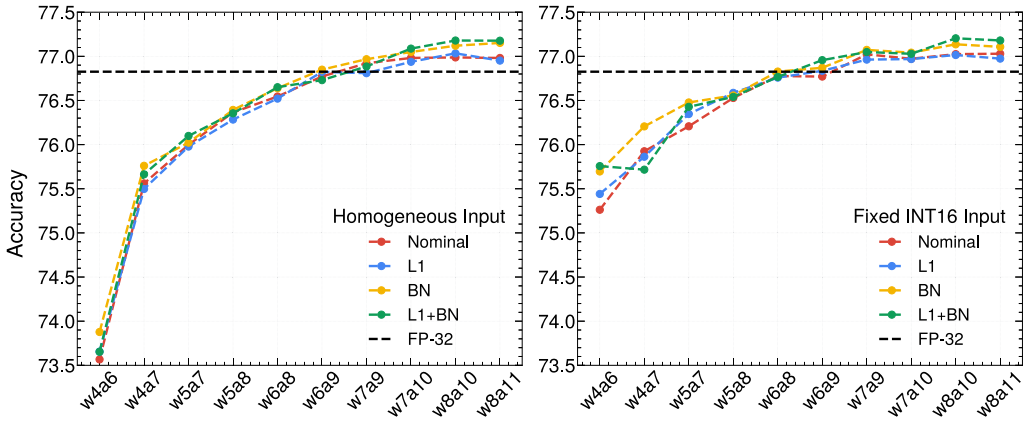

Fig. 2. Model performance using homogeneous quantization. The precision of weights is indicated after "w" and activations after "a." The test accuracy of a FP-32 NN serves as a reference point. Models are trained with L_1 regularization and BN. We can see (1) 16-bit input improves the model performance of all bit settings, (2) larger activation bit widths improve accuracy, and (3) L_1 and BN (applied alone or together) show no positive impact on performance.

usage. In the next subsection, we explore mixed-precision quantization. We fix the input bit width to INT16. This could be further optimized, but this choice makes direct comparison with other work easier [16, 24, 25, 34].

4.2 Mixed-precision Quantization

4.2.1 Brute-force Search. Mixed-precision quantization aims to improve performance by keeping certain layers at a higher precision than others. The basic problem with going beyond homogeneous quantization is that—when implemented naively—the search space for determining the bit setting is exponential to the number of layers. Our model architecture's MLP search space is significantly smaller than deep CNNs such as ResNet-50 [35], because our MLP only has three hidden layers. However, assuming we have 5-bit width options, finding the mixed-precision setting for our MLP classifier, with 4 fully connected layer weights and activations, has a search space of $((2)(4))^5 = 32,768$ combinations. It is impractical, especially for applications that need frequently retrained models or that need DNNs, to search this space exhaustively. Several methods have been proposed to address this problem of manually searching for the optimal bit configuration [9, 22, 58, 63, 64]. We use Reference [22], which is based on the Hessian information, and we observe the relative position of Hessian-based solutions within the *brute-force search* space.

4.2.2 Hessian-aware Quantization. As discussed in the Section 4.1, performance greatly benefited from higher precision in activations suggesting certain layers are more sensitive to quantization than others. We use the work first proposed in HAWQv2 [22] to determine the relative sensitivity of each layer for the baseline 32-bit floating point implementation of the model. The sensitivity metric is computed using the Hutchinson algorithm,

$$\mathrm{Tr}(H) \approx \frac{1}{k}\sum_{i=1}^{k} z_i^{\top} H z_i = \mathrm{Tr}_{\mathrm{Est}}(H), \tag{9}$$

where $H \in \mathbb{R}^{d\times d}$ is the Hessian matrix of second-order partial derivatives of the loss function with respect to all d model parameters, $z \in \mathbb{R}^d$ is a random vector whose component is independent and

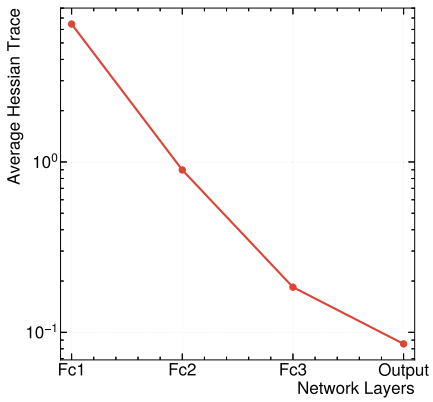

Fig. 3. Average Hessian trace of each fully connected (Fc) layer in the MLP. The Hessian is used as a sensitivity metric to quantization, where layers are ranked based on their trace. The first two layers are significantly larger than the others, signifying they are more prone to error at lower bit widths. The average Hessian traces are used to assign each layer a bit setting, i.e., layers with higher traces are assigned larger precision.

identically distributed sampled Rademacher distribution, and k is the number of Hutchinson steps used for trace estimation. Figure 3 shows the average Hessian trace (our sensitivity metric) of each layer in the baseline model, with logarithmic scaling. The first two layers are the most sensitive, with the first layer more sensitive than the second by a factor of 7. Thus, the first two layers in the network must have a larger bit-width setting, while the last two layers can be quantized more aggressively. While the Hessian traces provides a sensitivity metric, this does not directly translate to a bit configuration. Instead, Reference [22] assigns the bit width of each layer i by checking the corresponding Ω term, defined as

$$\Omega = \sum_{i=1}^{L} \Omega_i = \sum_{i=1}^{L} \overline{\mathrm{Tr}}(H_i) \|Q(W_i) - W_i\|_2^2, \tag{10}$$

where Q is the quantization function, $\|Q(W_i) - W_i\|_2^2$ is the squared L_2 norm of the quantization perturbation, and $\overline{\mathrm{Tr}}$ is the average Hessian trace. We apply the same technique as Reference [22], where the amount of second-order perturbation, Ω, is calculated for a given set of quantization schemes, and the minimal Ω is chosen. This procedure is fully automated without any manual intervention.

We follow the procedure outlined in HAWQ [65] to constrain Equation (10) by the total BOPs. We formulate an **integer linear programming (ILP)** optimization problem, where the objective is to minimize Ω_i while satisfying the constraints. Linear programming solvers are a well-established method for optimization, and recent applications have been applied in memory management for deep learning [39]. In our case, we set up an ILP problem to automatically determine the bit settings of our classifiers for various BOPs limits, and we compare these solutions with the brute-force and homogeneous quantization methods.

4.2.3 QAT Results. With the information provided in Figure 2, we apply all possible bit settings based on the initial implementation in homogeneous quantization. We explore the weight bit width $b_W = \{4, 5, 6, 7, 8\}$, and we set the activation bit width $b_a = b_W + 3$ to prevent saturation and further reduce the search space. All models are trained for 100 epochs, with L_1 regularization, and

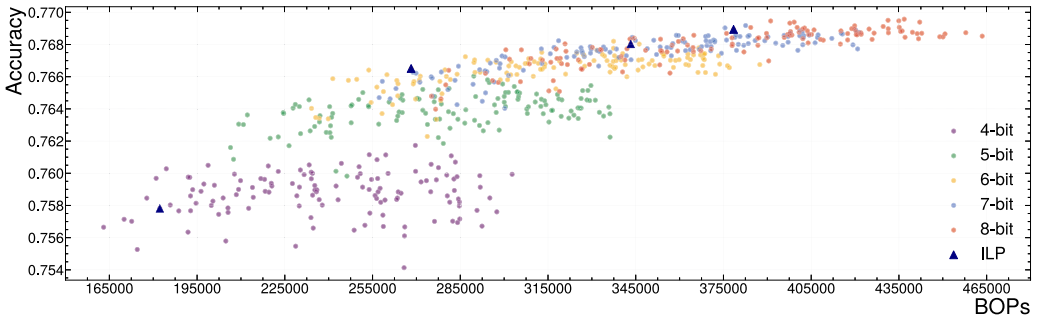

Fig. 4. Brute-force search quantization using weight bit widths $b_w = \{4, 5, 6, 7, 8\}$. Each data point is color-coded based on the bit width of the first fully connected layer. It is importance in quantization coincides with the observed clusters, with higher performing models using larger bit widths. Solutions based on ILP are presented. All ILP solutions make tradeoffs based on the quantization error and bit width and typically are among the lowest BOPs in their respective cluster.

all models use quantized inputs with INT16. Figure 4 presents the model accuracy against BOPs for all combinations of weight bits b_W. Data points are color coded based on the bit precision of the first layer. Several data points indicate a complete or nearly complete recovery to baseline accuracy (76.853%). The majority of points can be clustered based on the bit width of the first layer, since the model accuracy generally increases as the first layer's bit width increases. We can also see in Figure 4 that the bit width of the first fully connected layer greatly impacts the final model performance. Among the top 100 best-performing models, 66 had the first dense layer as INT8, and 33 had INT7 weights. This coincides with the average Hessian traces shown in Figure 3, showing the first layer is the most sensitive layer to quantization, by a factor of 7×, compared to the second most sensitive layer. Among the top models, we observed the frequency of 7-bit and 8-bit in later layers decrease significantly. The bit width of the later layers has fewer effects on the classification than the first two layers.

The ILP solutions to Equation (10) are also shown. The solutions are obtained with respect to four different BOPs constraints, from 250 k to 400 k in steps of 50 k. As expected, as the BOPs constraint increases, the selected precision of the first two layers increases. Hence, we begin to see more ILP solutions closer to the 8-bit cluster. The ILP solutions also tend to be positioned toward the lower end of BOPs in their local cluster. With brute-force search quantization and the ILP solutions shown side by side, the advantages of using the Hessian information become clearer. Although an optimal solution is not guaranteed, the Hessian method provides a stable and reliable solution to mixed-precision quantization. This is ideal for deep learning models that need to be quantized to meet the resource constraints and inference times of the LHC 40-MHz collision rate.

5 CONVERSION INTO QONNX

5.1 Intermediate Representations

To increase interoperability and hardware accessibility, the **Open Neural Network Exchange (ONNX)** format was established to set open standards for describing the computational graph of ML algorithms [3]. ONNX defines a common and wide set of operators enabling developers and researchers greater freedom and choice between frameworks, tools, compilers, and hardware accelerators. Currently, ONNX offers some support for quantized operators, including `QuantLinear`, `QLinearConv`, and `QLinearMatMul`. However, ONNX falls short in representing arbitrary precision and ultra-low quantization, below 8-bit precision. To overcome these issues, recent work [52]

introduced **quantize-clip-dequantize (QCDQ)** using existing ONNX operators and a novel extension with new operators, called QONNX, to represent QNNs. QONNX introduces three new custom operators: `Quant`, `Bipolar`, and `Trunc`. The custom operators enable uniform quantization and abstract finer details, making the intermediate representation graph flexible and at a higher level of abstraction than QCDQ.

For these reasons, we represent HAWQ NNs in the QONNX format, leveraging HAWQ's ultra-low precision and QONNX's abstraction to target two FPGA synthesizing tools, hls4ml and FINN [7, 61].[1] We also include the ONNX format in our model exporter for representing QNNs. In the next subsections, we describe the setup, export procedure, and validation steps to represent HAWQ NNs in the QONNX and QCDQ intermediate representations.

5.2 Model Translation

In PyTorch, exporting to ONNX works via tracing. This is the process of capturing all the operations invoked during the forward pass on some input. PyTorch provides the means for tracing through the `torch.jit` API. Tracing a model will return an executable that is optimized using the PyTorch just-in-time compiler. The executable contains the structure of the model and original parameters. Tracing will not record any control flow like if-statements and loops. The returned executable will always run the same traced graph on any input, which may not be ideal for functions or modules that are expected to run different sets of operations depending on the input and model state. The executable is then used to build the ONNX graph by translating operations and parameters within the executable to standard ONNX operators. In general, all PyTorch models are translated to ONNX using this process, and we extend this existing system to build support for QONNX operators in HAWQ.

The layers in HAWQ and operators in QONNX both require extra steps to support tracing and export. For each quantized layer in HAWQ, we implement a corresponding "export" layer. These dedicated export layers implement the forward pass and specify the equivalent QONNX operators based on the original layer parameters. This is accomplished by registering *symbolic functions* via `torch.onnx.register_custom_op_symbolic`. These symbolic functions decompose HAWQ layer operations into a series of QONNX nodes. Because we are using custom QONNX nodes, we also must register them via the `torch.onnx` API. Together, these preliminary steps define the HAWQ-to-QONNX translation. During the export process, the exporter looks for a registered symbolic function for each visited operator. If a given model contains quantized HAWQ or standard PyTorch layers, then it can be traced and finally translated to standard ONNX and QONNX operators. Because tracing records computations, the input can be random as long as the dimensions and data type are correct. The model exported with ONNX and QONNX operators is shown in Figure 5(a). With these additions, our exporter can perform the following:

(1) export models containing HAWQ layers to QONNX, with custom operators to handle a wide range of bit widths while keeping the graph at a higher level of abstraction; and
(2) export models containing HAWQ layers to standard ONNX with INT8 and UINT8 restrictions.

5.3 Post-Export

5.3.1 Optimization. To create firmware using hls4ml or FINN, the QONNX graph is expected to be normalized, i.e., to undergo several optimization steps. The QONNX software utilities [52]

[1]The main focus in exporting QNNs is the QONNX intermediate format. However, the QONNX software toolkit enables conversion to QCDQ format. This allows HAWQ to target hls4ml and FINN, and indirectly all other ONNX inference accelerators and frameworks.

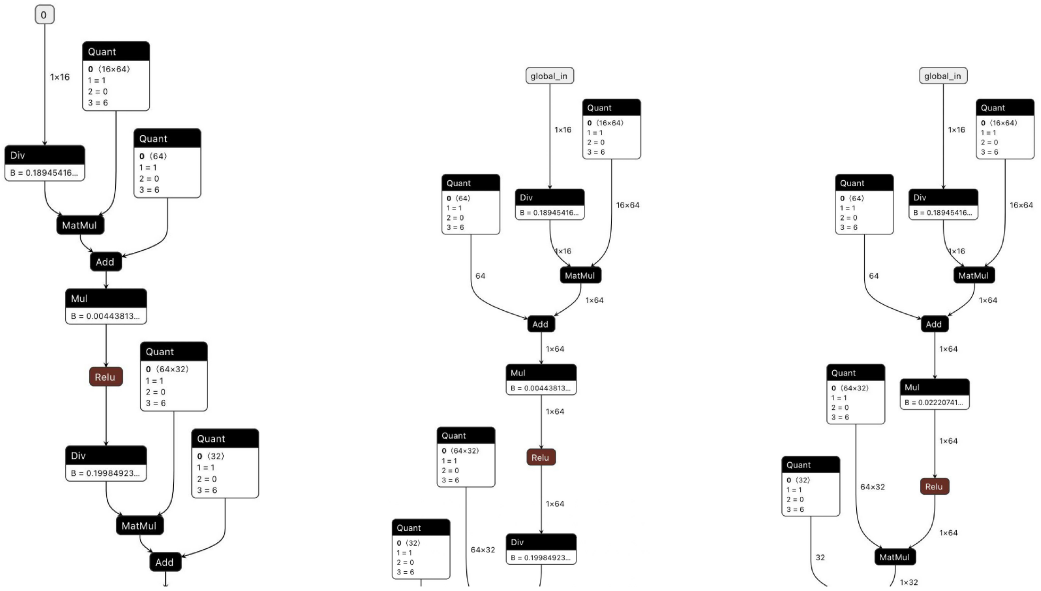

Fig. 5. The QONNX graph in its three stages after exporting. (a) The first layers of the model including the quantized fully connected layer before any optimizations. (b) The first layers of the model after post-clean-up operations: constant folding, shape inference, tensor, and node renaming. (c) The final optimization step: node merging across ReLU activations. All QNNs implemented in HAWQ can be exported to an QONNX or ONNX intermediate representation and undergo transformations as described in each stage.

provide these transformations, as shown in Figure 5(b), where shape inference and constant folding are applied to the graph. Figure 5(c) shows the last optimization step; we merge scaling factors across ReLU activation functions. For reasons related to the underlying implementation of HAWQ, there are two scaling operations before and after specific layers. For a detailed explanation of these scaling factors, see Section 2.1. To reduce the number of operations needed in firmware we combine the scaling factors in cases where the ReLU function is used. This cannot always be done, and it is dependent on the activation function used.

5.3.2 Graph Evaluation. Afterexporting, we evaluate the model using the QONNX software package [52], confirming a successful translation of our model from HAWQ to QONNX. While the main focus has been MLPs, exporting is not limited to this one architecture. All HAWQ layers now support QONNX export via the implemented symbolic functions. Moreover, with the QONNX software package, it is easy to transform, optimize, evaluate, and validate the exported HAWQ models.

6 HARDWARE GENERATION

In this section, we explain where HAWQ fits within the hls4ml hardware generation workflow. The total resources used, BOPs, and classification performance for different bit-width configurations are shown and discussed.

6.1 hls4ml Ingestion

The hls4ml workflow automatically performs the translation of the architecture, weights, and biases of NNs, layer by layer, into code that can be synthesized to RTL with HLS tools. The first part of this workflow entails training a NN for a task as usual with PyTorch, Keras, QKeras, or

HAWQ. For HAWQ, a QONNX graph must be exported from the model, but this step can (optionally) be performed for all the frameworks (and, eventually, this will be the preferred flow). Next, hls4ml translates the QONNX graph into an HLS project that can subsequently be synthesized and implemented on an FPGA or ASIC in the final step of the workflow.

The LHC imposes various constraints on both software and hardware, and these constraints must adapt as experiments evolve. Therefore, it is crucial to minimize the hardware footprint to maintain flexibility for future modifications. All results presented are synthesized for a Xilinx Kintex Ultrascale FPGA with part number `xcu250-figd2104-2L-e`. We report the usage of different resources: **digital signal processor units (DSPs)**, **flip-flops (FFs)**, and **look-up tables (LUTs)**. We do not report the **block RAM (BRAM)**, a dense memory resource, usage, because its only use in the design is by the softmax activation, whose numerical precision is the same for all quantization schemes. The softmax layer implemented in HLS uses tables to store the results of the exponential function. The precomputed values, stored in BRAM on an FPGA, remove this computation during inference. Only the "bare" firmware design needed to implement the NN is built with RTL synthesis using Vivado 2020.1. All NNs are maximally parallelized. In hls4ml, parallelization is configured with a "reuse factor" that sets the number of times a multiplier is used to compute the layer output. A fully parallel design corresponds to a reuse factor of one. All resource usage metrics are based on this "bare" implementation after RTL synthesis, and all designs use a clock frequency of 200 MHz, which is a standard FPGA clock rate commonly used in LHC experiments.

6.2 Synthesis Results

The LHC imposes various constraints on both software and hardware, and these constraints must adapt as experiments evolve. Therefore, it is crucial to minimize the hardware footprint to maintain flexibility for future modifications. Figure 6 shows the resource usage versus the accuracy of the implemented designs. The quantization bit-width settings were chosen at random. Higher-performing models use more resources. This is expected, as the top 100 performing models use larger bit widths for the first layer, which is the largest layer in the model. As such, we expect to see more resources as accuracy increases. LUTs have the most linear relationship with accuracy, whereas FFs and DSPs also increase with accuracy. The relationship between BOPs and resources, presented in Figure 7, also shows a linear relationship between LUTs and BOPs, which scale with the bit width and weight matrix dimensions. The number of LUTs used depends on the bit width, because, at low bit widths, addition and multiplication are implemented with LUTs. However, DSPs are used at larger bit widths, because they become much more efficient. DSPs offer custom datapaths that efficiently implement a series of arithmetic operations, including multiplication, addition, **multiply-accumulate (MAC)**, and work-level logical operations. DSP datapaths are less flexible than programmable logic, but they are more efficient at multiplying and MAC operations. This is shown in Figure 7 as DSP usage increases dramatically at points with larger BOPs. Switching from LUTs to DSPs depends on the target device and Vivado HLS internal biases toward DSPs for certain bit widths. The shift toward DSPs occurs with 11 or wider bits in Vivado 2020.1, with multiplications lower than this limit implemented using LUTs. The result of these operations is stored in FFs, displaying a steady increase with fewer variations than that seen in DSPs. The number of FFs up to 250k BOPs rise at a constant pace, with deviations beginning to appear thereafter. The inconsistencies between the number of FFs for neighboring BOPs suggest that there is a weaker correlation between the two. The deviations arise from the precision needed for intermediate accumulations and the total FFs needed will vary network to network.

The **baseline (BL)** model is synthesized after adjusting the weights without any fine-tuning. In hls4ml, parameters and computations are performed using fixed-point arithmetic, and each layer in the model can be quantized after training by specifying a reduced precision. Fixed-point

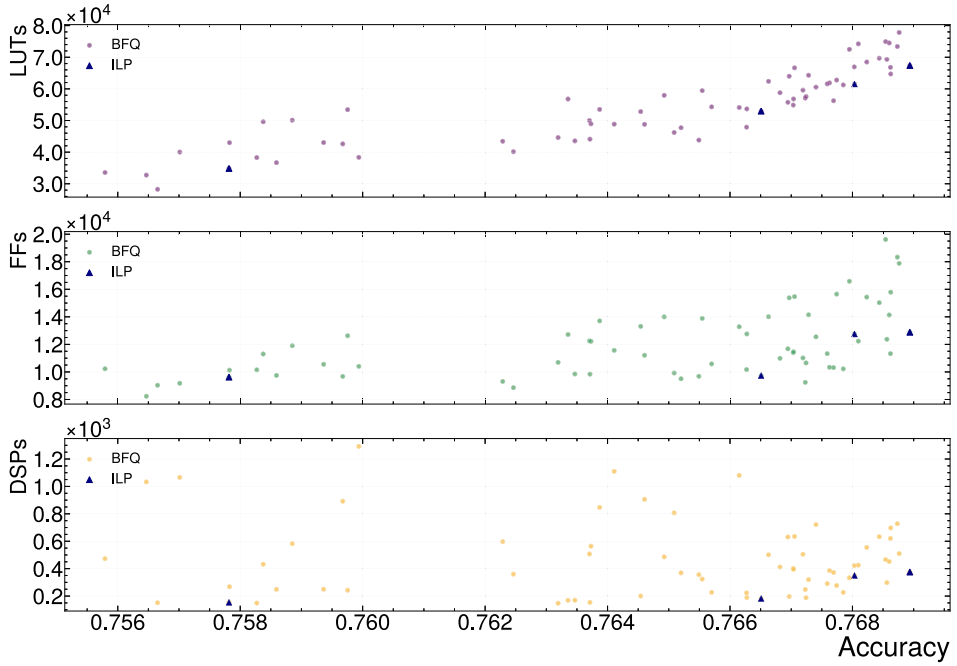

Fig. 6. Resource usage for a subset of brute-force quantization (BFQ) using weight bit widths $b_W = \{4, 5, 6, 7, 8\}$. LUT, FF, and DSP usage versus accuracy are shown, with quantization schemes that perform better among the highest resource users. All solutions to the ILP problem from BOPs constraint are presented. Extra logical elements are needed to maintain accuracy while considerable reduction in all metrics can be achieved with a 1–2% drop in accuracy.

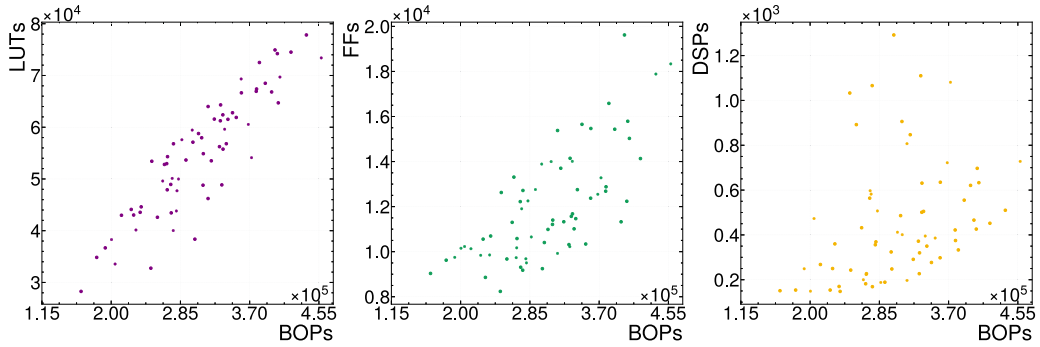

Fig. 7. Resource usage for a subset of QNNs in a brute-force attempt to an optimal mixed-precision quantization scheme. LUT, FF, and DSP usage versus BOPs are shown, with LUTs having the most linear relationship to BOPs. This relationship weakens with larger bit widths, as DSPs can implement MAC operations more efficiently. In all designs, FFs are the only type of memory utilized in fully connected layers, and the total used can drastically vary for neighboring BOPs, implying a weaker relationship between the two.

data types model the data with `I` integer bits (including the sign) and `F` fractional bits, denoted `ap_fixed<W,I>`, where `W = I + F`. The BL model uses `ap_fixed<16,6>` for all parameters and computations and is fully unrolled, i.e., maximally parallelized, as in previous results. We compare the BL logical synthesis results with the homogeneous and a Hessian-aware quantization model.

From Table 1, homogeneous quantization begins to decline below INT8, and we use this quantization scheme to compare to BL. Of the multiple Hessian-aware solutions, we choose the solution given by the lowest BOPs constraint, i.e., the quantization scheme is 4, 4, 5, and 4 bits for the first, second, third, and final output layers, respectively. Table 2 shows the synthesis results for three models: BL, INT8 homogeneous quantization, and the Hessian-aware solution. With INT8 homogeneous quantization, there is a significant reduction in DSPs compared to the BL model, which is further reduced with mixed Hessian-aware quantization. We expect that as bit width increases, more MAC operations will be implemented in DSPs, which offer a much more efficient implementation than LUTs and FFs. Interestingly, there is only a minor decrease of FFs with INT8 from BL, compared to the other resources, but this is mostly attributed to INT16 inputs and INT8 activations. Larger inputs with INT8 weights require a greater amount of FFs to store and accumulate computations, but their utilization drastically decreases with lower weight bit widths. The MLP with a Hessian-aware quantization scheme uses 42.2% fewer LUTs, 36.3% fewer FFs, and 95.7% fewer DSPs, compared to BL. As precision is reduced, the number of LUTs needed to compute outputs decreases. Most computations with lower precision can be implemented with LUTs; hence they have the strongest correlation with BOPs. However, this observed relationship weakens as the bit width increases and DSPs are used instead. The sudden uptick in LUTs and FFs is an outlier that originates from the softmax activation. As mentioned previously, the softmax activation stores precomputed outputs and the sudden surge comes from lookup tables created to store all values with large bit widths. Table 2 also includes the automatic mixed-precision solution from AutoQKeras [16], denoted QB, a QNN optimized by minimizing the size of the model in bits. The QB solution reduces all resource metrics by a substantial amount by employing below binary and ternary quantization, and activations are 4-bits. The advantages are also seen in latency while accuracy only drops by a tolerable ∼4%. However, the most significant improvement is the reduction in DSP usage compared to INT8. While the parameters have a width of 8 bits, activations require a higher level of precision. As a result, DSPs remain essential. In contrast to QB, the majority of parameters are limited to the values {−1, 0, 1}, where performing multiplications becomes significantly more straightforward using LUTs. In this study binary and ternary quantization was not explored as in AutoQKeras, but the total gains by leveraging mixed-precision are clearly shown.

The latency for these models, as estimated by Vivado HLS, is also shown in Table 2. Previous work demonstrated that the hardware implementation for this particular task aligns closely with the estimates provided by Vivado HLS [24]. Latency estimates are based on the specified clock, the loop transformations' analysis, and the design's parallelization. Pipelining and dataflow choices can greatly change actual throughput. However, the latency for the quantized models is about 30 ns longer than for BL. This can primarily be attributed to the additional scaling operations of the intermediate accumulations needed for lower precision quantities. Although the additional computation creates additional latency, the resources needed for these scaling layers are rather modest, approximately 1–3% relative to the rest of the design. So, there is a latency–resource tradeoff for the lower-precision computations. However, for the task at hand, the large reduction in resources is worth the increase in latency. The softmax activation is the other significant contributor to latency, with an estimated 10-ns runtime for all three quantized models presented in Table 2. As stated above, BRAMs are used for storing the precomputed outputs and the latency arises mainly from reading memory. Removing the softmax activation function from the implemented design is usually possible, especially if only the top-k classes are needed for further computation.

7 WORKFLOW WITH ADDITIONAL PHYSICS BENCHMARK

In addition to the jet tagging classifier, we demonstrate the end-to-end workflow on a second physics model planned for the high-granularity calorimeter in CMS [26] to show the generality

Table 2. Resource Usage of the Jet-tagging Model with Different Quantization Schemes Is Reported

Model	Acc. (%)	Latency (ns)	Resources			BOPs
			LUTs	FFs	DSPs	
Basline	76.85	65	60,272	15,116	3,602	4,652,832
INT8	76.45	95	54,888	14,210	671	281,277
Hessian	75.78	90	34,842	9,622	154	182,260
QB	72.79	60	16,144	4,172	5	122,680

Baseline (no quantization) achieves the highest accuracy with the most resources. The same bit-width quantization with INT8 reduces DSP and LUT usage, and Hessian-aware quantization significantly reduces all resource metrics. The mixed-precision model, QB, minimizing the total bits from AutoQKeras is shown. Both automatic solutions remove a considerable number of DSPs and LUTs needed for computations, and FFs to store intermediate accumulations. The Hessian is based on the ILP solution from the lowest BOPs constraint.

of the methodology. The **ECON-T Autoencoder (ECON-AE)** is a lossy data compression model aimed at reducing bandwidth needs while preserving critical information of the detector energy profile [20]. The NN architecture utilizes **convolutional layers (Conv)** to extract spatial features from image data, and **fully connected (FC)** layers. The training dataset is generated through simulated top-quark-pair events featuring 200 simultaneous collisions per bunch crossing within the CMS software framework. This dataset serves as a pragmatic surrogate, encapsulating typical energy patterns encountered in the high-granularity endcap calorimeter sensors. The performance of the autoencoder is evaluated based on how accurately it reproduces the original image after encoding and decoding. The **Earth mover's distance (EMD)** [56] is used to measure the difference between the raw and decoded data, where EMD quantifies the rearrangement cost of energy fractions when physically relocated. The EMD loss is not directly used in the algorithm training due to its computational complexity, a modified mean-squared-error loss function is employed, incorporating cell-to-cell distances, as an approximation of EMD, resulting in improved autoencoder performance as used in Reference [20]. Although ECON-AE is composed of encoder and decoder sections with the encoder ultimately targeting an ASIC responsible for compression and the decoder for an FPGA; we demonstrate our flow for comparison for an encoder FPGA implementation.

In Figure 8, the average Hessian Trace signifies greater importance to the first and final layers. Among these layers, the Conv and ConvTranspose layers emerge as particularly pivotal, playing a crucial role in both feature extraction and reconstruction processes. Such a high trace implies that small variations or perturbations in the input have a substantial impact on the loss function. We apply these traces to Equation (10) and set an ILP solver with various BOP limits as before. Table 3 shows the EMD, estimated latency, and final resource usage after place and route. The planned ECON-AE uses an ASIC design, here we use the same `xcu250-figd2104-2L-e` as before to simplify the workflow, again using the bare firmware design needed for the NN.

We train and compare three model scenarios for comparison (baseline, INT8, and Hessian), which are presented in Table 3. All resource and latency results are based on the encoder only. The baseline model is trained with FP32 weights then quantized to `ap_fixed<16,6>` for FPGA implementation. As seen in the jet tagging model, DSPs have the most noticeable impact on the firmware design, requiring a substantially smaller amount (75% and 82% for INT8 and Hessian, respectively) compared to the baseline. There is a shift to more logic cells for lower precision computations while retaining its EMD performance. Quantization can be seen as a regularizer preventing overfitting

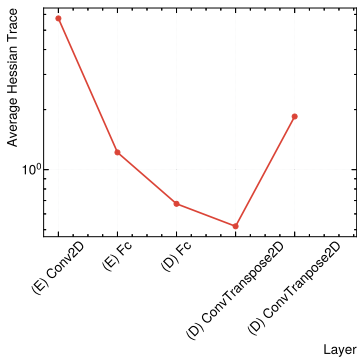

Fig. 8. The average Hessian trace of each layer is presented for the ECON-AE model. Encoder (E) and decoder (D) layers are also specified. As seen before, the first layer is the most "sensitive" to quantization with sharper minima and gradually declines.

Table 3. Performance Metrics for ECON-AE Model at Baseline, INT8, and Hessian Quantization are Reported

Model	EMD	Latency (ns)	Resources			BOPs
			LUTs	FFs	DSPs	
Baseline	1.29	95	48,107	27,147	1,696	732,375,268
INT8	1.15	100	73,078	36,690	408	79,421,668
Hessian	1.24	95	43,942	27,561	208	34,300,132

DSPs have the most noticeable impact on the firmware design, requiring 75% and 82% fewer resources for INT8 and Hessian, respectively. When comparing EMD, there's a 10% reduction with INT8 quantization, but this effect is less in lower precision Hessian models.

with INT8 models with a 10% EMD reduction; however, this effect is less in the lower precision Hessian but still comparable to the baseline EMD. The Hessian model is quantized to 4 and 8 bits for the Conv and FC layers, respectively. When comparing the INT8 and Hessian models, the impact is that less overall resources, LUTs and FFs, are required for the Conv and FC layers, and slightly more are required for scaling, which ultimately results in less resources for the Hessian case. The ECON-AE encoder is considerably smaller, consisting of only two layers, with larger activations requiring more FFs for registers and LUTs for scaling. Ultimately, the primary tradeoffs are latency and resource usage type, but careful considerations should be taken to the training procedure.

8 SUMMARY

The possible applications of HAWQ on edge devices and its automatic bit-setting procedure make it a valuable technique for physics research. In this article, we contributed to the HAWQ library by introducing an extension to convert NNs to ONNX and QONNX intermediate formats. Bridging HAWQ with firmware synthesis tools that ingest these formats makes it easier to deploy NNs to edge devices, enabling many potential use cases in science. While we emphasize FPGAs in the end-to-end workflow, the same procedural framework can equally applies to ASICs. Future studies targeting ASICs can do so with the HAWQ framework. As an initial case study, we employed a NN to classify jets at the LHC in a challenging benchmark task. We show that the Hessian-aware solution to a mixed precision quantization scheme provides a reliable solution. We then used our new exporter in HAWQ to translate multiple MLPs optimized with various bit settings into their QONNX IR. The models were successfully translated from HAWQ to a firmware implementation, and we have observed resource usage compared to total BOPs and accuracy. Furthermore, we compared the resource utilization for multiple different bit settings with the automatic bit selection process in Reference [22]; and we compared the Hessian-aware model with a homogeneous bit configuration and baseline. The Hessian-aware solution significantly reduced all resource metrics (LUTs, FFs, and DSPs), with the most significant improvements in DSPs and LUTs, using 95.7% and 42.2% fewer DSPs and LUTs compared to baseline, respectively. However, there are ways to enhance both total latency and hardware utilization with an emphasis on scaling factors. These scaling

factors, which are not quantized, may need 16+ bits for an accurate implementation, thereby contributing to the hardware cost. It is important to recognize scaling factors as dyadic, i.e., they can be efficiently implemented with integer multiplication and bit shifting. Additionally, scaling may be merged with the ReLU function as one hardware module. Currently, hls4ml cannot recognize scaling factors as dyadic and is left for future work. Finally we show the end-to-end workflow for a different physics task, on-detector data encoding, that includes convolutional layer types. The same workflow is compared against post-training quantization and uniform quantization-aware training, and we similarly find a reduction in resources using the automated Hessian approach.

REFERENCES

[1] Thea Aarrestad, et al. 2021. Fast convolutional neural networks on FPGAs with hls4ml. *Mach. Learn. Sci. Tech.* 2, 4 (2021), 045015. https://doi.org/10.1088/2632-2153/ac0ea1 arXiv:2101.05108 [cs.LG].

[2] ATLAS Collaboration. 2020. Operation of the ATLAS trigger system in Run 2. *J. Instrum.* 15, 10 (Oct. 2020), P10004.

[3] Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. ONNX: Open Neural Network Exchange. Retrieved from https://github.com/onnx/onnx

[4] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. 2021. MLPerf tiny benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1. Curran.

[5] Chaim Baskin, Natan Liss, Eli Schwartz, Evgenii Zheltonozhskii, Raja Giryes, Alex M. Bronstein, and Avi Mendelson. 2021. UNIQ: Uniform noise injection for non-uniform quantization of neural networks. *ACM Trans. Comput. Syst.* 37, 1–4, Article 4 (Mar. 2021), 15 pages.

[6] Biplob Bhattacherjee, Swagata Mukherjee, Rhitaja Sengupta, and Prabhat Solanki. 2020. Triggering long-lived particles in HL-LHC and the challenges in the rst stage of the trigger system. *J. High Energy Phys.* 08 (2020), 141. https://doi.org/10.1007/JHEP08(2020)141

[7] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfig. Technol. Syst.* 11, 3 (2018), 1.

[8] Anja Butter, et al. 2019. The machine learning landscape of top taggers. *SciPost Phys.* 7 (2019), 014.

[9] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13169.

[10] Jianfei Chen, Yu Gai, Zhewei Yao, Michael W. Mahoney, and Joseph E. Gonzalez. 2020. A statistical framework for low-bitwidth training of deep neural networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 883–894.

[11] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael W. Mahoney, and Joseph Gonzalez. 2021. ActNN: Reducing training memory footprint via 2-bit activation compressed training. In *Proceedings of the 38th International Conference on Machine Learning*. 1803–1813.

[12] François Chollet, et al. 2015. Keras. Retrieved from https://keras.io

[13] CMS Collaboration. 2020. *The Phase-2 Upgrade of the CMS Level-1 Trigger*. CMS Technical Design Report CERN-LHCC-2020-004. CMS-TDR-021.

[14] CMS Collaboration. 2022. *Neural Network-based Algorithm for the Identification of Bottom Quarks in the CMS Phase-2 Level-1 Trigger*. Technical Report CMS-DP-2022-021.

[15] Andrea Coccaro, Francesco Armando Di Bello, Stefano Giagu, Lucrezia Rambelli, and Nicola Stocchetti. 2023. Fast neural network inference on FPGAs for triggering on long-lived particles at colliders. *Machine Learning: Science and Technology* 4, 4 (2023), 045040. https://doi.org/10.1088/2632-2153/ad087a

[16] Claudionor N. Coelho, Aki Kuusela, Hao Zhuang, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nat. Mach. Intell.* 3, 8 (2021), 675.

[17] Evan Coleman, Marat Freytsis, Andreas Hinzmann, Meenakshi Narain, Jesse Thaler, Nhan Tran, and Caterina Vernierie. 2018. The importance of calorimetry for highly-boosted jet substructure. *J. Instrum.* 13, 01 (Jan. 2018), T01003.

[18] Allison McCarn Deiana, et al. 2022. Applications and techniques for fast machine learning in science. *Front. Big Data* 5 (2022), 787421. https://doi.org/10.3389/fdata.2022.787421

[19] Gage DeZoort, Savannah Thais, Javier Duarte, Vesal Razavimaleki, Markus Atkinson, Isobel Ojalvo, Mark Neubauer, and Peter Elmer. 2021. Charged particle tracking via edge-classifying interaction networks. *Comput. Softw. Big Sci.* 5 (2021), 1–13.

[20] Giuseppe Di Guglielmo, et al. 2021. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. *IEEE Trans. Nucl. Sci.* 68, 8 (2021), 2179–2186. https://doi.org/10.1109/TNS.2021.3087100 arXiv:2105.01683 [physics.ins-det].

[21] Zhen Dong, Yizhao Gao, Qijing Huang, John Wawrzynek, Hayden K. H. So, and Kurt Keutzer. 2021. Hao: Hardware-aware neural architecture optimization for efficient inference. In *Proceedings of the IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'21)*. IEEE, 50–59.

[22] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 18518–18529.

[23] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 293–302.

[24] Javier Duarte, et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *J. Instrum.* 13 (27 7 2018), P07027.

[25] Javier Duarte, Nhan Tran, Ben Hawks, Christian Herwig, Jules Muhizi, Shvetank Prakash, and Vijay Janapa Reddi. 2022. FastML science benchmarks: Accelerating real-time scientific edge machine learning. In *Proceedings of the 5th Conference on Machine Learning and Systems*.

[26] Javier Duarte, Nhan Tran, Ben Hawks, Christian Herwig, Jules Muhizi, Shvetank Prakash, and Vijay Janapa Reddi. 2022. FastML science benchmarks: Accelerating real-time scientific edge machine learning. arXiv:2207.07958. Retrieved from https://arxiv.org/abs/2207.07958

[27] Abdelrahman Elabd, et al. 2022. Graph neural networks for charged particle tracking on FPGAs. *Front. Big Data* 5 (23 3 2022). https://doi.org/10.3389/fdata.2022.828666

[28] Nicolò Ghielmetti, et al. 2022. Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml. *Mach. Learn. Sci. Tech.* (2022).

[29] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence*, G. K. Thiruvathukal, Y.-H. Lu, J. Kim, Y. Chen, and B. Chen (Eds.).

[30] Google. 2020. QKeras. Retrieved from https://github.com/google/qkeras

[31] Ekaterina Govorkova, et al. 2022. Autoencoders on FPGAs for real-time, unsupervised new physics detection at 40 MHz at the Large Hadron Collider. *Nat. Mach. Intell.* 4, 2 (1 2 2022), 154. https://doi.org/10.1038/s42256-022-00441-3

[32] Ekaterina Govorkova, Ema Puljak, Thea Aarrestad, Maurizio Pierini, Kinga Anna Woźniak, and Jennifer Ngadiuba. 2022. LHC physics dataset for unsupervised New Physics detection at 40 MHz. *Sci. Data* 9 (2022), 118. https://doi.org/10.1038/s41597-022-01187-8

[33] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. 152–159.

[34] Benjamin Hawks, Javier Duarte, Nicholas J. Fraser, Alessandro Pappalardo, Nhan Tran, and Yaman Umuroglu. 2021. Ps and Qs: Quantization-aware pruning for efficient low latency neural network inference. *Front. Artif. Intell.* 4 (2021), 676564.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770.

[36] Qijing Huang, Dequan Wang, Zhen Dong, Yizhao Gao, Yaohui Cai, Tian Li, Bichen Wu, Kurt Keutzer, and John Wawrzynek. 2021. Codenet: Efficient deployment of input-adaptive object detection on embedded fpgas. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 206–216.

[37] Yutaro Iiyama, et al. 2021. Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics. *Front. Big Data* 3 (12 1 2021), 44. https://doi.org/10.3389/fdata.2020.598927

[38] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Maching Learning, Volume 37 (ICML'15)*. JMLR, 448–456.

[39] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. 2020. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 497–511.

[40] Alex Karbachevsky, Cham Baskin, Evgenii Zheltonozhskii, Yevgeny Yermolin, Freddy Gabbay, Alex M. Bronstein, and Avi Mendelson. 2020. HCM: Hardware-aware complexity metric for neural network architectures. *CoRR* abs/2004.08906 (2020). arXiv:2004.08906 https://arxiv.org/abs/2004.08906

[41] Roman Kogler, et al. 2019. Jet substructure at the large hadron collider: Experimental review. *Rev. Mod. Phys.* 91, 4 (2019), 045003.

[42] Andrew J. Larkoski, Ian Moult, and Benjamin Nachman. 2020. Jet substructure at the large hadron collider: A review of recent advances in theory and machine learning. *Phys. Rept.* 841 (2020), 1–63.

[43] Joshua Lin, Marat Freytsis, Ian Moult, and Benjamin Nachman. 2018. Boosting $H \to b\bar{b}$ with machine learning. *J. High Energy Phys.* 10 (2018), 101. https://doi.org/10.1007/JHEP10(2018)101

[44] Xiaoxuan Liu, Lianmin Zheng, Dequan Wang, Yukuo Cen, Weize Chen, Xu Han, Jianfei Chen, Zhiyuan Liu, Jie Tang, Joey Gonzalez, Michael Mahoney, and Alvin Cheung. 2022. GACT: Activation compressed training for general architectures. In *Proceedings of the 39th International Conference on Machine Learning*. 14139–14152.

[45] Zechun Liu, Kwang-Ting Cheng, Dong Huang, Eric Xing, and Zhiqiang Shen. 2022. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'22)*.

[46] Vinicius Mikuni, Benjamin Nachman, and David Shih. 2022. Online-compatible unsupervised nonresonant anomaly detection. *Phys. Rev. D* 105, 5 (2022), 055006. https://doi.org/10.1103/PhysRevD.105.055006 arXiv:2111.06417 [cs.LG].

[47] S. Miryala, S. Mittal, Y. Ren, G. Carini, G. Deptuch, J. Fried, S. Yoo, and S. Zohar. 2022. Waveform processing using neural network algorithms on the front-end electronics. *J. Instrum.* 17, 01 (Jan. 2022), C01039. https://doi.org/10.1088/1748-0221/17/01/C01039

[48] Sandeep Miryala, Md Adnan Zaman, Sandeep Mittal, Yihui Ren, Grzegorz Deptuch, Gabriella Carini, Sioan Zohar, Shinjae Yoo, Jack Fried, Jin Huang, and Srinivas Katkoori. 2022. Peak prediction using multi layer perceptron (MLP) for edge computing asics targeting scientific applications. In *Proceedings of the 23rd International Symposium on Quality Electronic Design (ISQED'22)*. 1–6. https://doi.org/10.1109/ISQED54688.2022.9806285

[49] Eric A. Moreno, Thong Q. Nguyen, Jean-Roch Vlimant, Olmo Cerri, Harvey B. Newman, Avikar Periwal, Maria Spiropulu, Javier M. Duarte, and Maurizio Pierini. 2020. Interaction networks for the identification of boosted $H \to b\bar{b}$ decays. *Phys. Rev. D* 102 (28 7 2020), 012010. https://doi.org/10.1103/PhysRevD.102.012010 arXiv:1909.12285 [hep-ex].

[50] Maxim Naumov, Utku Diril, Jongsoo Park, Benjamin Ray, Jedrzej Jablonski, and Andrew Tulloch. 2018. On periodic functions as regularizers for quantization of neural networks. arXiv:1811.09862 [cs.LG].

[51] Jennifer Ngadiuba, et al. 2021. Compressing deep neural networks on FPGAs to binary and ternary precision with HLS4ML. *Mach. Learn. Sci. Tech.* 2 (2021), 015001.

[52] Alessandro Pappalardo, et al. 2022. QONNX: Representing arbitrary-precision quantized neural networks. arXiv:2206.07527 [cs.LG]. Retrieved from https://arxiv.org/abs/2206.07527

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Vol. 32. Curran Associates, Inc., 8024.

[54] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. 2020. hls4ml LHC Jet Dataset (30 Particles).

[55] Shah Rukh Qasim, Kenneth Long, Jan Kieseler, Maurizio Pierini, and Raheel Nawaz. 2021. Multi-particle reconstruction in the High Granularity Calorimeter using object condensation and graph neural networks. *EPJ Web Conf.* 251, 03072. https://doi.org/10.1051/epjconf/202125103072

[56] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vis.* 40 (2000), 99. https://doi.org/10.1023/A:1026543900054

[57] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. 2018. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc.

[58] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8815–8821.

[59] Albert M, Sirunyan, et al. 2020. Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques. *J. Instrum.* 15, 06 (Jun. 2020), P06005.

[60] Albert M, Sirunyan, et al. 2020. Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV. *J. Instrum.* 15, 10 (Oct. 2020), P10017. arXiv:2006.10165 [hep-ex].

[61] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, 65.

[62] Stylianos I. Venieris and Christos-Savvas Bouganis. 2019. fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 2 (2019), 326–342.

[63] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*. 8604. arXiv:1811.08886

[64] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of ConvNets via eifferentiable neural architecture search. arXiv:1812.00090. Retrieved from https://arxiv.org/abs/1812.00090

[65] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. 2020. HAWQV3: Dyadic neural network quantization. arXiv:2011.10680. Retrieved from https://arxiv.org/abs/2011.10680

Received 2 June 2023; revised 7 February 2024; accepted 21 April 2024